Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

Luis Muñiz Pasarín

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Bot de Telegram para la comunicación con PLCs de Beckhoff

FEBRERO DE 2018

Índice

1. Objetivos y alcance del trabajo

Este proyecto surge a raíz de la importancia que hoy en día está adquiriendo todo lo relacionado con la Industria 4.0, sobre todo en el ámbito de las comunicaciones y la interacción con las máquinas y procesos.

La causa del desarrollo de este trabajo es la problemática de la comunicación entre los PLCs y los usuarios. Para entender esto se deben tener en cuenta los comienzos de la automatización industrial, donde uno de los principales problemas era cómo establecer una comunicación entre los PLCs instalados y los operarios/ingenieros de las fábricas o de las empresas subcontratadas. Esto se solucionó con la incorporación de los HMI y los SCADA, que permiten visualizar valores, estados del proceso y en según qué casos, realizar modificaciones.

Conforme la tecnología fue avanzando, los dueños y dirigentes de las fábricas se dieron cuenta de que se estaban quedando muy obsoletos. Ya no solo bastaba con los HMI y SCADA a nivel de automatización, sino que en general debería haber una conexión entre todo el proceso y todos los sectores implicados. La conclusión evidente era que sus procesos y tecnologías se podían mejorar notablemente, traduciéndose esto en un posible aumento del rendimiento y de los ingresos de sus plantas.

A partir de la primera década del siglo XXI, algunas grandes marcas como Siemens, y otras más pequeñas, como Beckhoff, desarrollaron aplicaciones que instaladas en un ordenador o en una PDA permitían acceder remotamente a parte de la información contenida en los PLCs. También mejoraron su software de automatización, brindando la posibilidad de recibir avisos y alarmas del proceso vía SMS o con llamadas pregrabadas.

Aún así, todas estas soluciones que se han ido incorporando carecen de una característica clave sobre la que pivota este proyecto, que es la de lograr un modo sencillo, flexible e intuitivo de comunicación entre PLCs y trabajadores. Por lo tanto, se planteó la siguiente pregunta: ¿Cuál sería la forma más sencilla posible de comunicarse con un PLC?

Con el nacimiento y expansión de las redes sociales y las *apps* de mensajería, la respuesta a la pregunta anterior era clara: lo más sencillo sería poder chatear con los PLCs. Dentro de estas aplicaciones, Telegram es la única que ofrece una API para la programación de *bots*, que permiten realizar aplicaciones de todo tipo. Así mismo, por la parte industrial, Beckhoff es la única marca que ofrece una API de comunicación.

Por lo tanto, el objetivo principal de este proyecto es la creación de un *bot* de *Telegram* que permita comunicarse con los PLCs de Beckhoff, empleando un lenguaje más o menos natural. Esto implica que el usuario no tendrá la necesidad de escribir algo como: "Bot: Conectar PLC Ip… puerto… ". Si no que será posible que el *bot* entienda un lenguaje más natural, por ejemplo: "Hola, quiero conectarme al PLC con ip… y puerto… por favor".

Para lograr esta flexibilidad en el lenguaje, se creará un diccionario en el que cada fila contendrá una serie de palabras intercambiables entre sí. Cada operación que se pueda realizar con el *bot* irá asociada al reconocimiento de una determinada combinación de palabras de este diccionario. Además la aplicación se programará de tal manera que permita cambiar la posición de cualquiera de estas palabras "clave", sin que esto afecte al resultado del reconocimiento. De esta forma se aportará al usuario una libertad "controlada" a la hora de comunicarse con el *bot*.

Se partirá del estudio de las APIs mencionadas y se irán investigando las distintas posibilidades que ofrecen, hasta lograr un *bot* con unas características útiles y de sencillo manejo, que permita realizar tareas como la consulta y modificación de variables, consulta de estado del PLC, arrancar y parar el PLC, gestión de avisos y alarmas, creación de proyectos de Twincat (Software de programación de PLCs de Beckhoff) y otras funcionalidades que se consideren oportunas. Debido a la capacidad de Telegram de enviar mensajes en formato nota de voz, se estudiará la posibilidad de incorporar el reconocimiento de voz en la aplicación y se analizará su efectividad. Para esta parte se pretende lograr una forma de procesar el audio y convertirlo a texto, para que posteriormente el texto reconocido se trate como un mensaje normal.

Telegram, como otras *apps* de mensajería, también permite el envío de archivos a través de los chat. Por este motivo, como funcionalidad extra, se añadirá al *bot* la capacidad de gestionar los archivos enviados, almacenando su referencia en una base de datos, de forma que los usuarios puedan descargarlos en cualquier momento.

También se tendrá en consideración la premisa de sencillez y claridad, ya que no se pretende realizar una aplicación con mil opciones y que resulte compleja de utilizar.

Todo esto teniendo en cuenta que el proyecto surge como una idea de I+D, por lo que habrá objetivos a los que se podrá llegar, otros que se podrán conseguir parcialmente y algunos que se podrán plantear como ampliaciones del proyecto.

Para la consecución de estos objetivos será preciso pasar por las siguientes etapas

- **Estudio de las APIs de Beckhoff y Telegram**: Se estudiarán y analizarán las dos APIs para conocer las posibilidades que ofrecen y las características que se pueden implementar, cumpliendo con la premisa de sencillez y claridad para el usuario.

- **Programación del *bot* de Telegram**: Con las características a implementar fijadas y teniendo en cuenta que se deben seguir investigando las APIs en busca de funcionalidades interesantes a añadir, se procederá a la realización del programa. Una parte principal de este será el modo de reconocimiento de las órdenes del usuario, buscando la mayor naturalidad y flexibilidad posible.

- **Realización de programas en Twincat 3**: Como parte del proyecto, será necesario realizar uno o varios programas sencillos con los que probar las funcionalidades implementadas en el *bot*, para esto se dispone de un panel PC de Beckhoff en la empresa, que corre el software Twincat 3.

- **Análisis de resultados y posibles ampliaciones**: Una vez terminada la parte principal del proyecto, se estudiarán otras funciones interesantes a añadir, que pueden requerir de otras APIs o conceptos.

2. Contexto del proyecto

El proyecto está enmarcado en la denominada Industria 4.0 y la aplicación se ha realizado utilizando Telegram y Beckhoff. A continuación se describen brevemente la industria 4.0 y estas compañías.

2.1.- INDUSTRIA 4.0

Actualmente, las industrias del mundo están llevando a cabo, más o menos rápido, una gran actualización de sus fábricas y su forma de entender el proceso de producción. Esta importante renovación ha recibido el nombre de Industria 4.0.[1]

El concepto Industria 4.0[2] fue inicialmente introducido por el gobierno alemán en 2013, para describir una nueva visión de la fabricación, en la que todos sus procesos están interconectados mediante el Internet de las cosas (IoT). Esta renovación está introduciendo unos cambios tan profundos que ya se la conoce como cuarta revolución industrial. Debido a la innovación que supone y los posibles puestos de trabajo que puede generar, no solo es el sector privado el interesado en potenciar este sector, sino que las administraciones públicas también están promoviendo este nuevo modelo con subvenciones y programas de I+D.

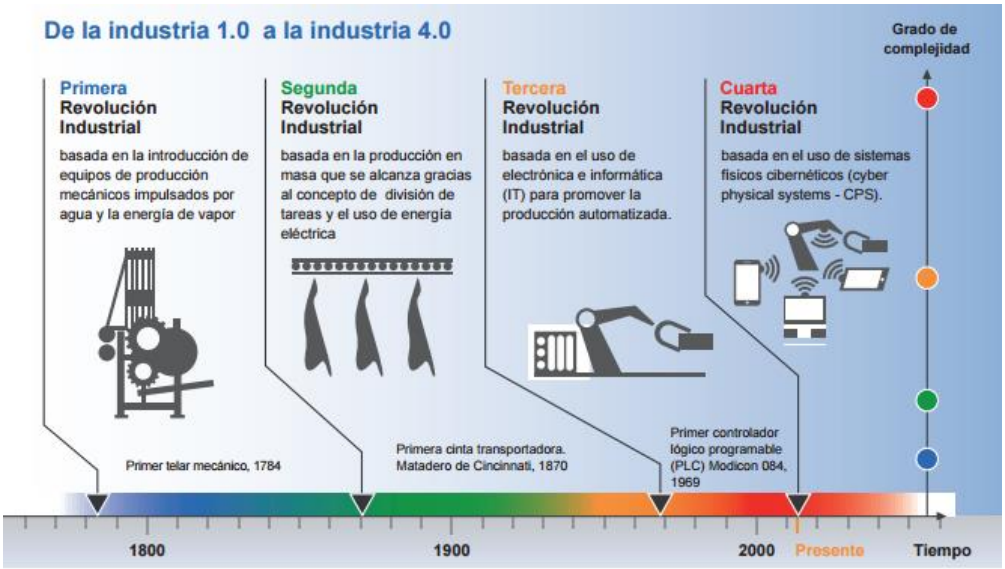


Figura 2.1.- Imagen que representa la evolución industrial desde la primera revolución industrial [3].

Se podría decir que la industria 4.0 se estructura en torno a seis tecnologías:

2.1.1.- IoT y Sistemas Ciberfísicos

El Internet of Things, *IoT*, se refiere al empleo de la intercomunicación de todo tipo de dispositivos dotados con capacidad de procesamiento a través del uso de internet, especialmente de forma inalámbrica, con el objetivo de controlar un determinado proceso físico. Estos dispositivos están conectados entre sí y a su vez con la red global.

2.1.2.- Fabricación 3D

La impresión 3D, ha permitido introducir la personalización total en la industria. Además ha hecho posible que sea muy sencillo producir lotes pequeños de productos, desde pequeñas piezas hasta grandes máquinas y diferentes prototipos.

2.1.3.- Big Data, Data Mining y Data Analytics

Hoy en día es cada vez más importante el tratamiento de la información, sobre todo a la hora de economizar y mejorar la rentabilidad de un determinado proceso productivo. El análisis de datos, obtenidos a través de sensores y detectores de distintos tipos, a lo largo de un proceso puede proporcionar información muy valiosa acerca del comportamiento del sistema analizado. Esta información puede servir para prevenir y prever problemas y averías, así como eventos adversos y su relación con las características de funcionamiento de la planta. Además, a partir de toda esta información se pueden realizar simulaciones que permiten predecir qué recursos van a ser necesarios en un futuro, pudiendo optimizar su uso de forma automática y proactiva.

2.1.4.- Inteligencia Artificial

Debido al gran volumen de datos generados a la hora de realizar Big Data, son necesarias herramientas y tecnologías que sean capaces de procesar en tiempo real esta enorme cantidad de información, así como algoritmos capaces de aprender de forma

autónoma a partir de la información que reciben (técnicas de Machine Learning, Deep Learning y Artificial Intelligence).

2.1.5.- Robótica Colaborativa

Con este término se hace referencia a una nueva generación de robots industriales, cuya principal novedad es la cooperación con humanos, sin las típicas restricciones de seguridad habituales en la mayoría de instalaciones robóticas actuales. Estos robots se caracterizan, entre otras cosas, por su flexibilidad, accesibilidad y relativa facilidad de programación.

Figura 2.2.- Ejemplo de robótica colaborativa[4]

2.1.6.- Realidad virtual y aumentada

Algunas de las principales funcionalidades de estas dos tecnologías es la del entrenamiento y la formación de los operarios de las fábricas. Gafas de realidad aumentada ya se están empleando en diversas industrias del mundo, facilitando la vida a los operarios, por ejemplo en el montaje o modificación cuadros eléctricos, o en el mantenimiento de máquinas o coches[5].

Figura 2.3.- Ejemplo de aplicación de realidad aumentada para el mantenimiento de coches.

2.2.- TELEGRAM

Telegram es una aplicación de mensajería desarrollada en el año 2013 por los hermanos Nikolai y Pavel Durov [6]. Debido a su éxito con VKontakte[7], la principal red social rusa y segunda en Europa por detrás de Facebook, no tuvieron la necesidad de enfocar Telegram como un proyecto comercial, si no que se centraron en la seguridad y en proporcionar una buena experiencia de usuario, sin introducir publicidad. De hecho, es administrada por una organización sin ánimo de lucro, cuya sede se encuentra en Berlín.

En Telegram los mensajes que se envíen pueden ser cifrados, si el usuario activa dicha opción, y no dejan ningún rastro en sus servidores, además se autodestruirán en un tiempo que cada usuario puede determinar. Aunque no se puede decir que la aplicación sea 100% segura, algo que nunca se puede garantizar, sí se puede afirmar que es un servicio fiable y relativamente superior a sus competidores.

Entre las características que la hacen distinta destacan:

- **Canales en Telegram**: Desde finales de 2015 Telegram comenzó a agregar canales a la aplicación. Estos canales tienen la función de informar o divulgar cualquier tipo de información. Existen canales públicos y privados, en estos últimos un administrador debe aceptar tu petición.

- **Bots:** Los *bots* son un potente recurso que ofrece Telegram para que cualquier persona o empresa pueda crear un sistema de interacción automática con los usuarios. Existen multitud de *bots*, como *bots* que informan del tiempo, deportes y otros. Telegram ofrece una API oficial muy completa para poder programarlos.

- **API de Telegram**: Facilita el desarrollo de múltiples contenidos, como los *bots*. Cualquier usuario puede acceder a ella y se va actualizando regularmente, por lo que las posibilidades a futuro son muy variadas.

- **Número de teléfono oculto**: Al crear una cuenta en Telegram, se nos asigna un nombre de usuario y ese será nuestro identificador en la red. Así, a diferencia de Whatsapp, cuando queramos chatear con una persona, no es necesario darle nuestro número de teléfono, si no que con nuestro nombre de usuario ya bastaría.

- **Telegram Web y escritorio**: Telegram ofrece la posibilidad de utilizar su servicio de mensajería tanto en el móvil como en su versión web o de escritorio. Gracias a que no requiere un número de teléfono móvil, se facilita esta posibilidad, ya que con un usuario y contraseña podemos conectarnos.

- **Integraciones con IFTTT**: Desde diciembre de 2017, Telegram incorporó la compatibilidad con el servicio de automatizaciones de tareas IFTTT. Esto permite realizar tareas como mandar correos, *tweets* y un largo etcétera desde Telegram.

- **Enviar vídeos de gran tamaño**: A diferencia de WhatsApp que solo permite enviar vídeos de hasta 16 MB, Telegram ofrece la posibilidad de compartir vídeos de hasta 1.5 Gb.

- **Grupos de 5.000 personas**: Los supergrupos de Telegram tienen una capacidad de hasta 5000 usuarios, lo que resulta muy práctico para enviar fácilmente información a miles de personas, garantizando la interacción entre los miembros.

- **Envío de distintos tipos de archivos**: En Telegram se pueden enviar archivos de cualquier tipo, como RAR, PDF, Torrents, etc… Esto contrasta con los servicios que ofrece WhatsApp, que solo permite enviar documentos, imágenes, vídeos, ubicación y contactos, además con un tamaño limitado por archivo.

Además, Telegram, en comparación con sus competidores, ofrece un menor consumo de datos. En la figura 2.4 se puede observar el consumo de Mb para 1 hora de uso de distintas aplicaciones de mensajería, habiendo enviado en todas ellas el mismo número de mensajes. Los términos “Bajo”, “Medio” y “Alto” hacen referencia al uso de la aplicación, partiendo desde un uso *light* (Bajo) hasta uno intensivo (Alto).

Aplicación	Bajo	Medio	Alto
	0.42	0.87	3.75
	0.65	1.39	6.23
	0.76	1.60	6.60
	0.77	1.59	5.84
	1.56	3.37	15.26
	2.05	4.14	13.94
	3.08	6.36	22.22

Figura 2.4. Consumos de aplicaciones de mensajería por hora [8]. *De arriba a abajo son: Telegram, WhatsApp, Google Hangouts, Line, Viber, Facebook Messenger y Skype.*

2.3.- BECKHOFF

Beckhoff[9] es una marca de PLCs que ofrece sistemas abiertos para la automatización de procesos. Su rango de productos abarca desde PCs industriales, módulos de entrada/salida y componentes Fieldbus, Drive Technology para Motion y software de control. Beckhoff dispone de multitud de productos que pueden emplearse en industrias de todo tipo.

Dentro del sector de la automatización, Beckhoff es una empresa que apuesta por la innovación, de hecho en la última versión de su software de automatización se proporciona la posibilidad de realizar la programación en C++ y está totalmente integrado con Visual Studio 2013/2015. También dispone de varias APIs para la comunicación y automatización de tareas, con posibilidad de programarlas en diferentes lenguajes.

Dentro de la industria 4.0, Beckhoff está apostando actualmente por pantallas multitáctiles y por la innovación en los buses de comunicación. Garantizan también la utilización de sus productos en atmósferas explosivas.

Figura 2.5 Últimas pantallas multitáctil de Beckhoff.

Desde su fundación en la década de los 80, Beckhoff ha evolucionado profundamente. En la actualidad cuenta con más de 3350 empleados alrededor del mundo, 34 filiales, 18 oficinas de ventas y más de 75 distribuidores.

Beckhoff ha ido mejorando su facturación año a año. En la figura 2.6 se puede observar la evolución anual de la empresa desde sus comienzos.

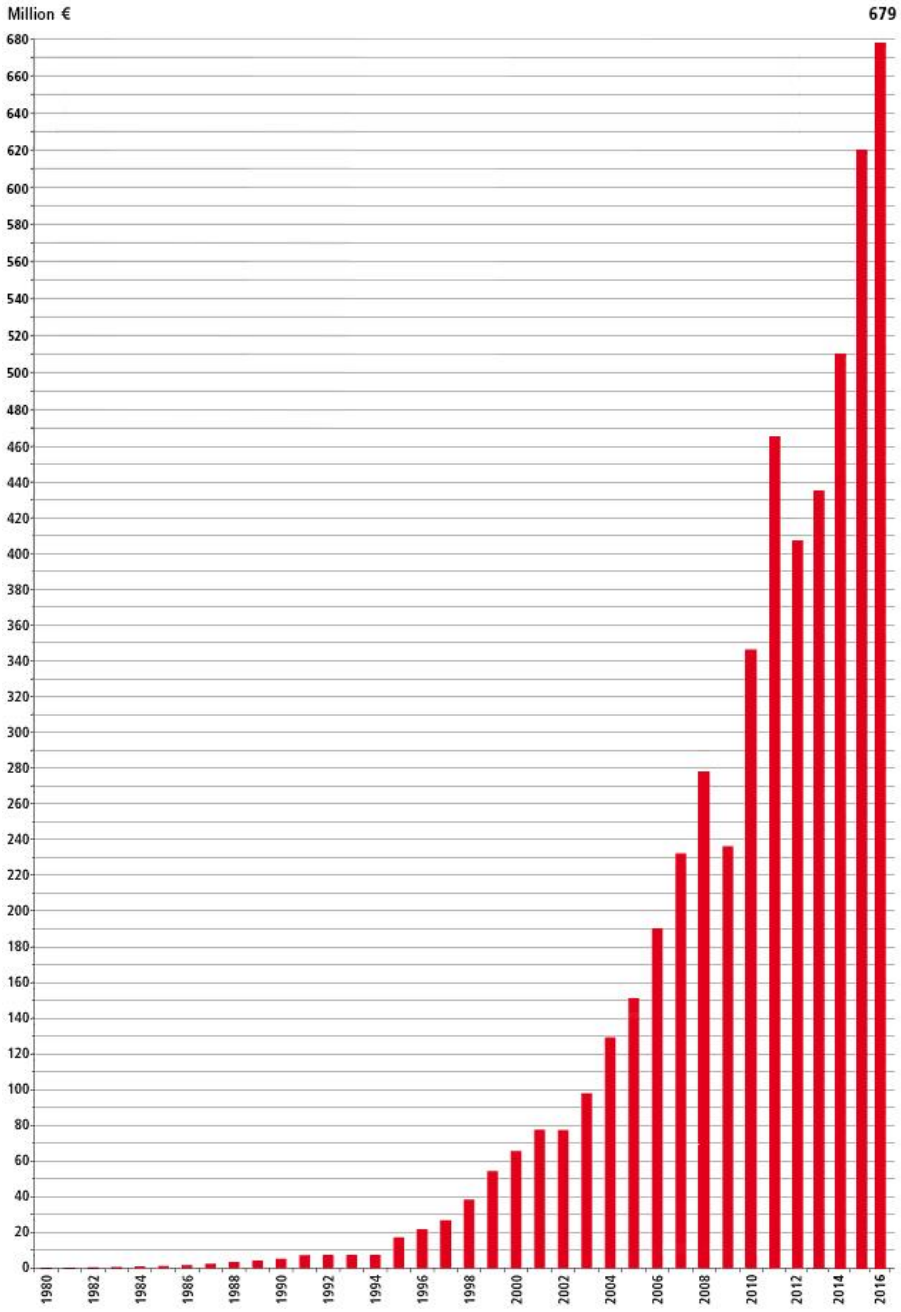


Figura 2.6 Evolución de la facturación, en millones, de Beckhoff desde los años 80.

3. Decisiones de diseño y desarrollo

3.1.- ELECCIONES INICIALES

A la hora de valorar las opciones a elegir para crear este tipo de aplicación, tan poco habitual, surgieron algunas preguntas, como *¿Por qué usar Telegram y Beckhoff? ¿Existe un bot que ya hace todo esto? ¿Qué opciones profesionales hay?*

Este proyecto nace como idea de I+D en la empresa Ivolt Proyectos e Instalaciones S.L., por lo que los requisitos en cuanto a las tecnologías a emplear han venido dados por las necesidades de dicha empresa. No obstante, esto no quiere decir que no haya una justificación clara al por qué de las decisiones tomadas.

Debido a su dilatada experiencia en el campo de la automatización industrial, los profesionales de Ivolt se han dado cuenta de que sería muy ventajoso disponer de algún tipo de plataforma o aplicación con la que comunicarse con PLCs o sobre la que realizar pequeñas pruebas, resulte sencillo, sobre todo a la hora de hacer comprobaciones rápidas en las instalaciones durante las puestas en marcha o mantenimientos, sin falta de usar un ordenador. Además se añadía el requisito o más bien, la prioridad, de que esta aplicación no estuviera sujeta a licencias ni ningún otro tipo de restricción ajena a la empresa. Esto, sumado a que una gran parte de los proyectos de la empresa se realizan utilizando Beckhoff, que a su vez cuenta con una API de comunicación disponible gratuitamente, sentaron las bases del proyecto desde el punto de vista industrial.

Con la marca de PLC seleccionada, faltaba la otra parte de la ecuación, la interfaz con la que establecer la comunicación con la API de Beckhoff. Como requisito del proyecto se había planteado que la aplicación a realizar fuera portable y sencilla de utilizar, buscando una interacción tipo conversacional. Fue en este momento donde se decidió apostar por los *bots* de Telegram, cuya API está totalmente accesible en la red y existe una gran cantidad de información.

En lo referente a los *bots* de Telegram, no hay ningún *bot* del que se tenga conocimiento que haya sido desarrollado con el mismo objetivo que el que se plantea en este trabajo. Como la naturaleza de Telegram es puramente informática, la mayoría de los *bots* se dedican a resolver tareas de ese campo, dejando de lado su potencial aplicación en otros sectores, como el industrial.

Con las bases del trabajo fijadas, lo siguiente fue decidir el entorno de programación. Se decidió utilizar Visual Studio por ser uno de los entornos más conocidos, por tener el estudiante conocimientos sobre este y porque Twincat 3, el software de programación de Beckhoff, está integrado con Visual Studio, por lo que a la hora de realizar pequeños programas para hacer pruebas, no sería necesario utilizar otro entorno.

En cuanto al lenguaje de programación utilizado, había diversas opciones: Java, C++, Html, C#, etc. Finalmente se optó por C#, ya que es un lenguaje con el que los trabajadores de Ivolt están familiarizados y podrían ayudar con las dudas o problemas que fueran surgiendo. Además, se decidió utilizar C# con vistas a futuros proyectos de la empresa, en los que cada vez se está utilizando más este lenguaje.

3.2.- PLANTEAMIENTO GENERAL DEL PROGRAMA

Como se ha mencionado anteriormente, para esta aplicación se van a utilizar Beckhoff y Telegram. Inicialmente el proyecto se planteó en torno a dos API, una de Beckhoff para la comunicación y otra de Telegram para la interacción, aunque posteriormente, debido a características añadidas se fueron incluyendo más APIs y librerías externas. En este apartado se pretende justificar las decisiones iniciales tomadas.

Llegado este punto, la pregunta que puede tenerse en la cabeza es la siguiente: ¿Qué es exactamente lo que se va a utilizar de Beckhoff y Telegram?

Beckhoff ofrece gratuitamente una API de comunicación llamada “Twincat ADS (*Automation Device Specification*)[10]”. Esta API permite establecer una conexión remota con cualquier PLC de Beckhoff compatible y obtener información de este. Con esta API se

pueden leer variables del PLC, obtener su estado, se puede parar y arrancar el PLC, etc… Su uso no requiere ninguna licencia y está disponible en varios lenguajes de programación (se usa C#). Esta API, a diferencia de la de Telegram, no resulta sencilla de entender ni utilizar, pero en la página oficial de ayuda[11] de Beckhoff hay una gran cantidad de información.

Para la parte de Telegram se va a utilizar su API de *bots*[12]. Esta API permite realizar todo tipo de *bots* y además cuenta con la ventaja de que está muy bien documentada y se actualiza regularmente con nuevas características. El intercambio de mensajes con el *bot* se realiza a través de peticiones HTTPS a los servidores de Telegram, por lo que una conexión a internet es necesaria.

Tras decidir las herramientas a utilizar, lo siguiente fue plantear la estructura general de cómo se iba a organizar el programa. Debido a las distintas funcionalidades que se fueron añadiendo al programa, la organización y planteamiento del código fue cambiando. El principal problema de cara a cómo diseñar el programa fue decidir cómo se iba a realizar el reconocimiento de texto, así como las valoraciones acerca del reconocimiento de voz, que es uno de los extras añadidos a la aplicación.

El planteamiento inicial era identificar directamente las palabras que el usuario introducía y compararlas con una serie de frases previamente definidas, para identificar lo que el usuario quería hacer.

Tras mejorar los conocimientos de C# se decidió que sería mejor comprobar la existencia de ciertas palabras clave en el mensaje recibido, según cada posible operación, para lograr esa flexibilidad que se buscaba en un principio y evitar que la interacción fuera del estilo de MS-DOS (ejemplo: Comando 1=”Conectar Ip…Puerto…”).

Conforme se fueron añadiendo funcionalidades y adquiriendo nuevos conocimientos, el planteamiento de cómo se debían reconocer los mensajes fue cambiando. Lo mejor sería poder abstraerse de si el mensaje contiene tildes o mayúsculas/minúsculas y tener una forma de reconocer varias palabras como iguales, por ejemplo que “conectar”, “conecta” y “conéctame” devuelvan el mismo resultado. Fue en este punto donde se tomó

la decisión de crear un diccionario en el que las filas representen un conjunto de palabras intercambiables entre sí, para proporcionar al usuario una cierta flexibilidad en la escritura o dictado de mensajes, tal y como se ha descrito en los objetivos del proyecto. Por estos motivos, se determinó que lo mejor sería que toda la parte de procesamiento del lenguaje estuviera implementada en una librería dinámica (dll), que se pudiera ir actualizando sin falta de modificar el programa principal.

En general, el funcionamiento principal del programa se puede entender fácilmente con la figura siguiente:

Figura 3.1. Esquema del funcionamiento general del programa.

En cuanto al reconocimiento de voz, el algoritmo de la figura 3.1 es aplicable, salvo que entre la fase 1 y 2 habría una nueva etapa que sería la del procesamiento del audio recibido. La decisión que se tomó fue que tras recibir una nota de voz por el *bot*, este audio

se convirtiera a texto para posteriormente pasarlo a la librería de reconocimiento del lenguaje.

Para la parte de convertir audio a texto se decidió se decidió utilizar la librería que trae por defecto Visual Studio, Speech Recognition. Esta decisión se tomó en base a criterios económicos, ya que la librería es gratuita y también porque para el estado al que se pretendía llegar del proyecto, lo que se planteaba era estudiar si era factible introducir esta funcionalidad en el programa, para posteriormente si se siguiera desarrollando el trabajo, añadir una librería o framework más versátil y preciso.

Además, el formato de las notas de voz obligaba a un pre-procesamiento del audio, ya que se generan en .ogg y la librería no admite ese tipo. Se decidió convertirlas a .wav y para ello se recurrió al empleo de FFmpeg [19], una aplicación de conversión de formatos que se llama desde la aplicación desarrollada en Visual Studio.

3.3.- OTRAS ALTERNATIVAS

Este proyecto, al no ser muy común, no cuenta con una alternativa clara en cuanto a las herramientas a utilizar. En el caso de Beckhoff, es la única marca de PLCs que ofrece una API oficial de comunicación gratuita y Telegram es la única herramienta de mensajería con una API que permite la creación de *bots*.

Las alternativas de este proyecto se encuentran más en la parte de programación y algoritmia, sobre todo en lo que concierne al reconocimiento de texto y voz.

En cuanto al reconocimiento de texto, se valoraron otras opciones como el uso de la librería “Speech Recognition”[14] de Microsoft, que permite identificar una serie de palabras prefijadas en una cadena de texto. Este método es muy similar a una de las ideas iniciales descartadas durante la fase de diseño del programa, mencionada anteriormente. El problema de esta opción es que no permite introducir una cierta flexibilidad a la hora del reconocimiento, es una comparación igual a igual, es decir, si se busca la palabra “casa”,

identifica si esa palabra exacta existe en la cadena de texto, pero no admite variaciones. Por ejemplo no reconocería "Casa" como una opción válida.

También se analizó la posibilidad de realizar el reconocimiento aplicando machine learning, para lo cual existe, entre otras opciones, el framework "Accord.Net"[15]. Debido a la complejidad que acarreaba y al tiempo necesario para utilizarlo correctamente se descartó su uso por el momento. No obstante, es una de las opciones que se tienen en reserva de cara a futuras mejoras del proyecto.

Para la parte de reconocimiento de voz, se valoró el uso de la librería "API Speech" de Google[16], que es una de las mejores que existen para esta tarea, debido al uso de potentes redes neuronales y complejos algoritmos desarrollados por Google, que permiten reconocer el contexto del usuario e incluso procesar correctamente el mensaje en situaciones ruidosas.

Otra API muy buena para el reconocimiento de voz es "Bing Speech" de Microsoft[17], que al igual que la de Google incorpora redes neuronales. Además, cuenta con tres modos: Dictado, Interacción y Conversación. Según qué modo se haya seleccionado mejorará el reconocimiento en dicha situación.

Estas dos últimas opciones, aunque hubieran aportado muy buenos resultados, se descartaron por tener un coste económico asociado, una por límite de transacciones y otra por el uso de los servidores *cloud* a la hora de procesar los audios. Sin embargo, si el proyecto se sigue desarrollando, en la empresa tienen la intención de posiblemente pagar por una de las dos opciones para integrarla en el programa.

4. Creación de un *bot* de Telegram

Este proyecto, como ya se ha mencionado anteriormente y como su título indica, se basa en la creación de un *bot* de Telegram para la comunicación con PLCs. Por lo tanto, una de las primeras cosas que se tuvo que hacer fue crear un *bot*.

Aunque suene redundante, para crear un *bot* hay que utilizar otro *bot*, llamado BotFather. Este *bot* actúa como "Master" de todos los *bots* que cada usuario cree en Telegram. Para crear un nuevo *bot* hay que introducir en un chat con BotFather el comando "/newbot". A continuación se tiene que introducir un nombre y un nombre de usuario para el Bot, que se pueden modificar a posteriori. Por último, se generará automáticamente un *token*, una clave alfanumérica única e identificativa del *bot* que se ha creado, que es lo que permitirá comunicarse con él a través de la aplicación de Visual Studio.

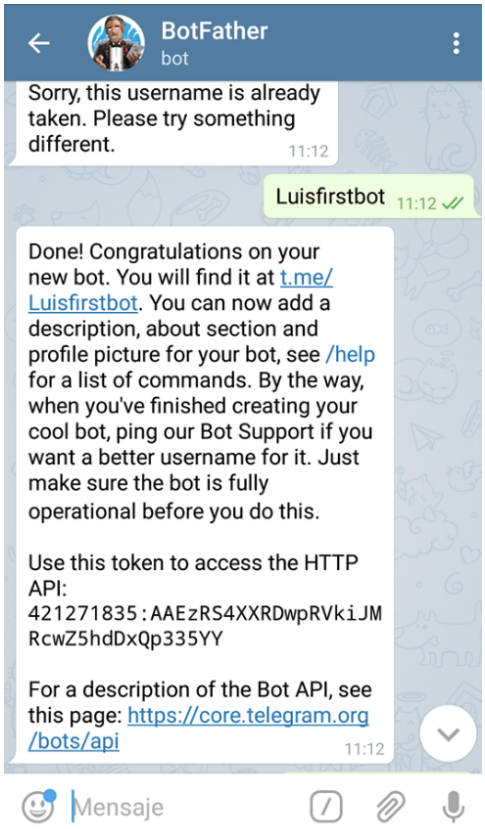


Figura 4.1. BotFather proporciona un token único e identificativo de cada bot creado por el usuario.

Al *bot* creado para este proyecto se le ha asignado por el momento el nombre "Pruebabot" y el nombre de usuario "Luisfirstbot". Además, se le puede añadir al *bot* una descripción, un mensaje inicial y un icono, entre otras características. Con el *bot* creado lo siguiente es empezar una conversación con él, para ello se le da al botón "Iniciar" cuando se tenga una conversación abierta con él.

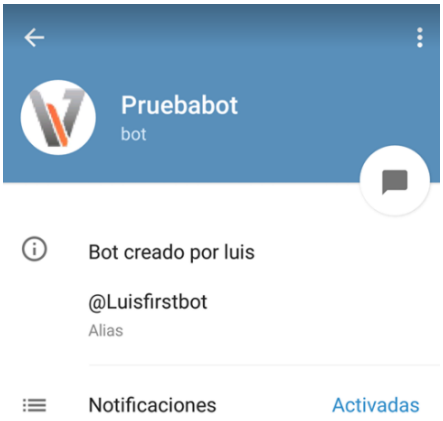


Figura 4.2. Pantalla de información sobre el bot creado.

Figura 4.3. A la izquierda, pantalla que se visualiza cuando se inicia una conversación con un bot. A la derecha, estado incial del chat tras haber pulsado en Iniciar.

5. Análisis del software

A lo largo de este apartado se describe toda la aplicación desarrollada en Visual Studio, mostrando la forma en la que se ha programado, los métodos utilizados y la relación entre las clases, así como los programas de prueba creados en Twincat 3. También se detalla al final un manual de uso de la aplicación.

Antes de comenzar con un análisis más profundo, en la figura 5.1 se puede observar un esquema general del funcionamiento de la aplicación desarrollada.

