

Proyecto Fin de Grado

Ingeniería de Telecomunicación

Firmware para soporte de un chip wifi de terminal de teleasistencia

Autor: Francisco Javier Cruz Sánchez

Tutor: Fernando Cárdenas Fernández

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023



Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Firmware para soporte de un chip wifi de terminal de teleasistencia

Autor:

Francisco Javier Cruz Sánchez

Tutor:

Fernando Cárdenas Fernández

Profesor titular

Dpto. de ingeniería telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Grado: Firmware para soporte de un chip wifi de terminal de teleasistencia

Autor: Francisco Javier Cruz Sánchez

Tutor: Fernando Cárdenas Fernández

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

Resumen

En este proyecto, a partir de un módulo WiFi (ESP-8266) y otro módulo GSM (MC60) se ha implementado un cliente SIP que permita al usuario el intercambio de mensajes con otros usuarios. Para ello se ha establecido una comunicación serie con ambos módulos, mediante la cuál se han configurado y establecido conexión con el servidor SIP.

Además se han configurado dos servidores SIP: kamilio, utilizado para el desarrollo y las pruebas y Asterisk, que ha permitido crear un escenario más realista. Ambos servidores han sido instalados y configurados. Se han creado los usuarios necesarios para las pruebas. En el caso de Asterisk se ha creado un contexto específico que permita el procesamiento de los mensajes.

La implementación del cliente SIP creado permite el registro del usuario, la autenticación HTTP (si es requerida) y el envío periódico de peticiones de registro para evitar la caducidad de la sesión. También se encarga de crear las peticiones de envío de mensajes instantáneos, así como el procesamiento de cualquier respuesta o petición proveniente del servidor, generando de forma automática la respuesta si es necesario y posible.

Todas estas funcionalidades del cliente se han puesto a disposición del usuario en una intuitiva interfaz gráfica.

Índice

Resumen	7
Índice	9
1 Introducción	11
2 Fundamentos teóricos	15
2.1. <i>Módulo ESP-8266</i>	15
2.1.1 Comandos	16
2.1.2 Configuración de uso	20
2.2. <i>Módulo GSM MC60</i>	21
2.1.3 Comandos AT	22
2.1.4 Configuración de uso	26
2.3. <i>Protocolo SIP</i>	27
2.3.1 Registro en servidor SIP.	27
2.3.2 Mensajería instantánea en SIP	28
3 Metodología	11
3.1 <i>Diseño experimental</i>	11
3.1.1 Casos de uso	15
3.2 <i>Configuración del entorno de pruebas</i>	17
Primer bloque de pruebas	17
Segundo bloque de pruebas	20
3.3 <i>Implementación del protocolo SIP</i>	25
3.3.1 ClienteSIP	26
3.3.2 Interfaz ComSerie	29
3.3.3 Librería serial	30
3.3.4 ComSerieESP	31
3.3.5 ComSerieGSM	33
3.3.6 Interfaz gráfica y estructura del proyecto	35
4 Resultados y análisis	39
4.1 <i>Implementación del protocolo SIP en el módulo Wi-Fi. Bloque I de pruebas</i>	39
Requisitos previos.	39
Comprobación de la conexión serie con ESP-8266	40
4.1 <i>Implementación del protocolo SIP en ambos módulos. Bloque II de pruebas.</i>	46
Implementación del cliente SIP en el módulo ESP8266	46
Implementación del cliente SIP en el módulo MC60	48
Pruebas con sistema de interfaz gráfica	51
5 Conclusión	61

1 INTRODUCCIÓN

La tecnología de comunicación ha experimentado un crecimiento exponencial en las últimas décadas, permitiendo una conexión instantánea entre personas en diferentes partes del mundo. En este contexto, el protocolo SIP (Session Initiation Protocol) se ha convertido en una pieza fundamental para la transmisión de voz y video a través de Internet.

El presente trabajo se centra en la implementación de un cliente SIP en dos módulos ampliamente utilizados en el ámbito de la comunicación: el ESP-8266 y el MC60. Estos módulos, diseñados por los fabricantes Espressif Systems y Quectel respectivamente, ofrecen capacidades de conectividad y comunicación que permiten la transmisión de mensajes SIP entre usuarios.

El objetivo principal de este proyecto es desarrollar un cliente SIP funcional y eficiente, capaz de establecer sesiones de comunicación entre usuarios utilizando los módulos ESP-8266 y MC60. Esto implica la implementación de las funcionalidades básicas de un cliente SIP, como el registro en un servidor SIP y la transmisión de mensajes de texto.

El módulo ESP-8266, conocido por su conectividad Wi-Fi, se utilizará como plataforma para el desarrollo del cliente SIP. Este módulo ofrece un amplio conjunto de características que facilitará la implementación de las funcionalidades requeridas.

Por otro lado, el módulo MC60, que combina las capacidades GSM y GPRS, permitirá la comunicación de mensajes SIP a través de la red celular. Esto brindará una mayor flexibilidad en términos de conectividad, ya que no se limitará a entornos con acceso Wi-Fi, sino que podrá aprovechar la infraestructura de comunicación móvil existente.

En este proyecto, se utilizará el servidor SIP Asterisk como infraestructura central para el enrutamiento de las sesiones SIP. Asterisk es una plataforma de código abierto que ofrece una amplia gama de funcionalidades de telefonía IP y comunicaciones unificadas. Su flexibilidad y escalabilidad lo convierten en una elección ideal para implementar servicios de comunicación basados en SIP.

Se configurará y personalizará el servidor Asterisk para adaptarlo a los requisitos específicos del cliente SIP implementado en los módulos ESP-8266 y MC60. Esto implicará la definición de extensiones, configuración de troncales y la implementación de reglas de enrutamiento adecuadas para establecer los mensajes SIP entre los usuarios.

Además del servidor Asterisk, se utilizará también Kamailio para la realización de un bloque de pruebas. Este servidor de fácil configuración permitirá crear un escenario donde módulo y servidor se encuentre en la misma red local, teniendo un mayor control sobre el traspaso de mensajes SIP.

Se desarrollará una interfaz gráfica intuitiva que permita a los usuarios interactuar con el cliente SIP de manera fácil y eficiente. Esta interfaz gráfica proporcionará un entorno visual para realizar la conexión con el dispositivo, la configuración de este y la visualización de mensajes SIP.

2 FUNDAMENTOS TEÓRICOS

En este capítulo se abordan los fundamentos teóricos relacionados con el protocolo SIP y los módulos Wi-Fi y GSM. Estos conocimientos son esenciales para comprender el contexto en el que se llevará a cabo la implementación del protocolo en dichos módulos.

2.1. Módulo ESP-8266

El módulo WiFi ESP8266, desarrollado por Espressif, es una solución compacta y de bajo costo diseñada específicamente para proyectos de IoT¹ (Internet de las cosas). Este chip proporciona conectividad inalámbrica a dispositivos electrónicos, permitiéndoles comunicarse y transferir datos a través de redes WiFi. Gracias a su diseño eficiente y versátil, el módulo ESP8266 se ha convertido en una opción popular entre los desarrolladores de proyectos de IoT, ofreciendo una amplia gama de funcionalidades y una fácil integración con otros componentes. Su capacidad para conectarse a Internet y comunicarse con otros dispositivos lo hace ideal para aplicaciones de monitoreo remoto, automatización del hogar, control de dispositivos, entre otros. Con su tamaño compacto y bajo consumo de energía, el módulo WiFi ESP8266 ofrece una solución conveniente y rentable para proyectos que requieren conectividad inalámbrica y acceso a Internet.

Las principales características técnicas² del módulo son las siguientes:

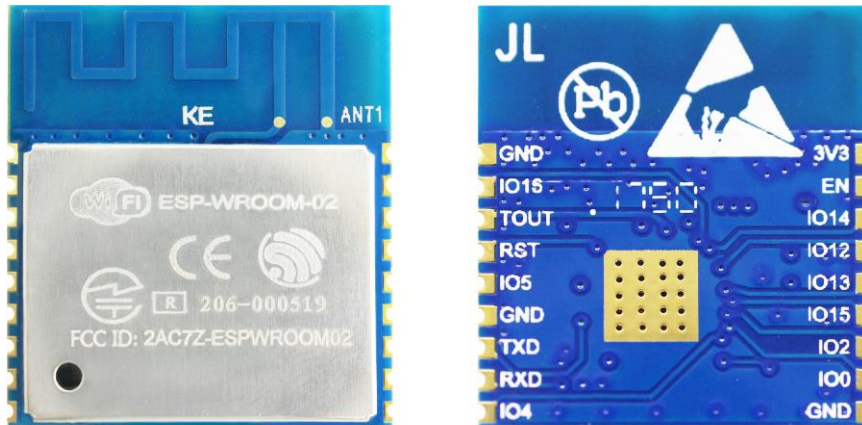
- Capacidades completas de conexión Wi-fi. Este dispositivo está preparado tanto para conectarse a un punto de acceso disponible como para crear una red Wifi y actuar él mismo como punto de acceso. Los protocolos que implementa son: 802.11 b/g/n por lo que su rango de frecuencia estará entre los 2.4GHz y los 2.5GHz. Tiene una potencia máxima de transmisión de 20 dBm.
- A nivel hardware, está compuesto por un procesador de 32 bits. Posee entradas y salidas de tipo UART, GPIO, PWM y I2C, entre otros. También contiene un ADC con 10 bits de precisión. En este proyecto únicamente se usará la comunicación UART. El módulo debe alimentarse entre 2.5 y 3.6 V. Además posee un modo de bajo consumo.
- Implementa la pila de protocolo UDP-TCP/IP. Como protocolo de nivel de aplicación implementa HTTP. En lo relativo a la seguridad de la conexión WiFi permite el uso de WPA o WPA2. Para la encriptación implementa WEP, TKIP o AES.
- En lo relativo para la comunicación con el módulo pueden usarse comandos AT a través de una conexión serie (método que se usará en este proyecto). Además permite la conexión a través de un servidor Cloud.

¹ <https://www.espressif.com/en/products/socs/esp8266>

² ESP8266 Hardware Design Guidelines. Version 2.7

https://www.espressif.com/sites/default/files/documentation/esp8266_hardware_design_guidelines_en.pdf

Para el desarrollo del proyecto, se utilizará la placa de pruebas ESP-WROOM-02, que incorpora el chip ESP8266. Esta placa ofrece ventajas adicionales, como una antena WiFi con una ganancia de 20 dBi, lo que mejora la recepción y transmisión de señales inalámbricas. Además, el módulo ESP-WROOM-02 está configurado para utilizar el oscilador interno del ESP8266, que funciona a una frecuencia de 26 MHz³. Esta configuración garantiza un rendimiento óptimo y una mayor estabilidad en las comunicaciones WiFi. Con el uso de la placa ESP-WROOM-02, se aprovechan las características mejoradas de este módulo específico, lo que contribuye a un desarrollo más eficiente y fiable del proyecto.



El módulo ESP8266 se controla mediante una comunicación serie, a través de la cual podemos enviar comandos de tipo AT para controlar todas sus funciones. Estos comandos AT nos permiten consultar el estado del módulo y configurar diversos parámetros según nuestras necesidades. Al utilizar una interfaz de comunicación serie, podemos interactuar con el módulo de forma sencilla y controlar sus funcionalidades de manera eficiente. Los comandos AT nos brindan la flexibilidad necesaria para obtener información del módulo, realizar configuraciones específicas y gestionar sus operaciones. Esta comunicación serie basada en comandos AT es una forma conveniente y ampliamente utilizada para interactuar con el módulo ESP8266 y aprovechar todas sus capacidades en nuestros proyectos.

2.1.1 Comandos

Los comandos AT⁴ mantienen una estructura consistente, donde se utiliza el prefijo "AT+" seguido de un comando específico para enviar instrucciones al módulo. Existen diferentes formas de ejecutar un comando AT:

- Envío directo del comando sin necesidad de incluir parámetros: Se envía el comando completo al módulo utilizando la comunicación serie. Por ejemplo, "AT+RST" para reiniciar el módulo.
- Para enviar comandos AT que requieren parámetros, se seguirá la siguiente sintaxis: AT+<x>=valor_param1,valor_param2,valor_param3. Los parámetros deben especificarse en el orden correcto, de acuerdo con las especificaciones del comando en cuestión.
- Si un parámetro es una cadena de caracteres, deberá encerrarse entre comillas. Por ejemplo, si se desea enviar un valor de texto "ejemplo" como parámetro, se codificaría de la siguiente forma: AT+<x>="ejemplo".
- En el caso de los parámetros numéricos, se especificarán directamente sin comillas. Por ejemplo, si se desea enviar un valor numérico 123 como parámetro, se codificaría de la siguiente manera: AT+<x>=123.
- Para obtener el valor actual de los parámetros relacionados con un comando, se puede enviar el

³ ESP-WROOM-02 Datasheet. Version 3.2 https://www.espressif.com/sites/default/files/documentation/0c-esp-wroom-02_datasheet_en.pdf

⁴ Estos comandos son normalizados en la recomendación UIT-T V.250

comando seguido de un signo de interrogación. Por ejemplo, si se desea obtener el valor actual de un parámetro utilizando el comando `AT+<x>`, se enviaría `AT+<x>?` al módulo. Es importante destacar que no todos los comandos admiten esta funcionalidad de consulta de parámetros. Algunos comandos pueden no tener parámetros asociados o no permitir la consulta de valores actuales. En esos casos, el módulo responderá con una respuesta indicando la incapacidad de proporcionar la información solicitada.