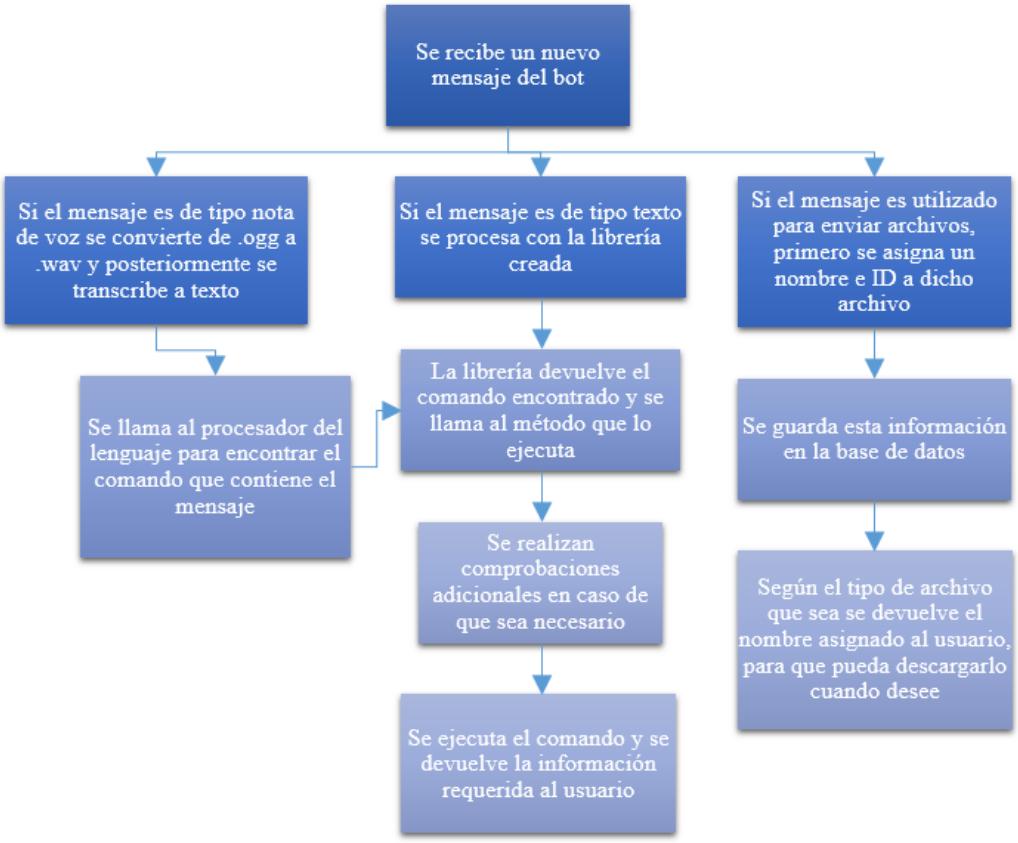


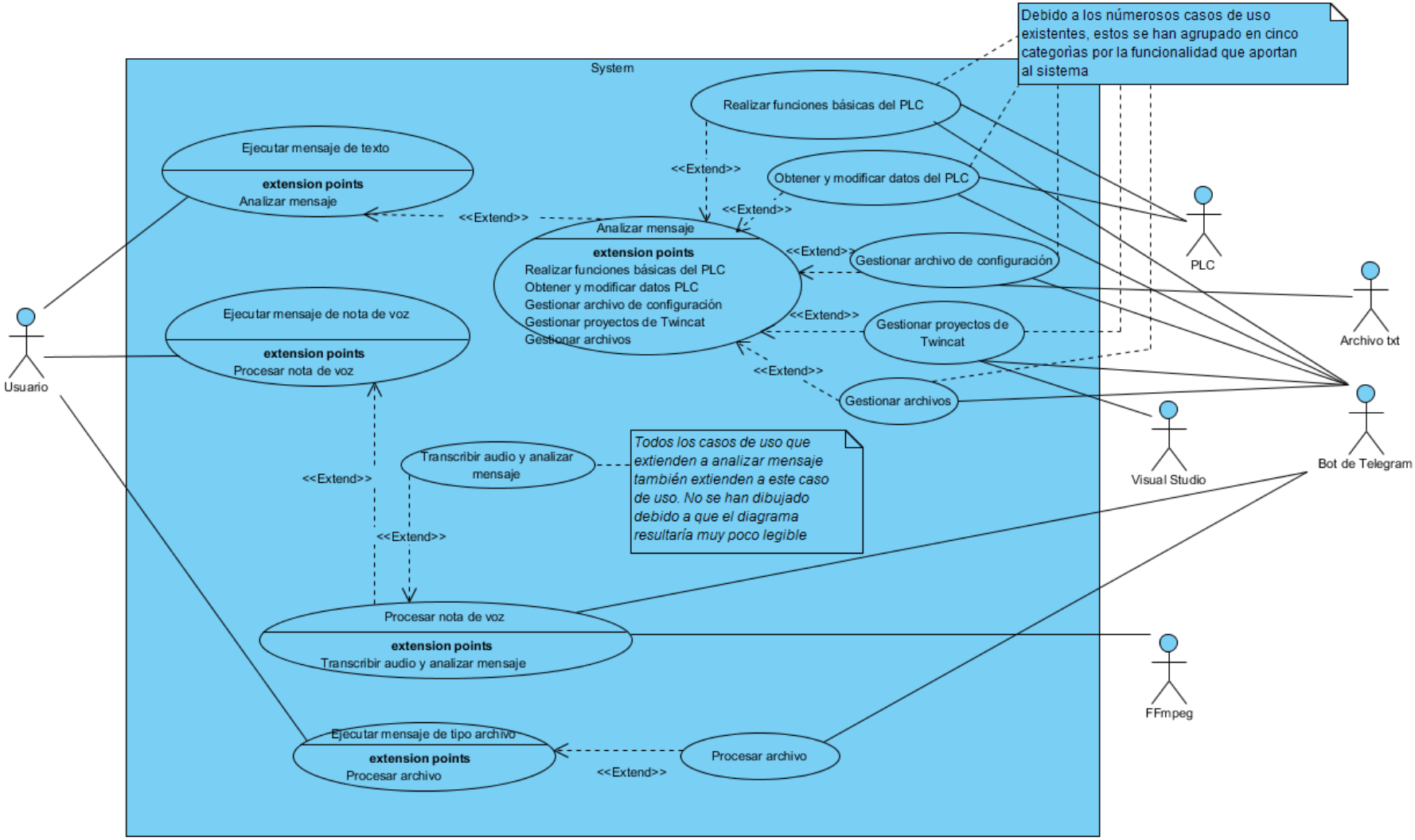
Figura 5.1.- Esquema general del funcionamiento del programa.

5.1.- CASOS DE USO

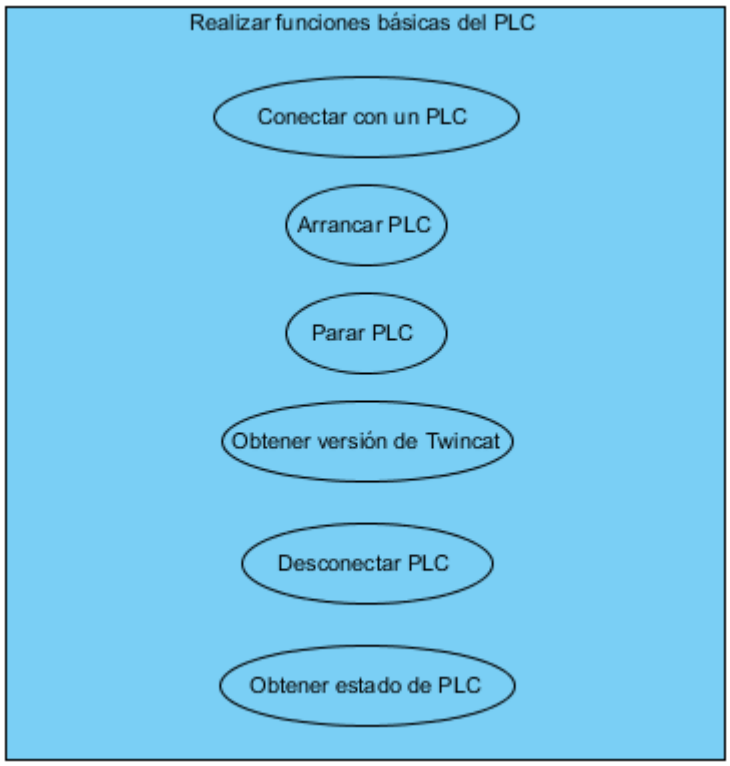
Cada usuario del *bot* puede realizar hasta 29 acciones distintas, agrupadas en tres categorías según el mensaje que se haya enviado al *bot* (texto, voz, archivo). A lo largo de este sub-apartado se describirá cada una de las operaciones posibles.

En la página siguiente se puede observar el esquema general de los casos de uso de la aplicación. Debido al número de operaciones que el usuario puede realizar, los casos de uso del *bot* se han agrupado en cinco categorías: Realizar funciones básicas del PLC, obtener y modificar datos del PLC, gestionar archivo de configuración, gestionar proyectos de Twincat y gestionar archivos.

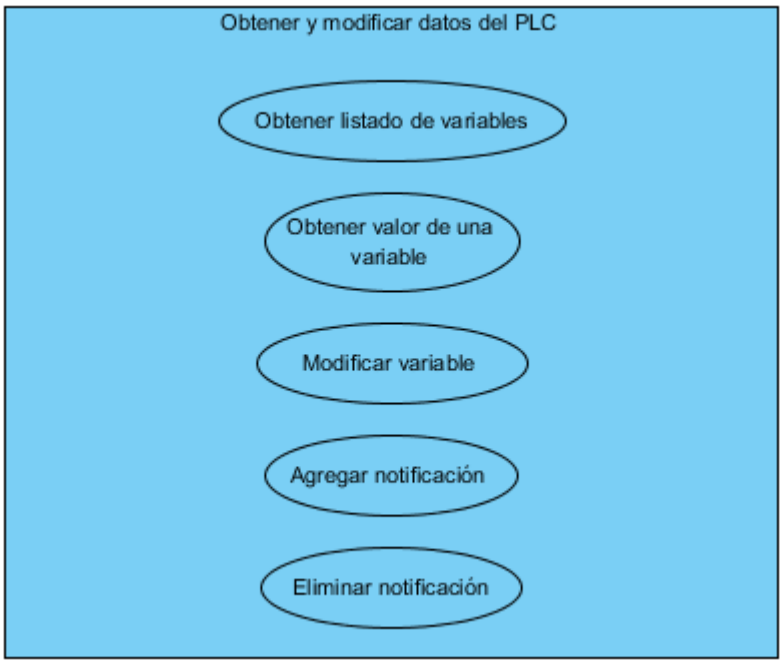
5.1.1.- Diagrama general de casos de uso



5.1.2.- Diagrama de casos de uso contenidos en "Realizar funciones básicas del PLC"

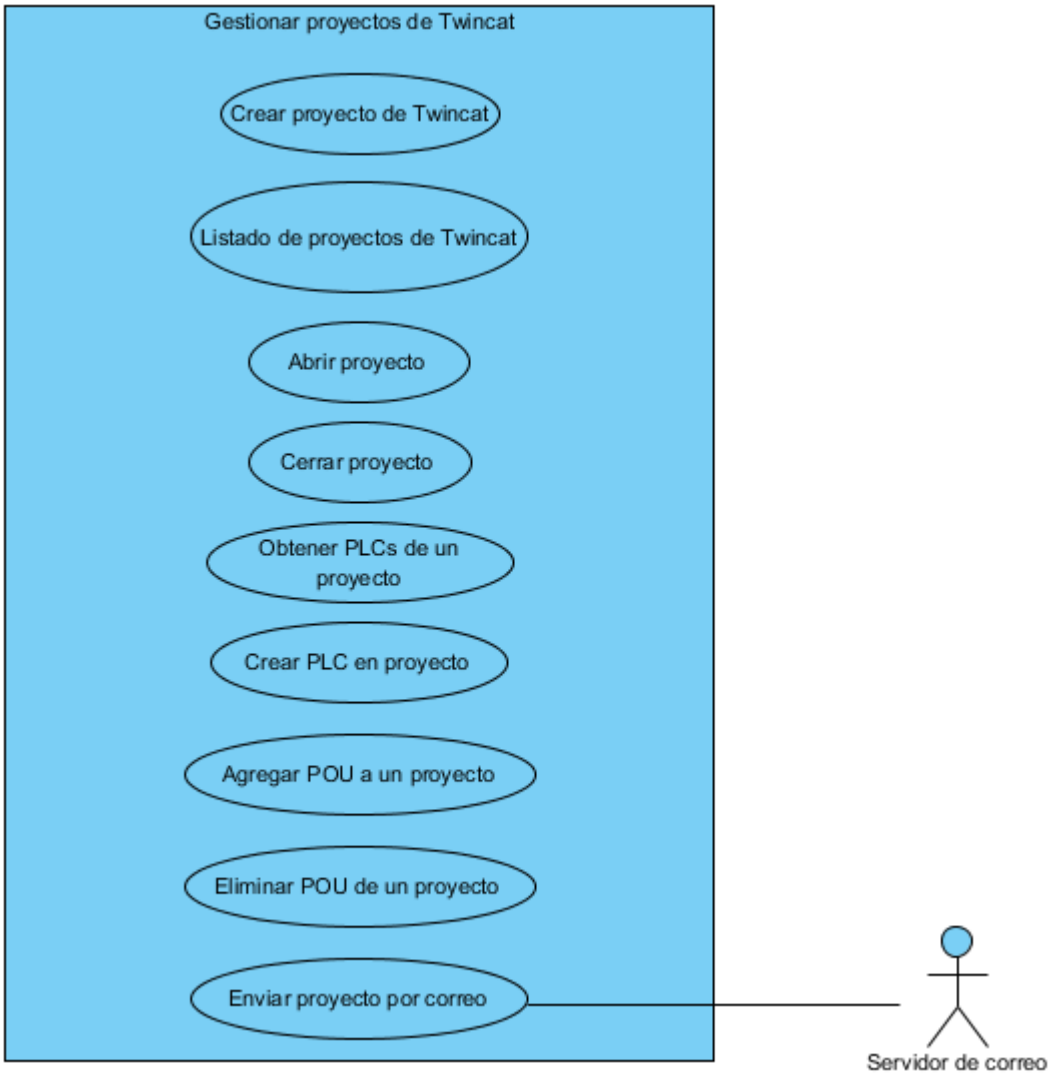


5.1.3.- Diagrama de casos de uso contenidos en "Obtener y modificar datos del PLC"

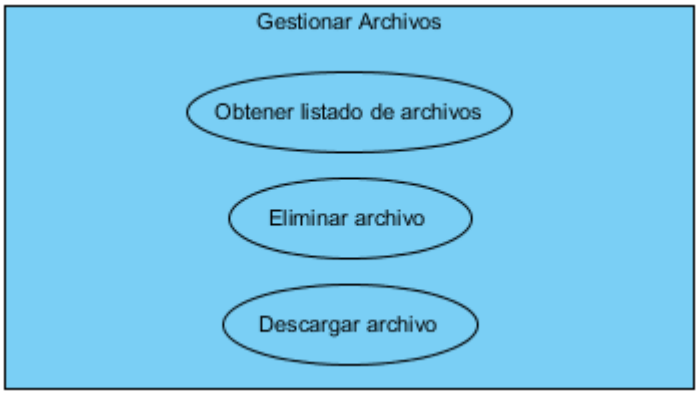


5.1.4.- Diagrama de casos de uso contenidos en "Gestionar archivo de configuración"

5.1.5.- Diagrama de casos de uso contenidos en "Gestionar proyectos de Twincat"



5.1.6.- Diagrama de casos de uso contenidos en "Gestionar archivos"



5.1.7.- Ejecutar mensaje de texto

Caso de uso: Ejecutar mensaje de texto	
Actores que intervienen	Usuario, *bot* de Telegram
Precondición	Tener descargada la aplicación de Telegram
Resultado	Se envía un mensaje de texto al *bot*
Descripción	El usuario abre una conversación con el *bot* de Telegram. En el apartado de texto escribe una o varias frases. Finalmente da al botón de enviar y el *bot* recibe el mensaje

Escenario principal

1. El usuario abre un chat con el *bot*.
2. Se pulsa sobre el apartado de escritura y se teclean una o varias palabras/frases.
3. Se da al botón de enviar.
4. El *bot* recibe el mensaje y esto provoca la activación de un evento en la aplicación desarrollada, que recibe el mensaje enviado.

5.1.8.- Ejecutar mensaje de nota de voz

Caso de uso: Ejecutar mensaje de nota de voz	
Actores que intervienen	Usuario, *bot* de Telegram
Precondición	Tener descargada la aplicación de Telegram
Resultado	Se envía una nota de voz al *bot*
Descripción	El usuario abre una conversación con el *bot* de Telegram y pulsa sobre el icono de adjuntar nota de voz. Mantiene el icono pulsado mientras esté grabando el mensaje. Finalmente la nota de voz se envía al *bot*

Escenario principal

1. El usuario abre un chat con el *bot*.
2. Se pulsa sobre el icono de notas de voz.
3. Se mantiene pulsado el icono durante el tiempo que se desee grabar el mensaje.
4. Cuando se deja de pulsar el icono la nota de voz es enviada al *bot*, lo que dispara un evento en el programa, que recibe el audio enviado.

5.1.9.- Ejecutar mensaje de tipo archivo

Caso de uso: Ejecutar mensaje de tipo archivo	
Actores que intervienen	Usuario, *bot* de Telegram
Precondición	Tener descargada la aplicación de Telegram
Resultado	Se envía un archivo al *bot*
Descripción	El usuario abre una conversación con el *bot* de Telegram. Se pulsa sobre el icono para adjuntar archivos y se selecciona un archivo del móvil u ordenador desde el que se esté ejecutando Telegram. Tras añadir el archivo, se pulsa el botón de enviar para mandar el archivo al *bot*

Escenario principal

1. El usuario abre un chat con el *bot*.
2. Se pulsa sobre el icono de adjuntar archivos.
3. Se selecciona el archivo a adjuntar y una vez decidido se pulsa sobre el botón de enviar.
4. El *bot* recibe el archivo enviado por el usuario, lo que dispara un evento del programa desarrollado, que recibe este archivo.

5.1.10.- Analizar mensaje

Caso de uso: Analizar mensaje	
Actores que intervienen	Usuario, *bot* de Telegram
Precondición	Se debe haber enviado al *bot un mensaje* de tipo texto
Resultado	Se obtiene el comando a ejecutar, si lo hubiera
Descripción	Tras recibir un nuevo mensaje de texto, el programa llama a una librería creada, que se encarga de procesarlo para obtener el comando que contiene

Escenario principal

1. El mensaje recibido se envía a la librería de procesamiento del lenguaje.
2. La librería elimina las tildes que el texto pudiera contener y se comparan las palabras del mensaje con las de un diccionario, formado por una serie de palabras “clave” asociadas a los comandos que el *bot* puede reconocer.
3. Se obtiene una secuencia de números, que representa las filas del diccionario en las que se ha encontrado una palabra del mensaje.
4. La secuencia de números se analiza y se obtiene el comando asociado.

5.1.11.- Procesar nota de voz

Caso de uso: Procesar nota de voz	
Actores que intervienen	Usuario, *bot* de Telegram, FFmpeg
Precondición	Se debe haber enviado un mensaje al *bot* de tipo nota de voz
Resultado	Se obtiene el archivo de la nota de voz en formato wav.
Descripción	El usuario envía una nota de voz al *bot*. Tras recibirla, el programa desarrollado descarga la nota de voz, que por defecto está en formato .ogg. Una vez descargada se llama a FFmpeg, un programa externo, para que realice la conversión a .wav

Escenario principal

1. El usuario envía una nueva nota de voz al *bot*.
2. Tras recibirla, se llama a un método que se encarga de descargar la nota de voz.
3. La nota de voz descargada se convierte al formato .wav gracias al uso de FFmpeg.

5.1.12.- Transcribir audio y analizar mensaje

Caso de uso: Transcribir audio y analizar mensaje	
Actores que intervienen	Usuario, *bot* de Telegram
Precondición	Se ha recibido una nota de voz y se ha convertido a formato .wav
Resultado	Se obtiene la transcripción del audio recibido a texto
Descripción	Tras recibir una nota de voz por el *bot* y convertirla a .wav, se llama a la librería “Speech Recognition” para realizar el reconocimiento de audio a texto. Tras configurar los parámetros oportunos se obtiene una transcripción a texto de la nota de voz y con la librería de procesamiento se obtiene el comando que contiene el mensaje.

Escenario principal

1. Se recibe una nueva nota de voz y se convierte al formato .wav.
2. Se utiliza el motor de reconocimiento de voz por defecto para obtener la transcripción a texto de la nota de voz.
3. El texto reconocido se pasa como parámetro a la librería de reconocimiento del lenguaje.
4. Se procesa el texto y se obtiene el comando encontrado, si lo hubiera.

5.1.13.- Procesar archivo

Caso de uso: Procesar archivo	
Actores que intervienen	Usuario, *bot* de Telegram
Precondición	Se debe haber enviado un mensaje al *bot* de tipo archivo
Resultado	La referencia del archivo se añade a la base de datos
Descripción	Tras recibir un archivo con el *bot*, se le asigna un nombre y un identificador y se agrega a la base de datos creada.

Escenario principal

1. Se recibe un nuevo archivo por el *bot*.
2. Se llama a un método que se encarga de asignar un nombre e identificador al archivo.
3. Se guarda la referencia al archivo en la base de datos creada.
4. El *bot* manda un mensaje al usuario en el que le informa del nombre asignado al archivo enviado, para que pueda descargarlo cuando quiera.

5.1.14.- Conectar con un PLC

Caso de uso: Conectar con un PLC	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	El *bot* debe haber recibido un nuevo mensaje y este haberse procesado, obteniendo la orden de conectarse a un PLC. El PLC de Beckhoff debe estar encendido y conectado a la red.
Resultado	Se establece la conexión con el PLC de Beckhoff
Descripción	El usuario introduce un mensaje solicitando la conexión a un PLC, especificando su dirección Ams y puerto, o indicando un nombre de PLC guardado en el archivo de configuración. Tras procesar el mensaje y comprobar que la Ams y puerto del PLC son válidos, se establece la conexión con el PLC y el *bot* devuelve un mensaje al usuario indicando que la operación se ha realizado con éxito.

Escenario principal

1. El usuario ha enviado un nuevo mensaje al *bot* y se ha procesado, obteniendo un comando que indica que se ha solicitado la conexión con un PLC.
2. Se llama al método para conectarse a un PLC.
3. Se comprueba que el mensaje contenga una dirección Ams y puerto.
4. Se comprueba que la Ams y puerto tengan una estructura válida. La Ams tiene que tener el formato de una Ip y el puerto un número de 1 a 4 dígitos.
5. Se establece la conexión con el PLC de Beckhoff.
6. El *bot* manda un mensaje al usuario diciéndole que se ha establecido la conexión.

Escenarios alternativos

3a. El mensaje no tiene una dirección Ams ni puerto, si no el nombre de un PLC.

1. Se busca el nombre introducido en el archivo de configuración.
    1a. El PLC solicitado no existe en el archivo de configuración.
        1. El *bot* informa sobre esto al usuario y queda a la espera de recibir un nuevo mensaje.
2. Se accede al archivo de configuración y se obtiene la dirección Ams y puerto del PLC solicitado.
3. El flujo continúa en el paso 4.

4a. La dirección Ams o puerto no tienen el formato adecuado.

1. El *bot* manda al usuario un mensaje informativo y se queda a la espera de un nuevo mensaje.

5a. La dirección Ams y puerto son sintácticamente correctos pero no se logró establecer una conexión con un PLC.

1. El *bot* informa al usuario de que la configuración es correcta pero no pudo establecer la conexión.
2. El *bot* queda a la espera de un nuevo mensaje.

5.1.15.- Obtener el listado de variables de un PLC

Caso de uso: Obtener el listado de variables de un PLC	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	El *bot* debe haber recibido un nuevo mensaje y este haberse procesado, obteniendo el comando de devolver el listado de variables del PLC. Se debe tener una conexión establecida con el PLC de Beckhoff
Resultado	Obtención del listado de variables del PLC
Descripción	El comando identificado es el de solicitar el listado de variables del PLC. Tras comprobar que se tiene una conexión establecida, el *bot* devuelve al usuario el listado de variables del PLC.

Escenario principal

1. El usuario ha enviado un nuevo mensaje al bot y se ha procesado, obteniendo un comando que indica que se ha solicitado el listado de variables del PLC.
2. Se llama al método para devolver el listado de variables.
3. Se comprueba que exista al menos una variable.
4. El *bot* manda un mensaje al usuario indicándole el número total de variables encontradas y el nombre de cada una.

Escenario alternativo

3a. No existe ninguna variable en el PLC o no se ha podido obtener el listado.

1. El *bot* manda al usuario un mensaje informativo y se queda a la espera de un nuevo mensaje.

5.1.16.- Obtener el valor de una variable

Caso de uso: Obtener el valor de una variable del PLC	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	El *bot* debe haber recibido un nuevo mensaje y este haberse procesado, obteniendo el comando de devolver el valor de una variable del PLC. Se debe tener una conexión establecida con el PLC de Beckhoff
Resultado	Se obtiene el valor que tiene en un instante determinado una variable del PLC
Descripción	El usuario solicita que se le informe del valor que tiene una variable del PLC. Si la variable solicitada existe en el PLC, se obtiene su valor y el *bot* informa al usuario.

Escenario principal

1. Tras procesar el mensaje que el usuario ha enviado al bot, se ha obtenido el comando que solicita la lectura del valor de una variable.
2. Se llama al método para devolver el valor de la variable solicitada.
3. Se comprueba que exista dicha variable.
4. El *bot* manda un mensaje al usuario indicándole el valor que tiene la variable

Escenario alternativo

3a. No existe la variable en el PLC o no se ha podido acceder al PLC.

1. El *bot* manda al usuario un mensaje informativo y se queda a la espera de un nuevo mensaje.

5.1.17.- Modificar una variable

Caso de uso: Modificar el valor de una variable	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	El *bot* debe haber recibido un nuevo mensaje y este haberse procesado, obteniendo el comando de modificar una variable del PLC. Se debe tener una conexión establecida con el PLC de Beckhoff
Resultado	Se modifica el valor que tiene en un instante determinado una variable del PLC
Descripción	El usuario manda un mensaje en el que desea cambiar el valor de una variable. Tras comprobar que dicha variable existe en el PLC, se accede a este y se modifica su valor. Finalmente el *bot* informa al usuario si la operación se realizó con éxito.

Escenario principal

1. El usuario ha enviado un nuevo mensaje al *bot* y tras procesarlo se ha obtenido un comando que indica que se ha solicitado la modificación de una variable del PLC.
2. Se llama al método que se encarga de modificar variables.
3. Se comprueba que el mensaje contenga un número a asignar a la variable.
4. Se modifica la variable del PLC.
5. El *bot* manda un mensaje al usuario informándole de la correcta modificación de la variable.

Escenario alternativo

3a. El mensaje no contiene un número (Incluidos el True y False).

1. El *bot* manda al usuario un mensaje indicando que no se ha introducido un número y que no se ha podido modificar la variable.

4a. Se ha intentado modificar la variable pero el formato del valor que se desea introducir no es correcto, por ejemplo: asignar el valor 8.5 en una variable de tipo entero.

1. El *bot* manda al usuario un mensaje indicando que el valor que se desea asignar no es válido según el tipo de variable.

4b. Se ha intentado modificar una variable pero no se ha podido acceder a ella.

1. El *bot* manda al usuario un mensaje indicando que es posible que la variable no exista o que no haya una conexión establecida con un PLC.

5.1.18.- Obtener la versión de Twincat

Caso de uso: Obtener la versión de Twincat	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	Se debe tener una conexión establecida con el PLC de Beckhoff. El *bot* debe haber recibido un nuevo mensaje y este haberse procesado, obteniendo el comando de obtener la versión de Twincat
Resultado	Se obtiene la versión de Twincat que tiene el programa cargado en el PLC
Descripción	El usuario solicita que se le informe de la versión de Twincat que tiene el PLC. Si la conexión está establecida, se obtiene la versión de Twincat del PLC y el *bot* se encarga de informar al usuario.

Escenario principal

1. El usuario ha enviado un nuevo mensaje al *bot* y tras procesarlo se ha obtenido un comando que indica que se ha solicitado la obtención de la versión de Twincat.
2. Se llama al método que se encarga de la obtención de la versión de Twincat.
3. Se obtiene el valor número de la versión.
4. El *bot* manda un mensaje al usuario informándole acerca de la versión de Twincat.

Escenario secundario

3a. No se ha podido obtener la versión de Twincat por un fallo de conexión.

1. El *bot* informa al usuario acerca de lo ocurrido.

5.1.19.- Obtener el estado del PLC

Caso de uso: Obtener el estado del PLC	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	Se ha recibido un nuevo mensaje y el comando encontrado tras procesarlo es el de la obtención del estado del PLC. Se debe tener una conexión establecida con el PLC de Beckhoff
Resultado	Se obtiene el estado actual del PLC
Descripción	El usuario envía un mensaje con el que desea saber el estado del PLC. Si se tiene una conexión establecida con un PLC de Beckhoff, entonces se obtiene su estado y el *bot* devuelve esta información al usuario, que puede ser "Run" o "Stop"

Escenario principal

1. El usuario ha enviado un nuevo mensaje al *bot* y tras procesarlo se ha obtenido un comando que indica que se ha solicitado la obtención de la versión de Twincat.
2. Se llama al método que se encarga de la obtención de la versión de Twincat.
3. Se obtiene el valor número de la versión.
4. El *bot* manda un mensaje al usuario informándole acerca de la versión de Twincat.

Escenario secundario

3a. No se ha podido obtener la versión de Twincat por un fallo de conexión.

1. El *bot* informa al usuario acerca de lo ocurrido.

5.1.20.- Arrancar un PLC

Caso de uso: Arrancar un PLC	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	Se ha recibido y procesado un nuevo mensaje del *bot* y se ha identificado el comando de arrancar un PLC. Se debe tener una conexión establecida con el PLC de Beckhoff
Resultado	Se pone el PLC en modo "Run"
Descripción	El usuario envía un mensaje solicitando el arranque del PLC. Tras comprobar que existe una conexión con un PLC, se modifica el estado de este y el *bot* informa al usuario acerca del nuevo estado

Escenario principal

1. El usuario ha enviado un nuevo mensaje al *bot* y tras procesarlo se ha obtenido un comando que indica que se debe arrancar el PLC.
2. Se llama al método que se encarga del arranque del PLC.
3. Se arranca el PLC.
4. El *bot* manda un mensaje al usuario informándole acerca del estado actual del PLC.

Escenario secundario

3a. No se ha podido arrancar el PLC por un fallo de conexión.

1. El *bot* informa al usuario acerca de lo ocurrido.

5.1.21.- Parar un PLC

Caso de uso: Parar un PLC	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	Se debe tener una conexión establecida con el PLC de Beckhoff. Se ha recibido y procesado un nuevo mensaje del *bot* y se ha identificado el comando de parar un PLC
Resultado	Se pone el PLC en modo "Stop"
Descripción	El usuario envía un mensaje solicitando la parada del PLC. Tras procesar el mensaje y comprobar que existe una conexión con un PLC, se modifica el estado de este y el *bot* informa al usuario acerca del nuevo estado

Escenario principal

1. El usuario ha enviado un nuevo mensaje al *bot* y tras procesarlo se ha obtenido un comando que indica que se debe parar el PLC.
2. Se llama al método que se encarga de parar el PLC.
3. Se para el PLC.
4. El *bot* manda un mensaje al usuario informándole acerca del estado actual del PLC.

Escenario secundario

3a. No se ha podido parar el PLC por un fallo de conexión.

1. El *bot* informa al usuario acerca de lo ocurrido.

5.1.22.- Agregar notificación

Caso de uso: Agregar notificación	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	Se debe tener una conexión establecida con el PLC de Beckhoff. El *bot* ha recibido un mensaje y se ha identificado el comando de agregar una notificación.
Resultado	Se agrega una notificación de una variable del PLC
Descripción	El usuario envía un mensaje solicitando que una variable se agregue al sistema de notificaciones. Debe indicar un valor límite y la operación asociada que desea detectar: Mayor, menor o igual. Si el formato del mensaje es correcto y la variable existe en el PLC, entonces se agrega la notificación y el *bot* informa al usuario

Escenario principal

1. El usuario ha enviado un nuevo mensaje al *bot* en el que solicita la creación de una nueva notificación.
2. Se llama al método que se encarga de agregar notificaciones.
3. Se comprueba que la variable de interés existe y que el formato del mensaje es correcto.
4. Se comprueba que la notificación no existe todavía.
5. Se agrega la notificación al PLC y se guarda su valor y el nombre de la variable en un diccionario.
6. Finalmente, el *bot* informa al usuario de que todo fue correctamente.

Escenario secundario

3a. No existe la variable solicitada.

1. El *bot* informa al usuario de que la variable solicitada no existe.

3b. El formato del mensaje no es correcto.

1. El *bot* informa al usuario sobre el formato del mensaje que ha de introducir.

4a. La notificación ya existe.

1. El *bot* informa al usuario de que la notificación que quiere agregar ya existe.

5a. Se produce un error de conexión al intentar agregar la notificación en el PLC.

1. El *bot* indica al usuario que se ha producido un error de conexión.

5.1.23.- Eliminar notificación

Caso de uso: Eliminar notificación	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	El *bot* ha recibido un mensaje y se ha identificado el comando de eliminar una notificación. Se debe tener una conexión establecida con el PLC de Beckhoff
Resultado	Se elimina una notificación de una variable del PLC
Descripción	El usuario envía un mensaje con el que desea eliminar una notificación. Si esta existe en el sistema entonces se elimina y el *bot* informa al usuario sobre la operación realizada

Escenario principal

1. El usuario ha enviado un nuevo mensaje al *bot* en el que solicita la eliminación de una notificación.
2. Se llama al método que se encarga de eliminar notificaciones.
3. Se comprueba que la notificación existe (por el nombre de la variable).
4. Se elimina la notificación.
5. El *bot* manda un mensaje al usuario en el que le informa sobre la notificación eliminada.

Escenario secundario

3a. No existe la notificación pedida.

1. El *bot* informa al usuario de que la notificación no existe.

4a. No se ha podido eliminar la notificación por un fallo en la conexión.

1. El *bot* informa al usuario de que no tiene conexión con un PLC.

5.1.24.- Desconectarse de un PLC

Caso de uso: Desconectarse de un PLC	
Actores que intervienen	Usuario, PLC de Beckhoff, *bot* de Telegram
Precondición	Se debe tener una conexión establecida con el PLC de Beckhoff y haber recibido un mensaje en el que se pide la desconexión de un PLC
Resultado	Se cierra la conexión previamente establecida con un PLC
Descripción	El usuario manda un mensaje en el que solicita la desconexión de un PLC. Se comprueba que la conexión estuviera establecida y se realiza la desconexión. Finalmente el *bot* informa al usuario

Escenario principal

1. El usuario ha enviado un nuevo mensaje al *bot* en el que solicita la desconexión de un PLC.
2. Se llama al método que se encarga de realizar la desconexión.
3. Se realiza la desconexión del PLC.
4. El *bot* manda un mensaje al usuario en el que le informa que se ha realizado la desconexión.

Escenario secundario

3a. No hay ninguna conexión establecida

1. El *bot* informa al usuario de que para desconectarlo es necesario que se haya conectado a un PLC previamente.

5.1.25.- Descargar archivo

Caso de uso: Descargar archivo	
Actores que intervienen	Usuario, *bot* de Telegram
Precondición	Se debe haber recibido un nuevo mensaje en el que se solicita la descarga de un archivo
Resultado	Se obtiene el archivo solicitado
Descripción	El usuario envía un mensaje en el que pide la descarga de un determinado archivo. Se comprueba la existencia de la referencia a dicho archivo en la base de datos. En caso favorable, el *bot* accede a los servidores de Telegram y descarga el archivo solicitado, enviándolo al chat abierto con el usuario

Escenario principal

1. El usuario ha enviado un nuevo mensaje en el que solicita la descarga de un archivo.
2. Se llama al método que se encarga de la descarga de archivos.
3. Se busca el archivo solicitado en la base de datos, para obtener la referencia.
4. Se solicita al *bot* la descarga del archivo mediante un método al que se le pasa como parámetro la referencia obtenida.
5. El *bot* descarga el archivo y se lo envía al usuario.

3a. El archivo solicitado no existe en la base de datos.

1. El *bot* manda un mensaje al usuario en el que le dice que el archivo solicitado no está en la base de datos.

5.1.26.- Obtener listado de archivos

Caso de uso: Obtener listado de archivos	
Actores que intervienen	Usuario, *bot* de Telegram
Precondición	Se ha recibido un mensaje en el que se pide el listado de archivos de la base de datos. Debe haber archivos en la base de datos
Resultado	Se obtiene el listado de archivos
Descripción	Se accede a la base de datos para obtener el listado de archivos. Finalmente, el *bot* devuelve al usuario este listado

Escenario principal

1. Se recibe un mensaje en el que se solicita el listado de archivos de la base de datos.
2. Se llama al método que se encarga de devolver el listado de archivos.
3. Se accede a la base de datos y se obtiene el listado de archivos.
4. El *bot* se encarga de mandar un mensaje al usuario con el nombre de los archivos encontrados.

5.1.27.- Eliminar un archivo

Caso de uso: Eliminar un archivo	
Actores que intervienen	Usuario, *bot* de Telegram
Precondición	El archivo debe existir en la base de datos y se debe haber recibido un mensaje en el que se solicite su eliminación
Resultado	Se elimina de la base de datos la referencia a un archivo
Descripción	El usuario envía un mensaje con el que desea eliminar un archivo. Si el nombre del archivo solicitado existe en la base de datos se elimina y el *bot* informa al usuario

Escenario principal

1. El usuario ha enviado un mensaje en el que solicita la eliminación de un archivo de la base de datos.
2. Se llama al método que se encarga de la eliminación de archivos.
3. Se accede a la base de datos para comprobar si el archivo existe. En caso afirmativo se elimina.
4. El *bot* informa al usuario de que se ha eliminado el archivo.

Escenario alternativo

3a. El archivo no existe.

1. El *bot* informa comunica al usuario que está intentado eliminar un archivo que no existe.

5.1.28.- Obtener PLCs del archivo de configuración

Caso de uso: Obtener PLCs del archivo de configuración	
Actores que intervienen	Usuario, *bot* de Telegram, archivo txt
Precondición	El usuario ha pedido la obtención del listado de PLCs del archivo de configuración.
Resultado	Se obtiene el listado de PLCs del archivo de configuración
Descripción	Se accede al txt de configuración y el *bot* devuelve al usuario el listado de PLCs encontrados en forma de teclado, en el que cada botón representa un PLC

Escenario principal

1. Se recibe un mensaje en el que se solicita el listado de PLCs del archivo de configuración.
2. Se llama al método que se encarga de devolver el listado de PLCs.
3. Se accede al archivo y se devuelve un Array con los PLCs encontrados.
4. Se llama a un método que se encarga de crear un teclado con tantos botones como PLCs se hayan encontrado.
5. El *bot* envía el teclado al usuario.

Escenario alternativo

3a. No existe ningún PLC en el archivo de configuración.

1. Se cancela la secuencia y el *bot* informa al usuario de que no ha podido devolver el listado porque no hay ningún PLC.

5.1.29.- Agregar PLCs al archivo de configuración

Caso de uso: Agregar PLCs al archivo de configuración	
Actores que intervienen	Usuario, *bot* de Telegram, archivo txt
Precondición	El usuario ha solicitado la agregación de un nuevo PLC al archivo de configuración. Debe existir el archivo de configuración
Resultado	Se agrega un PLC al archivo de configuración
Descripción	El usuario solicita la inclusión de un nuevo PLC en el archivo de configuración, indiciando su dirección Ams y puerto. Si el mensaje tiene el formato correcto se accede al txt de configuración y se agrega el PLC. Finalmente el *bot* informa al usuario de la operación realizada

Escenario principal

1. Se recibe un mensaje en el que se solicita la inclusión de un nuevo PLC en el archivo de configuración.
2. Se llama al método que se encarga de agregar PLCs.
3. Se comprueba que se haya introducido una Ip y puerto válidos.
4. Se llama al método que se encarga de abrir el archivo y agregar el PLC solicitado.
5. El *bot* envía un nuevo mensaje al usuario informándole de que se ha añadido correctamente el PLC solicitado.

Escenario secundario

3a. No se han introducido una Ip o puerto con un formato válido.

1. El *bot* informa al usuario de que ha habido un error al procesar los datos del PLC, indicándole que si lo desea, se los repita.

5.1.30.- Eliminar PLCs del archivo de configuración

Caso de uso: Eliminar PLCs del archivo de configuración	
Actores que intervienen	Usuario, *bot* de Telegram, archivo txt
Precondición	El usuario debe haber mandado un mensaje solicitando la eliminación de un PLC del archivo de configuración. El PLC solicitado debe estar en el archivo de configuración
Resultado	Se elimina un PLC del archivo de configuración
Descripción	El usuario solicita la eliminación de un PLC del archivo de configuración, indicando su nombre. Se accede al txt de configuración para comprobar que el PLC existe. Si esto es cierto, entonces se elimina dicho PLC del archivo. Por último el *bot* informa al usuario

Escenario principal

1. Se recibe un mensaje en el que se solicita la eliminación de un PLC del archivo de configuración.
2. Se llama al método que se encarga de eliminar PLCs.
3. Se comprueba que se haya introducido un nombre de PLC que existe en el archivo.
4. Se llama al método que se encarga de acceder al archivo y eliminar el PLC solicitado.
5. El *bot* envía un nuevo mensaje al usuario informándole de que se ha eliminado el PLC solicitado.

Escenario secundario

3a. El PLC solicitado no existe en el archivo de configuración.

1. El *bot* envía un nuevo mensaje al usuario diciéndole que el PLC solicitado no existe.

5.1.31.- Crear un proyecto de Twincat

Caso de uso: Crear proyecto de Twincat	
Actores que intervienen	Usuario, *bot* de Telegram, Visual Studio
Precondición	El usuario debe haber enviado un mensaje en el que solicitó la creación de un nuevo proyecto de Twincat, el cual no debe existir previamente
Resultado	Se crea un nuevo proyecto de Twincat 3
Descripción	El usuario manda un mensaje en el que solicita la creación de un nuevo proyecto de Twincat con un determinado nombre. Tras comprobar que ese proyecto no existe, se crea y el *bot* informa sobre ello

Escenario principal

1. Se recibe un mensaje en el que se solicita la creación de un nuevo proyecto de Twincat.
2. Se llama al método que se encarga de la creación de proyectos de Twincat.
3. Se comprueba que no exista un proyecto con el mismo nombre.
4. Se crea el proyecto y se espera hasta finalizar su creación.
5. Se guarda el proyecto.
6. El *bot* envía un nuevo mensaje al usuario informándole de que se ha creado el proyecto solicitado.

Escenario alternativo

3a. Se ha encontrado un proyecto con el mismo nombre.

1. Se cancela la creación del proyecto y el *bot* envía un mensaje al usuario informándole de que el proyecto ya existe.

5.1.32.- Abrir proyecto de Twincat

Caso de uso: Abrir proyecto de Twincat	
Actores que intervienen	Usuario, *bot* de Telegram, Visual Studio
Precondición	El proyecto a abrir debe existir y el usuario debe haber solicitado al *bot* su apertura
Resultado	Se abre un proyecto de Twincat
Descripción	El usuario solicita que se abra un determinado proyecto de Twincat. Tras comprobar que el proyecto existe, se procede a su apertura y una vez abierto el *bot* informa sobre ello

Escenario principal

1. Se recibe un mensaje en el que se solicita la apertura de un proyecto de Twincat
2. Se llama al método que se encarga de abrir proyectos de Twincat.
3. Se comprueba que el proyecto solicitado no esté abierto.
4. Se abre el proyecto.
5. El *bot* envía un mensaje al usuario informándole de que se ha abierto el proyecto solicitado.

Escenario alternativo

3a. El proyecto ya está abierto.

1. Se cancela la apertura del proyecto y el *bot* manda un mensaje al usuario informándole de lo ocurrido.

5.1.33.- Cerrar proyecto de Twincat

Caso de uso: Cerrar proyecto de Twincat	
Actores que intervienen	Usuario, *bot* de Telegram, Visual Studio
Precondición	El proyecto a cerrar debe estar previamente abierto y el usuario debe haber mando un mensaje solicitando su cierre
Resultado	Se cierra un proyecto de Twincat
Descripción	El usuario solicita que se cierre un determinado proyecto de Twincat. Tras comprobar que el proyecto está abierto, se cierra y el *bot* informa al usuario

Escenario principal

1. Se recibe un mensaje en el que se solicita el cierre de un proyecto de Twincat
2. Se llama al método que se encarga de cerrar proyectos de Twincat.
3. Se comprueba que el proyecto solicitado esté abierto.
4. Se cierra el proyecto.
5. El *bot* envía un mensaje al usuario informándole de que se ha cerrado el proyecto solicitado.

Escenario alternativo

3a. El proyecto ya está abierto.

1. Se cancela el cierre del proyecto y el *bot* manda un mensaje al usuario informándole de lo ocurrido.

5.1.34.- Listado de proyectos de Twincat

Caso de uso: Listado de proyectos de Twincat	
Actores que intervienen	Usuario, *bot* de Telegram, Visual Studio
Precondición	El usuario debe haber mandado un mensaje solicitando el listado de proyectos de Twincat
Resultado	Se obtiene el listado de proyectos de Twincat
Descripción	El usuario manda un mensaje en el que pide el listado de proyectos de Twincat. Tras procesar el mensaje se obtienen los proyectos y el *bot* se encarga de mostrar el listado al usuario

Escenario principal

1. Se recibe un mensaje en el que se pide el listado de proyectos de Twincat.
2. Se llama al método que se encarga de obtener los proyectos de Twincat.
3. Se devuelve un Array con el listado de nombres de los proyectos.
4. El *bot* envía un mensaje al usuario con el listado de proyectos.

5.1.35.- Crear PLC en proyecto de Twincat

Caso de uso: Crear PLC en proyecto de Twincat	
Actores que intervienen	Usuario, *bot* de Telegram, Visual Studio
Precondición	El proyecto debe estar abierto y se debe de haber procesado un nuevo mensaje en el que el comando solicitado es el de creación de un PLC en el proyecto.
Resultado	Se crea un PLC en un proyecto de Twincat
Descripción	El usuario solicita la creación de un nuevo PLC en el proyecto. Tras comprobar que dicho PLC no existe todavía, se agrega al proyecto. Finalmente el *bot* informa al usuario

Escenario principal

1. Se ha recibido y procesado un mensaje en el que se pide la creación de un nuevo PLC en el proyecto.
2. Se llama al método que se encarga de agregar PLCs a un proyecto.
3. Se comprueba que no exista un PLC con ese mismo nombre en el proyecto.
4. Se agrega el PLC al proyecto.
5. El *bot* envía un mensaje al usuario comunicándole que se ha agregado el PLC solicitado.

Escenario alternativo

3a. Ya existe un PLC con el nombre solicitado.

1. Se cancela la agregación del nuevo PLC y el *bot* manda un mensaje al usuario informándole de lo ocurrido.

5.1.36.- Obtener los PLCs de un proyecto de Twincat

Caso de uso: Obtener los PLCs de un proyecto de Twincat	
Actores que intervienen	Usuario, *bot* de Telegram, Visual Studio
Precondición	El proyecto debe estar abierto y se debe haber procesado un mensaje en el que el comando solicitado es el de obtener los PLCs de un proyecto de Twincat
Resultado	Se obtiene el listado de PLCs de un proyecto
Descripción	El usuario solicita el listado de PLCs que contiene un proyecto abierto de Twincat. Se obtiene el listado y el *bot* se encarga de mostrárselo al usuario

Escenario principal

1. Se ha recibido y procesado un mensaje en el que se solicita el listado de PLCs de un proyecto abierto de Twincat.
2. Se llama al método que se encarga de obtener el listado de PLCs.
3. Se obtiene un Array con los nombres de los PLCs.
4. El *bot* devuelve al usuario el listado de PLCs.

5.1.37.- Agregar POUs a un PLC existente en un proyecto de Twincat

Caso de uso: Agregar POUs a un PLC existente en un proyecto de Twincat	
Actores que intervienen	Usuario, *bot* de Telegram, Visual Studio
Precondición	El proyecto debe estar abierto y se debe haber procesado un mensaje en el que el comando es el de agregar un nuevo POU a un PLC de un proyecto de Twincat
Resultado	Se crea un nuevo POU en un PLC del proyecto
Descripción	El usuario envía un mensaje con el que solicita la creación de un nuevo POU. Tras definir el tipo de POU y lenguaje de programación mediante unos teclados que el *bot* muestra al usuario, se crea el nuevo POU y el *bot* le informa sobre ello

Escenario principal

1. Se ha recibido y procesado un mensaje en el que se solicita la creación de un nuevo POU en un PLC del proyecto.
2. Se llama al método que se encarga de agregar POUs.
3. Se comprueba que el PLC solicitado existe y que no existe un POU con el mismo nombre.
4. Se genera un teclado que el *bot* manda al usuario, para que este seleccione el tipo de POU a crear.
5. Tras seleccionar el tipo de POU se envía otro teclado al usuario para que seleccione el tipo de lenguaje de programación que desea para el POU.
6. Se genera el POU en el PLC solicitado.
7. El *bot* manda un mensaje al usuario informándole de la creación del POU.

Escenario alternativo

3a. El POU solicitado ya existe o el PLC no se encuentra en el proyecto abierto.

1. Se cancela la creación del POU y el *bot* manda un mensaje al usuario con el que le informa al usuario del fallo ocurrido.

5a. El POU seleccionado es de tipo función.

1. Se genera un teclado adicional para que el usuario indique el tipo de dato que ha de devolver la función.
2. Se continúa en el paso 6.

5.1.38.- Eliminar POUs de un PLC existente en un proyecto de Twincat

Caso de uso: Eliminar POUs de un PLC existente en un proyecto de Twincat	
Actores que intervienen	Usuario, *bot* de Telegram, Visual Studio
Precondición	El proyecto debe estar abierto y se debe haber procesado un nuevo mensaje en el que el usuario solicita la eliminación de un POU
Resultado	Se elimina un POU de un PLC del proyecto
Descripción	El usuario solicita la eliminación de un POU indicando su nombre. Si dicho POU existe, entonces se borra y el *bot* se encarga de informar al usuario

Escenario principal

1. Se ha recibido y procesado un mensaje en el que se solicita la eliminación de un POU de un PLC del proyecto.
2. Se llama al método que se encarga de eliminar POUs.
3. Se comprueba que el PLC solicitado existe y que existe el POU solicitado.
4. Se elimina el POU del PLC solicitado.
5. El *bot* manda un mensaje al usuario informándole de la eliminación del POU.

Escenario alternativo

3a. El PLC no existe o el POU no existe en el PLC.

1. El *bot* manda un mensaje al usuario en el que le informa de que el PLC no existe o que el POU solicitado no existe en el PLC.

5.1.39.- Enviar un proyecto por email

Caso de uso: Enviar un proyecto por email	
Actores que intervienen	Usuario, *bot* de Telegram, Visual Studio, Servidor de correo
Precondición	El usuario debe haber enviado un mensaje en el que solicita el envío del proyecto por correo electrónico. Se debe tener un proyecto abierto de Twincat
Resultado	Se manda por correo el proyecto
Descripción	El usuario manda un mensaje en el que solicita el envío por correo del proyecto. Tras comprobar que el mensaje tiene el formato adecuado, se genera un zip del proyecto, el cual se envía como adjunto del correo. Por último, el *bot* informa al usuario sobre el envío realizado

Escenario principal

1. Se ha recibido y procesado un mensaje en el que se solicita el envío por correo del proyecto abierto de Twincat.
2. Se llama al método que se encarga de enviar proyectos.
3. Se comprueba que el mensaje contenga un correo y una contraseña.
4. Se añaden los credenciales, usuario y contraseña al correo.
5. Se genera un zip del proyecto.
6. Se crea un nuevo correo y se adjunta el zip generado anteriormente.
7. Se envía el proyecto al correo indicado en el mensaje recibido.
8. El *bot* manda un mensaje al usuario informándole de que se ha enviado el proyecto.

Escenario alternativo

3a. No se ha introducido un correo o una contraseña.

1. Se cancela el envío del proyecto y el *bot* manda un mensaje al usuario informándole del fallo ocurrido.

4. a No se ha introducido la contraseña correcta.

1. Se cancela el envío y el *bot* informa al usuario acerca de lo ocurrido.

5.2.- ORGANIZACIÓN DE LAS CLASES DEL PROGRAMA

Todos los proyectos de c# están fuertemente orientados a objetos, de hecho, al crear el proyecto, el programa por defecto que se genera ya es una clase de nombre "Program", que contiene el main y desde la cual se puede empezar a programar. Como esto no suponía ningún conflicto con el planteamiento de este trabajo, se partió de esa clase "Program" para realizar el programa. Es importante mencionar que debido al uso de librerías externas, hay clases que se han utilizado pero que no se han creado como tal en el proyecto. Para que quede más claro que clases se han creado y cuáles no, se ha realizado el listado siguiente con las creadas y las más importantes de las externas.

Clases propias

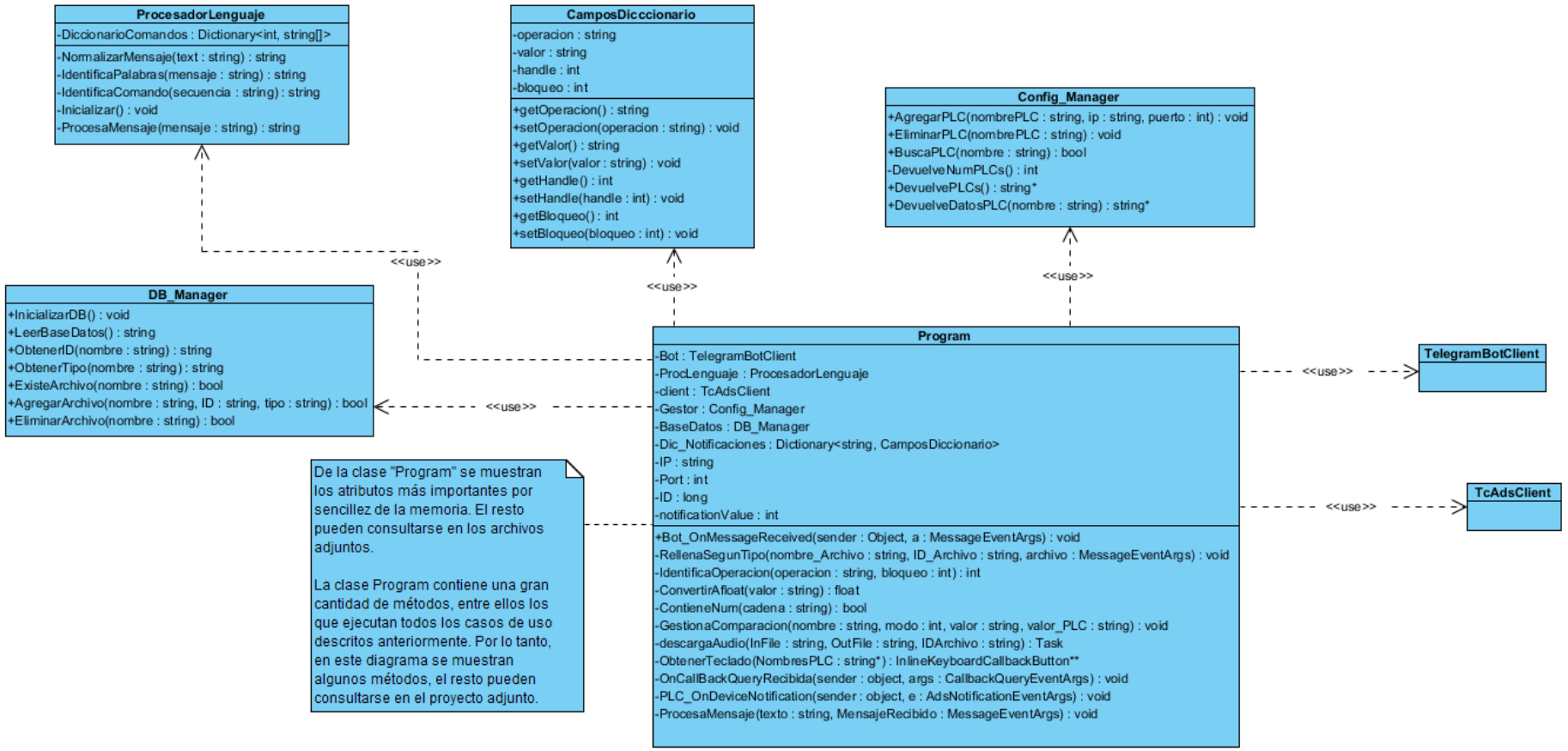
1. Program
2. ProcesadorLenguaje
3. Config_Manager
4. DB_Manager
5. CamposDiccionario

Clases externas

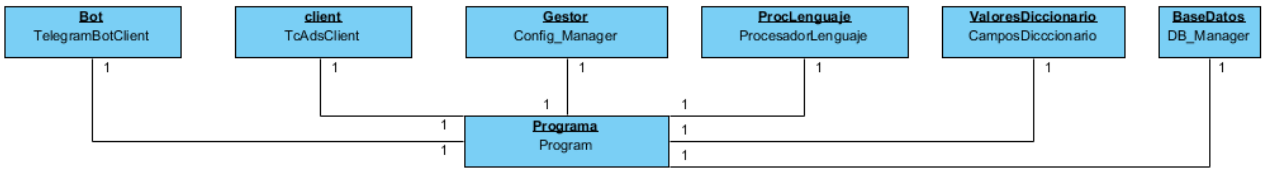
1. TelegramBotClient
2. TcAdsClient

Las clases anteriores son las principales, aún así debido a que c# está profundamente orientado a objetos, se emplean otras clases para realizar ciertas funciones en el programa. Estas clases se nombrarán en la descripción de los métodos, pero no se hará un mayor análisis debido a que esto complicaría y alargaría la memoria innecesariamente.

5.2.1.- Diagrama de clases



5.2.2.- Diagrama de objetos



5.2.3.- Clase Program

Es la clase principal del proyecto. Contiene el main y todos los métodos que representan los casos de uso de la aplicación, desde los cuales se llaman a las distintas clases auxiliares utilizadas.

5.2.4.- Clase ProcesadorLenguaje

Esta clase es de las más importantes del proyecto. Se encarga de procesar un mensaje de texto que se le pase como parámetro de uno de sus métodos. Tras recibir el mensaje, identifica palabras clave en él para posteriormente devolver a la clase "Program" el comando o funcionalidad reconocida, si es que la hubiera.

5.2.5.- Clase Config_Manager

Se encarga de manipular toda la parte de gestión de un archivo que contiene configuraciones de PLCs. Se encarga de leer los datos, devolverlos, añadir PLCs nuevos o eliminar los ya existentes según los requisitos de los usuarios.

5.2.6.- Clase DB_Manager

Esta clase contiene toda la parte de gestión de una base de datos que se ha creado para la manipulación de los archivos enviados a través del *bot*. Su funcionalidad está descrita con más detalle en apartados posteriores.

5.2.7.- Clase CamposDiccionario

Se encarga de almacenar los parámetros necesarios para la gestión de las notificaciones, es decir, contiene propiedades como tipo de operación, valor límite, etc…, además de métodos *get* y *set* de cada una.

5.2.8.- Clase TelegramBotClient

Es la clase utilizada, de la API de Telegram, para definir el *bot* y permitir las comunicaciones con los servidores de Telegram. Es la que se encarga de gestionar el envío y recepción de mensajes con el *bot*.

5.2.9.- Clase TcAdsClient

Es la clase principal de la librería de Twincat. Permite establecer la conexión con el PLC deseado y realizar todas las funcionalidades de interacción con un PLC.

5.3.- DESCRIPCIÓN DETALLADA DE LAS CLASES AUXILIARES Y SUS MÉTODOS

Para que el programa se pueda entender, se procederá a explicar primero las clases auxiliares, para posteriormente describir la clase principal, Program. De lo contrario, uno se perdería al leer los contenidos de esta clase, ya que en ella se hace referencia al resto de clases.

5.3.1.- Clase ProcesadorLenguaje

Como se ha explicado en apartados anteriores, para la parte de procesamiento del lenguaje se creó una librería (de tipo dll), la cual permite realizar modificaciones en el reconocimiento de texto sin falta de recompilar el programa principal. Esto aporta grandes ventajas de cara a un futuro, ya que se podría ir mejorando el algoritmo de reconocimiento o incluso sustituirlo sin falta de tener que actualizar el *bot*.

Para la generación de esta librería se creó un proyecto aparte de Visual Studio, en el que se agregó la clase “ProcesadorLenguaje”, que se describe a continuación.

Para el reconocimiento de palabras, se recurre al uso de un diccionario interno, que contendrá en la primera columna un número entero, que identifica la fila, y en la segunda columna un Array de strings, en el cual se incluyen las distintas opciones deseadas para el reconocimiento de una misma palabra.

Identificador de la fila	Listado de palabras
0	Conectar, conectarme, conéctame, conectes, enlazar, enlázame, enlazarme, enlaces
1	Listado, lista, conjunto
2	Ver, mostrar, muéstrame, muestres, dime, devuélveme
3	Escribir, escribe, escribes, modificar, modifica, modifiques, cambiar, cambia, cambies, asignar, asigna, asignes
4	Estado, modo
5	Parar, para, párame, pares, detener, detén, detengas, desactivar, desactives, desactiva, desactívame
6	Arrancar, arranca, arráncame, arranques, activar, activa, actívame, actives
7	Avisar, avísame, avises, notificar, notifícame, notifiques, informar, infórmame, informes
8	Eliminar, elimina, elimines, elimíname, borrar, borra, bórrame, borres
9	Desconectar, desconecta, desconéctame, desconectes
10	Descargar, descarga, descárgame, descargues
11	Disponibles, almacenados, guardados
12	Crear, crea, créame, crees, generar, genera, genérame, generes, agregar, agrega, agrégame, agregues
13	Abrir, abre, ábreme, abras
14	Cerrar, cierra, ciérrame, cierres
15	Contenidos, generados, contiene, incorporados
16	Enviar, envíame, envía, envíes, mandar, mándame, manda, mandes
17	PLC, PLCs, autómata, autómatas
18	Variables, variable
19	Versión
20	Twincat, programa
21	Notificación, notificaciones
22	Archivo, archivos

23	Configuración
24	Proyecto, proyectos
25	POU, POUs

Tabla 5.1.- Diccionario de palabras para identificar el comando contenido en un mensaje.

Para cargar las palabras en el diccionario se dispone de un método público llamado "Inicializar", que se encarga de añadir cada secuencia de palabras al diccionario.

De esta forma, lo que se pretende conseguir es que cada palabra que se desea reconocer cuente con una serie de opciones válidas sustitutivas.

Una de las características clave cuando se recibe un nuevo mensaje es normalizarlo, es decir, quitarle las posibles tildes que pueda contener. Esto se realiza con el método privado "NormalizarMensaje". Este se encarga de descomponer la cadena entrada, identificar donde están los acentos y recomponer la cadena sin las tildes.

Posteriormente se busca en el diccionario si alguna de las palabras del string de entrada está contenida en el diccionario. Para esto se utiliza una expresión literal de comparación y por cada palabra "válida" se va formando una secuencia en el formato: n° fila de la primera palabra – n° fila de la segunda palabra - ….etc.

Con el método anterior se consigue una secuencia que identifica cada una de las filas del diccionario en las que se ha encontrado una palabra del mensaje. Por ejemplo si el usuario hubiera escrito "Hola, quiero saber el listado de variables", este método devolvería la secuencia: 1-18-.

Después, estos números se introducen como parámetro del método "IdentificaComando", que se encarga de reconvertir esa secuencia numérica en una secuencia de palabras e identificar el comando asociado. Siguiendo con el ejemplo anterior, la secuencia 1-18- sería reconvertida a "listado Variables".

Con los métodos de los string en c# se buscan las coincidencias, según los comandos que se han programado para el *bot*. En este caso se recurre mucho al uso del método "indexOf", que permite comprobar si una determinada cadena existe en otra que se pase como parámetro. Cabe mencionar que este método puede devolver resultados erróneos según qué palabras se utilicen por lo que hay que tener cuidado, ya que si por ejemplo se busca el string "conectar", este está contenido tanto en "conectar" como en "desconectar", por lo que se debe diferenciar cual es el correcto en cada caso. Además se añade a "indexOf" la opción "IgnoreCase", para que no se tengan en cuenta las mayúsculas/minúsculas.

```
private string IdentificaComando(string secuencia)
{
    StringBuilder SecInString=new StringBuilder();
    string Comando="null";
    int number;
    //Se divide el string de entrada por guiones, ya
    //que tiene la estructura 1-2-3-4...
    string[] SecArray = secuencia.Split('-');

    //Se recorre la secuencia para realizar las equivalencias
    //con el diccionario. Cuando encuentra una guarda la primera palabra
    //de ese valor del diccionario.
    for (int i=0;i<SecArray.Length-1;i++) //Se usa el -1 porque el último elemento es un espacio vacío que añade el split.
    {
        Console.WriteLine(SecArray[i]);
        int.TryParse(SecArray[i], out number);
        SecInString.Append(DiccionarioComandos[number].First<string>());
        SecInString.Append(" ");

    }
    //Conversión de stringbuilder a string para
    //poder usar las propiedades de string
    string WordSeq = SecInString.ToString();
    Console.WriteLine(WordSeq);

    //Según la secuencia de palabras se identifica a que comando se corresponde
    if (WordSeq.IndexOf("Conectar", StringComparison.OrdinalIgnoreCase) >= 0 &&
        WordSeq.IndexOf("Desconectar", StringComparison.OrdinalIgnoreCase) < 0)
    {
        Comando = "Conectar";
    }
    else if(WordSeq.IndexOf("listado", StringComparison.OrdinalIgnoreCase) >= 0 &&
            WordSeq.IndexOf("Variables", StringComparison.OrdinalIgnoreCase) >= 0)
    {
        Comando = "Listado_Variables";
    }
```

Figura 5.2.- Parte inicial del método IdentificaComando, que se encarga de reconvertir la secuencia númerica de entrada a un string que contenga las palabras identificadas, para comprobar qué método debe ejecutarse.

El método principal de la clase es "ProcesaMensaje", también público. Se encarga de comprobar si un string de entrada contiene un comando que se deba ejecutar o no. Es el encargado de llamar al resto de métodos de la clase, descritos anteriormente, hasta obtener un string con el nombre del comando que se va a tener que ejecutar.

```
/*Método que se encarga de procesar el mensaje de entrada*/
public string ProcesaMensaje(string mensaje)
{
    StringBuilder ComandoID=new StringBuilder();
    string MensajeNormalizado;
    string SecuenciaPalabras;
    string Identificador;
    //Primera parte del mensaje devuelto siempre
    //Sigue la estructura Comando...
    ComandoID.Append("Comando:");

    //Se normaliza el mensaje recibido para quitar tildes
    MensajeNormalizado=NormalizarMensaje(mensaje);
    //Obtención del codigo de la secuencia encontrada
    SecuenciaPalabras = IdentificaPalabras(MensajeNormalizado);

    //Si no hubo coincidencias no hay comando que ejecutar
    if(SecuenciaPalabras=="No hay coincidencias")
    {
        ComandoID.Append("null");
    }
    else
    {
        //Si se encontraron coincidencias se identifica el comando asociado
        Identificador = IdentificaComando(SecuenciaPalabras);
        ComandoID.Append(Identificador);
    }

    //Se devuelve la información del comando que se debe ejecutar
    return (ComandoID.ToString());
}
```

Figura 5.3.- Método principal de la librería que se encarga de procesar los mensajes e identificar los comandos.

5.3.2.- Clase Config_Manager

Para establecer una conexión con un PLC, normalmente se debe introducir una IP y un puerto. En este proyecto se ha añadido la opción de que el usuario pueda decir algo como “Conéctame al PLC PLC_1” y que el *bot* busque ese PLC en un archivo de configuración que contenga la IP y el puerto correspondientes. Como en otras ocasiones, esto se podría haber hecho de otras formas, por ejemplo con una base de datos, pero como este trabajo también tiene una finalidad de aprendizaje en la empresa en la que se ha realizado, para familiarizarse con las distintas características de c#, se optó por usar un txt y emplear las funciones de gestión de archivos de c#.

Por lo tanto, para gestionar toda la parte de este archivo de configuración utilizado para la conexión con PLCs, se creó la clase Config_Manager.

El primer método de esta clase es el que permite agregar un nuevo PLC al archivo de configuración y recibe el nombre "AgregarPLC". La estructura de los datos cuando se desea agregar un nuevo PLC es la siguiente: Nombre PLC, IP, Puerto. Es importante tener en cuenta que los parámetros anteriores están separados por tabulaciones, ya que en caso de que se agregue un PLC manualmente, abriendo el txt correspondiente, se deben separar con tabulaciones los datos y no con espacios.

Para eliminar PLCs del archivo de configuración, se realiza una copia temporal del txt y se van pasando los datos de un archivo a otro, para finalmente sustituir el txt original por este nuevo. Esta funcionalidad la integra el método "EliminarPLC".

Otro método de la clase Config_Manager es el que se encarga de buscar un PLC en el archivo de configuración. Este método recibe el nombre "BuscaPLC", el cual devuelve "true" si el nombre de PLC que se le pase como parámetro existe en el archivo de configuración.