Después de enviar un comando, el módulo responderá con un mensaje que indica el resultado de la operación. Las respuestas pueden ser "OK" para indicar éxito, "ERROR" en caso de error, u otras respuestas específicas según el comando ejecutado.

La sintaxis y el uso de los comandos AT se normalizan y estandarizan en los documentos de referencia proporcionados por el fabricante del módulo. Estos documentos suelen incluir un manual o una guía de comandos AT que especifica la sintaxis correcta, los parámetros admitidos y las respuestas esperadas.

En el caso del módulo ESP8266, el fabricante Espressif Systems proporciona la documentación oficial que describe los comandos AT admitidos. El documento más relevante es el "ESP8266 AT Instruction Set" que detalla los comandos AT y su uso. De él se han seleccionado los comandos de utilidad para la realización de este proyecto:

AT: Comando de prueba que responde "OK" en el caso de estar el módulo funcionando correctamente.

AT+RST: Comando que permite reiniciar el módulo.

El comando **ATE** se utiliza para activar o desactivar el modo eco en el módulo ESP8266. El modo eco determina si el módulo retransmite o no los comandos que recibe antes de enviar una respuesta. Hay dos posibles valores para el comando ATE:

- **ATE0:** Este comando desactiva el modo eco.
- **ATE1:** Este comando activa el modo eco.

Después de enviar el comando ATE con el valor correspondiente, el módulo responderá con "OK" si el cambio se ha realizado correctamente. En caso de que haya algún error, el módulo responderá con "ERROR".

Los comandos disponibles para la configuración de la conexión Wi-Fi son:

AT+CWMODE_CUR y **AT+CWMODE_DEF:** Permiten establecer el funcionamiento del módulo como cliente WIFI, punto de acceso o mixto. El primer comando realiza el cambio pero no guarda dicha información en memoria flash. La segunda opción permite mantener la configuración aunque se apague el dispositivo. Se le pasa un único argumento que puede tomar los siguientes valores:

- 1: En el caso de configurarse como una estación wifi
- 2: Para la configuración como punto de acceso.
- 3: Se configura tanto como punto de acceso como estación.

Por ejemplo, si se desea configurar el módulo como una estación WiFi y que dicha configuración sea almacenada en memoria flash, el comando a enviar es: `AT+CWMODE_DEF=1`.

Mediante **AT+CWJAP_DEF** y **AT+CWJAP_CUR** se configura la conexión Wi-Fi guardándose, o no, en memoria flash respectivamente. El módulo debe estar configurado previamente en el modo estación. Los parámetros necesarios para la ejecución de este comando son:

- SSID de la red Wifi
- Contraseña de la red wifi
- BSSID del punto de acceso, opcional.

En el caso de producirse la conexión de forma exitosa devolverá 'OK'. En el caso de producirse algún error se informará con los siguientes códigos:

- 1: El punto de acceso no responde tras al espera del correspondiente timeout
- 2: Contraseña incorrecta
- 3: No se puede encontrar el punto de acceso correspondiente
- 4: Conexión fallida.

Por ejemplo, para configurar la conexión a una red llamada "red_wifi" con contraseña "1234" se deberá utilizar el siguiente comando: **AT+CWJAP_CUR="red_wifi","1234"**.

El comando **AT+CWLAP** se utiliza para obtener información sobre los puntos de acceso WiFi disponibles desde el módulo ESP8266. Cuando se envía el comando sin ningún parámetro, el módulo devuelve una lista de los puntos de acceso disponibles.

Una vez haya configurado la conexión Wifi, se puede configurar las conexiones con otros equipos de la red mediante los siguientes comandos:

AT+CIPSTATUS permite conocer el estado actual del módulo. Una vez ejecutado responde con **STATUS:n**, donde n puede tomar los siguientes valores:

- 2: El módulo está conectado a una red Wifi y ha podido obtener una dirección IP.
- 3: El módulo tiene actualmente una conexión TCP o UDP activa
- 4: La transmisión TCP/UDP del módulo está desactivada.
- 5: El módulo no está conectado a una red wifi.

En el caso de existir una o varias conexiones TCP/UDP se recibirá: **+CIPSTATUS:<link ID>,<type>,<remote IP>,<remote port>,<local port>,<tetype>** donde:

- Link ID: identifica a la conexión, puede haber hasta 4 conexiones simultáneas.
- Type informa del protocolo de transporte usado para dicha conexión
- remoteIP: IP del equipo al que se está conectado
- remotePort: puerto remoto
- Local port: número del puerto local que se usa para dicha conexión.
- Tetype: toma el valor 0 si el módulo actúa como cliente y 1 si actúa como servidor.

AT+CIPMUX: Activa o desactiva el uso de varias conexiones. Recibe un único parámetro que puede tomar dos valores:

- 0: Activa el modo de conexión única
- 1: Activa el modo de conexiones múltiples.

Como ejemplo, si se desea establecer múltiples conexiones se deberá realizar la siguiente configuración: AT+CIPMUX=1

AT+CIPMODE: permite seleccionar el modo de transmisión de datos. El módulo nos permite usar dos modos diferentes:

- Modo de transmisión normal. Al recibir un paquete IP el módulo lo retransmite por la conexión serie añadiendo el prefijo “+IPD” para indicar que los datos provienen del servidor. Para transmitir información al servidor hay que utilizar el comando AT+CIPSEND.
- Modo de transmisión continua. Este modo permite no tener que ejecutar ningún comando para el envío de información al servidor remoto. Una vez activo este modo, todo lo que se envíe a través de la conexión serie será retransmitido al servidor. De igual forma los paquetes IP recibidos se mostrarán por la conexión serie directamente, sin existencia de prefijos. Para salir de este modo de transmisión, el usuario deberá enviar la cadena ‘+++’ por la conexión serie. Este modo de transmisión únicamente está disponible si existe una única conexión.

Para ejecutar este comando habrá que incluir un único parámetro que tomará los valores:

- 0: Si se desea usar el modo de transmisión normal.
- 1: Si se desea configurar el modo de transmisión continua.

De forma predeterminada, el módulo está configurado para el modo de transmisión normal. El cambio de configuración mediante este comando no será guardado en memoria. Al encender el módulo siempre estará activo el modo de transmisión normal.

Por ejemplo, si se desea que el funcionamiento del módulo sea en modo de transmisión continua deberá enviarse el comando: AT+CIPMODE=1. Esto únicamente establece el modo de funcionamiento. Para activar dicho modo de funcionamiento hay que ejecutar el comando AT+CIPSEND una vez se haya establecido la conexión con el servidor.

AT+CIPSTART: permite establecer conexión con un servidor. Para ello hay que enviar los siguientes parámetros:

- Link ID: en el caso de permitir varias conexiones (AT+CIPMUX=1) se debe indicar el número con el que identificaremos a la conexión que estamos creando.
- Tipo: Deberá tomar los valores “TCP”, “UDP” o “SSL”
- IP del servidor al que queremos conectarnos.
- Puerto del servidor al que queremos conectarnos.

Dependiendo de si se ha realizado la conexión de forma correcta el módulo devolverá los siguientes mensajes:

- “ERROR”: No se ha podido realizar la conexión.
- “ALREADY CONNECT”: Existe otra conexión establecida con los mismos parámetros.

- “OK”: La conexión se ha realizado de forma correcta.

Para conectarnos al puerto 80 del servidor google.es, habría que ejecutar: AT+CIPSTART=1,"TCP","google.es",80. El primer parámetro se ha incluido porque suponemos que se va a querer establecer más de una conexión.

El comando **AT+CIPSEND** se utiliza para enviar información a través de las conexiones establecidas previamente. Dependiendo de si hay una única conexión o múltiples conexiones, los parámetros requeridos pueden variar:

- En el caso de una única conexión:
 - Tamaño: El tamaño de los datos que se desean transmitir, en bytes.
- En el caso de múltiples conexiones:
 - Identificador de la conexión: El número de identificación de la conexión a utilizar.
 - Tamaño: El tamaño de los datos que se desean transmitir, en bytes.
 - IP destino (opcional): La dirección IP del destino para las conexiones UDP.
 - Puerto destino (opcional): El número de puerto del destino para las conexiones UDP.

Después de enviar el comando, el módulo responderá con el símbolo '>' y esperará a que se indique la información que se va a transmitir. En ese momento, se puede enviar la información deseada.

Si el envío de la información se realiza correctamente, el módulo responderá con "SEND OK". En caso de que haya algún problema en el envío, se mostrará "SEND FAIL".

En el caso de tener activado el modo de transmisión transparente, simplemente se debe enviar el comando AT+CIPSEND sin ningún parámetro adicional para activar el modo. A partir de ese momento, cualquier dato que se reciba a través del puerto serie será transmitido a través de la conexión configurada previamente. Para salir del modo de transmisión continua, se debe enviar la cadena '+++'.
 Por ejemplo, si se desea enviar un paquete de datos con un tamaño de 100 bytes a una conexión ya establecida con identificador 1, se ejecutará: AT+CIPSEND=1,100

AT+CIPCLOSE: Su función es cerrar las conexiones abiertas. En el caso de tener varias conexiones abiertas se puede añadir un parámetro indicando el identificador de aquella que deseemos cerrar. La respuesta será 'OK' una vez se haya cerrado la conexión.

2.1.2 Configuración de uso

Para configurar el módulo con los comandos mencionados, se seguirán los siguientes pasos:

1. Desactivar el modo "echo" de los comandos ejecutados:
 - Enviar el comando ATE0 al módulo.
2. Configurar el módulo como estación WiFi:
 - Enviar el comando AT+CWMODE=1 al módulo.
3. Configurar la conexión WiFi utilizando el comando AT+CWJAP_CUR:

- Enviar el comando AT+CWJAP_CUR="<SSID>","<password>" al módulo.
 - Reemplazar <SSID> con el nombre de la red WiFi a la que se desea conectar.
 - Reemplazar <password> con la contraseña de la red WiFi.
4. Evitar que el módulo se conecte automáticamente a la red al reiniciarse:
- No es necesario enviar un comando adicional, ya que AT+CWJAP_CUR asegura que la configuración no se guarde automáticamente.
5. Activar la transmisión continua una vez conectado al servidor remoto:
- Enviar el comando AT+CIPMODE=1 al módulo.

Con estos comandos, el módulo se configurará adecuadamente para el envío y recepción de datos. Estos pasos se realizarán en la primera parte de la aplicación, asegurando que el módulo esté listo para establecer la conexión y transmitir datos.

2.2. Módulo GSM MC60

El módulo GSM/GPRS diseñado por Quectel proporciona conectividad a través de redes GSM y GPRS. Está equipado con una torre de protocolo IP/TCP que permite establecer conexiones de red utilizando los protocolos UDP,FTP, PPP, FTP y HTTP⁵. Esto facilita la transmisión de datos a través de la red.

Además de la transmisión de datos, el módulo también ofrece funcionalidades adicionales. Por ejemplo, permite el envío y recepción de mensajes SMS, lo que lo hace útil para aplicaciones de mensajería. También es capaz de realizar y recibir llamadas telefónicas con capacidad de transmisión de voz, lo que amplía su utilidad en aplicaciones de comunicación.

El módulo destaca por su tamaño compacto y bajo consumo de energía⁶, lo que lo hace ideal para su integración en dispositivos pequeños y portátiles. Esta combinación de características lo convierte en una solución versátil y conveniente para proyectos que requieran conectividad GSM/GPRS y capacidades de transmisión de datos, SMS y voz.

El módulo GSM/GPRS cuenta con diversas conexiones que permiten su integración en diferentes sistemas y aplicaciones. Estas conexiones incluyen⁷:

1. Tarjeta SIM: El módulo tiene la capacidad de aceptar hasta dos tarjetas SIM, lo que permite la conexión a diferentes operadores de red.
2. Puertos serie: Dispone de dos puertos serie. Uno de ellos se utiliza para la comunicación principal, mientras que el otro se destina al control y depuración del funcionamiento del módulo (debug port).
3. Entradas/salidas de audio analógico: El módulo cuenta con dos salidas de audio y una entrada de audio analógico. Estas conexiones permiten la reproducción y grabación de audio en aplicaciones específicas.
4. Conexión Bluetooth 3.0: El módulo está equipado con una conexión Bluetooth 3.0, lo que permite la comunicación inalámbrica con otros dispositivos compatibles.
5. Slots para antenas: Dispone de slots para la conexión de hasta tres antenas. Estos slots están diseñados para las antenas GSM (para la conectividad celular), GNSS (para la recepción de señales de posicionamiento global) y Bluetooth (para la comunicación inalámbrica Bluetooth).

Estas diversas conexiones proporcionan flexibilidad en la integración del módulo en diferentes sistemas y aplicaciones, permitiendo una amplia gama de funcionalidades y comunicaciones.

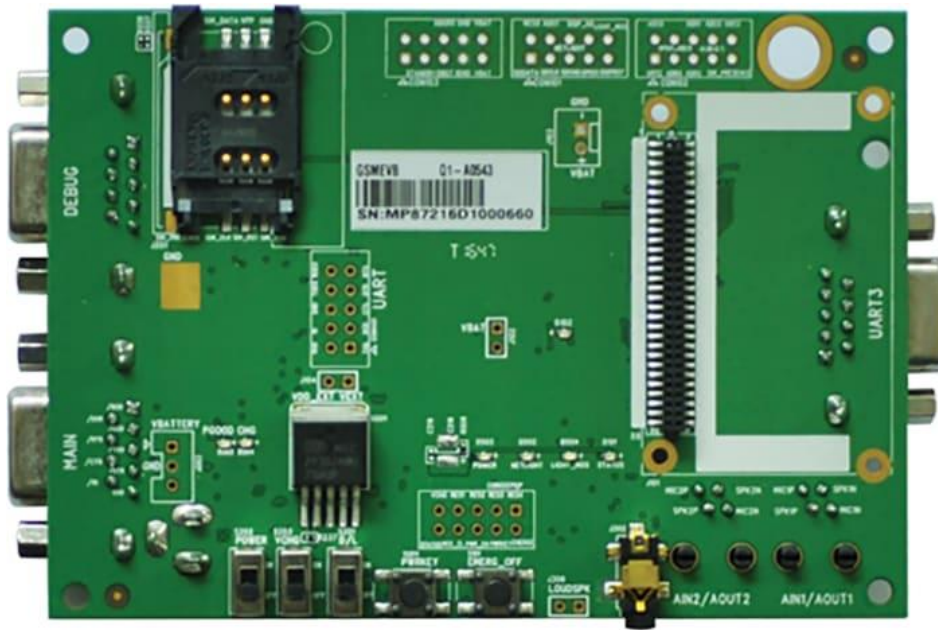
En el proyecto, se ha utilizado la placa de pruebas GSM EVB de Quectel, específicamente diseñada para el módulo MC60. Esta placa proporciona una interfaz conveniente y completa para aprovechar todas las

⁵ Quectel MC60 Specification V1.0 Apartado: Specifications for Data Function

⁶ <https://www.quectel.com/product/gsm-gprs-gnss-mc60>

⁷ Quectel MC60 Specification V1.0. Apartado: Interfaces

funcionalidades del módulo GSM⁸.



Algunos de los componentes y características incluidos en la placa son:

1. Puertos serie: La placa de pruebas GSM EVB cuenta con puertos serie que permiten la comunicación entre el módulo GSM y otros dispositivos o sistemas.
2. Circuito de alimentación: La placa de pruebas incluye un circuito de alimentación que suministra la energía necesaria para el funcionamiento del módulo y otros componentes de la placa.
3. Conectores de auriculares/micrófonos: La placa está equipada con conectores que permiten la conexión de auriculares y micrófonos, lo que facilita la utilización de las funciones de audio del módulo GSM, como la reproducción y grabación de sonido.
4. Slot para tarjeta SIM: La placa de pruebas tiene un slot destinado a la instalación de la tarjeta SIM, permitiendo así establecer la conexión a la red celular.
5. Antena: La placa de pruebas incluye una antena integrada, que asegura una buena recepción y transmisión de señales en la comunicación GSM.

Estos componentes y características de la placa de pruebas GSM EVB proporcionan una solución completa y lista para usar el módulo MC60 en el proyecto, facilitando su integración y puesta en marcha.

El módulo GSM MC60 se controla mediante comandos AT a través de una comunicación serie. Estos comandos AT permiten controlar y configurar todas las funciones del módulo, así como consultar su estado.

Al establecer una comunicación serie con el módulo, enviamos los comandos AT mediante una secuencia de caracteres específica. Estos comandos pueden ser utilizados para diversas acciones, como consultar la señal de red, enviar mensajes SMS, realizar llamadas telefónicas, configurar parámetros de red, entre otros.

2.1.3 Comandos AT

En el caso del módulo GSM Quectel MC60, el fabricante Quectel proporciona la documentación oficial que describe los comandos AT admitidos. El documento relevante es el "Quectel MC60 AT Commands Manual" que detalla los comandos AT y su uso. A partir de este manual, se han seleccionado los comandos más relevantes y útiles para la realización de este proyecto, permitiendo controlar y configurar el módulo de

⁸ GSM EVB User Guide V3.3

manera adecuada para cumplir con los requisitos específicos de comunicación y conectividad GSM.

El comando **AT+QEAUART** se utiliza para configurar las comunicaciones serie del módulo GSM Quectel MC60. Este comando recibe dos parámetros. El primer parámetro puede tomar los siguientes valores:

- "0": Desactiva la doble comunicación serie.
- "1": Activa la comunicación serie.

El segundo parámetro permite definir el puerto utilizado como puerto de debug. Puede tomar los siguientes valores:

- "2": El puerto UART 2 se utiliza como puerto de debug.
- "99": No hay puerto de debug configurado.

Mediante este comando, es posible activar o desactivar la comunicación serie y especificar el puerto de debug según las necesidades del proyecto. El puerto de debug se utiliza para realizar la depuración y el seguimiento de eventos y mensajes del módulo.

Por ejemplo, si se desea configurar la doble comunicación serie utilizando el puerto UART 2 como debug hay que ejecutar el comando **AT+QEAUART=1,2**.

El comando **AT+CMEE** se utiliza para establecer el formato de las respuestas y errores de los comandos ejecutados. Este comando se ejecuta con un único parámetro, el cual puede tomar los siguientes valores:

- "0": Desactiva la respuesta a los comandos.
- "1": Activa las respuestas en formato numérico.
- "2": Activa las respuestas en formato de texto.

Como ejemplo, las respuestas en formato texto se activan ejecutando: **AT+CMEE=2**

El comando **AT+QICSGP** se utiliza para establecer la configuración de la conexión TCP/IP en el módulo GSM. La sintaxis del comando es:

AT+QICSGP=<contextID>,<APN>,<username>,<password>

- El primer parámetro, <contextID>, especifica el ID del contexto de la conexión. En este caso, se utilizará el valor "1" para establecer una conexión GPRS.
- El segundo parámetro, <APN>, se refiere al Punto de Acceso (Access Point Name) al que se conectará el módulo. Este valor depende de la configuración proporcionada por el proveedor de servicios de telefonía móvil.
- El tercer parámetro, <username>, es el nombre de usuario utilizado para autenticarse en la red GPRS.
- El cuarto parámetro, <password>, es la contraseña asociada al nombre de usuario para la autenticación.

Es importante tener en cuenta que, dependiendo del proveedor de servicios de telefonía móvil utilizado, los últimos dos parámetros pueden no ser necesarios y deberán ser omitidos. Cada proveedor de servicios puede tener requisitos de configuración diferentes, por lo que es necesario consultar la documentación o contactar con el proveedor para obtener los valores correctos de APN, nombre de usuario y contraseña.

Una vez ejecutado este comando, el módulo queda listo para llevar a cabo la configuración

necesaria para activar la conexión TCP/IP.

Un ejemplo de este comando es: AT+QICSGP=1,"airtelwap.es", donde el módulo se conecta a la APN airtelwap.es sin necesidad de autenticación.

El comando **AT+QIREGAPP** se utiliza para activar la conexión TCP/IP en el módulo GSM. La sintaxis del comando es: AT+QIREGAPP=<apn>,<username>,<password>

Este comando permite establecer las credenciales de acceso a la red, incluyendo el nombre de punto de acceso (APN), nombre de usuario y contraseña. Si estos parámetros ya se han especificado previamente mediante el comando AT+QICSGP, es posible ejecutar únicamente el comando AT+QIREGAPP para activar la conexión TCP/IP utilizando las configuraciones previamente establecidas.

Es importante asegurarse de proporcionar los valores correctos de APN, nombre de usuario y contraseña según las especificaciones del proveedor de servicios de telefonía móvil utilizado.

Siguiendo el ejemplo del comando anterior, en este caso habría que ejecutar AT+QIREGAPP="airtelwap.es", o simplemente, AT+QIREGAPP, ya que la APN se definió en el comando anterior.

El comando **AT+QIACT** se utiliza para activar el conector GPRS en el módulo GSM. La sintaxis del comando es simplemente "AT+QIACT".

El comando **AT+QILOCIP** se utiliza para obtener la dirección IP local asignada al módulo GSM. Al ejecutar este comando, el módulo responderá con la dirección IP local actual.

El comando **AT+QIHEAD** se utiliza para controlar la visualización del tamaño de los datos recibidos al recibir un paquete IP.

La sintaxis del comando es: "AT+QIHEAD=<mode>", donde <mode> es el parámetro que define el modo de funcionamiento y puede tomar los siguientes valores:

- "0": En este modo, al recibir el paquete, se mostrarán directamente los datos por la conexión serie, sin indicar el tamaño.
- "1": En este modo, antes de mostrar los datos recibidos, se enviará una cadena de formato: "IPD(data length):", donde "data length" indica la cantidad de bytes que se han recibido.

Por ejemplo, si se desea mostrar el tamaño del paquete recibido del servidor hay que transmitir AT+QIHEAD=1

AT+QIDNSIP: Establece el modo en el que se indicará el servidor remoto al que se conectará. Su uso es: AT+QIDNSIP=<mod mode>, siendo mode:

- "0": Se indicará la dirección IP del servidor
- "1": Se utilizará un dominio para indicar el servidor.

Por ejemplo, para indicar el servidor mediante su IP hay que ejecutar: AT+QIDNSIP=0

El comando **AT+QIOPEN** se utiliza para establecer una conexión TCP o UDP con un servidor remoto.

La sintaxis del comando es:

"AT+QIOPEN=[<index>,<mode>,<IPaddress>/<domain name>,<port>", donde:

- "<index>": Es un parámetro opcional que indica el índice de conexión que se desea establecer. Este parámetro solo se debe especificar si está activo el modo de múltiples conexiones y se desea establecer una conexión en un índice específico. En caso contrario, se debe omitir.
- "<mode>": Es el tipo de conexión que se desea establecer y puede tomar los valores "TCP" o "UDP".
- "<IPaddress>/<domain name>": Es la dirección IP o el nombre de dominio del servidor remoto al que se desea conectarse. Esta información debe coincidir con la configuración realizada previamente mediante el comando anterior.
- "<port>": Es el número de puerto al que se desea realizar la conexión.

Al ejecutar el comando, se establecerá la conexión especificada. El módulo MC60 permite hasta 6 conexiones simultáneas si se activa el modo de múltiples conexiones.

La respuesta del módulo puede ser una de las siguientes:

- "OK": Indica que la estructura del comando es correcta.
- "ERROR": Indica que el comando es incorrecto.
- "ALREADY CONNECT": Indica que ya existe una conexión establecida, en el caso de que el modo de múltiples conexiones esté desactivado.
- "CONNECT OK": Indica que la conexión se ha establecido correctamente. Si existen múltiples conexiones, la respuesta del módulo incluirá el índice de la conexión establecida.
- "CONNECT FAIL": Indica que la conexión no se ha podido establecer. Si el modo de múltiples conexiones está activado, se incluirá el índice de la conexión que ha fallado.

El comando **AT+QIOPEN** es fundamental para establecer una conexión TCP o UDP y permitir la comunicación con un servidor remoto a través del módulo MC60.

Para aplicar este caso, supongamos que queremos conectarnos al servidor google.es en el puerto 80: **AT+QIOPEN=1,"TCP","google.es",80**

AT+QIPROMPT: Permite establecer el formato en el envío de datos a través de una conexión. Al ejecutar el comando de transmisión de datos el módulo muestra por la conexión serie el carácter ">" para indicar que está listo para la recepción de datos. Una vez se han transmitidos los datos correctamente devuelve "SEND OK". Ambos mensajes son configurables mediante este comando, que utiliza un único comando que puede tomar los siguientes valores:

- "0": No muestra ">", pero sí "SEND OK".
- "1": Muestra tanto ">" como "SEND OK"
- "2": No muestra ni ">" ni "SEND OK"
- "3": Muestra ambos mensajes, y además el índice de conexión junto con "SEND OK"

Si deseamos que, por ejemplo, se muestre tanto ">" como "SEND OK" hay que ejecutar: **AT+QIPROMPT=1**

AT+QISEND: Permite el envío de datos a través de las conexiones establecidas. Hay dos formas de uso:

1. Indicando en el comando el tamaño de los datos que se desea transmitir. Para ello debe invocarse con el siguiente esquema: AT+ QISEND=[<index>],< length>. Siendo "index" únicamente necesario cuando AT+QIMUX=1.
2. No se indica el tamaño de los datos. Para realizar el envío de los datos se debe transmitir control-Z al módulo. Si, por el contrario, se desea cancelar la ejecución del comando se envía ESC.