Para saber el número de PLCs del archivo de configuración se creó otro método, "DevuelveNumPLCs", que devuelve un entero que contiene el número de líneas del archivo de configuración que tienen datos. Otro método creado, "DevuelvePLCs", retorna un Array de strings con los nombres de los PLCs del archivo de configuración.

Por último, el método "DevuelveDatosPLC" es el encargado de devolver los datos del PLC que se le pase como parámetro. Para ello, se accede al txt de configuración y por cada línea que no esté vacía se obtienen los datos del PLC correspondiente, guardándolos en un Array de string. Posteriormente se evalúa si el nombre de ese PLC se corresponde con el buscado y si es así, se guardan y se devuelven sus datos.

```
/*Método que devuelve los datos de un plc
 * solicitado como parámetro*/
public string[] DevuelveDatosPLC(string nombre)
{
    var lineas_PLC = System.IO.File.ReadLines(@"C:\Users\luis.muñiz\Desktop\proyectos_twincat\Config_PLC.txt").Skip(1);
    string[] datos_PLC=new string[3];//Nombre,IP,Puerto--->(0,1,2)

    //Se busca el PLC solicitado
    foreach (var datosPLC in lineas_PLC)
    {
        if (datosPLC != "")
        {
            string[] datos_split = datosPLC.Split('\t');
            if (datos_split[0] == nombre)//Se ha encontrado el PLC
            {
                datos_PLC = datos_split;
                break;
            }
        }

    }
    //Se muestran los datos por consola y se sale del método
    Console.WriteLine("datos_PLC:"+datos_PLC);
    return datos_PLC;
}
```

Figura 5.4.- Método que devuelve los datos de un PLC solicitado como parámetro.

5.3.3.- Clase DB_Manager

A través del *bot* se puede enviar cualquier tipo de archivo, ya sea un documento, foto o vídeo. Por este motivo se consideró que sería útil ofrecer a los usuarios una manera de transferir archivos y poder descargarlos posteriormente si fuera necesario.

Esta clase contiene toda la configuración y gestión de una base de datos que se creó para almacenar la información de los archivos enviados a través del . La herramienta utilizada para la base de datos es SQLite [18], su elección vino dada porque es totalmente gratuita y es utilizada por una gran cantidad de personas, por lo que encontrar documentación no resulta especialmente difícil.

Entonces, cuando por ejemplo una foto se envíe a través del *bot*, su nombre, identificador y el tipo de archivo quedarán almacenados en la base de datos, para que el usuario pueda descargarlos cuando desee. Sin embargo, por el momento se utilizan los propios servidores de Telegram para almacenar los archivos, que cada 48 horas borran los datos. Para la finalidad de aprendizaje de este proyecto, esto no acarrea ningún problema, pero en una aplicación profesional se deberían almacenar los archivos en un servidor propio.

El método "InicializarDB" es el que se encarga de crear la base de datos si no existe, y agregarle una tabla de tres columnas: Nombre, ID, Tipo de archivo. Este método es una de las primeras acciones que se realizan al ejecutar el programa, ya que la base de datos debe estar creada desde el principio.

Otro método, "LeerBaseDatos", es el que permite leer los nombres de los archivos contenidos en la base de datos. Para realizar esta tarea se utiliza el comando "Select count*" y si este devuelve un número mayor que 0 entonces se comienza a leer la base de datos, almacenando los nombres de los archivos encontrados en un string, que posteriormente se devolverá al llamador del método.

También se han añadido métodos para comprobar si un archivo existe en la base de datos ("ExisteArchivo"), para obtener el nombre de un determinado archivo, su ID ("ObtenerID") o su tipo ("ObtenerTipo").

Como en cualquier base de datos, existen métodos para agregar y eliminar campos. A estos métodos se les asignó el nombre "AgregarArchivo" y "EliminarArchivo" respectivamente. Ambos son métodos que funcionan de forma muy similar, por lo que a continuación se describe únicamente la parte de agregar archivos a la base de datos.

El funcionamiento para agregar un nuevo archivo a la base de datos es el siguiente: Se abre la conexión con la base de datos y se busca en esta el nombre del archivo pasado como parámetro del método. Si se encuentra un archivo con ese nombre no se hace nada y se devuelve false, sino hay ningún archivo con ese nombre se agrega una nueva línea a la base de datos con el comando "Insert", en la que estarán contenidos el Nombre, ID y tipo del archivo. Finalmente se cierra la conexión.

5.3.4.- Clase CamposDiccionario

Esta clase se utiliza como tipo de dato en el diccionario que se creó en la clase "Program". Contiene una serie de propiedades privadas y sus homólogas públicas, con métodos *set* y *get* para modificar o leer su valor.

En esta clase se siguió una de las "buenas prácticas" recomendadas en c#, que es la de no acceder directamente a propiedades de las clases, es decir, se recomienda que por defecto todas se creen como privadas y que su valor sea gestionado a través de una propiedad pública equivalente que incluya los métodos *set* y *get*.

```
class CamposDiccionario
{
    private string _operacion;
    private string _valor;
    private int _handle;
    private int _bloqueo;

    //Métodos para devolver y asignar los valores
    //de las propiedades privadas declaradas
    public string operacion
    {
        get { return this._operacion; }
        set { this._operacion = value; }
    }
```

Figura 5.5.- Ejemplo de buenas prácticas en la gestión de las propiedades de las clases.

5.3.5.- Clase TcAdsClient

La clase TcAdsClient es la que se utiliza para la comunicación con los PLCs de Beckhoff. Contiene todas las operaciones necesarias para las distintas acciones que se puedan llevar a cabo. La guía de esta clase se puede leer en la página oficial de soporte de Beckhoff, Infosys [11].

5.3.6.- Clase TelegramBotClient

La clase TelegramBotClient es la que se utiliza para la creación del *bot* y para la gestión de los mensajes enviados. La documentación a cerca de sus métodos y características se puede encontrar en esta página de github [19].

5.4.- DESCRIPCIÓN GENERAL DE LA CLASE PRINCIPAL DEL PROYECTO, "PROGRAM"

"Program" es la clase principal del programa, es la que contiene el main y todos los comandos que se pueden pedir al *bot*, es decir, es la encargada de ejecutar todos los casos de uso descritos anteriormente. Además es la que se encarga de llamar al resto de clases creadas cuando la acción solicitada lo requiera.

En la clase Program se declaran los objetos del resto de las clases del proyecto. En un primer párrafo se declaran los objetos del *bot* y de la parte de Beckhoff. Posteriormente se declaran las variables que almacenarán la Ip y puerto de la conexión, así como el id del usuario y el identificador de notificaciones.

Posteriormente se declaran los objetos de las clases necesarias para la parte de creación de proyectos de Twincat de forma remota. Tras esto se crea el diccionario que se utilizará para el almacenamiento de la información de las notificaciones. Como se puede observar en el diagrama de clases, este diccionario incluye a la clase "CamposDiccionario" descrita anteriormente.

Por último se declaran e instancian los objetos de las clases Config_Manager y DB_Manager.

5.4.1.- Main del programa

El main del programa contiene la creación del objeto de la clase "Program" y la subscripción a tres eventos, dos para recibir los mensajes del *bot* y un tercero, "OnCallbackQuery", para recibir la información contenida en los botones de unos teclados que se crearán más adelante. También se inicializan en esta parte el procesador del lenguaje y la base de datos.

```
//Main del programa
public static void Main(string[] args)
{
    //Creación de un objeto de la clase programa
    Program programa = new Program();
    programa.client.Synchronize = false;


    //Se añaden los tres eventos asociados a los mensajes del Bot
    programa.Bot.OnMessage += programa.Bot_OnMessageReceived;
    programa.Bot.OnMessageEdited += programa.Bot_OnMessageReceived;
    programa.Bot.OnCallbackQuery += programa.OnCallBackQueryRecibida;

    //Se Inicializa el procesador del lenguaje
    programa.ProcLenguaje.Inicializar();

    //Se Inicializa la base de datos
    programa.BaseDatos.InicializarDB();

    /*Se comienzan a recibir mensajes del bot*/
    programa.Bot.StartReceiving();
    Console.ReadLine();
    programa.Bot.StopReceiving();

}
```

Figura 5.6.- Main del programa.

5.4.2.- Recepción y preparación del mensaje recibido

Uno de los principales métodos de la clase "Program" es el asociado al evento que se dispara cuando se recibe un nuevo mensaje del *bot*, "Bot_OnMessageReceived". A partir de este se derivan el resto de métodos del programa. En "Bot_OnMessageReceived" se identifica si el mensaje recibido es de tipo texto, voz o de envío de archivos y desde este punto se llaman a los otros métodos que correspondan.

Lo primero que se hace en este método es guardar el valor del ID del usuario que ha mandado el mensaje. Después se identifica el tipo de mensaje y se actúa en consecuencia.

Para los mensajes de tipo texto se llama al método "ProcesaMensaje", el cual a su vez llamará al procesador del lenguaje para identificar posibles comandos.

Los mensajes de tipo nota de voz necesitan de un pre-procesamiento antes de que se puedan utilizar. Por defecto, Telegram guarda las notas de voz en formato .ogg, pero analizando en profundidad el archivo que se genera, se observa que contiene el códec Opus y que Ogg simplemente actúa como contenedor. Por lo tanto, debido a que el motor de

reconocimiento de voz de Windows requiere que los archivos estén en formato .wav, es necesario realizar una conversión en tiempo de ejecución para poder procesar la nota de voz. Esta tarea se realiza con el programa FFMpeg [13]. Describir cómo funciona FFmpeg llevaría otro apartado independiente, por lo tanto para esta memoria se dará por sabido su funcionamiento.

Tanto para los mensajes de texto como para los de nota de voz, se llama al método "ProcesaMensaje". Este, se encarga de llamar al procesador del lenguaje, que devuelve el comando que contiene el mensaje.

```
//Método que se encarga de procesar el mensaje recibido
private void ProcesaMensaje(string texto, MessageEventArgs MensajeRecibido)
{
    try
    {
        string ComandoRecibido=ProcLenguaje.ProcesaMensaje(texto);//Obtención del comando a ejecutar
        Console.WriteLine(ComandoRecibido);//Se muestra por consola el comando recibido
        EjecutaComando(texto, ComandoRecibido, MensajeRecibido);//Ejecución del comando solicitado
    }
    catch(Exception err)
    {
        Bot.SendTextMessageAsync(ID, "Ha habido un error mientras procesaba el mensaje, "+
                                     "vuelve a repetírmelo por favor...");
    }
}
```

Figura 5.7.- Método auxiliar ProcesaMensaje, que llama al procesador del lenguaje y posteriormente a "EjecutaComando", que se encarga de gestionar el comando recibido.

Posteriormente se llama al método "EjecutaComando", que contiene un *switch* con el que se itera a lo largo de las distintas operaciones que el usuario puede realizar. Desde este *switch* se llama al método que corresponda.

La tercera parte del método "Bot_OnMessageReceived" es la que se encarga de gestionar los archivos enviados a través del *bot*. Para esto, como se ha mencionado anteriormente se recurrirá a una base de datos con SQLite. En este punto es importante destacar que Telegram no permite asignar un nombre a las fotos o vídeos que se desean enviar, por lo que para guardar la información de cada archivo se utilizará como punto de partida el identificador que le haya asignado Telegram. A partir de este se crearán dos parámetros, un nombre y un ID. Estos dos atributos se rellenarán con el identificador

asignado por Telegram a ese archivo, de forma que cada archivo tendrá un nombre e ID únicos. Además el nombre estará compuesto por el tipo de archivo enviado y su ID.

Tras asignar un nombre e ID al archivo se procede a agregarlo a la base de datos, siempre y cuando esta no lo contenga. Tanto si lo tiene agregado como si no, se informa al usuario con el resultado de la operación.

En el caso de que el archivo enviado sea un documento, se eliminan los espacios en blanco que pueda contener su nombre para que no haya problemas más adelante durante su agregación a la base de datos.

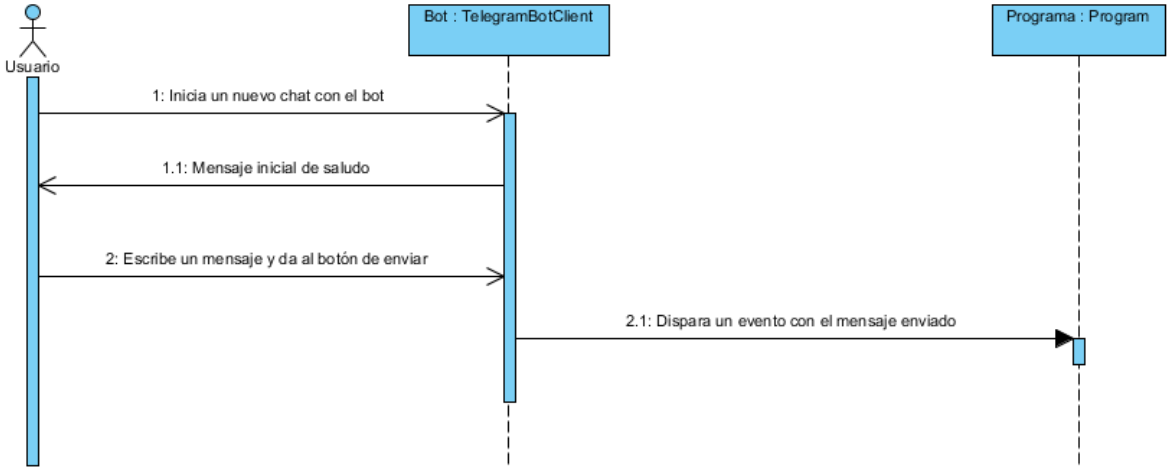
5.4.3.- Otros métodos

Algunos de los métodos que se observan en el diagrama de clases, como "RellenaSegunTipo" o "ConvertirAfloat", son métodos auxiliares que se emplean para ciertas tareas de los casos de uso. Debido a la dimensión de estos métodos lo mejor es que se lean en el código del programa adjuntado con este documento. Ya que los que realmente importan, de cara a la funcionalidad del sistema, son los descritos en los casos de uso.

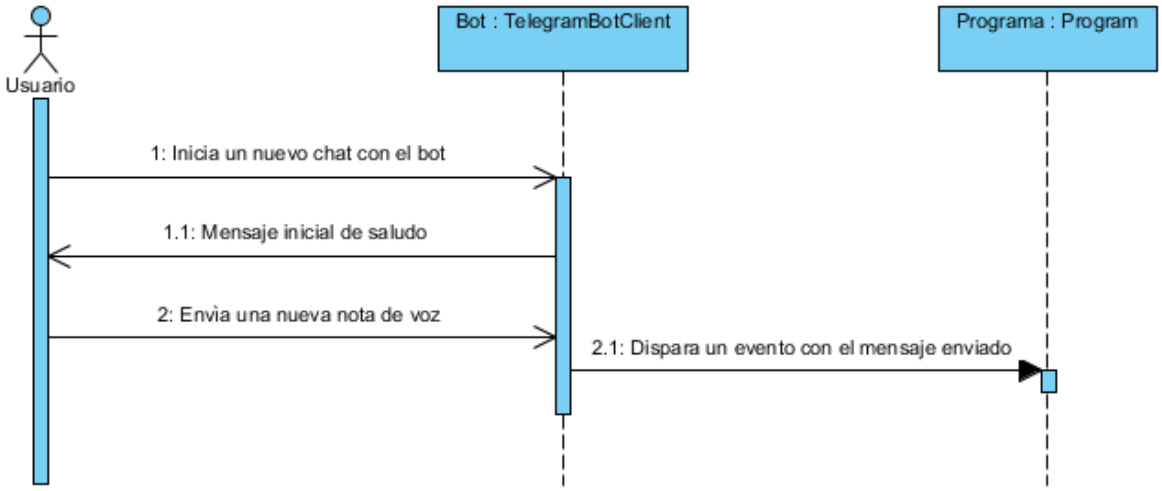
5.5.- DIAGRAMAS DE SECUENCIAS

Debido a los numerosos casos de uso de la aplicación, se van a realizar los diagramas de secuencias de los escenarios principales. En aquellos casos que resulten muy similares, como por ejemplo el de abrir o cerrar un proyecto de Twincat, o crear y borrar un POU, se mostrará uno de los dos.

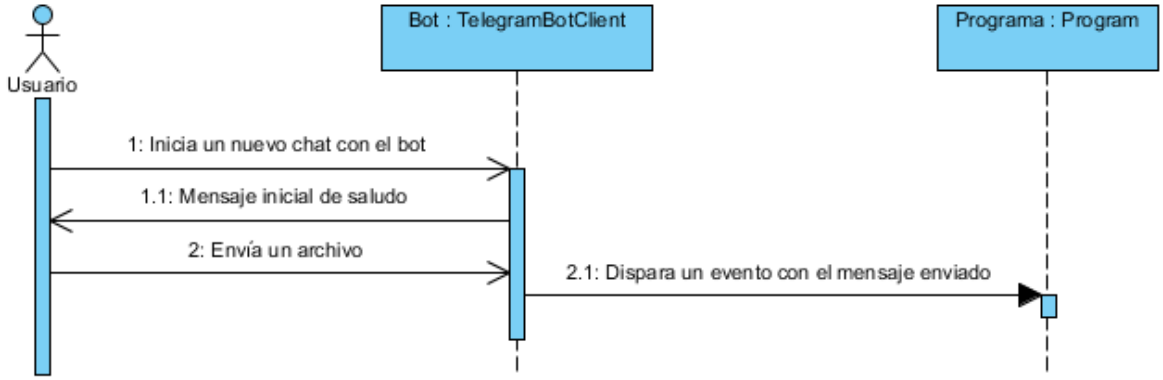
5.5.1.- Ejecutar mensaje de texto



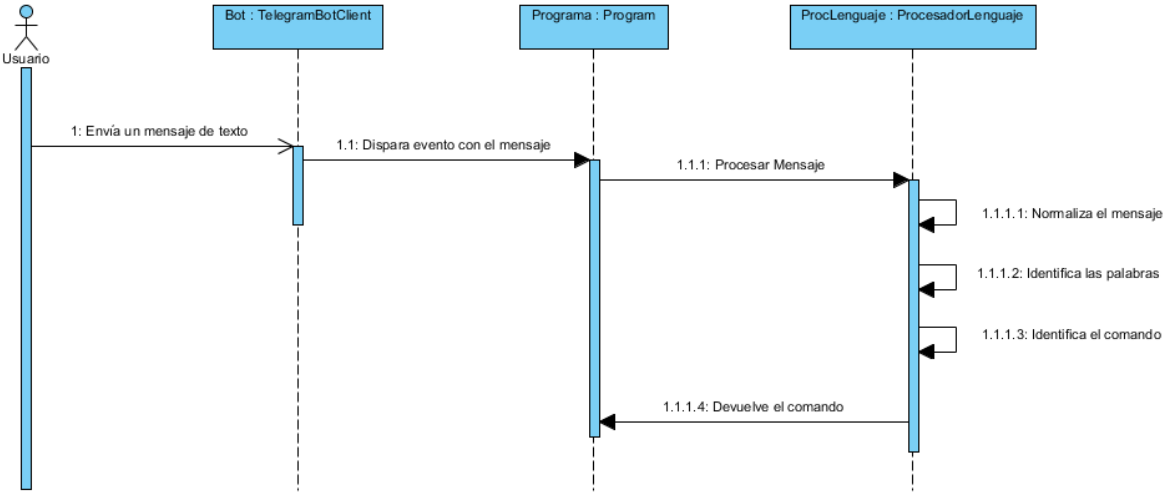
5.5.2.- Ejecutar mensaje de nota de voz



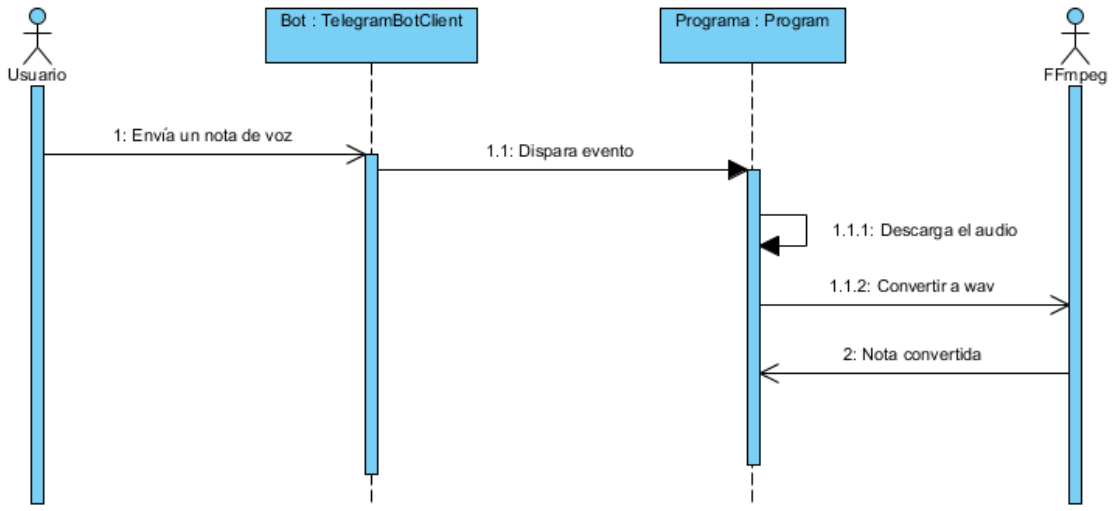
5.5.3.- Ejecutar mensaje de tipo archivo



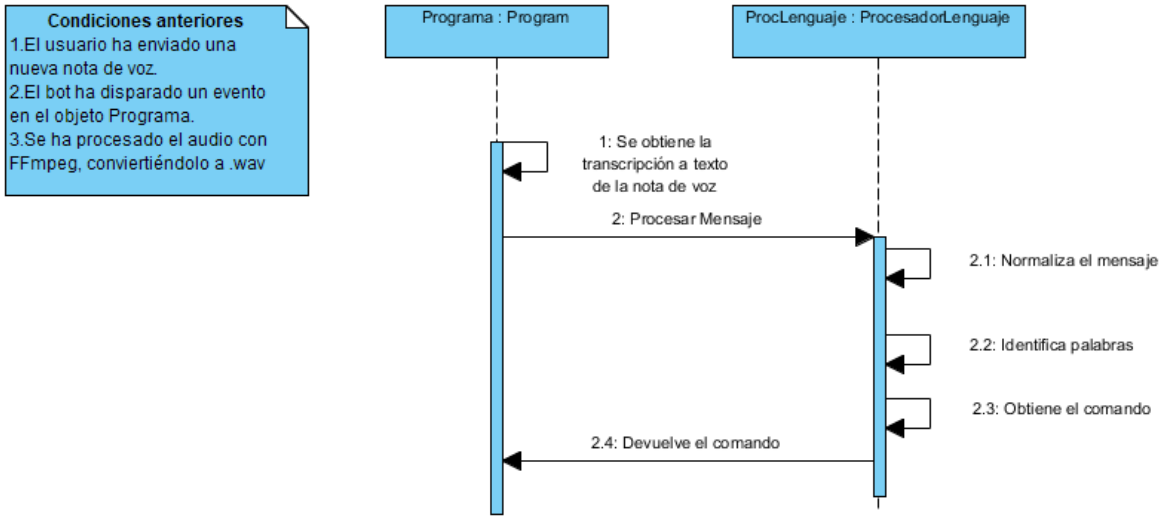
5.5.4.- Analizar mensaje



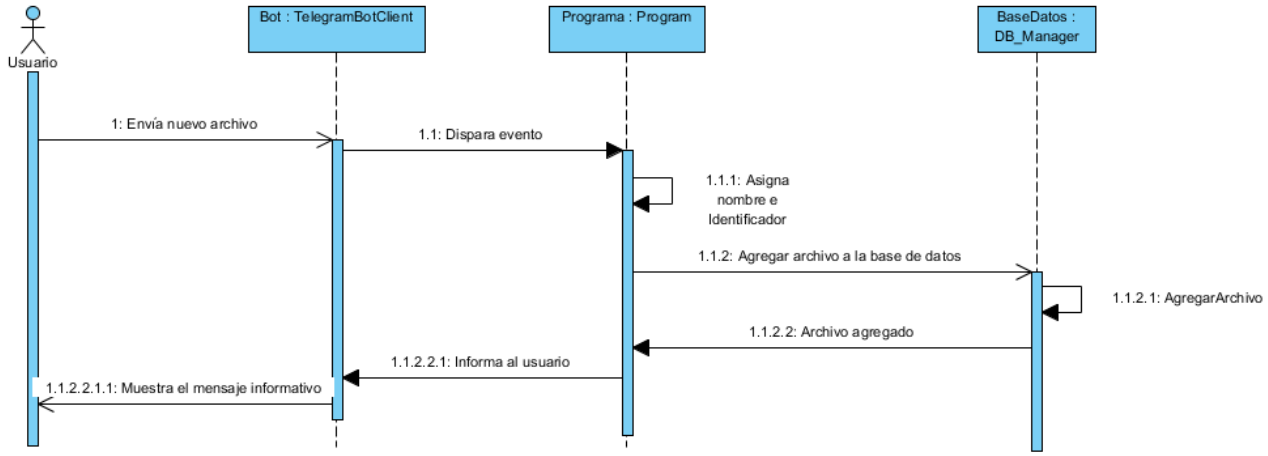
5.5.5.- Procesar nota de voz



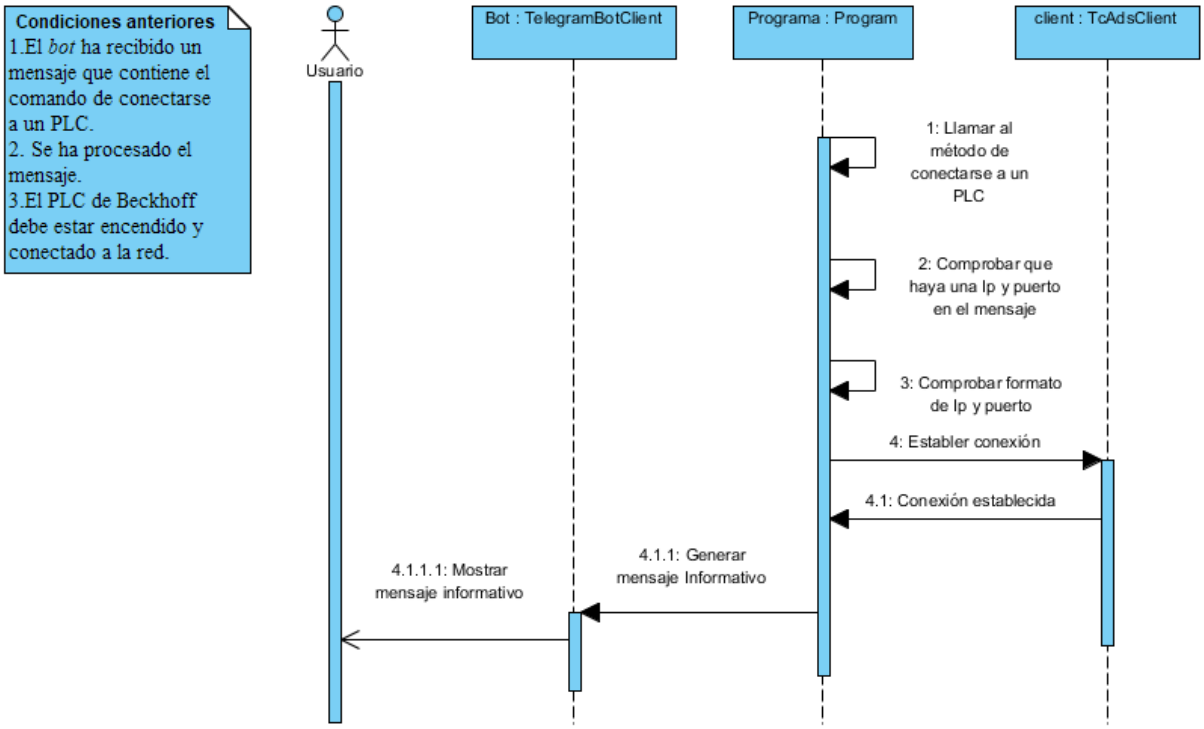
5.5.6.- Transcribir audio y analizar mensaje



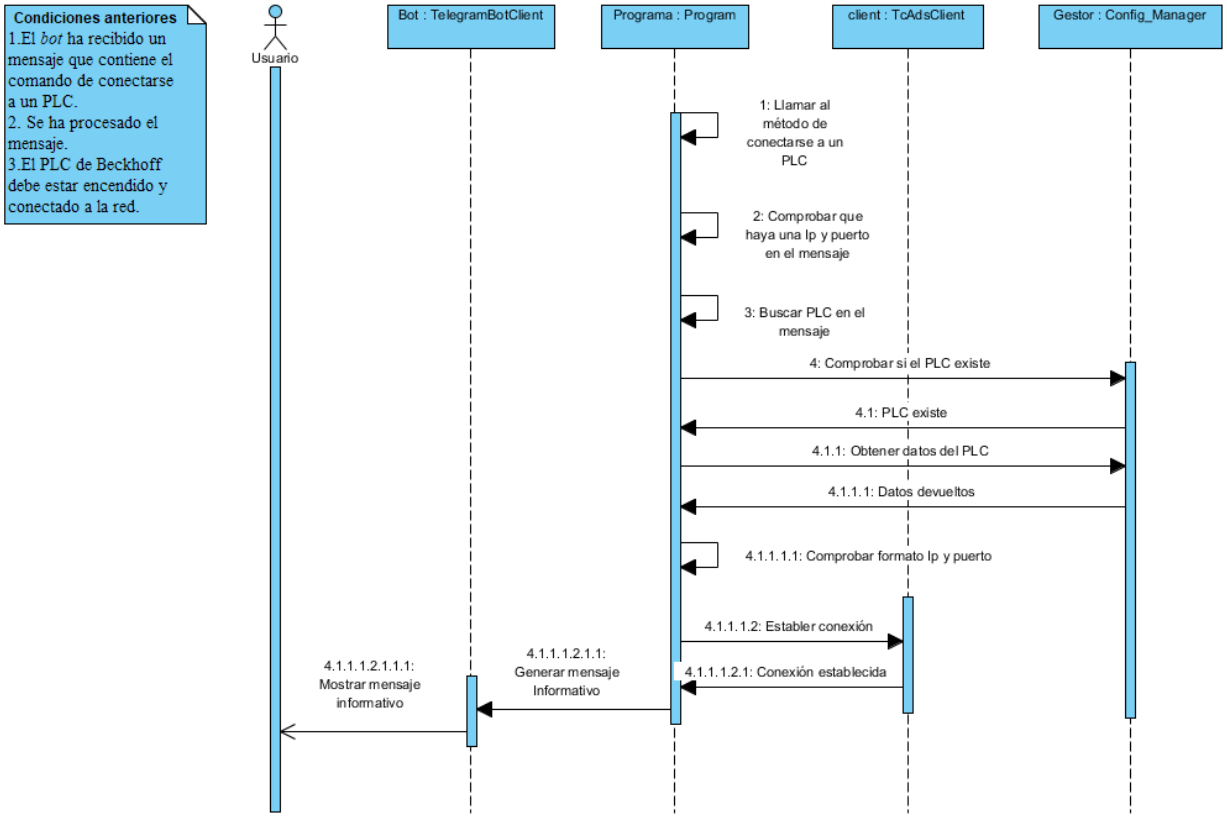
5.5.7.- Procesar archivo



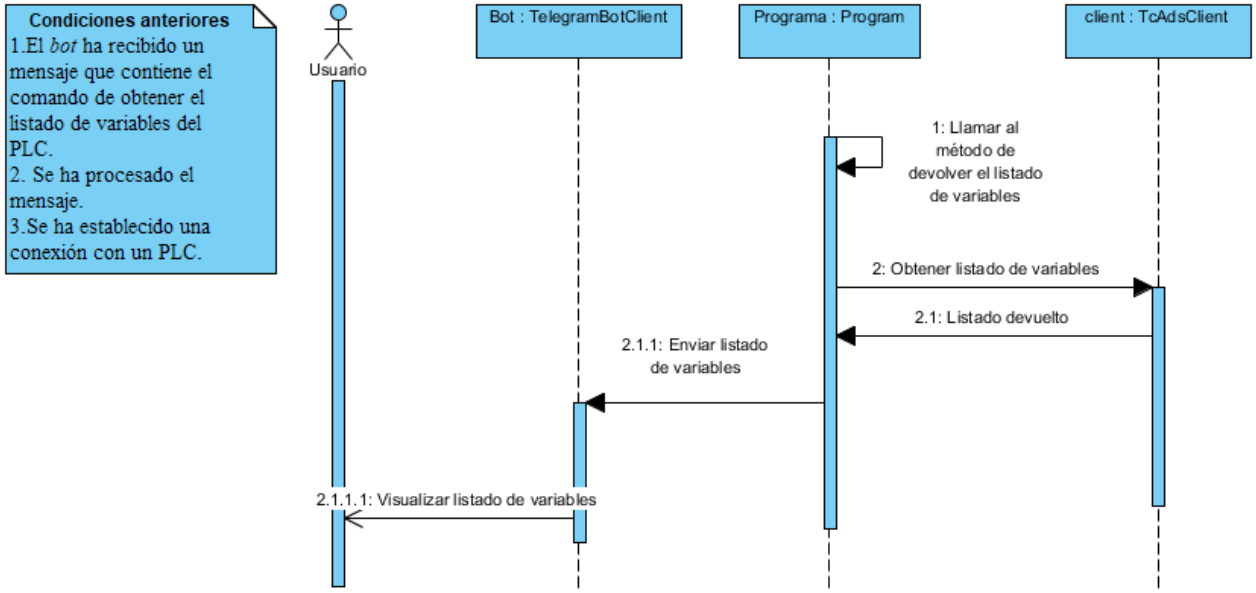
5.5.8.- Conectar con un PLC con Ams y puerto



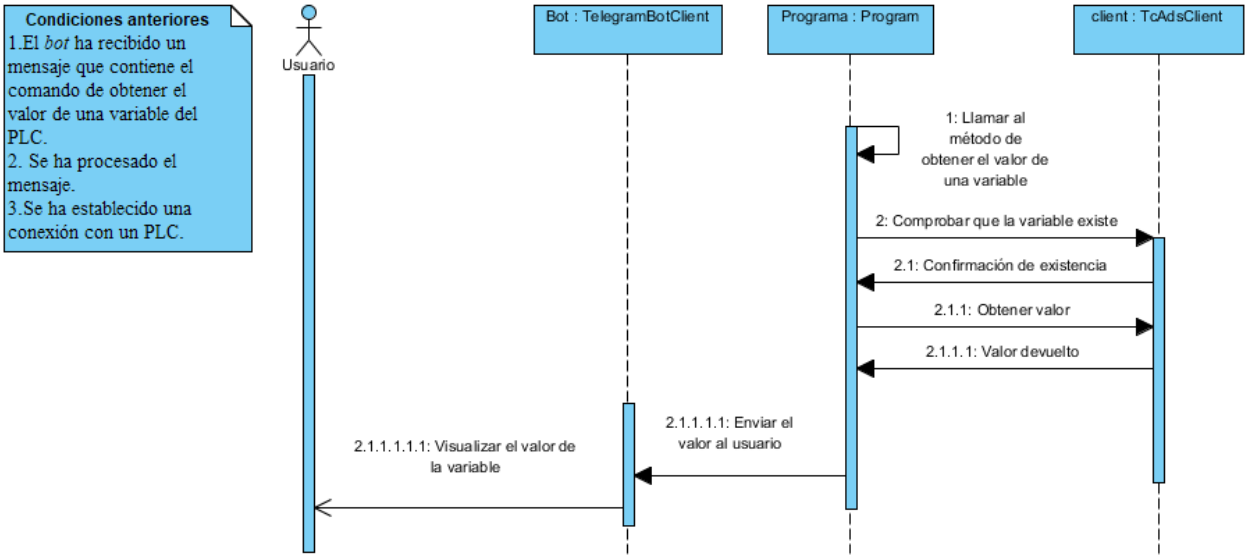
5.5.9.- Conectar con PLC sin Ams ni Puerto



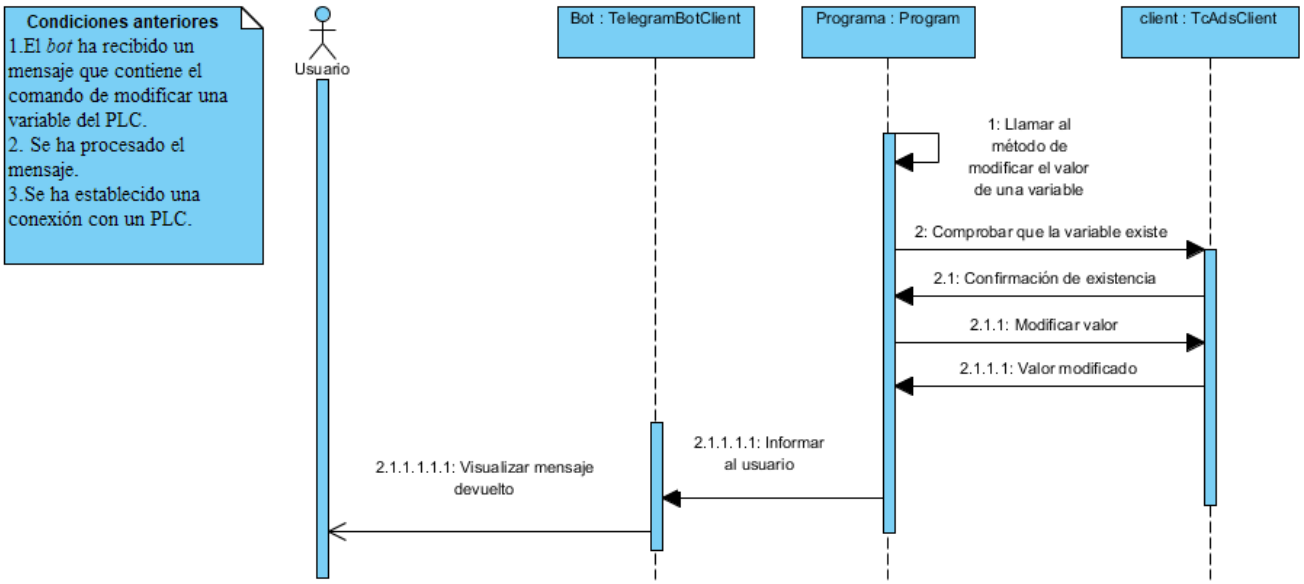
5.5.10.- Obtener el listado de variables de un PLC



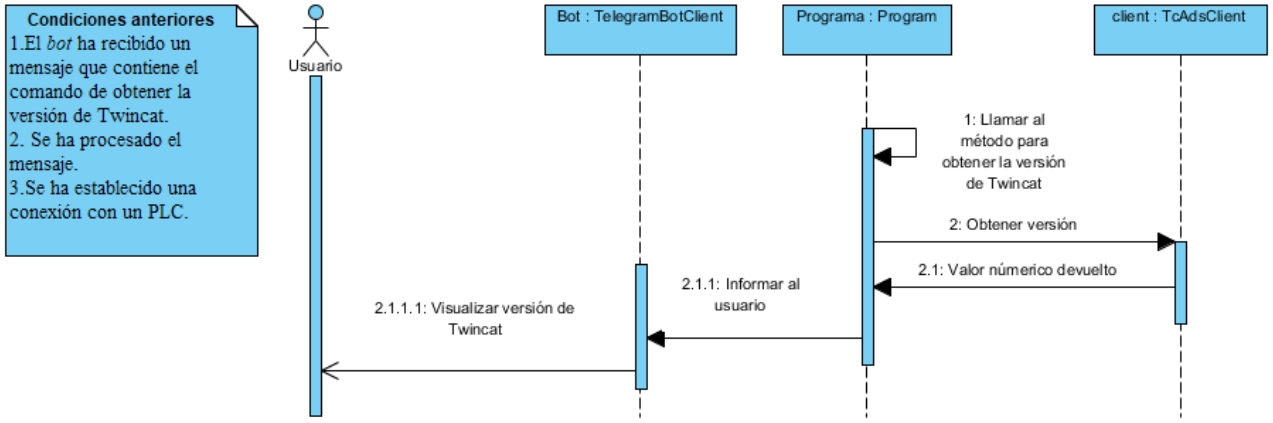
5.5.11.- Obtener el valor de una variable



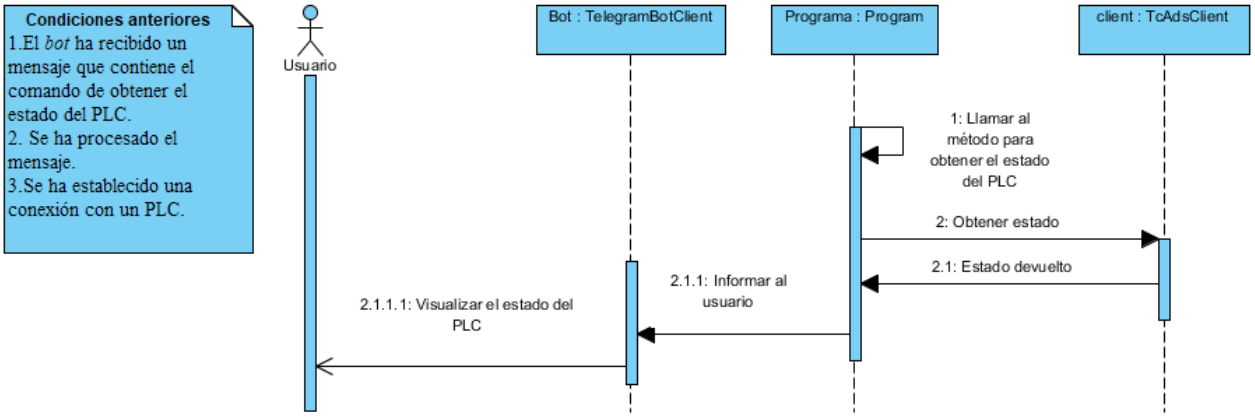
5.5.12.- Modificar variable



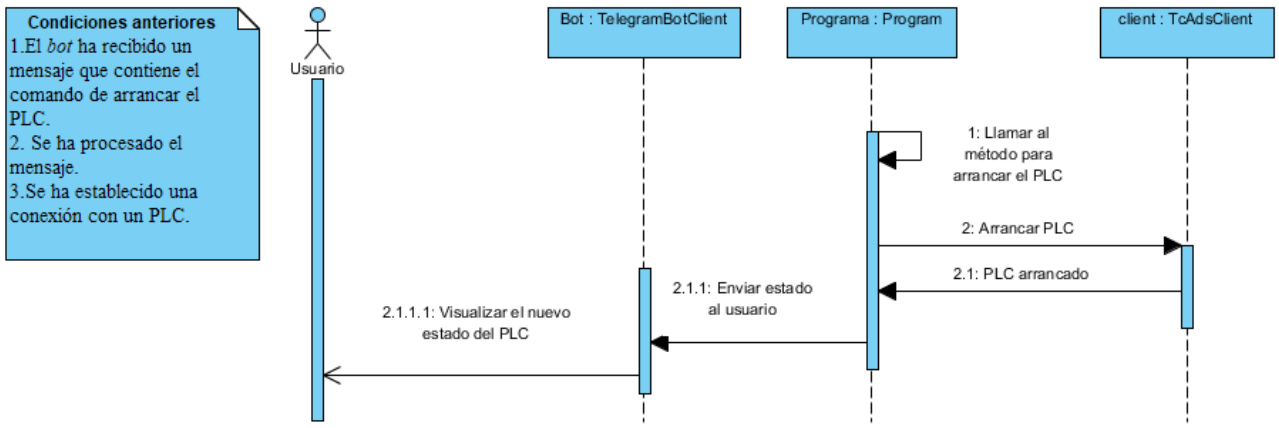
5.5.13.- Obtener versión de Twincat



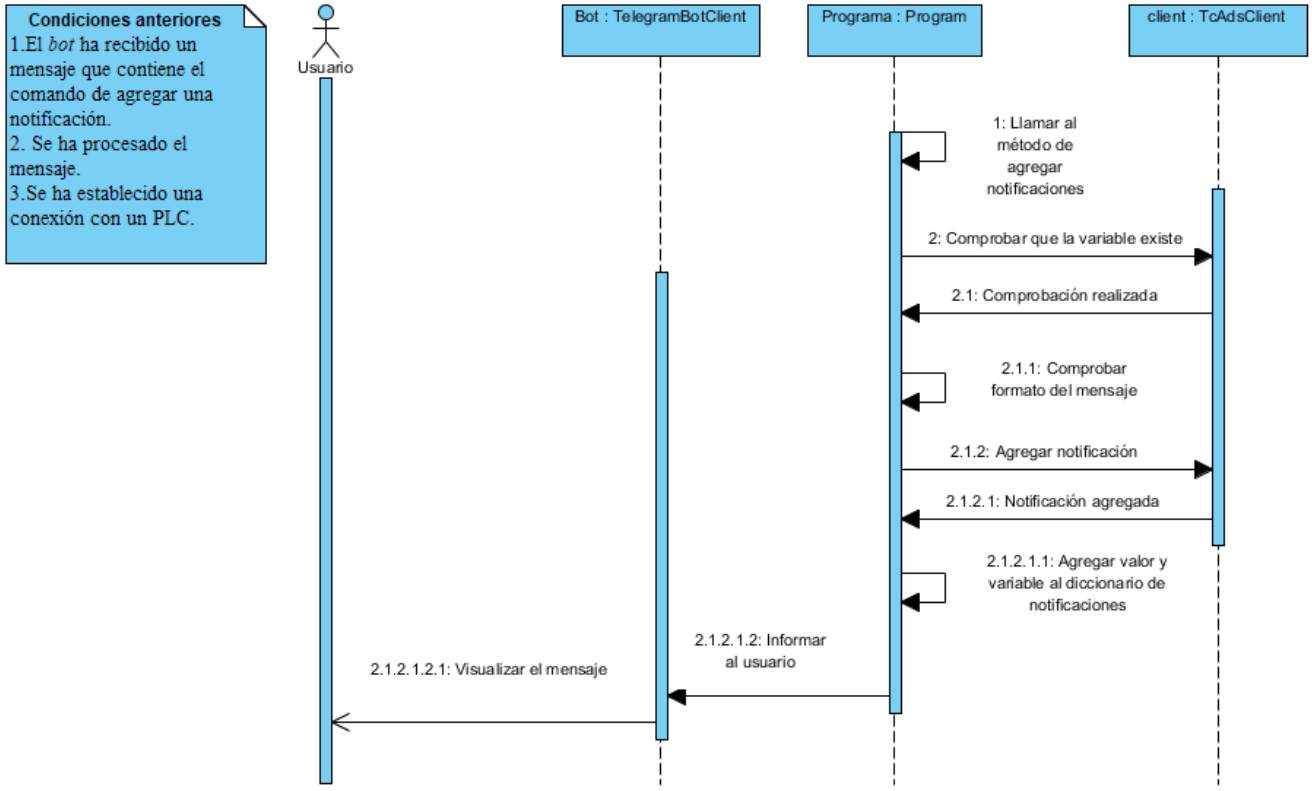
5.5.14.- Obtener estado del PLC



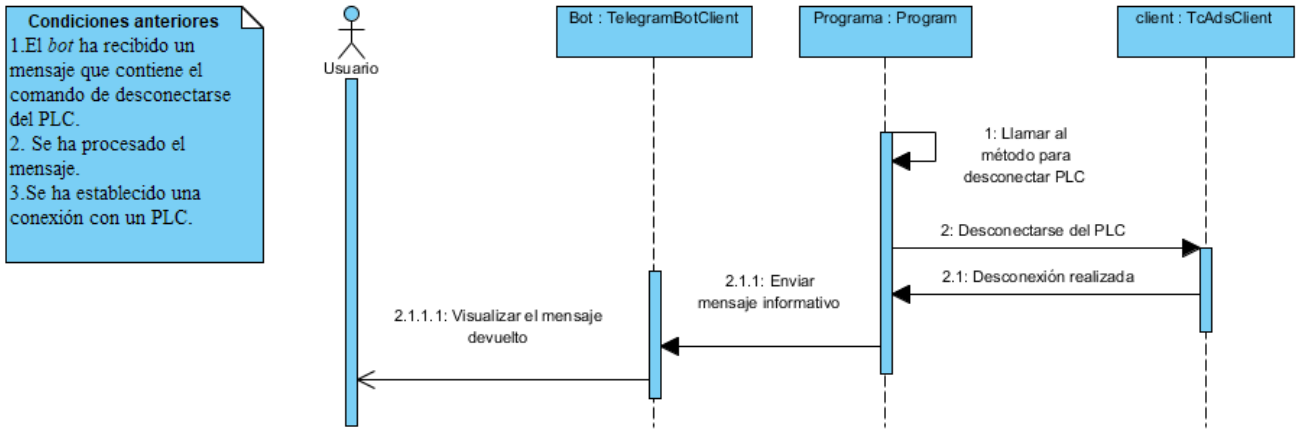
5.5.15.- Arrancar PLC



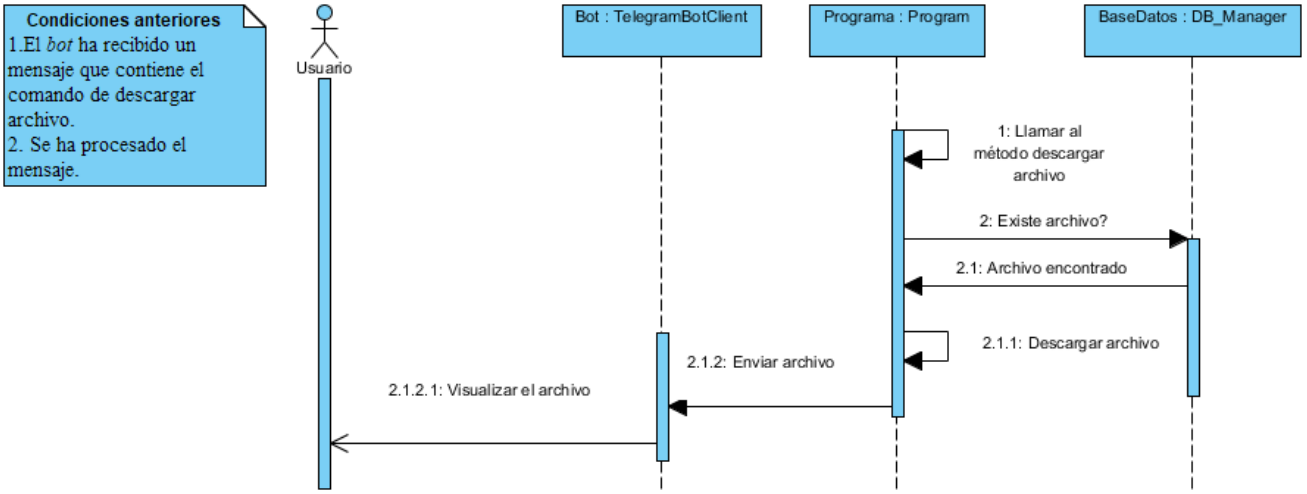
5.5.16.- Agregar notificación



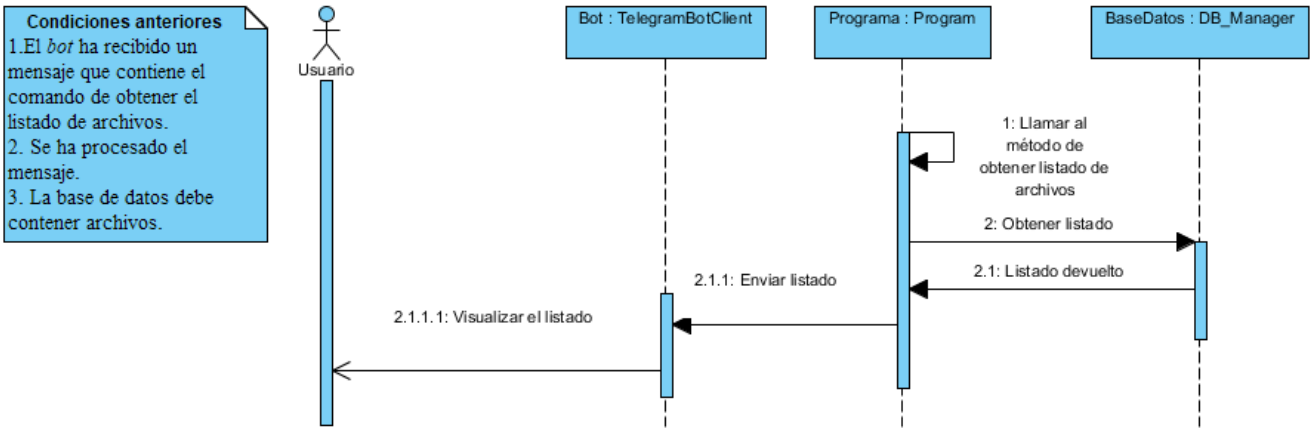
5.5.17.- Desconectarse de un PLC



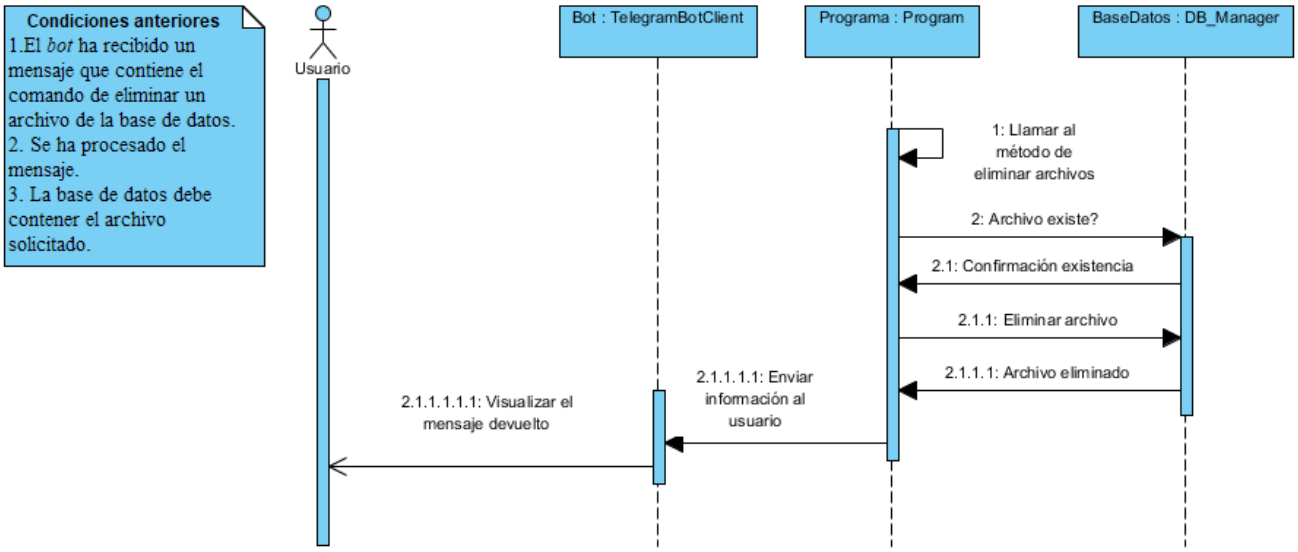
5.5.18.- Descargar archivo



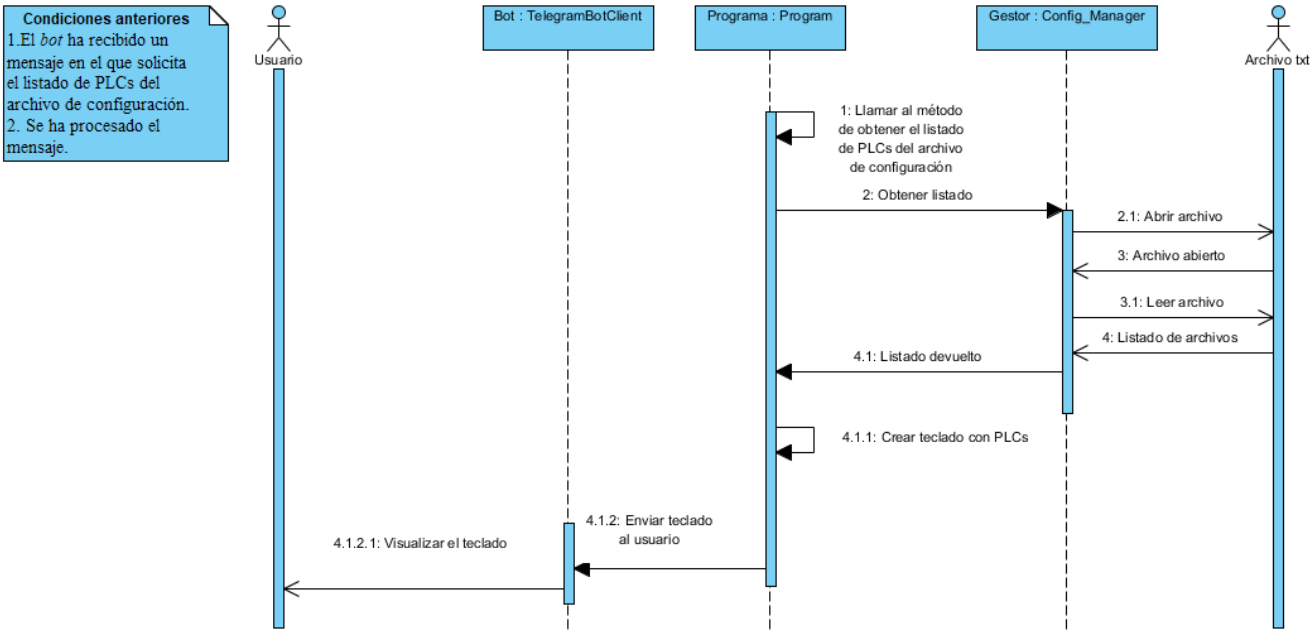
5.5.19.- Listado de archivos



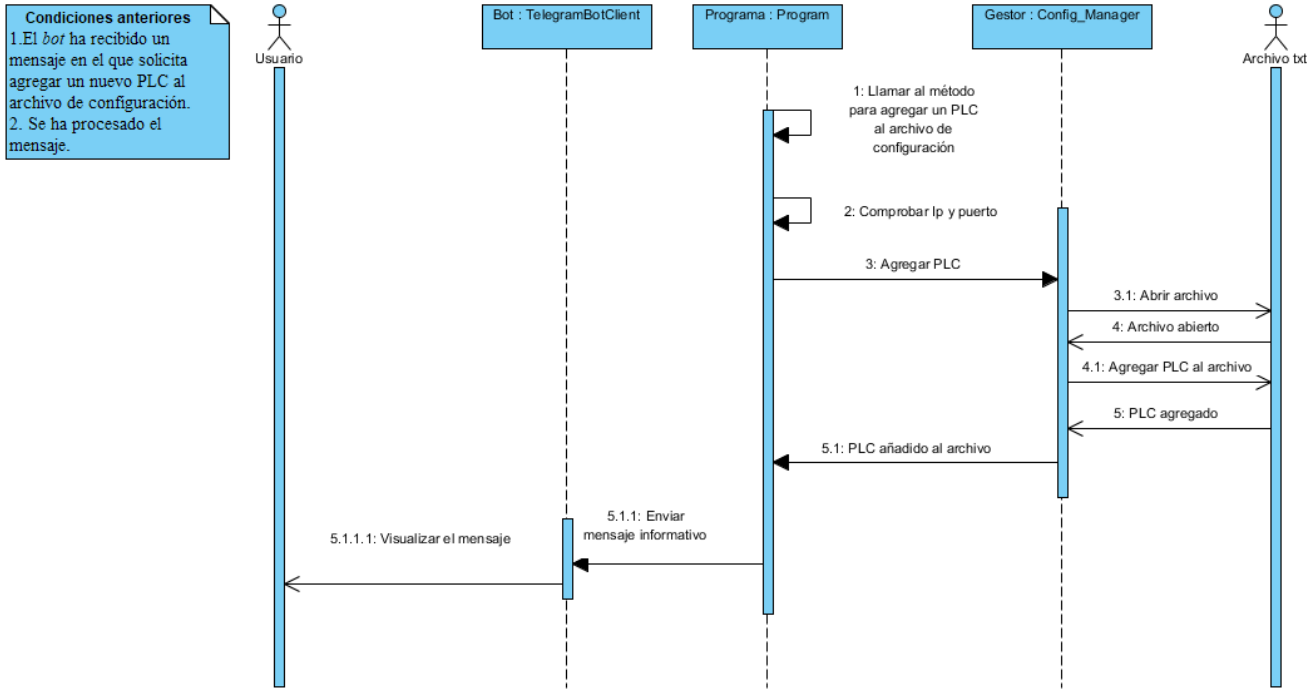
5.5.20.- Eliminar archivo



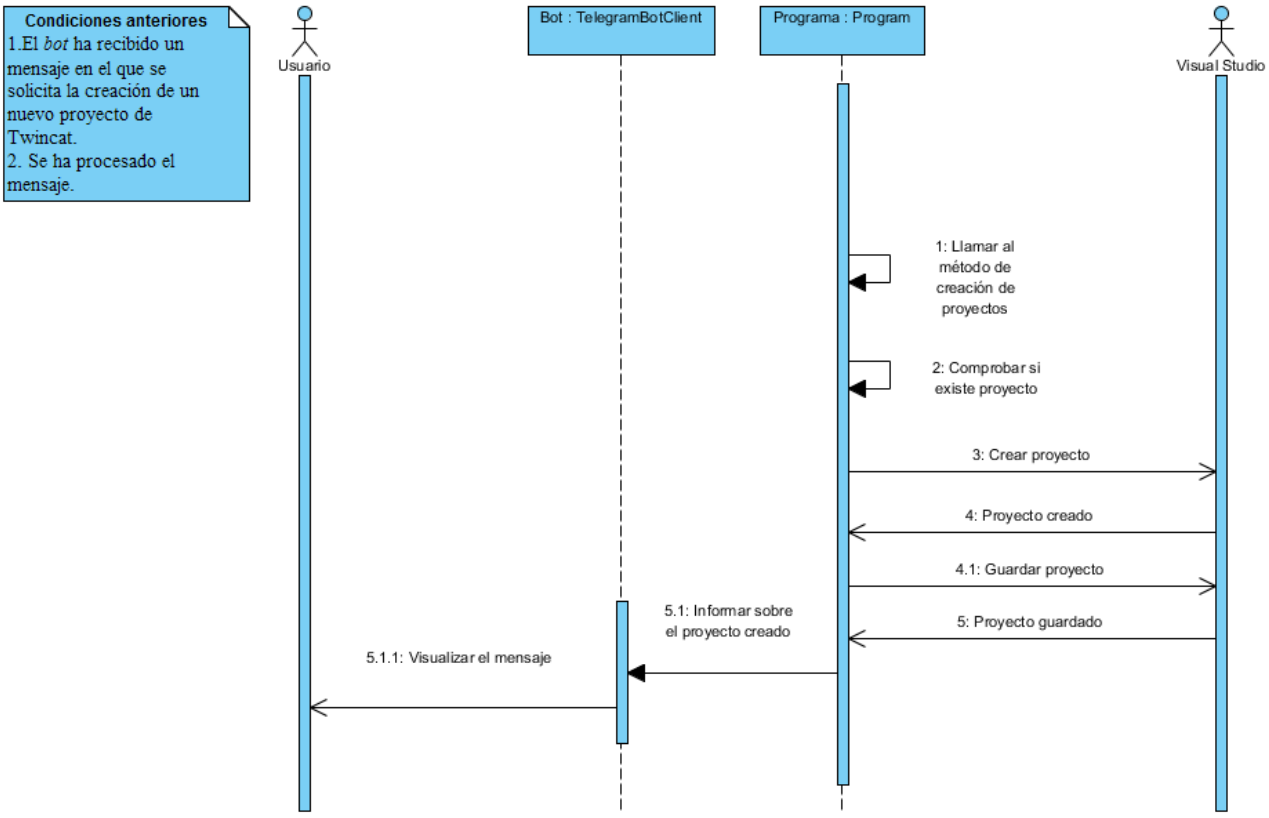
5.5.21.- Obtener PLCs del archivo de configuración



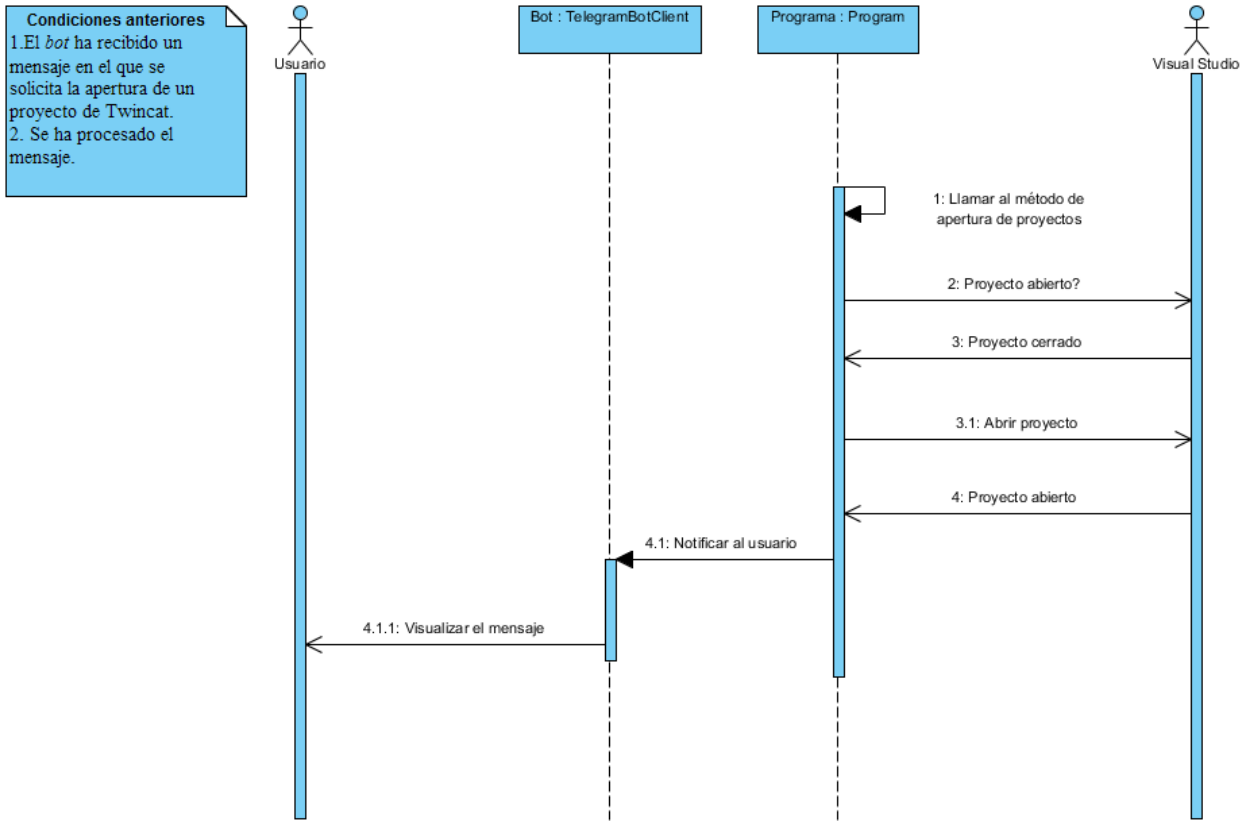
5.5.22.- Agregar PLCs al archivo de configuración