La respuesta del comando depende de la configuración del comando anterior. En cualquier caso, en caso de error porque la conexión no esté disponible se recibe "ERROR". Si la conexión está disponible pero se ha producido algún error en la transmisión se recibe "SEND FAIL".

Para transmitir 100 bytes de datos por la conexión 1, hay que ejecutar AT+QISEND=1,100

AT+QICLOSE: Permite cerrar las conexiones activas. En el caso de estar activo el modo de múltiples conexiones deberá indicarse el índice de la conexión que se desea cerrar.

El comando **AT+QIMUX** se utiliza para activar o desactivar el modo de múltiples conexiones TCP o UDP en el módulo MC60.

La sintaxis del comando es: "AT+QIMUX=<mode>", donde "mode" puede tomar los siguientes valores:

- "0": Este valor desactiva el modo de múltiples conexiones, lo que significa que solo se permite la existencia de una única conexión activa.
- "1": Este valor activa el modo de múltiples conexiones, lo que permite establecer hasta seis conexiones TCP o UDP simultáneas.

Al ejecutar el comando, se establecerá el modo de múltiples conexiones según el valor especificado. Si se activa el modo de múltiples conexiones, se podrán realizar hasta seis conexiones simultáneas con servidores remotos.

Si, por ejemplo, se desea activar las múltiples conexiones se ejecutaría: AT+QIMUX=1

2.1.4 Configuración de uso

En el proyecto en cuestión, se utilizará la siguiente configuración para el módulo MC60:

1. Se desactivará el "eco" de los comandos para que el módulo no retransmita los comandos ejecutados (ATE0).
2. Se desactivará el puerto serie de "debug" para simplificar la configuración (AT+QEAUART=1,99).
3. Se activarán las respuestas a los comandos en formato de texto para una mejor legibilidad (AT+CMEE=2).
4. Se utilizará la conexión GPRS con la siguiente APN de la compañía telefónica: "airtelwap.es" (AT+QICSGP=1,"airtelwap.es").
5. Una vez configurada la conexión móvil, se activarán los protocolos IP/TCP (AT+QIREGAPP).
6. Se activará la conexión GPRS utilizando el comando AT+QIACT.

7. Al recibir un paquete IP, se mostrará el tamaño de los datos recibidos (AT+QIHEAD=1).
8. Se utilizará un dominio para indicar el servidor remoto (AT+QIDNSIP=1).
9. No se mostrarán caracteres de control para enviar información al servidor remoto (AT+QIPROMPT=2).
10. El módulo se configurará para admitir una única conexión TCP o UDP (AT+QIMUX=0).

Estas configuraciones permitirán establecer una conexión GPRS y utilizar los protocolos IP/TCP para enviar y recibir datos a través de la red. Cabe mencionar que estas configuraciones son específicas para el proyecto en cuestión y pueden variar en otros escenarios de aplicación.

2.3. Protocolo SIP

El protocolo SIP fue normalizado por el IETF en la RFC 3261 en el año 2002, aunque posteriormente ha sufrido algunas modificaciones. Según esta RFC, se creó el protocolo SIP con el fin de crear un estándar de creación y control de sesiones de usuarios. Estas sesiones serían utilizadas para el intercambio de información, principalmente multimedia, entre los propios usuarios. Debido a ello se nombró SIP: Session Initiation Protocol.

Los mensajes SIP, tanto de petición como de respuesta, se componen de las siguientes partes⁹:

1. En primer lugar, se incluye una línea de comienzo cuyo contenido depende del tipo de mensaje:
 - a. Si es un mensaje de petición¹⁰ esta línea incluye: método, URI al que se realiza la petición y versión del protocolo SIP a usar.
 - b. Si es un mensaje de respuesta¹¹, se incluye en esta línea: Versión SIP, código de respuesta y una frase aclaratoria.
2. Tras la primera línea de comienzo se suceden los campos correspondientes al tipo de petición o respuesta. Cada campo irá en una línea diferente. El nombre del campo y el contenido deben ir separados por “:”.¹²
3. Se incluirá un retorno de carro y nueva línea para separar la sección de campos de la siguiente
4. Por último, se incluirá el cuerpo del mensaje, cuyo uso será necesario dependiendo del tipo de petición.

Este tipo de mensaje es tomado de la RFC 2616.

Debido al uso de mensajería instantánea, en este proyecto se utilizará SIP/2.0 como versión de SIP en la línea de comienzo de los mensajes.¹³

2.3.1 Registro en servidor SIP.

Para proceder al registro en el servidor SIP, el cliente envía una petición REGISTER al servidor. Esta petición debe contener los siguientes campos¹⁴:

⁹ RFC 3261. Apartado 7.

¹⁰ RFC 3261. Apartado 7.1

¹¹ Ibid. Apartado 7.2

¹² Ibid. Apartado 7.3.1

¹³ Ibid. Apartado 7.1

¹⁴ Ibid. Apartado 10.2

- **Request-uri:** URI del servidor al que se realiza la petición. Contendrá el dominio y el puerto destino.
- **To:** Nombre de usuario del que se realiza la petición del servidor. Debería seguirse del servidor al que se realiza la petición, por ejemplo usuario@dominioSIP.com
- **From:** Este campo se utiliza en el registro de una tercera persona. En caso contrario, este campo tendrá el mismo contenido que el campo anterior.
- **Call-ID:** Numero que identifica las transacciones de un mismo usuario.
- **CSeq:** Número que, en el caso del proceso de registro, ordena los mensajes involucrados en este. En la primera petición del proceso tomará el valor 1.
- **Contact:** Dirección IP del dispositivo desde el que se inicia sesión y puerto abierto para la recepción de peticiones INVITE o MESSAGE creadas por otros usuarios. Este campo no es obligatorio, pues dependiendo de la configuración del servidor utilizará la IP y puerto desde el que se envía la petición REGISTER.
- **Expires:** Tiempo (en segundos) que tardará en caducar la sesión del usuario. Este campo es opcional, ya que puede configurarse en el servidor.
- **Allow:** Tipo de peticiones que permite procesar el cliente SIP.

Una vez el servidor recibe y procesa la petición, envía la respuesta al cliente. El caso más probable es que, para iniciar sesión, el usuario necesite autenticarse. Para ello el servidor responderá con un mensaje: 401 Unauthorized¹⁵. Este mensaje contendrá un campo `www-Authenticated` que permitirá al usuario autenticarse mediante el método HTTP Digest¹⁶.

Ante esta respuesta, el cliente deberá repetir la petición REGISTER igual que la primera pero añadiendo el campo `authorization`¹⁷. Este nuevo campo contendrá los siguientes datos:

- **Username**
- **Realm:** este dato estaba contenido en el campo `www-authenticated` de la respuesta 401.
- **Nonce:** cadena de texto usada para codificar la contraseña del usuario (proveniente en la respuesta 401)
- **Uri:** usuario que se autentica, indicando el servidor como en el campo `to`.
- **Response:** contraseña del usuario cifrada
- **Algorithm:** método usado para el cifrado. En este proyecto se usará MD5.

Si todo ha funcionado correctamente, el servidor responderá con un mensaje “200 OK” quedando el usuario registrado. Tras este proceso, el cliente deberá reenviar la petición Register para evitar la caducidad de la sesión. Este reenvío deberá realizarse de forma periódica en un intervalo de tiempo igual o menor que el campo `expires`.

2.3.2 Mensajería instantánea en SIP

La petición de envío de mensajes instantáneos mediante el protocolo SIP fue normalizado en diciembre de 2002 en la RFC 3428. Esta norma propone el método MESSAGE «una extensión del protocolo SIP que permite la transferencia de mensajería instantánea»¹⁸.

En la cabecera del mensaje SIP, se deben incluir los mismos campos que en el caso del método REGISTER,

¹⁵ Ibid. Apartados 20.6 y 21.4.2

¹⁶ Este método de autenticación está definido en RFC 2617

¹⁷ RFC 3261. Apartado 20.7

¹⁸ RFC 3428 Abstract

pero solo se requieren los campos obligatorios, ya que los opcionales son específicos de la petición REGISTER. El campo "To" indica el destinatario del mensaje, mientras que el campo "From" contiene el nombre del usuario que desea enviar el mensaje. Además deberá incluir los siguientes campos adicionales:

- Content-type: tipo de contenido enviado en el cuerpo del mensaje.¹⁹
- Content-length: tamaño del cuerpo del mensaje en bytes.²⁰

Además toda petición MESSAGE deberá incluir un cuerpo donde se incluirá el contenido del mensaje.

El traspaso de mensajes es el siguiente²¹:

1. El cliente realiza la petición MESSAGE al servidor SIP (o proxy).
2. El servidor procesa el mensaje y realiza, de nuevo, la petición MESSAGE al cliente SIP donde ha iniciado sesión el destinatario del mensaje. Esta petición se envía a la dirección y puerto indicado en el campo Contact de la petición register. Dependiendo de la configuración del servidor SIP, también puede estar dirigido a la dirección y puerto origen de la petición register.
3. Una vez el destinatario ha recibido el mensaje, transmite la respuesta "200 OK" al servidor SIP.
4. El servidor indica al cliente que realizó la petición, que el mensaje ha llegado a su destino mediante la respuesta "200 OK".

La propia RFC 3428 advierte que la recepción de "200 OK" por parte del cliente que realizó la petición indica que el mensaje ha sido entregado a su destino, no que haya sido leído.

¹⁹ RFC 3261: Apartado 20.15

²⁰ Ibid. Apartado 20.14

²¹ RFC 3428. Apartado 10

3 METODOLOGÍA

En este capítulo se detalla la metodología empleada para llevar a cabo la implementación del protocolo SIP en los módulos Wi-Fi y GSM. Se describen los pasos y procedimientos seguidos, desde el diseño experimental hasta la propuesta de pruebas a realizar.

3.1 Diseño experimental

El diseño experimental se enfoca en definir los objetivos específicos de la implementación del protocolo SIP en los módulos Wi-Fi y GSM. Se establecen las funcionalidades y características que se desean lograr, como la capacidad de establecer sesiones SIP y el intercambio de mensajes SIP

Además, se definen los requisitos del entorno de pruebas, incluyendo los dispositivos y software necesarios para llevar a cabo las pruebas. Se determinan los criterios de evaluación y métricas para medir el rendimiento y la eficiencia de la implementación.

El objetivo final del proyecto es la comunicación de mensajes de usuarios SIP mediante los módulos ESP8266 y GSM. Para ello, el software debe cumplir una serie de requisitos. Estos se clasificarán en funcionales (Req-fun) y no funcionales (Req-Nfun). Además, cada requisito tendrá un identificador único numérico.

Los requisitos funcionales relativos al comportamiento de los dispositivos y la conexión serie que hay que abrir para poder usarlos son:

Req-fun-00: Uso similar de la aplicación independientemente del dispositivo seleccionado.

La aplicación funcionará de la misma manera para el usuario, sin importar el módulo que haya elegido. Esto significa que la comunicación serie se realizará de manera transparente, pero será diferente dependiendo del módulo utilizado. A pesar de estas diferencias, los resultados de las acciones realizadas por el usuario serán similares. Por lo tanto, el uso de la interfaz será independiente de la elección del tipo de dispositivo, brindando una experiencia consistente al usuario.

Dependencias: req-fun-01

Req-fun-001: Elección del tipo de dispositivo.

El usuario deberá poder indicar qué dispositivo está utilizando. Tendrá dos opciones posibles: ESP-8266 o GSM.

Req-fun-002: Realizar conexión serie con los dispositivos.

Deberá permitir al usuario realizar la conexión con el puerto serie que elija, permitiendo el envío y recepción de datos. Para ello deberá tener preconfigurados los parámetros necesarios para cada dispositivo, por ejemplo, la tasa de baudios.

Además, para la comunicación será necesaria la codificación y decodificación de los datos, enviados y recibidos respectivamente.

Dependencias: req-fun-00, Req-Fun-009

Req-fun-003: Comunicación de datos con los dispositivos.

Será necesario que el sistema pueda transferir datos con el módulo conectado a través de la conexión serie abierta. Para ello será necesaria la codificación y decodificación de los datos, enviados y recibidos respectivamente. La codificación a usar será UTF-8, permitiendo así que las cadenas sean legibles por el usuario.

Dependencias: req-fun-00

Req-fun-004: Listado de redes wifi disponibles.

El sistema debe permitir obtener el listado de redes que el módulo wifi tiene disponible para su conexión. Por cada red wifi, únicamente deberá mostrar el nombre de esta. Este requisito únicamente tiene sentido y es obligatorio si el usuario desea utilizar el módulo wifi.

Dependencias: Req-fun-001, Req-fun-002, Req-fun-003

Req-fun-005: Conexión de ESP8266 a una red wifi.

El sistema permitirá que el usuario escoja una red de la lista ofrecida por el requisito anterior (req-fun-04) y que indicando la contraseña de acceso, el módulo ESP8266 se conecte.

Dependencias: Req-fun-001, Req-fun-002, Req-fun-003, Req-fun-004

Req-fun-006: El usuario debe recibir información sobre las diferentes acciones que realiza.

El usuario recibirá información sobre si las acciones que ejecuta se realizan correctamente. En caso de existir algún error se le informará sobre este. Las acciones sobre las que el usuario será informado son:

- Apertura de la conexión serie con un dispositivo (Req-Fun-01)
- Conexión a una red Wifi (Req-Fun_05)
- Realización del registro en el servidor SIP (Req-fun-100)
- Recepción de mensajes SIP provenientes de otros usuarios (Req-fun-102)

Dependencias: Req-Fun-003, Req-Fun-00

Req-fun-007: Conexión UDP de los módulos con un servidor remoto.

Se permite la conexión del módulo con el servicio que corre en la IP (o dominio) y puerto especificado por el usuario. Esta conexión estará enfocada a la comunicación con el servidor SIP. Aunque los dispositivos permitan la existencia de más de una conexión concurrente se limitará a una única conexión.

Dependencias: Req-fun-001, Req-fun-002, Req-fun-003, Req-fun-004

Req-fun-008: Comunicación con el servidor remoto a través del módulo.

Se utilizará el dispositivo Wifi o GSM para transmitir y recibir datos del servidor remoto al que hemos establecido la conexión. Estos datos corresponderán a las peticiones y respuestas de los mensajes SIP que se usarán para la comunicación con el servidor.

Dependencias: Req-fun-001, Req-fun-002, Req-fun-003, Req-fun-004, Req-Fun-007. Req-fun-100

Req-fun-009: Desconexión del módulo.

El usuario tendrá la posibilidad de cerrar la conexión abierta con el dispositivo correspondiente. Antes de cerrar la conexión se revertirán los cambios efectuados en el módulo: configuraciones específicas, conexiones abiertas, etc.

Dependencias: Req-fun-01, Req-fun-02, Req-fun-05, Req-fun-07

A continuación, se exponen los requisitos funcionales necesarios para la implementación y uso del protocolo SIP:

Req-fun-100: Transmitir petición REGISTER con credenciales.

El sistema permitirá que el usuario introduzca las credenciales de inicio de sesión en el servidor SIP pudiendo generarse la petición REGISTER y transmitirla al servidor mediante el módulo correspondiente.

Dependencias: Req-fun-003, Req-fun-005, Req-fun-008, Req-Nfun-03

Req-fun-101: Transmitir petición de MESSAGE.

El usuario, una vez se haya efectuado el registro, podrá indicar un destinatario y un cuerpo del mensaje. Con estos datos, el sistema elaborará la petición MESSAGE que será transmitida al servidor a través del módulo que se esté usando.

Dependencias: Req-fun-003, Req-fun-005, Req-fun-008, Req-fun-100, Req-Nfun-03

Req-fun-102: Recepción de mensajes de otros usuarios.

Se recibirán y procederán mensajes SIP de otros usuarios, cuyo contenido y usuario origen hay que procesar y mostrar al usuario. Hay que estar registrado previamente en el servidor.

Dependencias: Req-fun-003, Req-fun-005, Req-fun-008, Req-fun-100, Req-Nfun-03

Reg-fun-103: Envío periódico de RE-REGISTER al servidor SIP

Una vez se ha realizado el registro, la aplicación a desarrollar debe enviar de forma periódica peticiones Register al servidor para evitar que caduque la sesión. Esto se realizará de forma paralela a las acciones que ejecute el usuario y de forma transparente para él.

Dependencias: Req-fun-003, Req-fun-005, Req-fun-008, Req-fun-100, Req-Nfun-03

Req-fun-104: Respuestas automáticas ante las peticiones del servidor SIP.

El propio software deberá procesar las respuestas y peticiones recibidas del servidor a través del módulo que se esté usando. Una vez procesada, deberá crear una respuesta, en caso de ser necesario y posible. Esta respuesta deberá ser transmitida de forma automática y transparente al cliente, notificándole únicamente en caso de error irresoluble. Este requisito conlleva la necesidad de almacenar un histórico de los mensajes transmitidos.

Dependencias: Req-fun-100, Req-fun-101, Req-fun-102, Req-fun-103, Req-Nfun-03

Además de estos requisitos funcionales que suponen funcionalidades específicas que debe incluir el proyecto hay otros requisitos que, aunque no supongan desarrollos propios pueden limitar o especificar aspectos concretos del diseño. Los requisitos no funcionales son:

Req-Nfun-01: Se establecerá un tiempo de espera máximo para la recepción de datos de los dispositivos.

Cuando se realice una lectura de datos de la conexión se establecerá un tiempo máximo de 5 segundos. Si en ese tiempo no se recibe ningún dato se establecerá que no hay información nueva.

Reg-Nfun-02: El módulo no tendrá obligatoriamente una dirección IP pública.

Debido a la naturaleza propia de las redes wifi, cuando el usuario utilice el módulo ESP-8266 no se podrá obtener una IP pública. Además, aunque se use el módulo GSM tampoco se puede asegurar que el dispositivo tenga asignado una IP pública. Esto es debido a que, por la escasez de direcciones, las compañías telefónicas utilizan métodos como CG-NAT haciendo que varios clientes compartan una dirección IP, asignando a sus dispositivos direcciones privada.

Por todo ello, el diseño deberá realizarse suponiendo que el dispositivo se conecta a una red privada teniendo en cuenta las limitaciones que esto supone, por ejemplo, no poder ejecutar servidores ni abrir puertos al que se tenga acceso desde internet.

Dependencias: Req-Nfun-03

Req-Nfun-03: El servidor SIP estará configurado para comunicarse con los clientes a través de la IP y puerto origen de la petición Register.

Por defecto, el servidor enviará las peticiones Message a la dirección indicada en el campo Contact de la petición de registro. En este caso, no se tomará en cuenta el valor de dicho campo, utilizando la dirección y puerto origen del inicio de sesión del cliente.

Este requisito es indispensable para satisfacer las exigencias del requisito Reg-Nfun-02.

Dependencias: Reg-Nfun-02, Ref-fun-101, Ref-fun-102, Ref-fun-103

3.1.1 Casos de uso

Como resultado visible del proyecto a se le proporcionará al usuario una única interfaz gráfica, es decir, para ambos dispositivos se utilizará el mismo software. Esto implica que una de las funcionalidades debe ser la elección del dispositivo que se desea controlar.

Caso de uso 1: Elección del dispositivo a controlar. En la UI el usuario podrá seleccionar el tipo de dispositivo conectado. Para el caso de este proyecto las posibles opciones serán:

- ESP-8266
- MC 60

La elección de este campo será obligatoria para poder hacer uso del resto de funciones.

Caso de uso 2: El usuario podrá elegir el puerto serie del equipo en el que se encuentra conectado el dispositivo y realizar la conexión. Se listarán los diferentes puertos que el equipo tiene disponibles y, una vez que haya seleccionado uno, podrá realizar la conexión.

Dependencias: Debe haberse realizado el Caso de uso 1.

Resultado: Se ha realizado la conexión y la configuración inicial del dispositivo. Informando al usuario de la conexión satisfactoria, o en caso contrario, del error producido.

Dependiendo del tipo de dispositivo que el usuario seleccione podrá realizar, o no, una serie de acciones. En concreto, si se elige ESP-8266, deberá conectarse a una red wifi.

Caso de uso 3: Conectarse a una red wifi. Se listan las redes a las que tiene acceso el módulo wifi. Una vez el usuario ha seleccionado una de ellas, podrá introducir una contraseña y ordenar al módulo su conexión.

Dependencias: Realización del caso de uso 1, caso de uso 2 y la elección del dispositivo ESP-8266.

Resultado: El módulo wifi se conecta a la red wifi seleccionada informando de los errores al usuario.

Una vez la placa está conectada a internet podremos llevar a cabo el registro en el servidor SIP. Para ello el usuario deberá indicar la IP o dominio en el que se encuentra alojado el servidor, el puerto y las credenciales de acceso (usuario y contraseña)

Caso de uso 4: Conexión e inicio de sesión en el servidor SIP.

Para la realización de este caso de uso el usuario deberá indicar la IP o dominio de la máquina que

aloja el servicio, así como el puerto. Debido a que la primera conexión consiste en la petición REGISTER, también debe indicar el usuario y la contraseña de registro SIP.

Dependencias: Realización de casos de uso 1 y 2. En caso de utilizar el módulo WiFi deberá haberse realizado el caso de uso 3.

Precondición: El dispositivo debe estar conectado a internet. Si se han realizado los casos de usos anteriores sin errores y la red wifi está conectada a internet no habría ningún problema

Resultado: El dispositivo se ha conectado con el servidor SIP y se ha realizado el registro. En el caso de existir algún error, el propio software intentaría solucionarlo (por ejemplo, el servidor exige autenticación proxy). En caso de no poderse realizar la conexión se avisa al usuario.

Una vez esté el usuario registrado en el servidor podrá recibir y enviar mensajes SIP a otros usuarios:

Caso de uso 5: Envío de mensajes SIP a otros usuarios.

El usuario indica el cuerpo del mensaje y el usuario receptor del mensaje

Precondición: Se debe haber realizado el registro SIP correctamente. Esto implica la realización de los casos de uso 4, 3 (en caso de módulo wifi), 2 y 1.

Resultado: Se genera la petición MESSAGE y se transmite al servidor.

Caso de uso 6: Recepción de mensajes SIP provenientes de otros usuarios.

El usuario recibirá por pantalla los mensajes provenientes de otros usuarios.

Precondición: Se debe haber realizado el registro SIP correctamente. Esto implica la realización de los casos de uso 4, 3 (en caso de módulo wifi), 2 y 1.

Resultado: En pantalla aparece el cuerpo del mensaje recibido y su usuario origen.

Actores: Este caso de uso es ejecutado por el propio dispositivo al recibir un mensaje del tipo MESSAGE proveniente del servidor SIP.

Además de esto, una vez que se ha iniciado sesión por primera vez, cada cierto tiempo hay que reenviar el mensaje REGISTER para que el servidor no caduque nuestra sesión.

Caso de uso 7: Envío automático y periódico de peticiones REGISTER.

Una vez se ha iniciado sesión se procederá a realizar peticiones REGISTER cada 60 segundos de forma automática para evitar que se caduque la sesión.

Precondición: Se debe haber realizado el registro SIP correctamente. Esto implica la realización de los casos de uso 4, 3 (en caso de módulo wifi), 2 y 1.

Resultado: Se realizan peticiones REGISTER de forma automática y transparente al usuario.

Actores: A diferencia del resto de casos de usos presentados, este no es realizado por el usuario, si no por el propio software que, mediante un temporizador que funciona continuamente ejecuta este caso de uso de forma ininterrumpida.

Por último, el usuario deberá tener la opción de desconectar el dispositivo y eliminar todas las conexiones.

Caso de uso 8: Desconexión del dispositivo

El usuario podrá desconectar la conexión serie realizada con el dispositivo. Para ello se desharán las configuraciones específicas realizadas, por ejemplo, la desconexión con servidores o la paralización de temporizador de RE-REGISTER.

Dependencias: Caso de uso 2 y 1.

Resultado: Se realiza la desconexión de la comunicación serie con el dispositivo.

Al desconectarse el módulo del servidor también se paraliza el caso de uso 7. Esto implica que al cabo de unos segundos la sesión en el servidor SIP caduca produciéndose el cierre de sesión.

Con estos casos de uso se cumplirían los diferentes requisitos expuestos en la primera parte de este punto. Además, estos casos de usos servirán como base para las pruebas, comprobando que cada uno de ellos se realizan sin ningún problema y obteniendo el resultado esperado.

3.2 Configuración del entorno de pruebas

Las pruebas se llevarán a cabo de manera incremental, lo que significa que a medida que se desarrollen los componentes de la aplicación, se realizarán pruebas individuales para cada uno de ellos. Para estructurar la ejecución de las pruebas, se dividirán en dos bloques: las pruebas relacionadas con el módulo WiFi y las pruebas que involucran tanto al módulo GSM como al módulo WiFi. La diferencia principal entre ambos bloques radica en el servidor SIP utilizado. En el primer bloque y en la primera parte del segundo bloque, no se utilizará una interfaz gráfica. Esta se añadirá en las pruebas finales del último bloque, una vez que se hayan probado todos los elementos individuales de la aplicación. Este enfoque incremental asegurará una implementación robusta y permitirá identificar y solucionar posibles problemas a medida que se avanza en el desarrollo del proyecto.