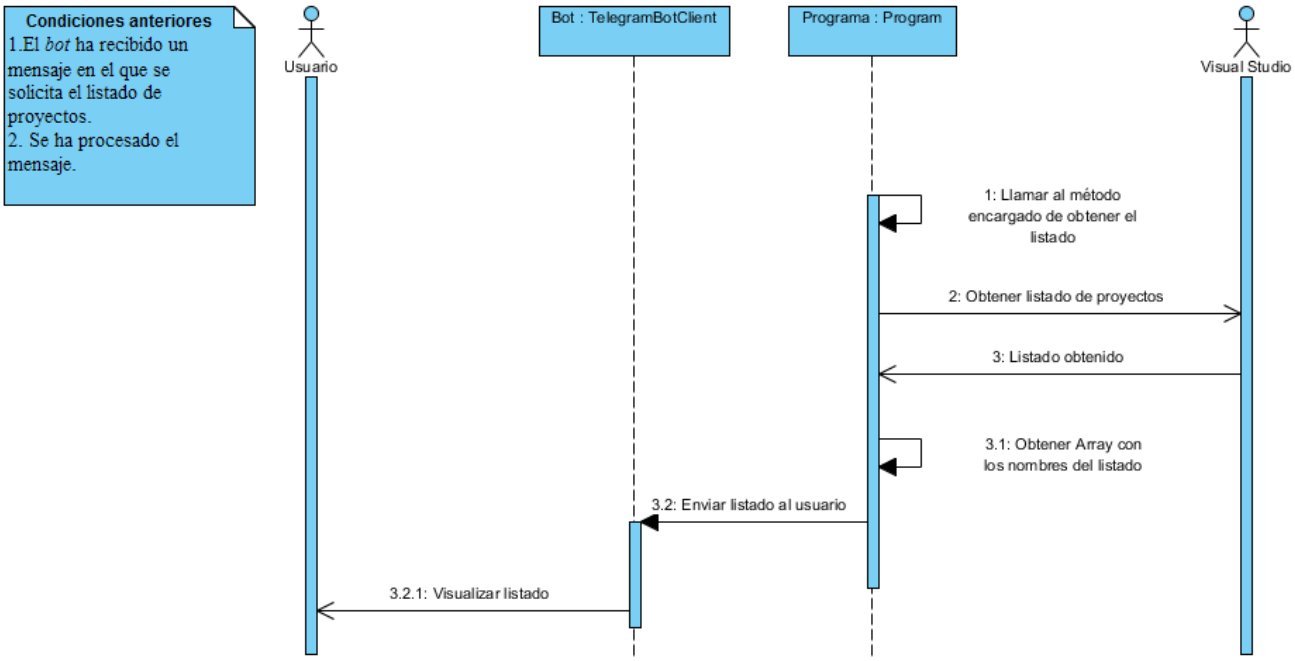
5.5.23.- Crear proyecto de Twincat



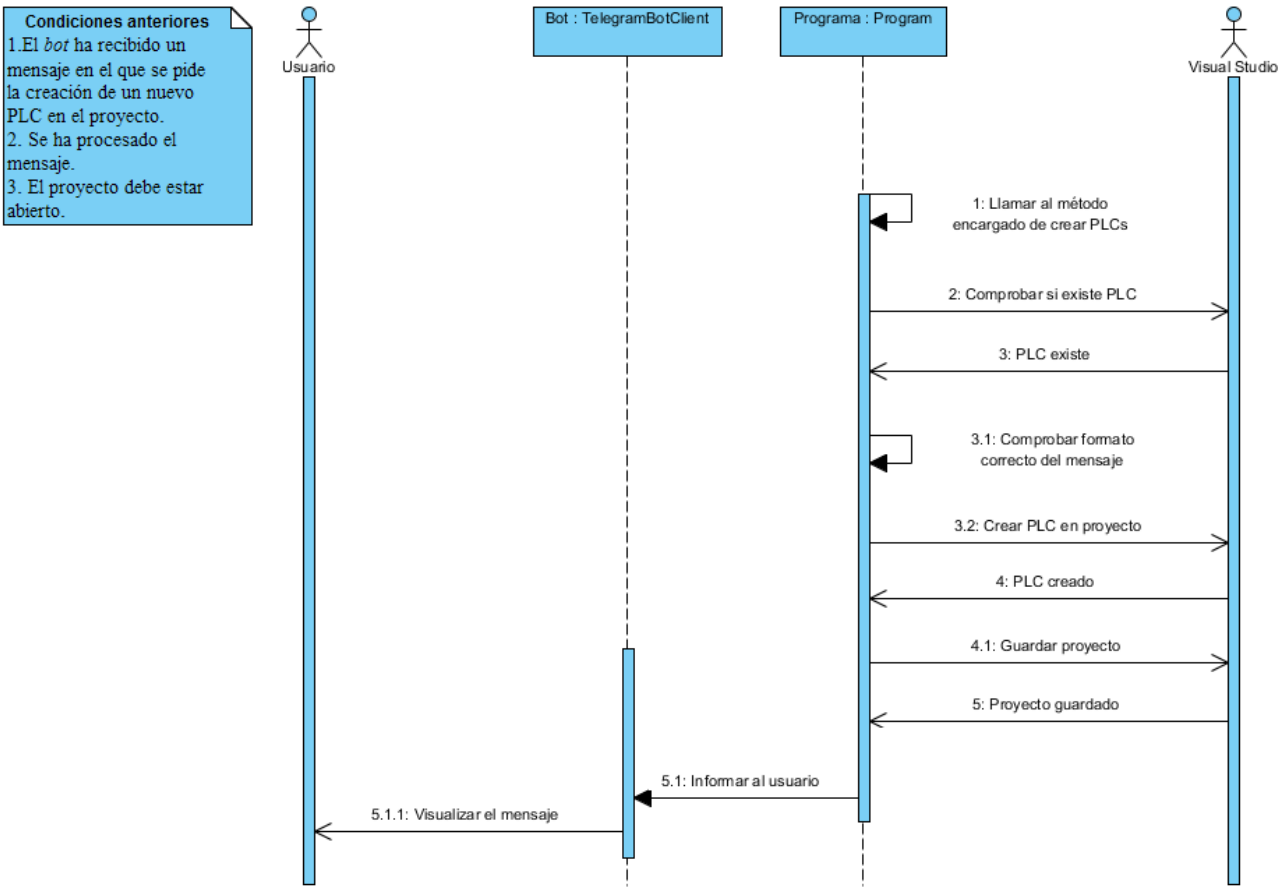
5.5.24.- Abrir proyecto de Twincat



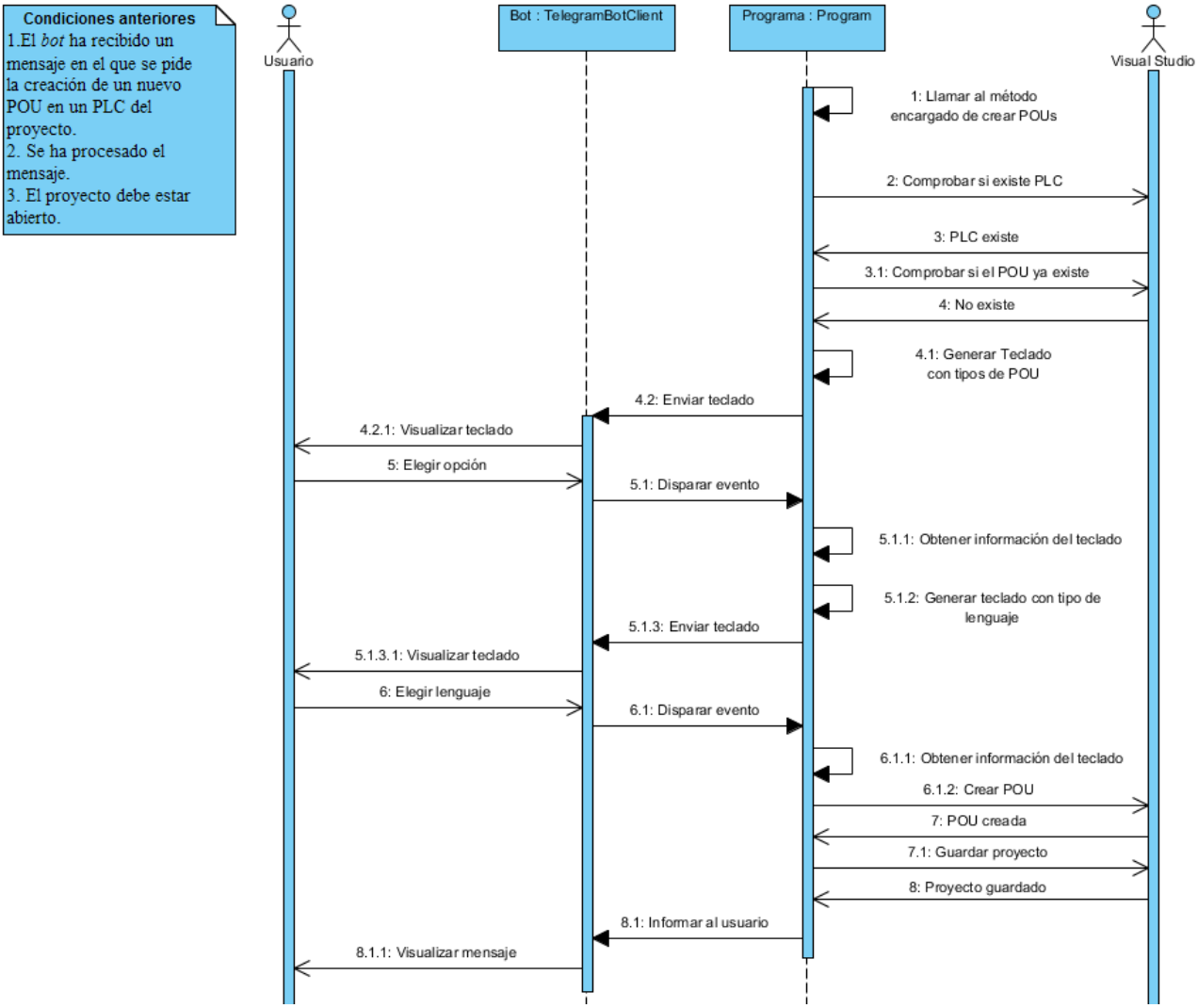
5.5.25.- Listado de proyectos de Twincat



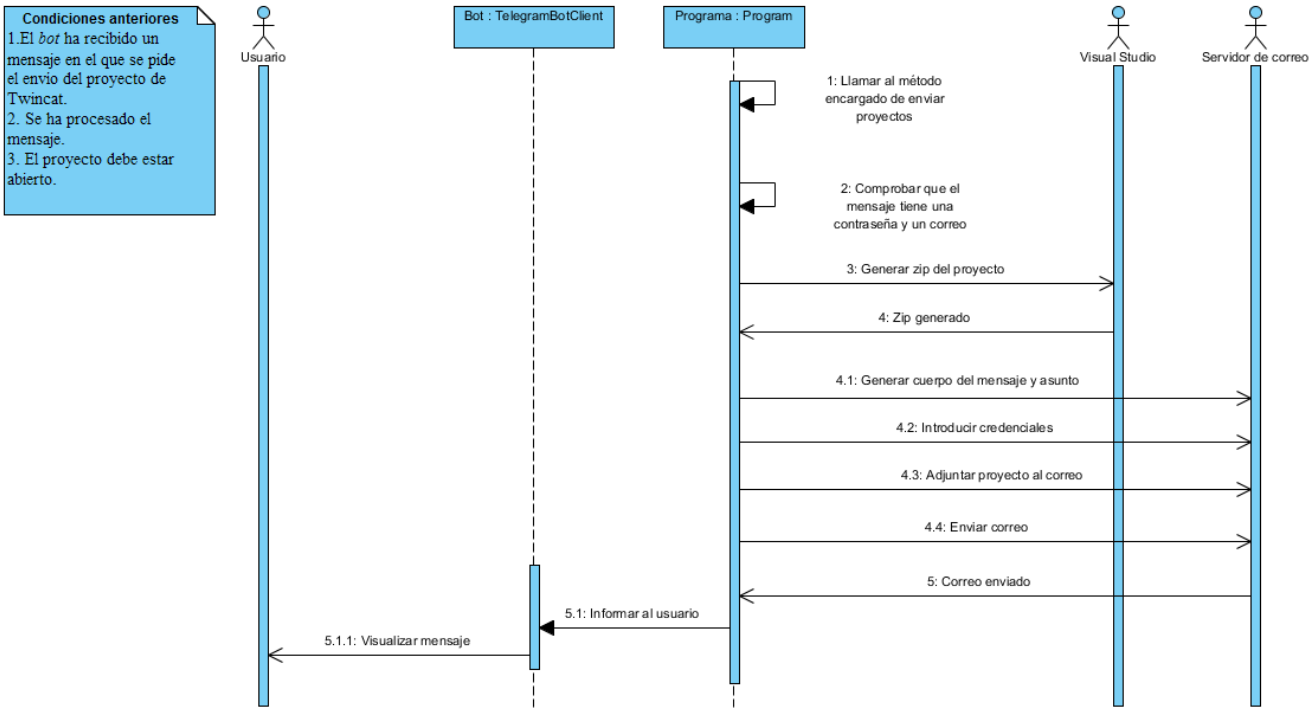
5.5.26.- Crear PLC en proyecto de Twincat



5.5.27.- Agregar POUs a un PLC existente en un proyecto de Twincat



5.5.28.- Enviar proyecto por correo



5.6.- PROGRAMACIÓN EN TWINCAT

Twincat 3 es el software que se utiliza para programar los PLC de Beckhoff. Como se ha mencionado anteriormente está integrado en Visual Studio, lo que facilita su uso.

Figura 5.8.- Ejemplo de la interfaz de Visual Studio junto con Twincat 3.

Twincat 3 está profundamente inspirado en Codesys, con el mismo sistema de POUs y también se programa según el estándar IEC 61131.

Para probar las distintas funcionalidades programadas se crearon varios programas de Twincat a lo largo del desarrollo del proyecto. Para el apartado de resultados se creó un sencillo programa de Twincat con únicamente 3 variables que se fueron modificando y realizando pruebas con ellas. No obstante, el *bot* podría leer cualquier programa, pero resultaría complejo de explicar y de utilizar un programa con muchas variables y POUs.

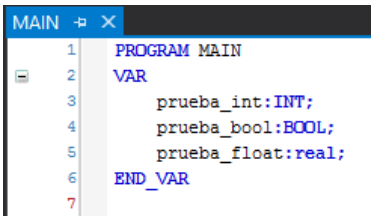


Figura 5.9.- Sencillo programa creado para el apartado de resultados.

Una guía sobre como configurar Twincat se puede leer en el anexo 11.1.

6. Manual de uso y Pruebas de Validación

Para comenzar a utilizar la aplicación lo primero es iniciar una conversación con el *bot*. Para ello se debe buscar en Telegram el nombre de usuario que se le asignó en el apartado 4, "@Luisfirstbot". A partir de este momento ya se puede empezar a enviar mensajes o archivos con el *bot*.

Figura 5.10.- Interfaz básica de Telegram. Arriba a la derecha se observa el icono de buscar (la lupa), lo que permite buscar bots.

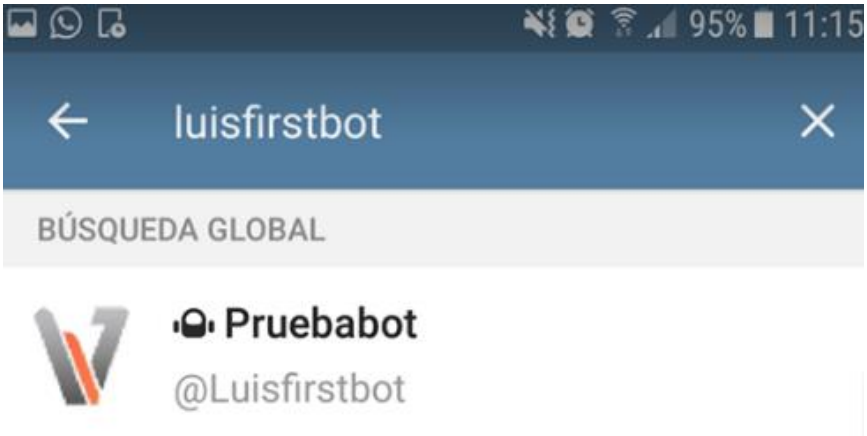


Figura 5.11.- Búsqueda del bot en Telegram.

Posteriormente se debe clicar sobre el *bot* para iniciar una conversación. Inicialmente se muestra un mensaje predefinido durante la creación de este.

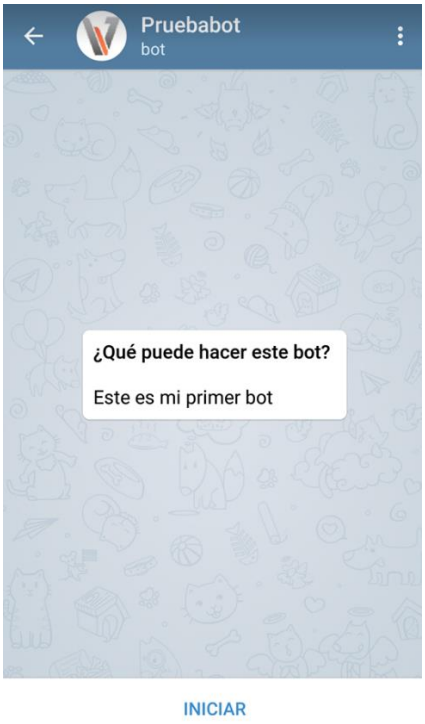


Figura 5.12.- Estado inicial tras comenzar una conversación con el bot.

Tras pulsar sobre el botón "Iniciar" ya se puede comenzar a interactuar con el *bot*.

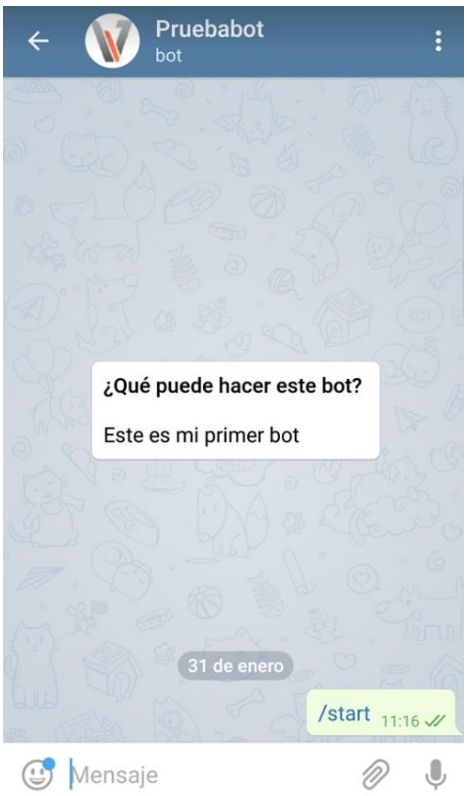


Figura 5.13.- Pantalla de chat con el bot.

En la figura siguiente se pueden observar los tipos de interacción disponibles con el *bot.*

Figura 5.14.- El icono rodeado con un círculo amarillo permite acceder a información general del bot. El rodeado de un círculo verde permite enviar archivos. El rodeado con un círculo rojo permite enviar notas de voz. Por último la zona central donde está situado el cursor permite escribir mensajes de texto.

El *bot* se ha diseñado para que la interacción con él sea de tipo conversacional. Si el usuario no introduce un comando correcto el *bot* le dirá que no le entiende. Además, debido a la información que proporciona el evento que dispara la API de Telegram cuando se recibe un nuevo mensaje, se tiene acceso al nombre del usuario, por lo que para que la interacción resulte más personal, en muchas respuestas se añade el nombre del usuario para que parezca que se está hablando con una persona.

Por otra parte, si el usuario introduce un comando bien "a medias", el *bot* le informará de que le ha entendido pero que algo ha fallado en el mensaje. También se notificará al usuario cuando el mensaje introducido sea correcto pero no se haya podido realizar por algún motivo.

En general, para utilizar el *bot* basta con saber los casos de uso del apartado 5.1 y si se tiene acceso, como es este caso, las palabras clave que el *bot* buscará para reconocer los comandos, ya mencionadas anteriormente (tabla 5.1).

6.1.- RESULTADOS CON MENSAJES DE TEXTO

A continuación se van a mostrar las pruebas realizadas y los mensajes obtenidos para los mensajes de texto. Para esta parte se ha empleado un sencillo programa de Twincat que contiene únicamente 3 variables, de esta forma no se hace difícil ni tediosa la comprobación.

6.1.1.- Conexión con un PLC especificando su IP (Ams) y puerto

Se puede observar que el método permite la introducción de palabras que no tienen que ver con el reconocimiento y que el algoritmo desarrollado simplemente busca las palabras clave.

Figura 6.1- Ejemplo de las pruebas realizadas para conectarse con un PLC especificando su Ams y su puerto.

Si la secuencia hubiera estado bien introducida por el usuario pero el PLC al que se desea conectar no estuviera online o no existiera, se informa al usuario con el mensaje de la figura 6.2.

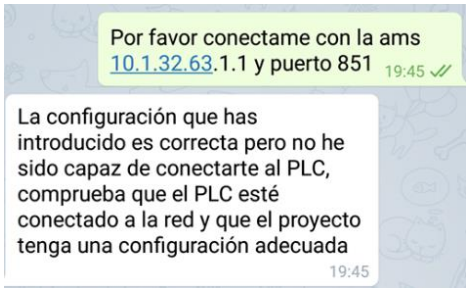


Figura 6.2.- Ejemplo de fallo en la conexión aunque la ip y el puerto sean correctos.

En el caso de que el usuario introduzca una ip inválida se informa también del fallo.

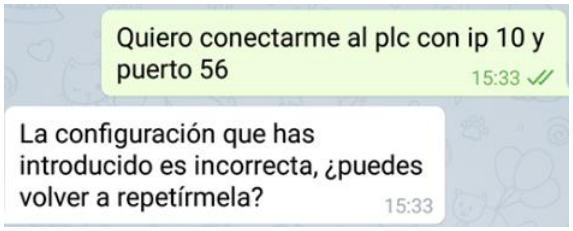


Figura 6.3.- Respuesta del bot ante una secuencia errónea del método de conexión.

6.1.2.- Obtención del listado de variables del PLC

La obtención del listado de variables también se ha probado satisfactoriamente. En este caso hay que tener en cuenta que el PLC además de devolver las tres variables creadas (prueba_float, prueba_int, prueba_bool) devuelve también variables internas, que pueden servir para el programador. En todo caso se podría evitar que estas variables se sacaran por pantalla si fuera necesario.

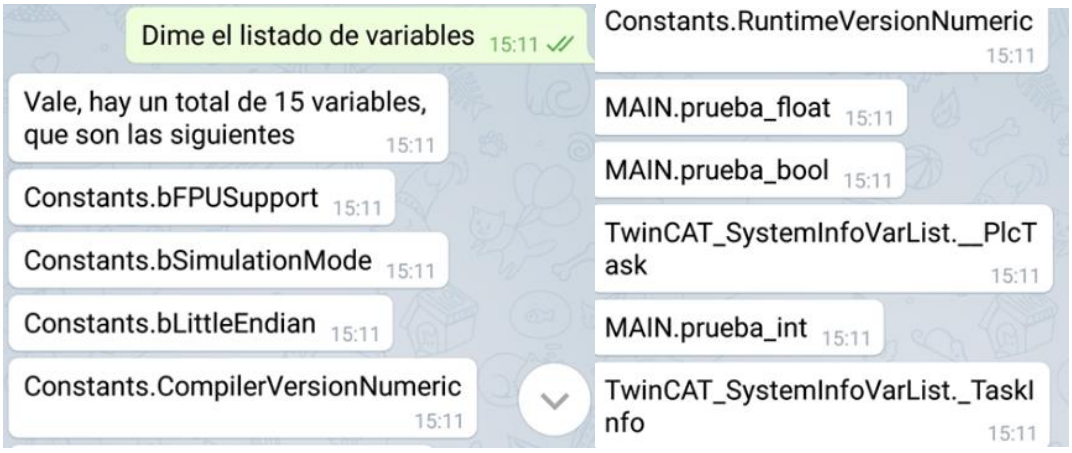


Figura 6.4.- Ejemplo de la obtención del listado de variables.

En el caso de que no hubiera ninguna variable creada o que no se hubiera establecido una conexión con algún PLC se devolvería un mensaje de error como el de la figura 6.5.

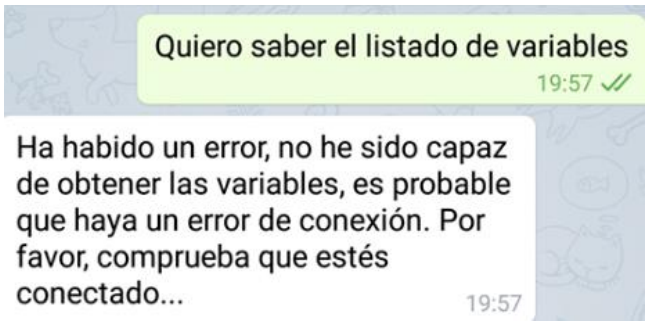


Figura 6.5.- Mensaje que devuelve el bot cuando no puede obtener el listado de variables. Como se puede comprobar, el bot entiende el mensaje pero no es capaz de devolver las variables al usuario por no tener conexión con un PLC.

6.1.3.- Obtener el valor de una variable

Para obtener el valor de una variable se debe introducir su nombre precedido del POU al que pertenece. Puede resultar un poco tedioso pero es el formato requerido por la api de Twincat.

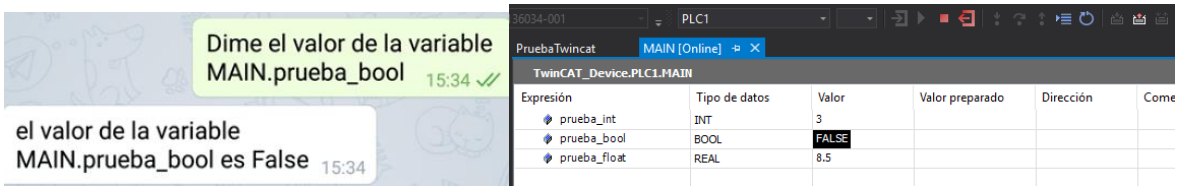


Figura 6.6.- Obtención del valor de la variable "prueba_bool". En la imagen de la derecha se puede ver el valor que marca el PLC, por lo que se puede comprobar que es correcta la respuesta del bot.

Figura 6.7.- Obtención de la variable "prueba_int".

En el caso de que la variable no exista o que haya un error de conexión se informa al usuario.

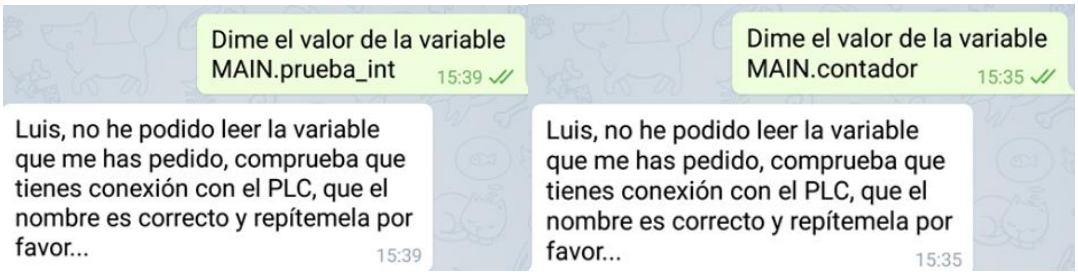


Figura 6.8.- La variable "prueba_int" existe en el PLC pero para demostrar el ejemplo se ha cerrado la conexión con este, por lo tanto el bot informa del "fallo". A la derecha se observa la respuesta ante una variable que de verdad no existe.

6.1.4.- Modificar el valor de una variable

En el proyecto se permite la modificación de variables de tipo simple, como enteros, booleanos y reales. En las figuras siguientes se comprueba el correcto funcionamiento de esta característica.

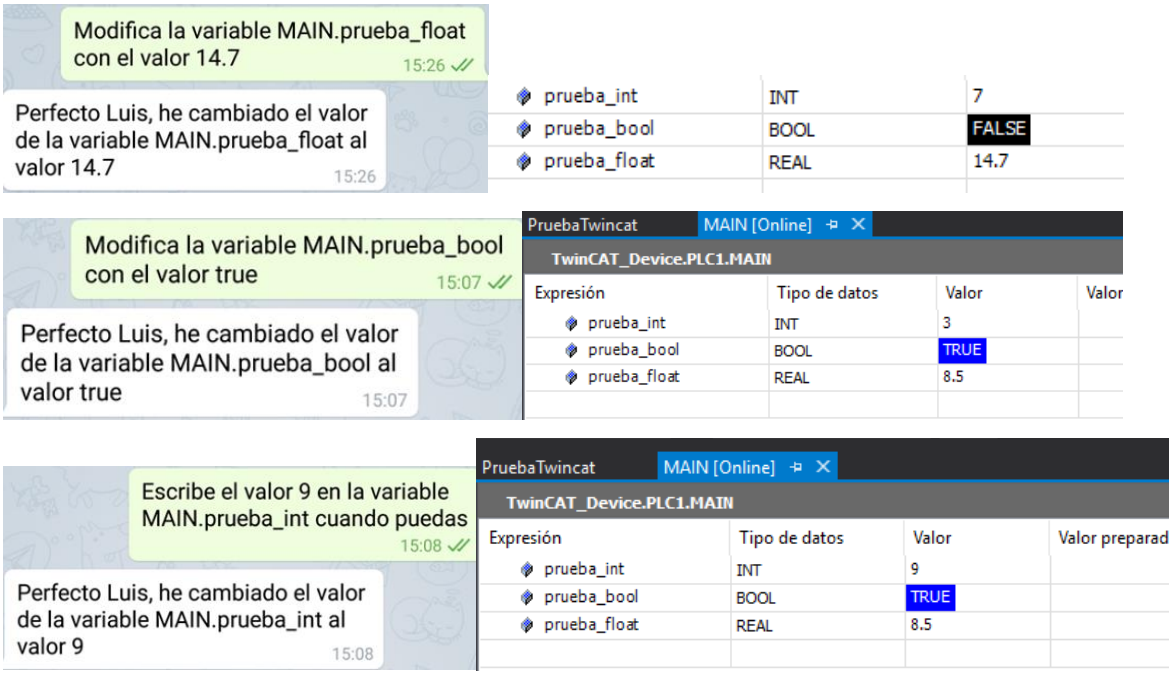


Figura 6.9.- Modificación del valor de una variable del PLC.

Podría darse el caso de que el usuario intentara introducir un valor incorrecto en una variable. El *bot* también respondería con un mensaje de error.

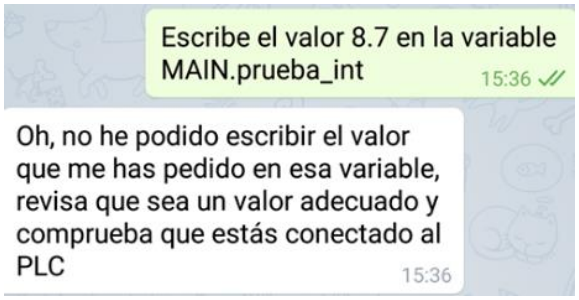


Figura 6.10.- Mensaje de error al intentar escribir un valor no válido con el tipo de una variable.

Si la variable que se desea modificar no está en el PLC, el *bot* avisa al usuario con un mensaje.

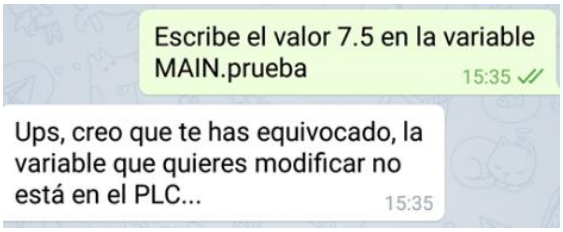


Figura 6.11.- La variable que el usuario quiere modificar no está en el PLC, por lo que el bot informa sobre ello.

6.1.5.- Obtener la versión de Twincat

Para obtener la versión de Twincat se debe establecer una conexión con un PLC. La versión se puede comprobar en el apartado System del proyecto de Twincat que se está cargando en el PLC.

Figura 6.12.- Obtención de la versión de Twincat. En la imagen de la derecha se observa la información proporcionada por Twincat 3, que informa sobre la versión del PLC (3.1).

Como en otras ocasiones, si hay algún error de conexión el *bot* manda un mensaje al usuario.

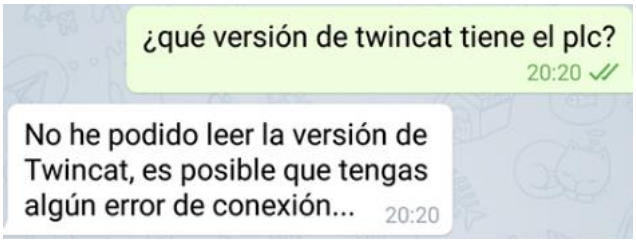


Figura 6.13.- Error en la obtención de la versión de Twincat.

6.1.6.- Obtención del estado del PLC

Cuando se está conectado a un PLC es posible obtener el estado en el que se encuentra. En la figura 6.14 se puede observar que el PLC está en modo Run y el *bot* devuelve el estado correctamente.

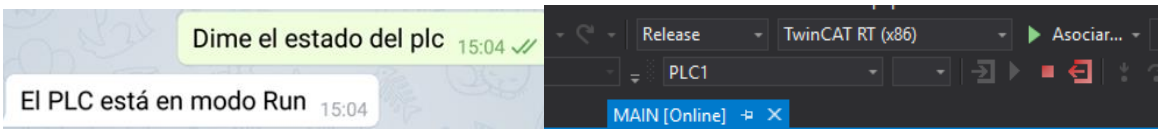


Figura 6.14.- Estado del PLC. En la imagen de la derecha se observa que el icono del cuadrado rojo está disponible, lo que significa que se puede parar el PLC, lo que implica que el PLC está en modo Run

Si hubiera algún error de conexión el *bot* responde consecuentemente.

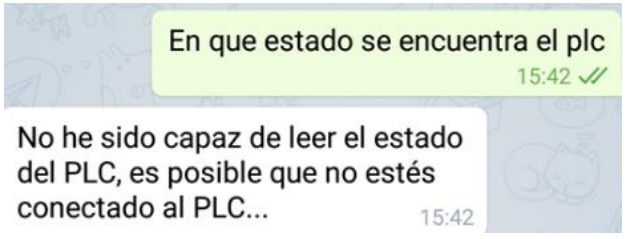


Figura 6.15.- Mensaje que devuelve el bot cuando no hay una conexión establecida con un PLC.

6.1.7.- Arrancar/Parar un PLC

Las pruebas realizadas demuestran que los métodos programados permiten parar y arrancar el PLC sin problemas.

Figura 6.16.- Arranque y parada de un PLC con el que se ha establecido una conexión.

Si se llama a alguna de estas funciones sin haber establecido previamente una conexión con un PLC, el *bot* devolverá un mensaje informando de lo ocurrido.

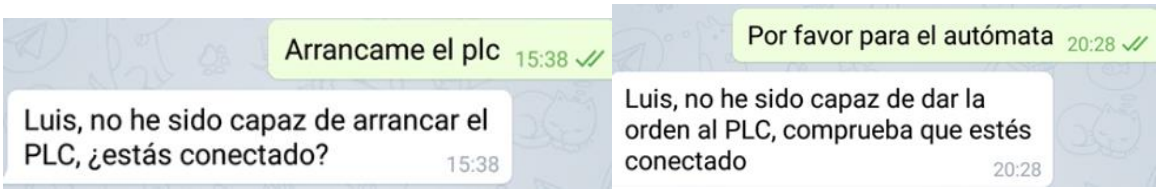


Figura 6.17.- Errores obtenidos al intentar arrancar y parar un PLC sin haberse conectado.

6.1.8.- Desconectarse de un PLC

Se puede comprobar que el *bot* es capaz de desconectar al usuario del PLC.

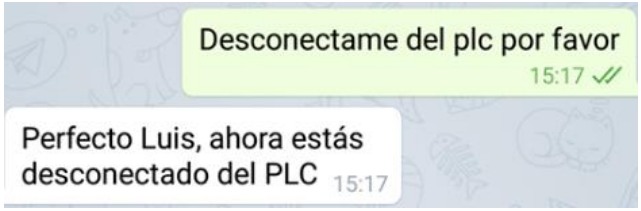


Figura 6.18.- Desconexión del PLC.

6.1.9.- Agregar notificaciones

Para las pruebas de las notificaciones se han agregado avisos sobre las 3 variables creadas en el programa de Twincat. Para recibir los avisos se han modificado manualmente

las variables empleando los comandos programados, en este caso el de modificar el valor de una variable

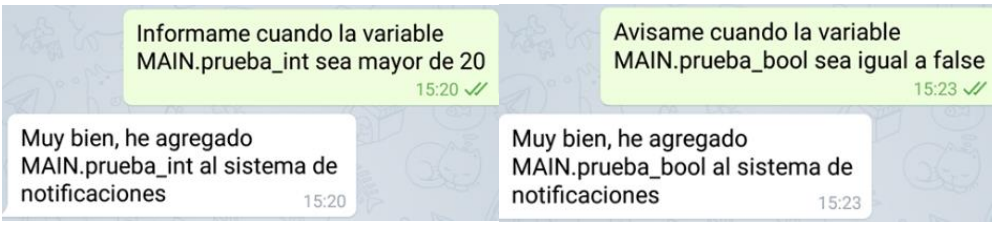


Figura 6.18.- Ejemplos para agregar una variable al sistema de notificaciones.