Primer bloque de pruebas

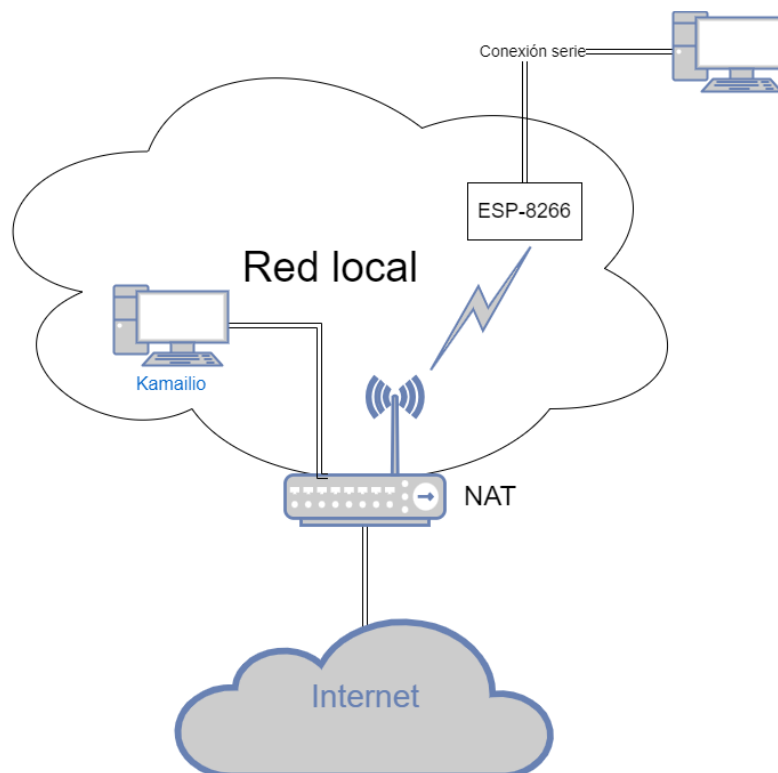
El bloque de pruebas inicial se centrará en la utilización exclusiva del módulo WiFi. El objetivo principal de este bloque es verificar la correcta implementación del protocolo SIP. Esto implica la realización de pruebas específicas que involucran el uso y funcionamiento del módulo WiFi. Se llevarán a cabo pruebas exhaustivas para asegurar que el intercambio de mensajes SIP se realiza de manera adecuada y que se cumplen los requisitos establecidos por el protocolo. Estas pruebas permitirán validar la funcionalidad básica de la comunicación SIP a través del módulo WiFi y asegurar un correcto inicio y control de las sesiones de usuario. Los dispositivos y software involucrados en este conjunto de pruebas son:

- Módulo ESP-8266. Dispositivo presentado en el punto 2.1. La configuración específica de la implementación usada para este proyecto, y por tanto, en estas pruebas es:
 - o Tasa: 115200 baudios
 - o Se establecerá un tiempo de timeout de 5 segundos para esperar la recepción de datos.
 - o Al abrir la conexión se desactivará el eco de los comandos transmitidos, por lo que se tomará como la configuración habitual.
- Servidor SIP Kamailio: Será el servidor SIP que se utilizará para realizar las pruebas de este bloque. El servicio se ejecutará en una máquina virtual Ubuntu por lo que estará en la misma red local que el módulo WiFi. La principal particularidad por la que se ha decidido utilizar este servidor en esta primera parte ha sido su sencillez de instalación, configuración y uso. Las características²² principales de este servidor son:
 - o Flexibilidad: su arquitectura modular, permite ejecutar el servicio base en equipos con bajos

²² Todas las características están detalladas en: <https://www.kamailio.org/w/features/>

- recursos software.
- Permite la ejecución del servicio sobre diferentes capas de transporte: UDP, TCP, TLS y SCTP. En lo referente a la capa de red está preparado tanto para IPv4 como IPv6.
 - Su funcionalidad de autenticación y autorización puede prepararse mediante sus propios ficheros configuración. Además nos brinda la posibilidad de utilizar una base de datos externa para almacenar los datos, o bien, enlazarlo con servidores Radius o Diameter externos.
 - Permite la autenticación Digest de los usuarios.
- Como cliente SIP se utilizará microSIP, este cliente SIP permitirá conectarnos al servidor instalado y comunicarnos con el dispositivo Wi-Fi, pudiendo comprobar el buen funcionamiento del envío y recepción de mensajes SIP entre usuarios.
 - Analizador de paquetes Wireshark. Se utilizará un analizador de paquetes de red, en este caso Wireshark. Gracias a este software se podrá estudiar el intercambio de paquetes que se producen entre Kamailio y el cliente SIP a desarrollar. Wireshark permite capturar todos los paquetes que recoge la interfaz de red del equipo donde se ejecuta. De cada paquete, se puede observar el contenido de las cabeceras que lo forman, desde la capa de enlace hasta la de aplicación.

El escenario utilizado en este bloque de pruebas sería:



Instalación y configuración del servidor Kamailio²³

Como se ha expuesto anteriormente, Kamailio tiene una arquitectura modular. Para facilitar su uso y configuración se instalará el módulo que le permite utilizar una base de datos externa para almacenar los datos de autenticación y autorización. En este caso se va a utilizar mysql. Para la instalación del software necesario

²³ Para la instalación se seguirá la guía oficial: <https://kamailio.org/docs/tutorials/devel/kamailio-install-guide-deb/>

se usarán los repositorios públicos de Ubuntu.

Para empezar, hay que instalar mysql. Para ello ejecutaremos como usuario root:

```
apt-get install default-mysql-server
```

A continuación, se procede con la instalación de Kamailio y el módulo correspondiente a mysql

```
apt install kamailio kamailio-mysql-modules
```

Se instalará el servicio quedando:

- Los ficheros de configuración en `/etc/kamailio`
- El ejecutable en `/usr/sbin/kamailio`
- El script de servicio propuesto en `/etc/init.d/kamailio`

Además se instalará kamctl, «un script de servicio que permite administrar los usuarios, dominios, alias y otras opciones del servidor»²⁴

El hecho de utilizar kamctl permitirá realizar menos cambios en los ficheros de configuración. Aún así hay que configurar esta herramienta. Para ello:

- Debemos especificar el dominio o ip donde se aloja el servidor. Esto se hace en el fichero `/etc/kamailio/kamctlrc`, modificando la propiedad `SIP_DOMAIN` cuyo valor pasará a ser la IP asignada a la interfaz de la red local donde se conectará el módulo wifi.
- En ese mismo fichero hay que configurar la base de datos. Para ello se dará el valor “MYSQL” a la propiedad `DBENGINE`. También se le puede dar valor a `DBRWPW` y `DBROPW`, que deberán tomar como valor las contraseñas de la base de datos para la escritura y lectura respectivamente. Por defecto sus valores son `kamailiorw` y `kamailioro`. Los usuarios para ambas contraseñas son, por defecto, `kamailio` y `kamailioro` que pueden modificarse mediante las propiedades `DBRWUSE` y `DBROUSE`, respectivamente. Por facilidad, se mantendrán los valores por defecto.

A continuación hay que configurar el propio servidor Kamailio. Para ello se modificará el fichero `/etc/kamailio/kamailio.cfg`. En este fichero habrá que añadir las siguientes líneas tras la primera que aparece (“#!KAMAILIO”):

- `#!define WITH_MYSQL`: Con ella activaremos el uso de mysql como base de datos externa (debe haberse instalado el módulo correspondiente).
- `#!define WITH_AUTH`: Permite la autenticación de usuarios.
- `#!define WITH_USRLOCDB`: Establece que los datos de los usuarios sean almacenado en la base de datos.

Una vez realizados todos los cambios necesarios en los ficheros de configuración se debe crear la base de datos. Para ello se ejecutará:

```
kamctl create
```

El script pedirá al usuario la contraseña del usuario root de mysql. Como resultado se crea una base de datos llamada kamailio que posee dos usuarios:

- Un usuario con permisos de lectura y escritura. Su usuario y contraseñas serán los valores `DBRWUSE` y `DBRWPW`, respectivamente
- Un usuario con permiso sólo de lectura. Su usuario y contraseña serán los valores de las propiedades `DBROUSE` y `DBROPW`

Ambos usuarios se crean con la limitación de poder ser accedidos únicamente desde el mismo equipo donde se ejecuta la base de datos (localhost).

²⁴ <https://manpages.ubuntu.com/manpages/trusty/man8/kamctl.8.html>

Una vez estén todas las configuraciones realizadas podemos añadir usuarios ejecutando:

```
kamctl add nombre_usuario contraseña_usuario
```

Habr  que realizar esta acci3n por cada usuario que deseemos a adir.

Realizaci3n de las pruebas

Mediante las pruebas realizadas en este bloque se comprobar  el buen funcionamiento de:

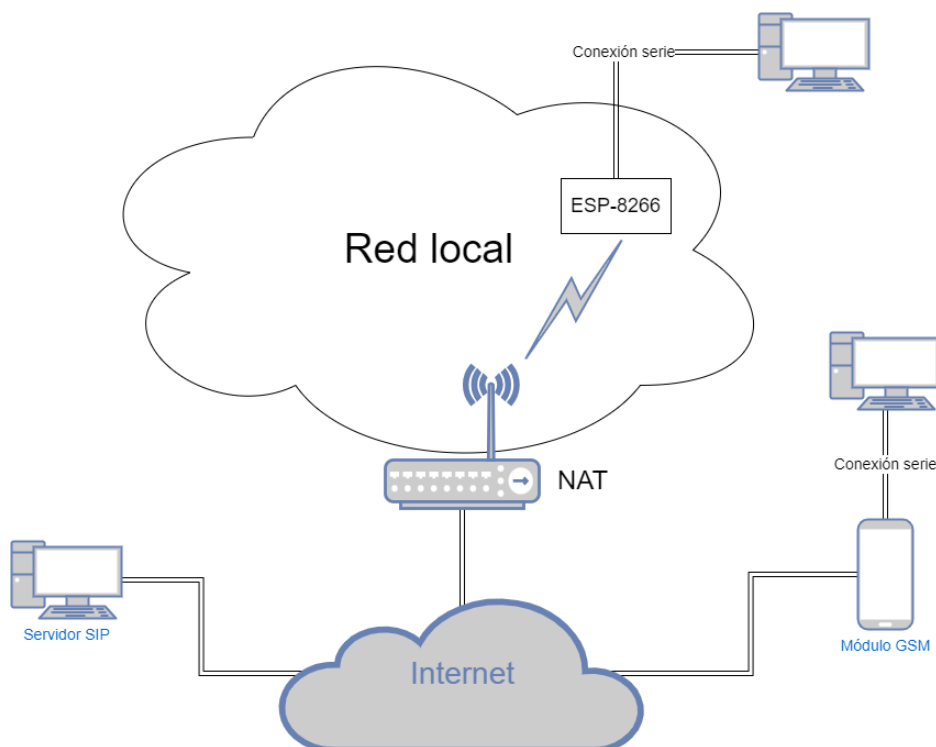
- La implementaci3n del cliente sip, lo que incluye: la realizaci3n de registro de nuevos usuarios, el env o de mensajes, la recepci3n de mensajes y el env o peri3dico de Re-Register para impedir la ca da de la sesi3n del usuario.
- La comunicaci3n del m3dulo Wi-Fi con un servidor. Esta es una consecuencia de la anterior, ya que la comunicaci3n entre el cliente sip implementado y el servidor se realizar  a trav s de este dispositivo.

Esto implica que para la realizaci3n de estas pruebas deben estar desarrollados los siguientes elementos:

- Comunicaci3n serie entre nuestro equipo y el dispositivo.
- Comunicaci3n a trav s de la red entre nuestro m3dulo Wifi y un servidor externo.
- Cliente SIP capaz de realizar peticiones a un servidor SIP y procesar las respuestas recibidas de este.
- Script de prueba donde se realicen diferentes peticiones SIP al servidor. Debido a que en este punto del desarrollo no se cuenta con una interfaz gr fica funcional se crear  un script que permita probar las diferentes funcionalidades del cliente desarrollado.