Cuando una variable del sistema de notificaciones cumple las condiciones de alarma, se avisa al usuario.

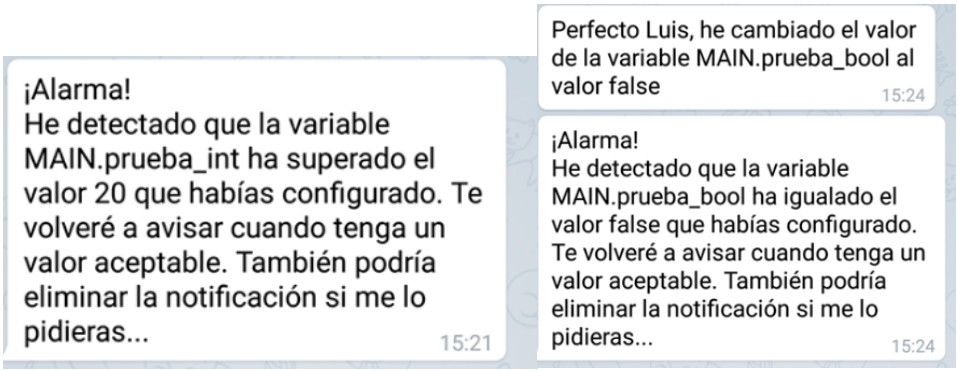


Figura 6.19.- En la izquierda se muestra el aviso de la variable "prueba_int". En la imagen de la derecha se modifica la variable "prueba_bool" para disparar la notificación.

Cuando una variable sobre la que se había informado vuelve a un valor que no cumple las condiciones de alarma, se notifica al usuario.

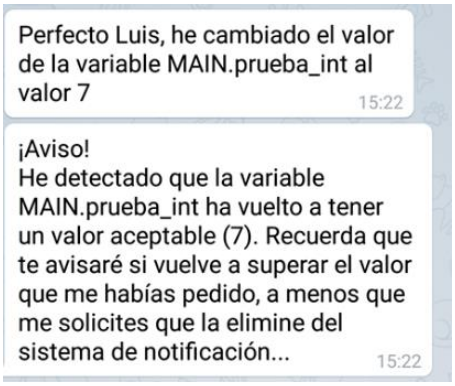


Figura 6.20.- La variable "prueba_int" ha vuelto a alcanzar un valor aceptable.

Si se intenta agregar una notificación sobre una variable que no existe en el PLC, entonces el *bot* se lo comunica al usuario.

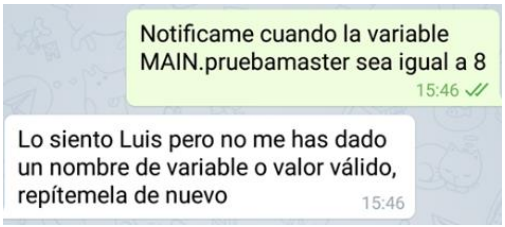


Figura 6.21.- No se pueden agregar notificaciones sobre variables que no existen en el programa del PLC.

6.1.10.- Obtención del listado de archivos de la base de datos

Se comprueba que se puede leer correctamente de la base de datos.

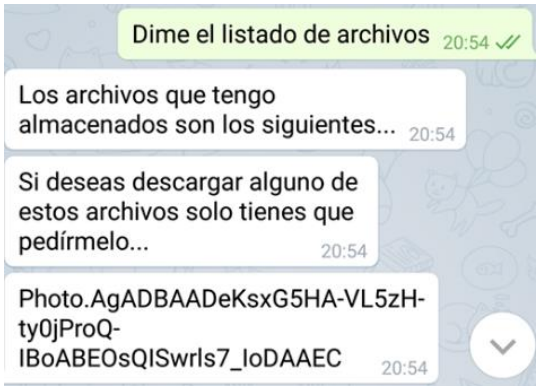


Figura 6.22.- Lectura de los archivos de la base de datos.

Si no se tuviera ningún archivo registrado el *bot* informaría sobre ello.

Figura 6.23.- Ejemplo de respuesta cuando no hay ningún archivo en la base de datos.

6.1.11.- Descarga de archivos

En este sub-apartado se prueba la descarga de archivos. Como se puede ver en la figura 6.24, si el archivo solicitado no existe el *bot* se lo comunica al usuario.

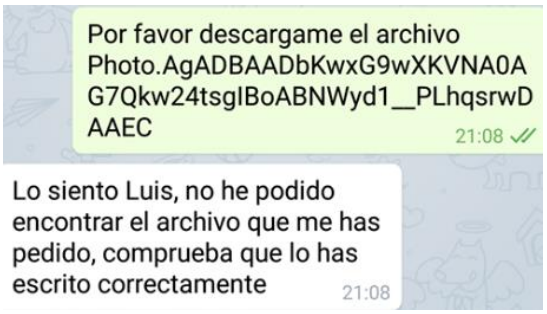


Figura 6.24.- Ejemplo de fallo al intentar descargar un archivo que no existe.

Si por el contrario, el archivo existe en la base de datos, entonces el *bot* lo descarga y lo manda al usuario.

Figura 6.25.- Ejemplo de descarga de un archivo almacenado en la base de datos. El bot devuelve el archivo tras buscarlo en los servidores de Telegram. En este caso se trataba del cartel de una película infantil.

6.1.12.- Eliminar un archivo de la base de datos

Para eliminar un archivo de la base de datos basta con introducir su nombre. Para probar que la eliminación se realizaba correctamente se probó a eliminar un archivo y volver a pedir al *bot* que lo eliminara de nuevo. Se comprobó que efectivamente el *bot* lo eliminaba, ya que la segunda vez ya no lo detectaba.

Figura 6.26.- Eliminación de un archivo de la base de datos (izquierda) y su posterior comprobación (derecha).

6.1.13.- Obtención del listado de PLCs del archivo de configuración

El listado de PLCs del archivo de configuración se muestra por pantalla en forma de teclado. De forma que el usuario puede conectarse pulsando sobre el PLC que desee.

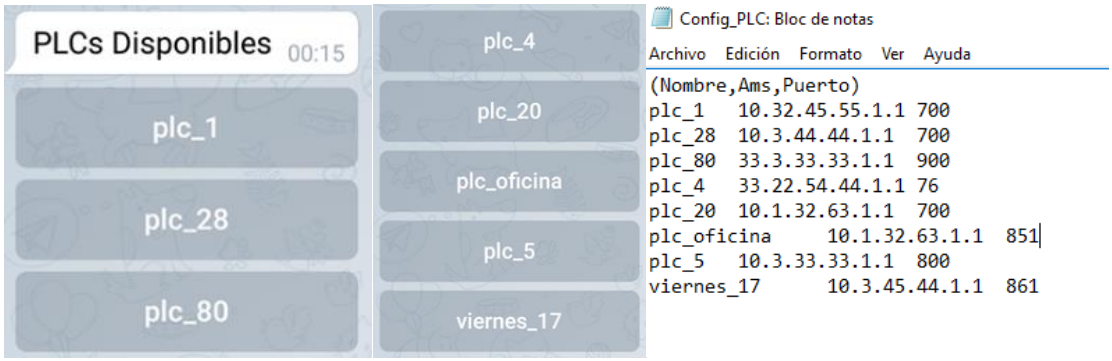


Figura 6.27.- Obtención del teclado que muestra el listado de PLCs del archivo de configuración. A la derecha se observa que los PLCs del txt son los que efectivamente muestra el bot.

Cuando se pulsa sobre uno de ellos se intenta establecer la conexión utilizando los datos almacenados en el txt.

Si la conexión se pudo establecer se informa al usuario, tal y como se ve en la figura 6.28. También se puede establecer una conexión poniendo el nombre del PLC al que se desea conectar.

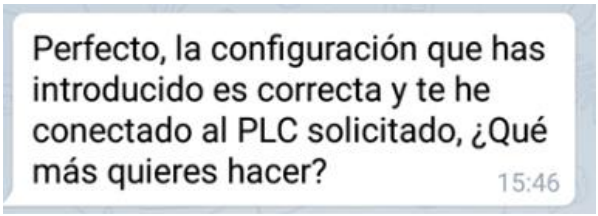


Figura 6.28.- Mensaje devuelto tras pulsar sobre un PLC y haber establecido la conexión.

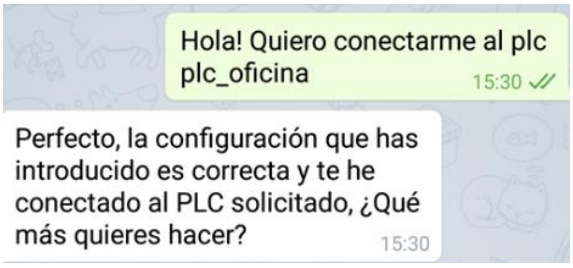


Figura 6.29.- Conexión directa con un PLC del archivo de configuración.

Si ha habido algún error de conexión se muestra el mensaje correspondiente (figura 6.30). De la misma manera, si se intenta conectar con un PLC que no está en el archivo de configuración se devuelve un mensaje de error (figura 6.31).

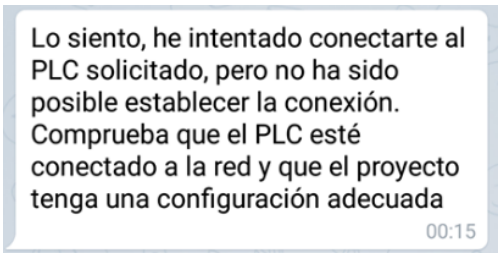


Figura 6.30.- Error al intentar conectarse con un PLC del archivo de configuración que no está Online.

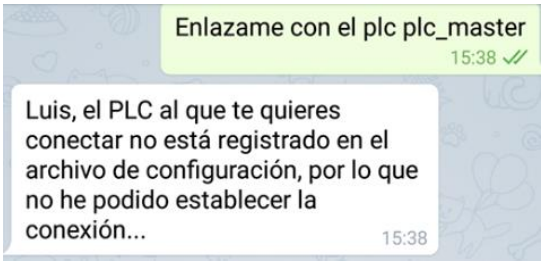


Figura 6.31.- Mensaje devuelto cuando se intenta establecer una conexión con un PLC que no está en el archivo de configuración.

6.1.14.- Agregar un PLC al archivo de configuración

Para añadir un PLC al archivo de configuración se debe indicar su nombre, dirección ams y puerto. En la figura 6.32 se prueba que el *bot* es capaz de añadir un PLC al archivo de configuración.

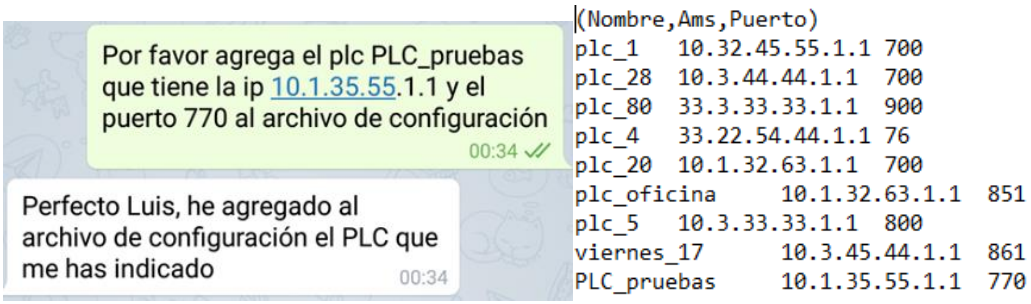


Figura 6.32.- A la izquierda se muestra el mensaje enviado al bot y que este ha respondido favorablemente. A la derecha se comprueba que efectivamente se ha agregado el PLC solicitado al txt de configuración.

Si se intenta añadir un PLC con un datos inválidos se avisa al usuario para que repita el mensaje.

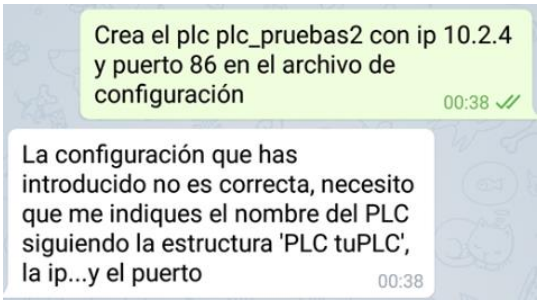


Figura 6.33.- Se comprueba que el bot informa al usuario cuando este intenta añadir un PLC con unos datos erróneos al archivo de configuración.

6.1.15.- Eliminar un PLC del archivo de configuración

Eliminar un PLC del archivo de configuración es muy similar a como se eliminaban los archivos de la base de datos. Basta con dar su nombre al *bot* y este lo eliminará. En la figura 6.34 se comprueba este funcionamiento.

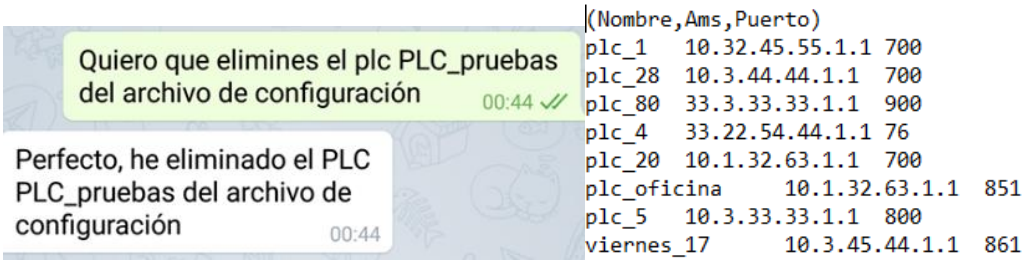


Figura 6.34.- Se comprueba que al eliminar un PLC realmente desaparece del archivo de configuración.

Si ahora se vuelve a solicitar al *bot* que elimine ese mismo PLC debería devolver un mensaje de error. Efectivamente en la figura 6.35 se comprueba que el método funciona como se esperaba.

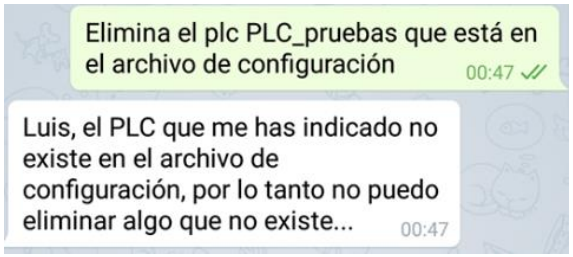


Figura 6.35.- Se comprueba que si un PLC no existe en el archivo de configuración el bot devuelve un mensaje informando.

6.1.16.- Obtención del listado de proyectos de Twincat

Si todo funciona como debería, al solicitar al *bot* el listado de proyectos debería devolver el nombre de los proyectos que encuentre en la carpeta predefinida en el programa. Se realizan a continuación las comprobaciones pertinentes.

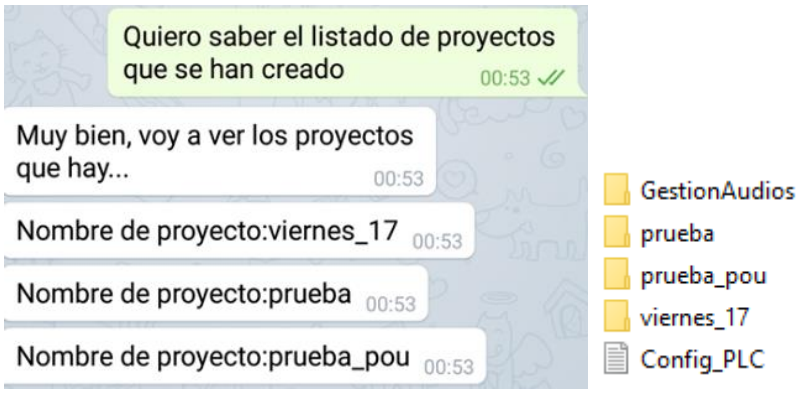


Figura 6.36.- En la imagen de la izquierda se comprueba que el bot devuelve los proyectos encontrados. En la derecha se ve la carpeta que está leyendo el bot y se observa que ha devuelto correctamente los proyectos existentes (GestionAudios es una carpeta donde se almacenan las notas de voz, no tiene proyectos, por eso no se incluye en el listado).

6.1.17.- Creación de un proyecto de Twincat

En este sub-apartado se prueba el método de crear proyectos de Twincat desde el *bot*. Como se observa en la figura 6.37, el *bot* realiza la operación tal y como se esperaba.

Se informa al usuario de que puede tardar un poco en crear el nuevo proyecto. Una vez creado se manda un nuevo mensaje informativo.

Figura 6.37.- Comprobación de que el bot es capaz de crear nuevos proyectos de Twincat.

A continuación se realiza una prueba para ver si el *bot* es capaz de reconocer un proyecto que ya está creado. Se utiliza para el ejemplo el mismo proyecto que en el caso anterior, Master_TFM.

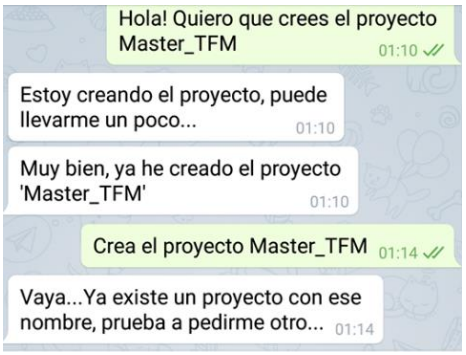


Figura 6.38.- Comprobación de que una vez creado un proyecto, si se vuelve a intentar crear otro con el mismo nombre el bot no lo permite.

6.1.18.- Abrir proyectos de Twincat

Se va a probar ahora que el *bot* es capaz de abrir proyectos de Twincat.

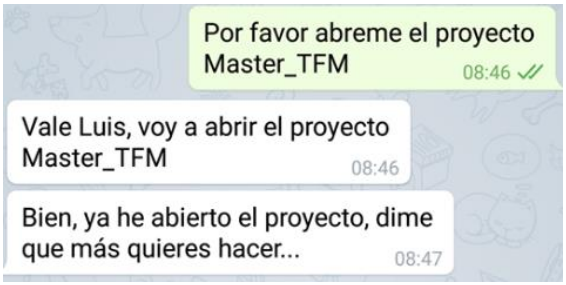


Figura 6.39.- Se prueba que el bot es capaz de abrir un proyecto de Twincat.

Si se intenta abrir un proyecto que ya estaba abierto, el *bot* debería responder con ese fallo.

Figura 6.40.- El bot es capaz de detectar que el proyecto indicado ya está abierto.

6.1.19.- Cerrar un proyecto de Twincat

Para probar esta funcionalidad se utilizó el mismo proyecto que en el sub-apartado anterior, "Master_TFM". El resultado se puede observar en la figura 6.41.

Figura 6.41.- Se comprueba que el bot es capaz de cerrar un proyecto (arriba). En la parte de abajo se hace la prueba de intentar cerrar un proyecto que ya estaba cerrado, dando como resultado que el bot lógicamente no es capaz de cerrar un proyecto que no está abierto.

6.1.20.- Crear PLCs en un proyecto abierto de Twincat

Se va a comprobar ahora si el *bot* es capaz de añadir PLCs a un proyecto de Twincat.

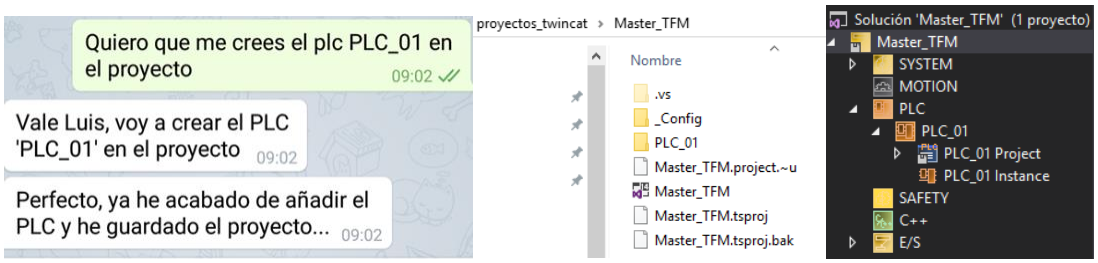


Figura 6.42.- Creación de un PLC dentro de un proyecto abierto de Twincat (izquierda). En la imagen de en medio se comprueba que la carpeta del proyecto ahora contiene otra carpeta con los datos del PLC añadido. En la imagen de la derecha se realiza la comprobación de que el PLC está en el proyecto de Visual Studio.

Si se intenta crear un PLC que ya está creado el *bot* debe indicar que no es posible esa operación.

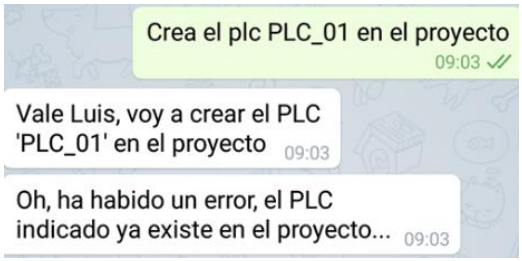


Figura 6.43.- Se comprueba que no se puede crear dos veces el mismo PLC en el proyecto.

6.1.21.- Mostrar el listado de PLCs de un proyecto

También se puede acceder al listado de PLCs creados en un proyecto.

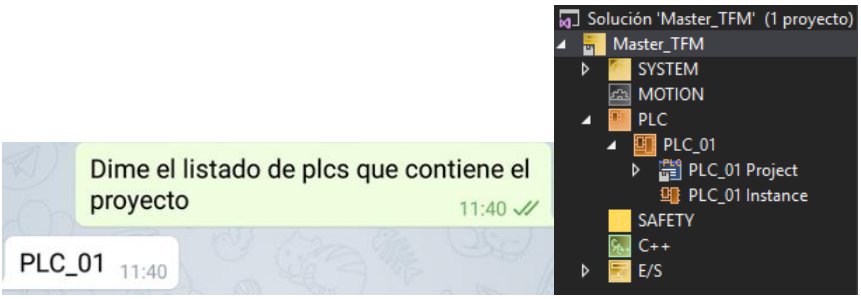


Figura 6.44.- Obtención del listado de PLCs de un proyecto de Twincat.

6.1.22.- Crear/borrar POUs de un PLC dentro de un proyecto de Twincat

El método de creación de POUs se realiza en varios pasos para que resulte intuitivo para el usuario.

Inicialmente se muestra al usuario un teclado donde debe pulsar sobre el tipo de POU que quiere crear. Después se le indica que debe elegir el tipo de lenguaje. En la prueba realizada se creó un POU de tipo programa y en lenguaje ST.

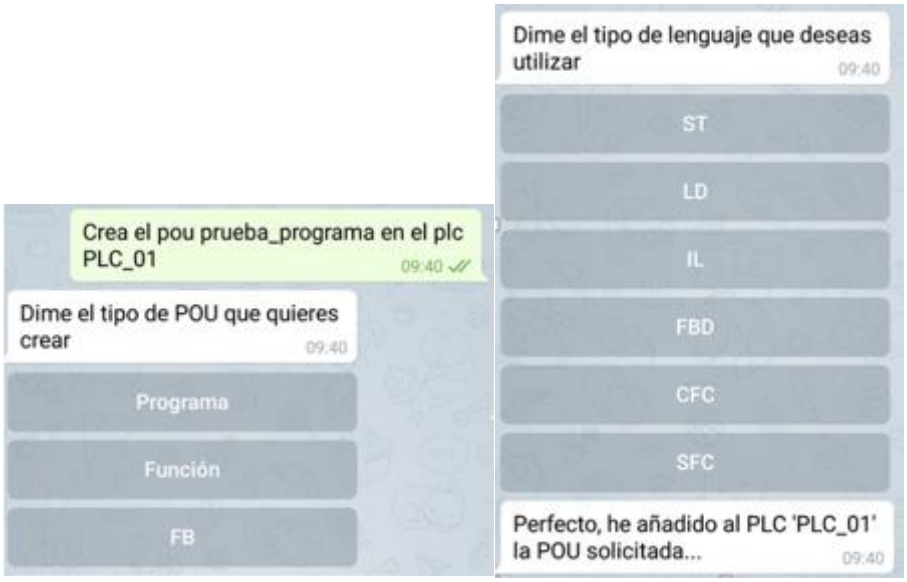


Figura 6.44.- Creación de un nuevo POU de tipo programa.

Posteriormente se comprueba que el POU creado ha sido añadido al PLC.

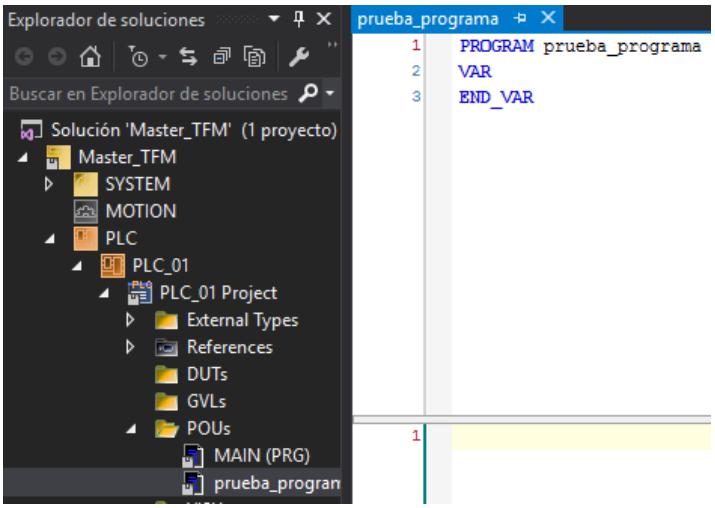


Figura 6.45.- Comprobación de la correcta creación del POU "prueba_programa".

La creación de POUs de tipo función es un poco distinta, en este caso el *bot* debe proporcionar un tercer teclado para elegir el tipo de dato devuelto. Para las pruebas realizadas se ha programado que los tipos posibles sean INT, BOOL o REAL.

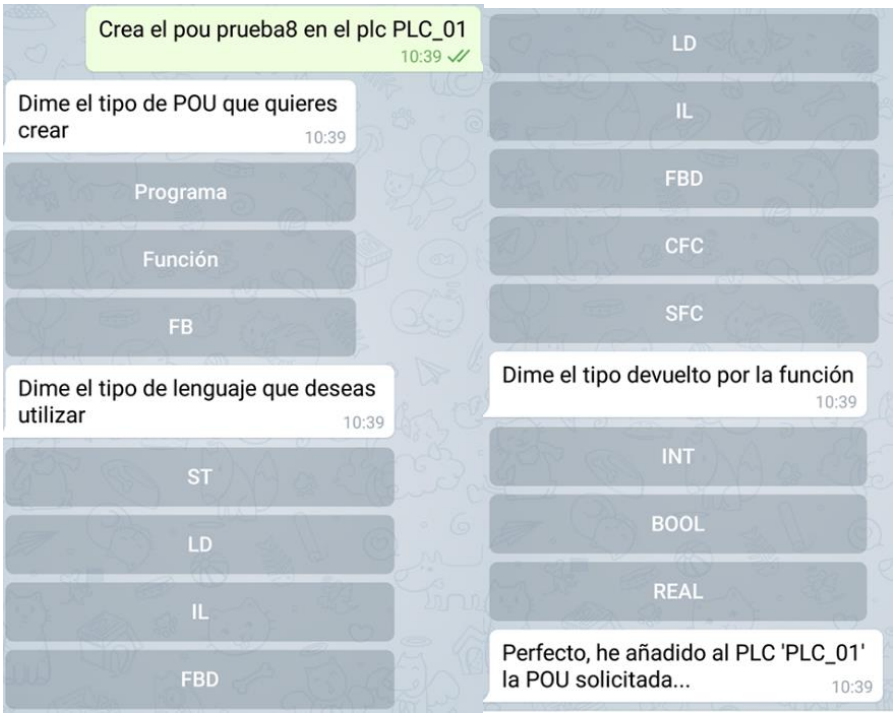


Figura 6.46.- Prueba realizada para comprobar que el bot es capaz de generar un nuevo POU de tipo función.

Como en anteriores ocasiones, se ha de comprobar que el *bot* puede detectar los POUs creados y si se le pide generar un nuevo POU con el mismo nombre que otro que ya existe debería avisar al usuario.

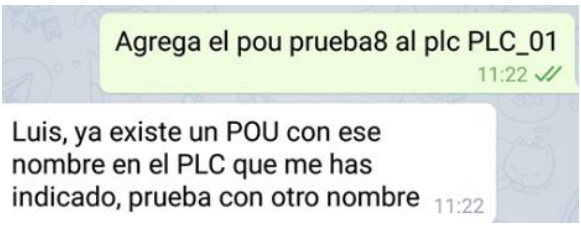


Figura 6.47.- Prueba para comprobar que no se puede crear un pou que ya existe.

El caso de borrar POUs funciona de manera similar al de su creación. En la figura 6.48 se realiza una prueba con el pou “prueba8”.

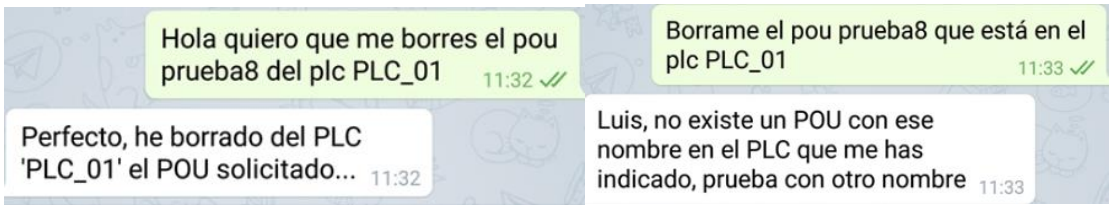


Figura 6.48.- En la imagen de la izquierda se observa como el bot es capaz de borrar el pou solicitado. En la imagen de la derecha, se comprueba que si se pide borrar un pou que no existe, el bot avisa al usuario.

6.1.23.- Obtención del listado de POUs de un PLC de un proyecto de Twincat

Se comprueba que el *bot* es capaz de acceder al listado de POUs que contiene un proyecto abierto de Twincat.

Figura 6.49.- En las imágenes de la izquierda y central se obtiene el listado de POUs del PLC solicitado. En la imagen de la derecha se comprueba que efectivamente los POUs devueltos existen en el PLC.

6.1.24.- Envío de proyectos

Una característica muy interesante que se ha añadido es la posibilidad de enviar por correo un determinado proyecto, utilizando el espacio de nombres “System.Net.Mail”.

Se ha realizado una prueba con el proyecto “Master_TFM” para comprobar el correcto funcionamiento de este método.

En primer lugar se debe tener en cuenta que el correo desde el que se enviarán los proyectos es el mío propio. Al solicitar el envío de un correo es necesario introducir mi

contraseña, por lo que he borrado esa parte de la foto para que no se vea. Además se genera un archivo tszip cada vez que se va a enviar un proyecto, que es el que se adjunta a este.

Figura 6.50.- Solicitud de envío del proyecto TFM_Master. En la imagen de la derecha se observa que se ha generado un zip del proyecto.

Por último, se comprueba que el correo ha sido enviado correctamente. Además se abre el correo para ver si el cuerpo del mensaje y el asunto son los que se habían programado.

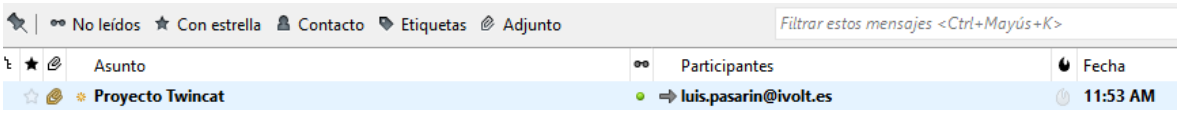


Figura 6.51.- El correo ha llegado correctamente.

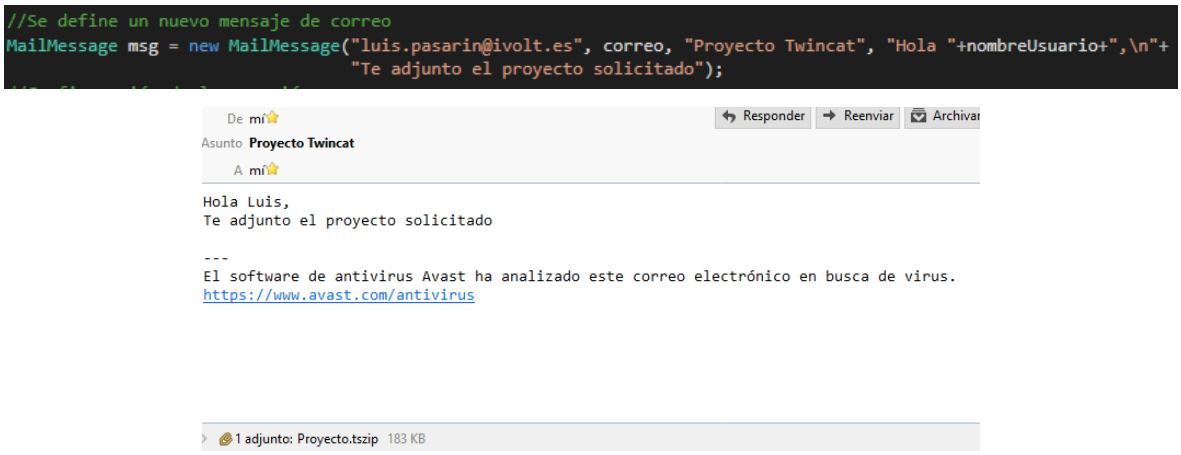


Figura 6.52.- Se comprueba que el asunto y el cuerpo programados en c# se corresponden con lo que ha llegado al correo indicado. Se observa que el proyecto ha sido enviado correctamente como datos adjuntos.

6.2.- RESULTADOS CON NOTAS DE VOZ

Las pruebas con el reconocimiento de voz se han realizado teniendo en cuenta ciertos aspectos de la API de Windows utilizada. No va a ser capaz de reconocer "puntos" ni "arrobas" ni siglas, por lo que la capacidad de ejecutar comandos complejos es limitada. Se probarán métodos que tengan solo letra como la obtención del listado de variables, arrancar y parar el PLC, etc.

Es importante tener en cuenta a la hora de ver las figuras siguientes que obviamente no es posible escuchar las notas de voz, pero se puede observar que en cuanto se recibe una nota de voz el *bot* responde.

Por ejemplo se ha probado a obtener el listado de archivos, cuyo resultado se puede observar en la figura 6.53.