Segundo bloque de pruebas

En la realizaci3n de estas pruebas se comprobar  el correcto funcionamiento del cliente SIP tanto en el m3dulo Wifi como GSM. Adem s, se probar  la interfaz gr fica de la aplicaci3n. En el escenario actual, el servidor SIP tendr  acceso directo a internet mediante una IP p blica sin necesidad de NAT. El escenario es:



Puede observarse como, en este nuevo escenario el módulo ESP-8266 y el servidor SIP no están en la misma red local. Además, el módulo wifi no tiene acceso directo a internet, si no que sus conexiones deben pasar a través de un NAT. Esto también puede ocurrir en el módulo GSM, ya que la operadora telefónica puede usar CG-NAT creando un direccionamiento privado para sus clientes móviles. Esto implica una configuración específica en el servidor SIP capaz de funcionar a través de NAT y con clientes en redes privadas. Para ello en vez de Kamailio se utilizará Asterisk. Además, se comprueba totalmente la implementación del cliente SIP al ser validada por dos servidores diferentes. Para ello, los dispositivos y software que se utilizarán en este conjunto de pruebas son:

- ESP-8266: Controlador Wifi. Utilizará la misma configuración que en el bloque anterior
- Módulo GSM MC60: Dispositivo presentado en el segundo apartado. La configuración específica de la implementación usada para este proyecto, y por tanto, en estas pruebas es:
 - o Tasa: 115200 baudios
 - o Se establecerá un tiempo de timeout de 5 segundos para esperar la recepción de datos.
 - o Al abrir la conexión, se desactivará el eco de los comandos transmitidos tomándose como la configuración habitual.
 - o Se desactivará la comunicación por el puerto DEBUG del dispositivo.
 - o Se configurará el dispositivo para que transmita las respuestas a los comandos en formato texto.
 - o Se utilizará la conexión GPRS.
 - o La APN correspondiente a la tarjeta SIM que se utilizará para las pruebas es: "airtelwap.es"
- Servidor SIP Asterisk. Según la documentación oficial Asterisk se define como un «framework para la construcción de aplicaciones de comunicaciones multiprotocolo en tiempo real»²⁵. Las principales funcionalidades²⁶ que posee son:
 - o Uso como centralita de llamadas, permitiendo la asignación de números, la realización de

²⁵ <https://www.asterisk.org/get-started/>

²⁶ <https://www.asterisk.org/get-started/applications/>

llamadas, el envío de mensajes y la conmutación del tráfico de comunicaciones al permitir la conexión a otros servidores.

- Servidor de buzón de voz.
- Puede utilizarse como una pasarela VoIP
- Centralita de centro de llamadas, pudiendo controlar la cola de llamadas entrantes, la redirección de llamadas, la configuración de extensiones, etc.

En este proyecto se utilizará como centralita de llamadas (o más bien, mensajes) al igual que Kamailio. De esta forma, con Asterisk registraremos los usuarios que tendrán acceso al servidor y configuraremos el envío de mensajes SIP entre ellos.

Al igual que Kamailio, Asterisk está compuesto por módulos que añaden diferentes funcionalidades al servicio. Asterisk incluye los siguientes servicios independientes²⁷:

- Asterisk: La funcionalidad base del servidor SIP
- DAHDI²⁸: Servicio que permite gestionar los controladores de dispositivos físicos.
- Libpri: Utilizado para manejar y controlar conexiones digitales.

Aunque la instalación, por defecto, incluye todos estos servicios, únicamente le daremos uso a Asterisk.

- Cliente microSIP
- Wireshark

Instalación y configuración del servidor Asterisk²⁹

A diferencia de Kamailio, Asterisk se instalará desde código fuente. Por ello el primer paso es la descarga de los archivos:

```
wget https://downloads.asterisk.org/pub/telephony/asterisk/asterisk-14-current.tar.gz
```

Únicamente se descargará el código propio de Asterisk. No será necesario el uso de DAHDI ni libpri, por lo que omitiremos sus descargas e instalaciones. Una vez descargado, lo descomprimimos:

```
tar -zxvf asterisk-14-current.tar.gz
```

Con el código fuente descomprimido se tendrá acceso a los archivos necesarios para la instalación. Para empezar debemos instalar los requisitos, para ello se ejecutará el script `./contrib/scripts/install_prereq`.

Cuando los pre-requisitos se hayan descargado e instalados se procederá a la configuración de la instalación de Asterisk. Para ello se ejecutará el script `configure` con los siguientes argumentos:

```
./configure --with-pjproject-bundled
```

En condiciones normales, este script no mostrará ningún error ya que los requisitos se han instalado en el paso anterior. Si no es así, deberán instalarse manualmente la librería que se indique. En caso de ser todo correcto se debe ejecutar:

```
make
```

Cuando finalice esta acción aparecerá en pantalla:

²⁷ <https://wiki.asterisk.org/wiki/pages/viewpage.action?pageId=4817506>

²⁸ <https://wiki.asterisk.org/wiki/display/DAHDI/DAHDI>

²⁹ Para la instalación se seguirá la documentación oficial: <https://wiki.asterisk.org/wiki/display/AST/Installing+Asterisk+From+Source>

```
+----- Asterisk Build Complete -----+
+ Asterisk has successfully been built, and +
+ can be installed by running:           +
+                                         +
+             make install                 +
+-----+
+----- Asterisk Build Complete -----+
```

Tal como se indica, se procede a instalar Asterisk ejecutando:

```
make install
```

A continuación, se procederá a la descarga de unos ficheros de configuración de ejemplo. Esto se hace por simplificar, para tener una base de configuración y únicamente tener que modificar los aspectos necesarios para las pruebas:

```
make samples
```

Los ficheros de configuración están en la ruta: `/etc/asterisk`. Se deberán realizar cambios en los siguientes ficheros:

- `extensions.conf`: en este fichero se configurará el manejo que realizará el servidor sobre los mensajes. Para ello se debe añadir al contenido del fichero:

```
[mensaje]
```

```
exten => _[0-9][0-9][0-9][0-9],1,Noop(Mensaje de ${MESSAGE(from)})
```

```
same => n,Noop(Mensaje para ${MESSAGE(to)})
```

```
same => n,Messagesend(pjsip:${EXTEN},${MESSAGE(FROM)})
```

```
same => n,Hangup(16)
```

Con ello se configura la numeración utilizada por los clientes, en este caso, un conjunto de cuatro dígitos. Además se establece el procesamiento que debe realizarse sobre las peticiones Message.

- `pjsip.conf`: En este fichero hay que configurar dos aspectos:
 - o La conexión que van a utilizar los usuarios. Para ello es necesario conocer el dominio o IP pública que tiene el servidor. Se eliminará el contenido que tenga el archivo y se añadirá:

```
[transport-udp]
```

```
type=transport
```

```
protocol=udp
```

```
bind=0.0.0.0:50060
```

```
local_net=192.0.0.0/24
```

```
external_media_address=dominio
```

```
external_signaling_address=dominio
```

Añadiendo en `local_net` la dirección de la red local del servidor.

- o Añadir los usuarios. Para ello por cada usuario se añadirá:

```
[USUARIO]
```

```
type=endpoint
```

```
transport=transport-udp
message_context=mensaje
context=extensiones-internas
disallow=all
allow=ulaw
auth=USUARIO
aors=USUARIO
direct_media=no
rtp_symmetric=yes
force_rport=yes
rewrite_contact=yes
```

```
[USUARIO]
type=auth
auth_type=userpass
password=PASSWORD
username=USUARIO
```

```
[USUARIO]
type=aor
max_contacts=2
```

Sustituyendo USUARIO por el nombre (o número) del usuario que se desea añadir y PASSWORD la contraseña que se le asignará a dicho usuario.

Para definir las características de un usuario es necesario crear tres bloques: endpoint, auth y aor. El primero de ellos permite establecer características propias relativas al procesamiento de las peticiones realizadas por dicho usuario. En concreto:

- La directiva `direct_media=no` indica que los usuarios no pueden dialogar entre sí, cualquier comunicación debe pasar por el servidor. Esto permite la comunicación aunque los clientes se encuentre tras NAT.
- `force_rport=yes`: Establece que toda la comunicación con el usuario debe realizarse a través del puerto en el que ellos inician la conexión, es decir, los usuarios no abrirán otros puertos de comunicación.
- `rewrite_contact=yes`: Indica que puede iniciarse sesión simultáneamente.
- `max_contacts=2`: Establece un máximo de dos sesiones simultáneas por usuario.

Por otra parte, auth permite establecer el modo de autenticación del usuario. Por último aor configura la gestión de los diferentes dispositivos donde se inicie sesión con dicho usuario. En nuestro caso se ha establecido que un usuario únicamente pueda iniciar sesión en dos dispositivos simultáneamente.

Con estos cambios el servidor queda configurado. La ejecución del servidor se realizará en modo debug con el fin de controlar las peticiones que realizan los usuarios y las acciones del servidor. Para ello se ejecuta:

asterisk -cvvvvvv

Realización de las pruebas

Mediante las pruebas realizadas en este bloque se comprobará el buen funcionamiento de:

- La implementación del cliente. Aunque ya haya sido validado en el bloque de pruebas anterior, se reconfirma su funcionamiento al utilizarse con otro servidor SIP
- La comunicación del módulo GSM con un servidor. Se comprobará que tanto la comunicación serie con el dispositivo, como su configuración y manejo por comandos AT se realiza de forma correcta. Además se utilizará para abrir la conexión con el servidor Asterisk.
- Las comprobaciones anteriores se realizarán con un script de pruebas parecido al del bloque de pruebas anterior. Una vez funcione correctamente, se probará la interfaz gráfica. Estas comprobaciones se realizarán tanto con el módulo Wifi como con el módulo GSM, probando todo el sistema en su conjunto.

Esto implica que para la realización de estas pruebas deben estar desarrollados los siguientes aspectos:

- Comunicación serie entre nuestro equipo y el dispositivo GSM.
- Comunicación a través de la red entre nuestro módulo GSM y un servidor externo.
- Cliente SIP capaz de realizar peticiones a un servidor SIP y procesar las respuestas recibidas de este.
- Script de prueba donde se realicen diferentes peticiones SIP al servidor. Este script se utilizará únicamente en la primera fase de las pruebas de este bloque.
- Interfaz gráfica que permita al usuario realizar todas las acciones necesarias. Con ella se realizan pruebas de conjunto de todo el sistema.

3.3 Implementación del protocolo SIP

En esta etapa, se lleva a cabo la implementación del protocolo SIP en los módulos Wi-Fi y GSM. Dicha implementación se realizará en Python. Se ha elegido este lenguaje de programación, por la facilidad de desarrollo interfaces gráficas y la existencia de librerías que permiten la comunicación serie con dispositivos.

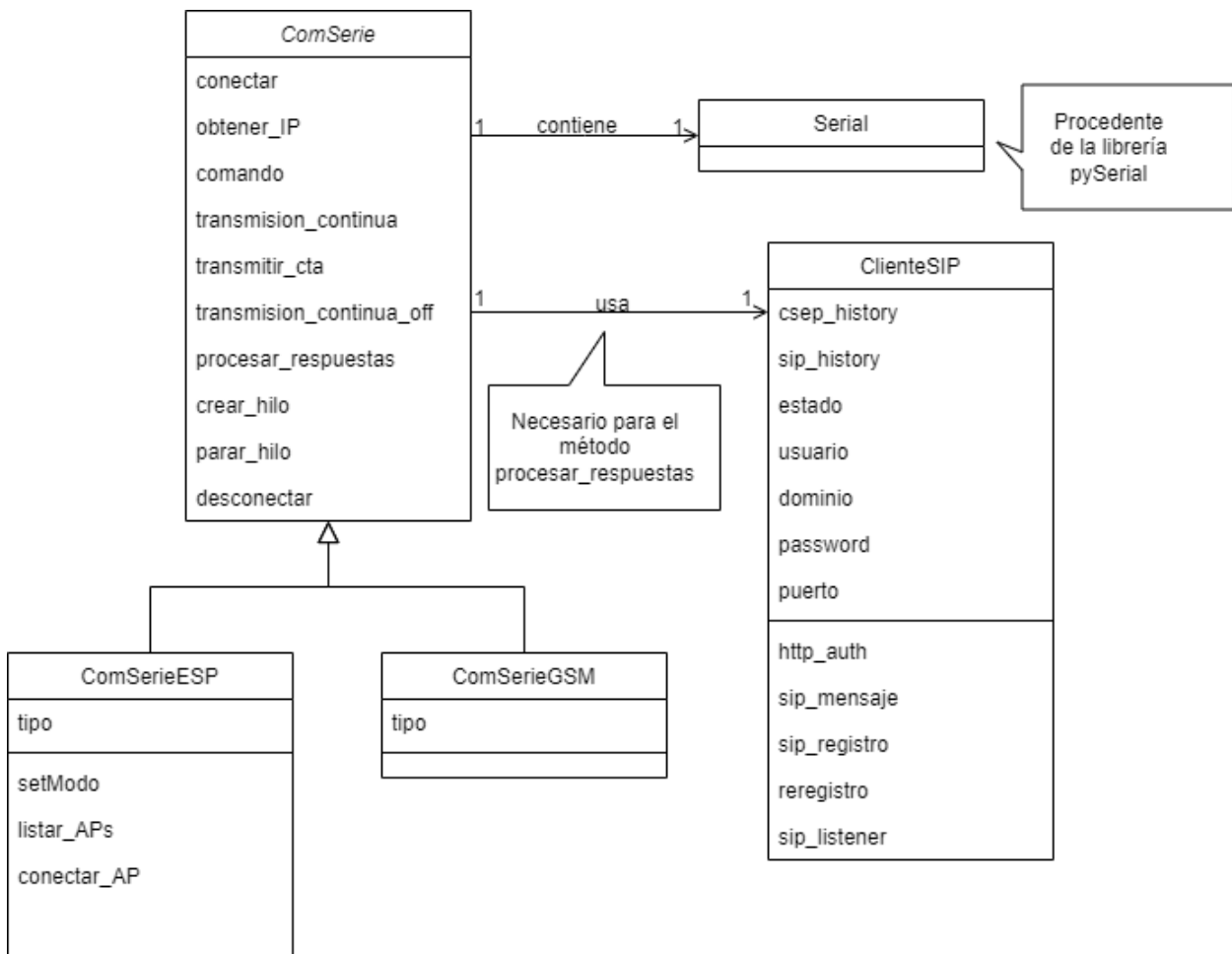
Durante la implementación, es importante seguir las especificaciones y estándares del protocolo SIP, asegurándose de que el código esté bien estructurado, optimizado y libre de errores.