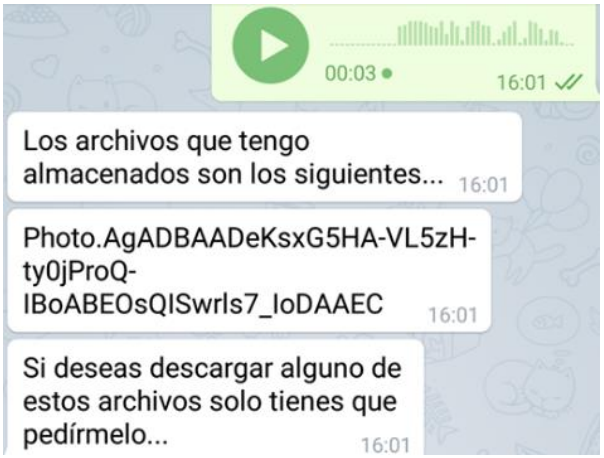


Figura 6.53.- Comprobación de que se puede obtener el listado de archivos con un comando de voz.

Figura 6.54.- Obtención del listado de variables con una nota de voz.

Figura 6.55.- Arranque y parada del PLC por voz.

Se comprueba de esta manera que el *bot* es capaz de descargar correctamente la nota de voz y utilizar FFmpeg para convertirla al formato wav. Del mismo modo, se confirma que el motor de reconocimiento realiza su trabajo, ya que el audio es transcrito a una cadena de texto que se pasa al procesador del lenguaje.

También funciona correctamente para la obtención del listado de proyectos, PLCs o POUs de un proyecto.

6.3.- RESULTADOS CON EL ENVÍO DE ARCHIVOS

A través del *bot* se pueden enviar fotos, documentos y vídeos. Se realizan en este sub-apartado las pruebas pertinentes para comprobar el correcto funcionamiento de los envíos. Se van a mandar tres archivos por el *bot* y posteriormente se va a comprobar pidiendo el listado de archivos que estos han sigo agregados a la base de datos.

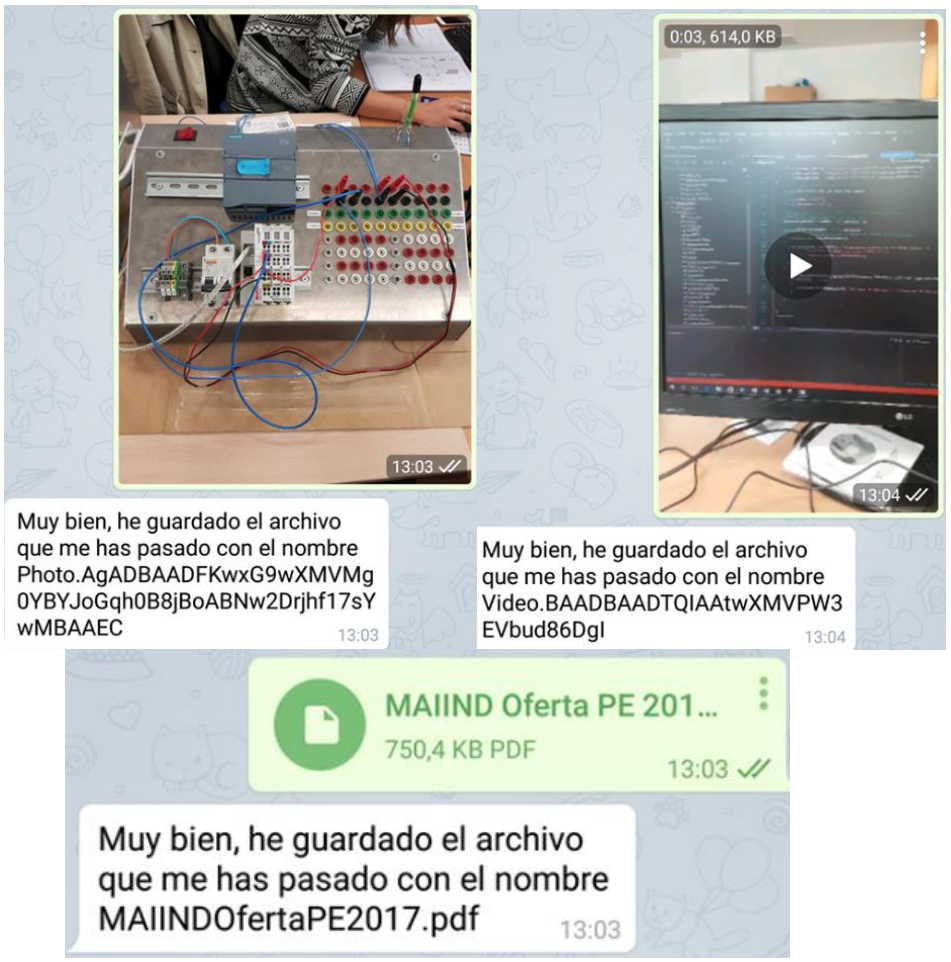


Figura 6.56.- Envío de un documento, una foto y un vídeo a través del bot.

Figura 6.57.- Obtención del listado de archivos de la base de datos.

Si ahora se intenta descargar uno de los archivos enviados, el *bot* debería devolverlo al usuario correctamente.

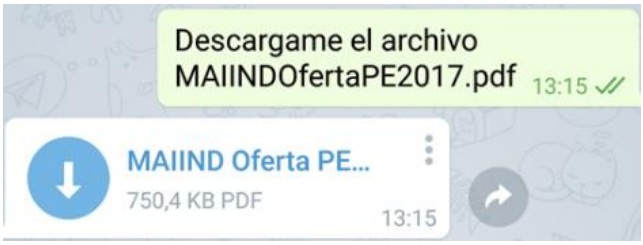


Figura 6.58.- Descarga de archivos enviados a través del bot.

7. Discusión de resultados

El objetivo inicial de este proyecto, que era la creación de un *bot* de Telegram capaz de reconocer comandos, utilizando un lenguaje más o menos flexible, para realizar ciertas operaciones sobre un PLC de Beckhoff, se ha cumplido. Además se han añadido otras características como el reconocimiento de voz, creación remota de proyectos de Twincat y envío de proyectos por correo electrónico. Por lo tanto se puede decir que el proyecto ha sido realizado con éxito, pero aún así hay cuestiones que todavía se pueden plantear, como: ¿Qué resultados se pueden mejorar? ¿Hay algún fallo reseñable?

En cuanto a la parte de reconocimiento de texto se ha podido comprobar que los resultados son bastante eficaces. Demostrando que la librería para el procesamiento del lenguaje cumple muy bien con su cometido. Del mismo modo, el método de reconocer palabras clave aporta una gran flexibilidad y da la opción al usuario de introducir frases más o menos coloquiales en el chat. No obstante hay que tener en cuenta que cada comando tendrá unas palabras clave necesarias para que sea reconocido correctamente, por lo que es posible que haya un pequeño período de adaptación al principio, por parte del usuario.

La parte de creación de proyectos de Twincat de forma remota proporciona una ayuda eficaz para preparar el esqueleto de un programa cuando no se está en el lugar de trabajo, o para modificar, añadir o eliminar, PLCs o POUs de un proyecto remotamente.

Un fallo que se ha encontrado en esta parte es que si se realizan demasiado rápido las operaciones de abrir el proyecto y crear PLCs o POUs, Visual Studio puede dar un error al usuario. Por ello, es recomendable esperar un tiempo prudencial entre operación y operación.

Una funcionalidad añadida que puede resultar bastante útil es la de enviar un proyecto por correo electrónico, de esta forma si se necesita mandar un trabajo a un compañero no hace falta estar con el ordenador, es posible que el *bot* se lo pueda reenviar.

El envío de otro tipo de archivos, como fotos, vídeos y documentos también puede resultar útil en ciertas situaciones. Por ejemplo que se haya hecho una foto a una parte de una instalación y que se quiera reenviar luego a un compañero o usarla en una memoria pero que sin darse cuenta se haya borrado del móvil. Utilizando el *bot* se ahorraría este problema, ya que en cualquier momento se podría volver a descargar cualquier foto o archivo.

No obstante hay que tener en cuenta que de momento los archivos se almacenan en los servidores de Telegram y que en la base de datos creada simplemente se guarda su referencia para poder descargarlos.

En cuanto al reconocimiento de voz, la api de Windows aporta unos resultados lejos de los esperados. Es capaz de reconocer comandos y órdenes sencillas pero no permite utilizar siglas ni símbolos como arrobas o puntos. Incluso muchas veces es necesario repetir el mensaje dos veces, debido a ruidos e interferencias durante el dictado. En general, estos resultados podrían mejorar notablemente con la incorporación de las APIs de Google o Bing, que son mucho más robustas y completas.

8. Conclusiones

Durante el desarrollo de este proyecto se explicó en qué consisten los *bots* de Telegram y las posibilidades que ofrecen la API de Beckhoff y la de Telegram. A raíz del estudio de las características de estas dos APIs se creó una base sobre la que comenzar a trabajar en el desarrollo del programa.

Como se pudo comprobar en el apartado 5, se han incorporado una gran cantidad de posibilidades al *bot*. Si ya el desarrollo y testeo del código llevó su tiempo, es importante destacar que adquirir los conocimientos sobre qué funciones utilizar de la api de Twincat no fue una tarea sencilla, ya que no resulta intuitiva ni sencilla de entender. Aún así, Beckhoff es una marca en constante crecimiento y poco a poco irán mejorando estos recursos, así como la documentación asociada.

Debido a la curva de aprendizaje en el uso de las APIs y en el propio conocimiento de c#, hubo una parte inicial en la que se realizaron pequeños programas y tutoriales, para familiarizarse con este lenguaje y con las APIs que se iban a utilizar.

Una de las cosas más importantes de este proyecto es que puede servir de base para otro tipo de trabajos, no hace falta que sea utilizando *bots* o no. Con los conocimientos adquiridos se podría realizar un programa de gestión de una instalación automatizada, en la que se usen PLCs de Beckhoff. Ya que se han adquirido buenos conocimientos sobre los proyectos de Twincat y la comunicación con los PLCs de esta marca.

También fue importante la elaboración de una librería para el reconocimiento de comandos en los mensajes recibidos. Aunque en un futuro se podría mejorar incorporando otros sistemas de aprendizaje automático, se han sentado las bases de un sistema que es perfectamente ampliable e independiente del código principal.

Dentro de esta parte de aprendizaje automático, se plantea la incorporación de un reconocimiento no literal de las palabras del diccionario, así como mejorar el *bot* para que pueda sugerir operaciones según lo que interprete que el usuario esté escribiendo.

Una de las tareas que se podrían mejorar en un futuro es la de crear una base de datos en la que además de guardar las referencias de los archivos enviados a través del *bot*, se guarden los propios archivos, a modo de repositorio online. De esta forma no habría una dependencia tan clara de los servidores de Telegram.

Otro punto que es importante mencionar es que muchos métodos se realizaron a modo de testeo en un entorno controlado, como es una oficina de trabajo. Por ejemplo, las pruebas con el PLC de la oficina habría que repetirlas en un entorno industrial, para comprobar su funcionamiento. Por otro lado, el *bot* debería estar subido en un servidor que estuviera disponible 24 horas al día. Actualmente el *bot* es una aplicación, un .exe, que debe ejecutarse para comenzar la interacción.

También habría que mejorar el método de envío por correo, para permitir enviar correos desde cualquier dirección. Ahora mismo el correo desde el que se envían los mensajes es el mío propio, fijado por código, por lo que no es posible modificarlo sin recompilar el programa. Se hizo así para no complicar la interacción con el usuario y que este no tenga que escribir un mensaje excesivamente largo, indicando dos correos (el de donde se envía y en el que se recibe) y una contraseña. No obstante es una mejora que podría llevarse a cabo sin una gran complicación.

En lo referente al envío de correos también se plantea como una mejora para el futuro la posibilidad de ocultar la contraseña del correo, creando algún tipo de interacción en la que cuando el usuario escriba no se visualicen las letras o números que esté introduciendo, imitando la forma en la que se introducen las contraseñas en servicios como Gmail, Hotmail, el correo de la universidad y otros tantos.

Por último, tal y como se ha descrito en este apartado y en el de resultados, los objetivos planteados se han cumplido y además se tienen opciones para las futuras ampliaciones, en caso de llevarlas a cabo. Además debido a que el proyecto se realizó como tarea de I+D de una empresa, los conocimientos adquiridos han servido para ayudar más en las tareas de programación de la empresa y para mejorar notablemente los adquiridos en el máster en cuanto a programación orientada a objetos y de programación de PLCs.

9. Planificación

Para la planificación del proyecto se ha partido de la base de un total desconocimiento de las APIs de Telegram y Beckhoff, calculando la carga de trabajo necesaria que una persona tendría que asumir para poder llevar a cabo este trabajo. El esquema general de la planificación se puede observar en el apartado 9.1.

La primera etapa del proyecto consistió en conocer más a fondo el sector de la Industria 4.0 y plantear como se iba a desarrollar el programa.

La segunda tarea a realizar en este proyecto fue la de realizar un estudio teórico práctico sobre c# y las APIs de Telegram y Beckhoff, realizando sencillos programas de ejemplo. No obstante es importante mencionar que la fase de estudiar las APIs de Telegram y Beckhoff fue algo que se siguió haciendo hasta el final.

La tercera etapa está compuesta por la familiarización con Twincat 3 y realización de programas sencillos para comunicarse con un PLC disponible en la oficina.

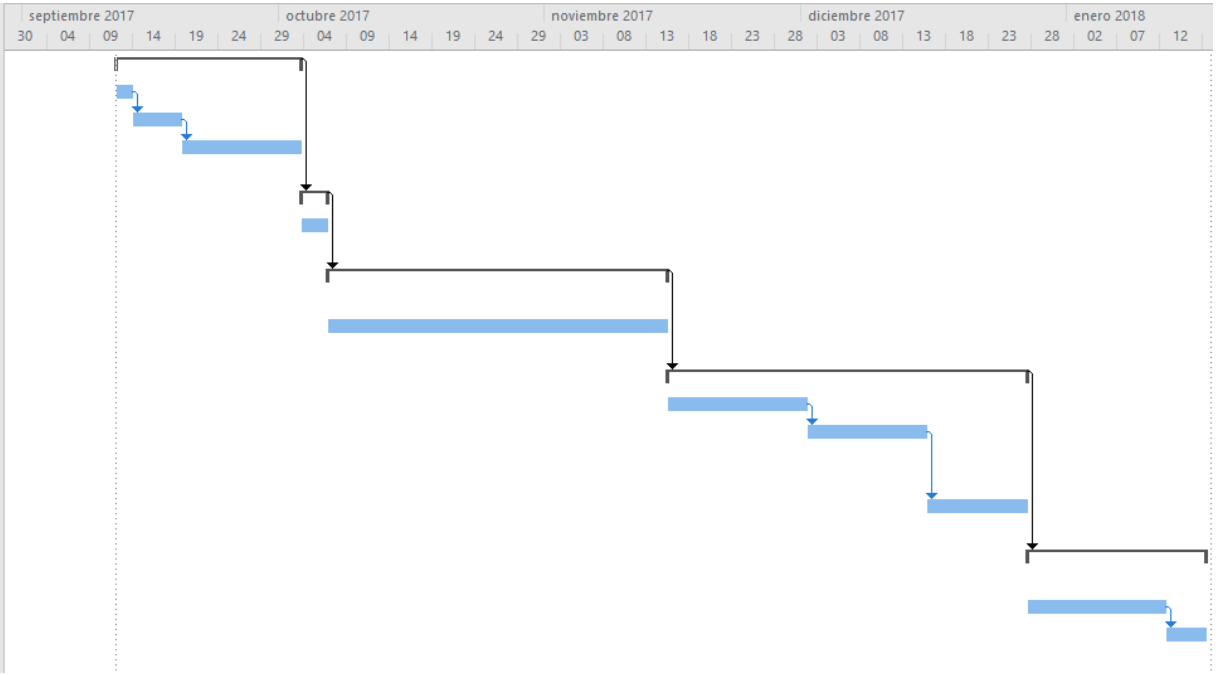
En una cuarta etapa, a la que se ha asignado el nombre “Desarrollo del Software Principal”, se llevó a cabo la realización de todo el programa, repasando constantemente las posibilidades de la API de comunicación de Beckhoff.

Una quinta etapa, llamada “Ampliaciones”, fue en la que se sugirieron y se desarrollaron cosas como el uso de comandos de voz, creación remota de proyectos y envío por correo de archivos, así como la de creación de una base de datos para la gestión de los archivos. También hubo una fase en la que se realizaron las pruebas pertinentes para probar todo el código.

Por último, se ha considerado una etapa final en la que se desarrolló toda la memoria del trabajo.

9.1.- GRÁFICO DE LA PLANIFICACIÓN

Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predec
	Investigación Inicial	**15,75 días**	**mar 12/09/17**	**mar 03/10/17**	
	Industria 4.0	16 horas	mar 12/09/17	mié 13/09/17	
	Familiarización con C#	30 horas	jue 14/09/17	mar 19/09/17	2
	Estudio de las Apis de Beckhoff y Telegram	80 horas	mar 19/09/17	mar 03/10/17	3
	Twincat 3	**3 días**	**mar 03/10/17**	**vie 06/10/17**	**1**
	Pruebas y ejemplos sencillos	24 horas	mar 03/10/17	vie 06/10/17	
	Desarrollo del Software Principal	**27,5 días**	**vie 06/10/17**	**mié 15/11/17**	**5**
	Programación del proyecto base	220 horas	vie 06/10/17	mié 15/11/17	
	Ampliaciones	**30 días**	**mié 15/11/17**	**mié 27/12/17**	**7**
	Reconocimiento de Voz	100 horas	mié 15/11/17	vie 01/12/17	
	Integración de la parte de creación remota de proyectos	80 horas	vie 01/12/17	vie 15/12/17	10
	Mejoras generales del código	60 horas	vie 15/12/17	mié 27/12/17	11
	Pruebas y desarrollo de la memoria	**15 días**	**mié 27/12/17**	**mié 17/01/18**	**9**
	Escritura de la memoria	100 horas	mié 27/12/17	vie 12/01/18	
	Pruebas y toma de datos para la memoria	20 horas	vie 12/01/18	mié 17/01/18	14



10. Presupuesto

Concepto	Cantidad	Precio unitario	Total
Investigación			
Horas de estudio teórico	126 Horas	12 €/hora	1.512 €
Horas de programación	484 Horas	25 €/hora	12.100 €
Horas de pruebas	100 Horas	18 €/hora	1.800 €
Hardware			
Ordenador portátil	1 ud	1000 € (25% anual)	250 €
Panel PC Beckhoff	1 ud	800 €	800 €
Software			
Visual Studio 2015 Community	1 ud	0 €	Gratuito
Twincat 3	1 ud	1400 € (16,66% anual)	233,24 €
Coste			
Presupuesto de ejecución			16.695,24 €
Gastos Generales		8 %	1335,61 €
Beneficio empresarial		6 %	1001,71 €
IVA		21 %	3.996,83 €
Presupuesto total			23.029,39 €

En Gijón, a 3 de Febrero de 2018

11. Anexo

11.1.- CONFIGURACIÓN DE TWINCAT

A lo largo del desarrollo del proyecto se ha utilizado constantemente Twincat 3, para realizar las pruebas sobre el PLC de Beckhoff del que se disponía en la oficina.

Como ya se ha comentado en otros apartados, Twincat 3 está integrado en Visual Studio. La forma de crear proyectos es igual a la de cualquier otro tipo de proyecto típico en este entorno.

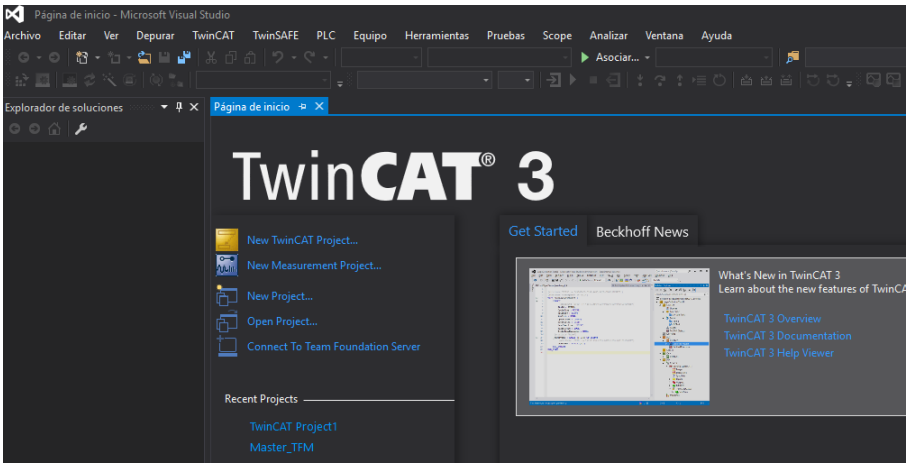


Figura 11.1.- Integración de Twincat 3 con Visual Studio.

En este apartado se explicarán los pasos que se han realizado para la conexión con el PLC de pruebas.

Figura 11.2.- Fotografía tomada al PLC de Beckhoff.

Lo primero que se hizo fue comprobar la Ip del PLC y hacerle un ping para asegurarse de que tanto la Ip del pc de desarrollo como la del PLC estaban en el mismo rango.

```
C:\Users\luis.muñiz>ping 192.168.0.69

Haciendo ping a 192.168.0.69 con 32 bytes de datos:
Respuesta desde 192.168.0.69: bytes=32 tiempo=40ms TTL=128
Respuesta desde 192.168.0.69: bytes=32 tiempo=18ms TTL=128
Respuesta desde 192.168.0.69: bytes=32 tiempo=51ms TTL=128
Respuesta desde 192.168.0.69: bytes=32 tiempo=20ms TTL=128

Estadísticas de ping para 192.168.0.69:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 18ms, Máximo = 51ms, Media = 32ms
```

Figura 11.3.- Comprobación de que las dos IP están en el mismo rango.

Lo siguiente que se hizo fue comprobar la dirección Ams que tenía asignada el PLC. En la figura 11.4 se puede observar la ams asignada.

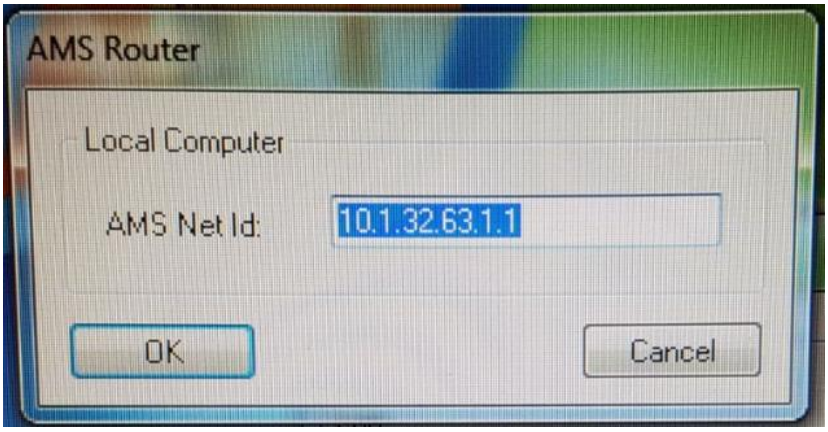


Figura 11.4.- Dirección Ams del PLC.

Con un proyecto de Twincat 3 creado, lo primero que hay que hacer para conectarse a un PLC es buscarlo en la sección de rutas. Una vez encontrado se hace doble clic y Twincat 3 establecerá la conexión.

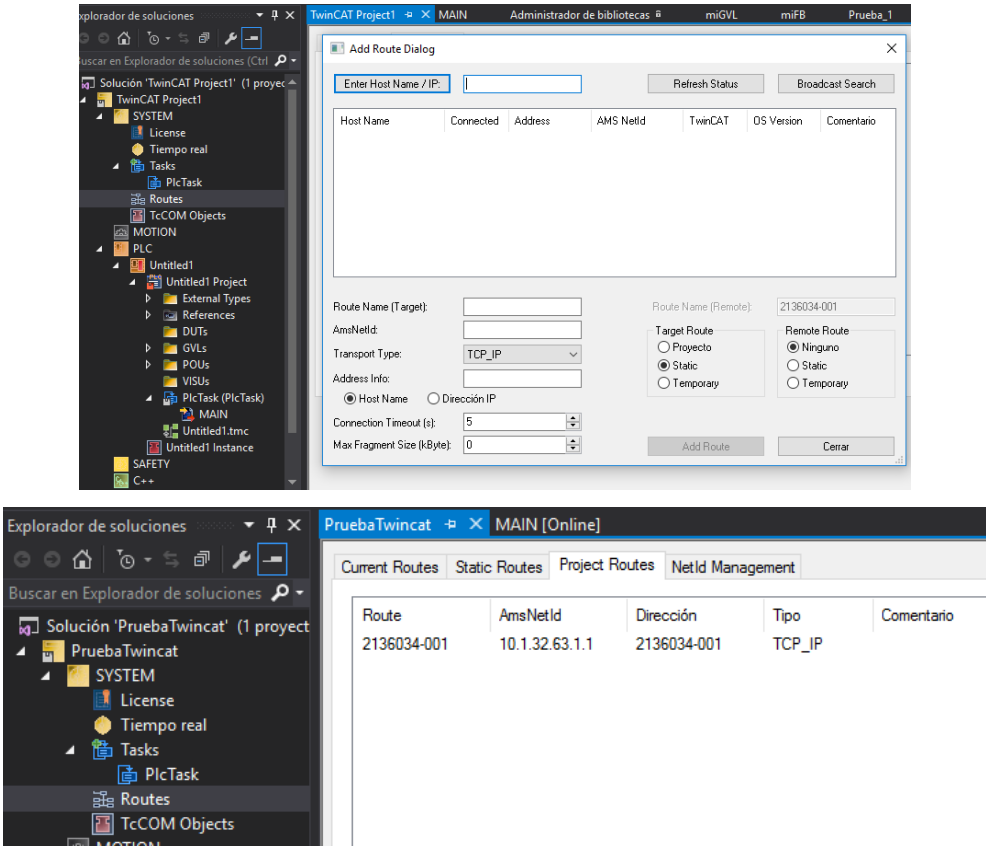


Figura 11.5.- Búsqueda de PLCs en la sección "Routes" de Twincat.

Si se pudo establecer la conexión se mostrará un icono verde para poder realizar el enlace final del programa. En la figura 11.6 se muestra lo que debería salir si todo fue bien.

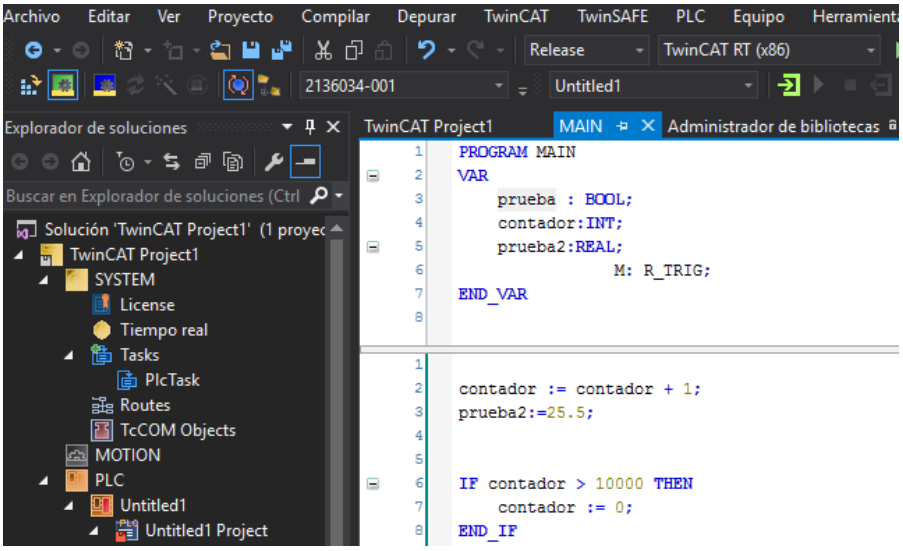


Figura 11.6.- Conexión establecida con el PLC seleccionado.

Pulsando sobre el icono que está en verde en la parte derecha de la figura 11.6 se establece la conexión con el PLC y ya se podrá dar a "play" para cargar el programa.

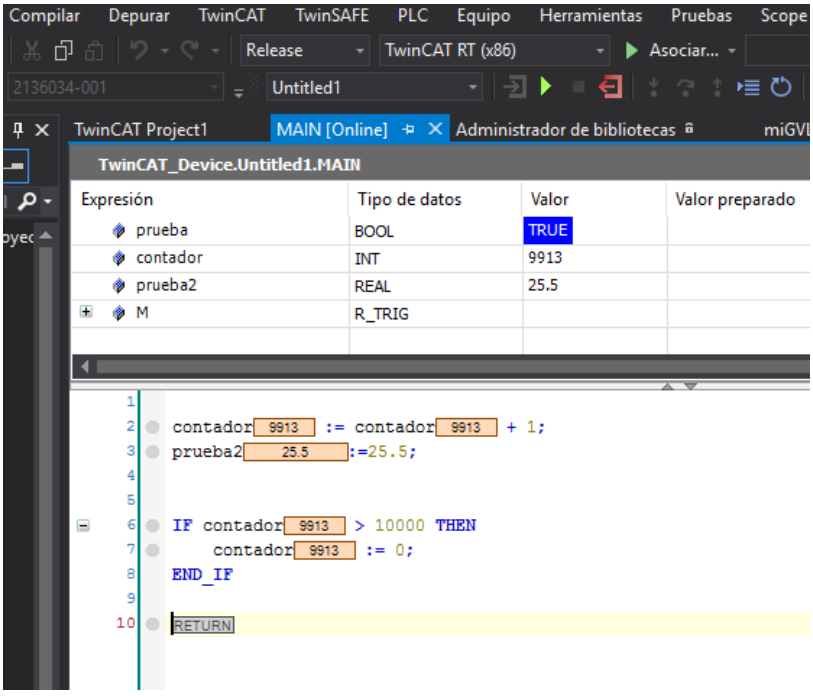


Figura 11.7.- Conexión establecida con el PLC y a la espera de dar al play.

Una vez que el programa está en modo Run se puede parar pulsando sobre el cuadrado rojo o desconectarse del PLC pulsando sobre el icono rojo a la derecha de este.

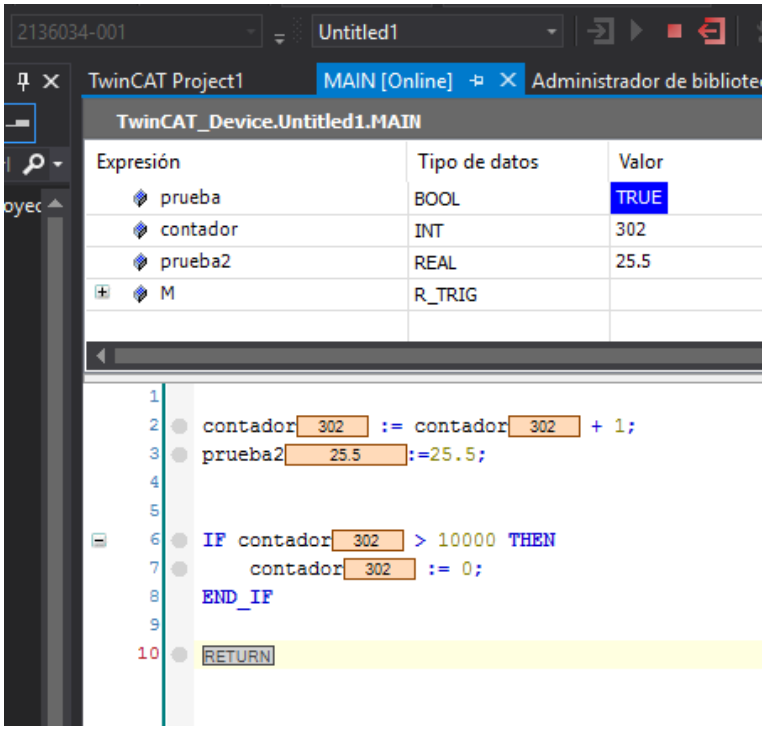


Figura 11.8.- Programa en modo Run. Se puede parar pulsando sobre el cuadrado rojo situado arriba a la derecha. También se puede "romper" la conexión con el PLC pulsando sobre el icono rojo situado a la derecha del cuadrado anterior.

12. Bibliografía

[1] «Industria 4.0», *IDEPA, Industria 4.0*. Disponible en: http://asturiasindustria40.es/.
[2] «Qué es industria 4.0 o cuarta revolución industrial», *Papeles de Inteligencia Competitiva*, 11-may-2016. .
[3] Grupo Franja, «DE LA INDUSTRIA 1.0 A LA 4.0». Disponible en: http://www.grupofranja.com/index.php/oftalmica/item/1763-de-la-industria-1-0-a-la-4-0.
[4] C. Scott, «Robot's Versatility is Thanks to 3D Printed Grippers from Materialise», *3DPrint.com | The Voice of 3D Printing / Additive Manufacturing*, 16-nov-2015. .
[5] «Inglobe Technologies | Smart manufacturing using AR in the era of Industry 4.0». Disponible en: http://www.inglobetechnologies.com/smart-manufacturing-ar-industry-4-0/.
[6] Vivienne Walt, «Meet the Reclusive "Mark Zuckerberg of Russia"», *Fortune*. .
[7] «Así es Vkontakte, el alocado Facebook ruso que ofrece películas y series gratis», *ELMUNDO*, 13-jun-2017. Disponible en: http://www.elmundo.es/f5/comparte/2017/06/13/593e76e146163fbf318b4621.html.
[8] «Características de Telegram que destacan de otras aplicaciones», *unocero*, 15-ago-2017. Disponible en: https://www.unocero.com/noticias/apps/caracteristicas-telegram-destacan-otras-aplicaciones/.
[9] «BECKHOFF New Automation Technology». Disponible en: https://www.beckhoff.es/es/default.htm?beckhoff/default.htm.
[10] «Twincat ADS Documentation». Disponible en: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_adscommon/html/tcadscommon_introads.htm&id=6479552931026062113.
[11] «Beckhoff Information System». Disponible en: https://infosys.beckhoff.com/english.php?content=../content/1033/tcinfosys3/html/startpage.htm&id=4495506442591881122.
[12] «Bots: An introduction for developers». Disponible en: https://core.telegram.org/bots.
[13] «FFmpeg». Disponible en: https://www.ffmpeg.org/.
[14] «Get Started with Speech Recognition». Disponible en: https://msdn.microsoft.com/en-us/library/office/hh361683(v=office.14).aspx.
[15] «Accord.NET Machine Learning Framework». Disponible en: http://accord-framework.net/.
[16] «API Speech: reconocimiento de voz», *Google Cloud Platform*. Disponible en: https://cloud.google.com/speech/?hl=es
[17] «Bing Speech API: Reconocimiento de voz | Microsoft Azure». Disponible en: https://azure.microsoft.com/es-es/services/cognitive-services/speech/.
[18] «About SQLite». Disponible en: https://sqlite.org/about.html.
[19] *Telegram.Bot: .NET Client for Telegram Bot API*. Telegram Bots, 2018.