El diseño del sistema se ha basado en la identificación de dos funciones generales que debe llevar a cabo la aplicación y que además son independientes. Estas son: la comunicación serie con los módulos y la implementación del protocolo SIP:

- Comunicación con módulos. Esta parte se encarga de realizar la comunicación serie con los módulos, configurándolos y estableciendo la conexión con el servidor.
- Protocolo SIP. Esta parte llevará a cabo todo el proceso de inicio de sesión en el servidor SIP, así como la comunicación con este. Procesará el contenido de los paquetes recibidos del servidor, generando una respuesta en caso de ser necesario. Además, permitirá al usuario enviar y recibir mensajes SIP.

Ambas partes están interconectadas por la interfaz de usuario que permitirá la comunicación y coordinación de ellas.

Esto da lugar al siguiente diagrama de clases:



Donde:

- **ComSerie** es una interfaz con las funciones comunes de comunicación con ambos dispositivos. Esta interfaz será implementada por las clases **ComSerieESP** y **ComSerieGSM** que se desarrollarán dependiendo de las necesidades del dispositivo Wifi y GSM respectivamente.
- **ClienteSIP**³⁰: Esta clase contendrá la implementación del cliente del protocolo SIP. Cada objeto instanciado de esta clase implica una sesión abierta con el servidor SIP.

La implementación de la recepción de mensajes del servidor difiere según el dispositivo seleccionado, lo que requiere la incorporación de un objeto **ClienteSIP** en las clases **ComSerieESP** y **ComSerieGSM**. Esta adaptación es necesaria para garantizar la correcta recepción y procesamiento de los mensajes SIP en cada dispositivo específico. La inclusión de un objeto **ClienteSIP** en estas clases permitirá gestionar adecuadamente la comunicación entre el módulo seleccionado (ESP o MC60) y el servidor SIP, asegurando así la correcta interpretación de los mensajes recibidos y su posterior procesamiento en la aplicación.

3.3.1 ClienteSIP

Para la implementación del protocolo SIP se ha creado la clase **ClienteSIP**. Esta clase se encarga de:

- Generar los mensajes de petición para tu transmisión al servidor.

³⁰ Como referencia para el diseño y creación de esta clase se han tomado varios ejemplos de la red: <https://github.com/SythilTech/Python-SIP>
<https://github.com/tayler6000/pyVoIP/blob/master/pyVoIP/SIP.py>

- Procesar la información de los mensajes SIP recibidos del servidor y crear un mensaje de respuesta en caso de ser necesario.

Para cada una de las posibles peticiones (mensajes, register, re-register) se ha creado un método encargado de montar la estructura con todos los campos necesarios.

La clase contiene un único constructor que recibe los siguientes parámetros:

- Ip: ip local del cliente
- Nombre de usuario para realizar el registro en el servidor SIP
- Dominio o IP del servidor SIP
- Contraseña del usuario.
- Puerto que utiliza el cliente SIP para la comunicación con el servidor. Si el cliente se conecte a través de un NAT a internet, no tiene sentido que se comunique al servidor el puerto local. En este caso el atributo tomará valor false.
- Mensaje_recibido: este parámetro hace referencia a la función que debe procesar el contenido de los mensajes SIP recibidos. Esta función recibirá dos parámetros: el remitente de dicho mensaje y su contenido.
- Avisos: referencia una función que recibe dos parámetros. Esta función será la encargada de procesar los posibles errores que se produzcan en el servidor. La función recibirá dos parámetros, el primero correspondiente al código de error y el segundo una breve descripción de este. Por ejemplo, esta función avisará de los errores de inicio de sesión, contraseña incorrecta, usuario inválido, etc.
- Envio_servidor: este parámetro también hace referencia a una función. En este caso, la función recibe el paquete que hay que transmitir al servidor. En nuestro caso esta función será la que se comunique con el módulo WIFI o GSM para realizar el envío. Esta función únicamente tendrá un parámetro correspondiente al texto que se transmitirá.
- Auth_username: En el caso de ser necesario un usuario diferente al anterior para el registro del servidor se indicará en este parámetro. En caso contrario, tomará el valor False. Este parámetro es opcional, tomando False como valor predeterminado.
- Account_port: Hace referencia al puerto del servidor SIP. Es opcional, siendo 5060 el valor predeterminado.
- Display_name: Nombre del usuario que se registra al servidor. Es opcional siendo '-' el valor predeterminado.

Estos valores recibidos por parámetros son asignados a los respectivos atributos para su posterior uso en cada uno de los métodos. Además de estos, al instanciar un objeto se crean dos atributos más:

- Histórico: diccionario que contiene cada mensaje que se envía al servidor. Como clave del diccionario se utiliza el identificador del mensaje.
- Estado: Indica el estado en el que se encuentra el cliente Sip con respecto del servidor. Al iniciar el objeto toma el valor "Desconectado".

Los métodos que dan funcionalidad a esta clase son:

- sip_mensaje(destinatario, cuerpo): Su objetivo es el envío de mensaje SIP a otro usuario registrado en el servidor. Para ello recibe dos parámetros: el usuario destino y el contenido del mensaje. El método devuelve el paquete SIP listo para ser transmitido.
- sip_registro(registro_frecuencia=3600): Este método genera el mensaje necesario para que un usuario se registre en el servidor SIP. Tiene un único argumento correspondiente al intervalo de reenvío del mensaje de registro. Por defecto el reenvío se realizará cada 3600 segundos.
- sip_listener(datos): Este método recibe en su único argumento el mensaje procedente del servidor SIP. Se encarga de analizarlo, dependiendo de los diferentes casos devuelve:
 - o Una cadena de texto como respuesta al servidor SIP, que deberá ser transmitida

- True: En este caso no es necesario responder al servidor.
- False: Si se ha recibido algún mensaje de error del servidor y no es posible responder.

Si el paquete procedente del servidor es un mensaje de otro usuario este método se encarga de llamar a la función que se ha indicado en el correspondiente parámetro del constructor para procesar el contenido.

El resto de los métodos existentes son necesarios para el funcionamiento de los objetos pero no están a disposición de uso del usuario. Estos son:

- `reregistro(register_string, registro_frecuencia)`: función encargada de realizar el envío del registro en el intervalo especificado por el usuario.
- `http_auth(authheader, method, address)`: Si en el momento de realizar el registro el servidor exige que la autenticación sea de tipo HTTP o proxy se usará este método para codificar la cabecera como nos exige el servidor. Este método llama a otros dos:
- `KD(secret, data)`: concatena en un único string los dos argumentos separados por “:”.
- `H(data)`: devuelve una cadena con el hash MD5 del argumento pasado como parámetro.

La comprobación de cada uno de estos métodos se ha realizado mediante el servidor Kamailio. Gracias a que este servidor se ejecutaba en una máquina con interfaz gráfica y en la misma red local que el módulo Wifi, se ha podido estudiar el paso de mensajes entre el cliente implementado y el servidor.

Uso y funcionamiento de la clase ClienteSIP

Después de presentar los diversos métodos en clase, procederemos a describir las acciones requeridas para crear y utilizar objetos de ClienteSIP.

Como prerequisites necesitaremos una función que sirva como vía de comunicación con el servidor SIP (en este proyecto se hará uso de los módulos mediante las clases que implementan ComSerie). También hay que definir la función encargada de procesar los mensajes recibidos de otros usuarios SIP.

Una vez tengamos ambas funciones definidas podemos llamar al constructor de la clase ClienteSIP para instanciar un objeto. Esto únicamente implica la creación del objeto y la asignación de los atributos, no se realiza ninguna petición al servidor.

El primer paso será realizar el registro. para ello se llamará al método `sip_registro`. Este método realizará dos acciones:

- Devolverá una cadena de texto correspondiente al mensaje SIP de Register que habrá que transmitir al servidor.
- Creará un hilo que ejecutará el método `reregistro` para que en el intervalo indicado se produzca la renovación del registro.

La respuesta recibida del servidor hay que pasarlo al objeto mediante el método `sip_listener`. Este se encargará de realizar otro tipo de petición en caso de ser necesario (por ejemplo, el servidor SIP al que nos intentamos registrar exige autenticación proxy). Una vez se haya recibido la respuesta de registro satisfactorio el estado del objeto pasa a ser “Conectado”. A partir de este momento cualquier recepción por parte del servidor deberá ser procesada por `sip_listener`. Los mensajes típicos que se recibirán son:

- Mensajes de asentimiento como respuestas a los Register enviados de forma periódica. Estos son tratados de forma invisible al usuario
- Mensajes procedentes de otros usuarios, para procesarlos se utiliza la función que el usuario ha indicado en el constructor del objeto.
- Mensajes de error con códigos 407, 401, 403 y 404. Se avisa al usuario mediante la función asignada al atributo avisos.

Cuando el usuario desee enviar un mensaje a otra persona deberá llamar al método `sip_message`. Este método

devolverá un String con el contenido del paquete que deberá transmitirse al servidor, el envío no es realizado por el objeto ClienteSIP de forma automática.

La respuesta del servidor ante la petición de envío de mensaje deberá ser procesada por sip_listener.

Al enviar un mensaje al servidor o realizar cualquier otra petición, el objeto Sip guarda un histórico (en el atributo creado para ello). De esta forma, mediante el identificador del mensaje, el método sip_listener puede relacionar las respuestas del servidor con el mensaje correspondiente.

3.3.2 Interfaz ComSerie

La interfaz ComSerie tiene como propósito permitir la interacción con los dos módulos utilizados en este proyecto. El objetivo es que el modo de uso de ambos módulos sea independiente del módulo específico que se esté utilizando, por lo tanto, se crea la interfaz ComSerie. Esta interfaz define los métodos fundamentales para la configuración, el envío de comandos AT y la recepción de datos.

Las clases que implementen esta interfaz deberán tener un único constructor. Este constructor recibirá un parámetro único que representa el puerto serie al que está conectado el dispositivo. Al ser invocado, el constructor establecerá una conexión serie con el dispositivo utilizando el puerto indicado y configurará los ajustes básicos del dispositivo.

Los métodos que se han definido en esta interfaz son:

- `cerrar_conexion()`: Cierra la conexión con el dispositivo serie.
- `comando(comando)`: Transmite un comando al dispositivo. Devuelve true o false indicando si se ha transmitido correctamente, o no. Comprueba únicamente la transmisión, no analiza la respuesta de la ejecución del comando.
- `obtener_IP()`: Ejecuta el comando correspondiente para obtener la dirección IP que ha obtenido el módulo a conectarse a la red (ya sea wifi o GSM). Devuelve la dirección IP o false en caso de que esta no se haya establecido aún.
- `conectar(tipo, ip, puerto)`: Conecta el módulo a un servidor remoto. Para ello hay que indicar el tipo de conexión ("UDP" o "TCP"), la ip remota y el puerto. Devuelve true si la conexión ha sido exitosa o false en caso contrario.
- `transmitir(contenido)`: Transmite la cadena pasada por parámetro al servidor con el que se ha establecido conexión. Devuelve true si el envío es exitoso y false en caso contrario.
- `transmisión_continua()`: Configura el dispositivo en el modo de transmisión continua.
- `transmitir_cta(data)`: Transmite los datos pasados como parámetro al servidor remoto. Es similar al método "transmitir", para los casos donde se ha activado la transmisión transparente en el dispositivo.
- `transmisión_continua_off()`: Desactiva la transmisión continua en el dispositivo.
- `procesar_respuesta(sip)`: Recibe como parámetro el objeto ClienteSIP que representa la sesión sip iniciada. Se encarga de recibir los datos del servidor, procesarlos y preparar la respuesta para transmitirla al servidor.
- `crear_hilo(sip)`: Crea un hilo donde se ejecuta en bucle el método procesar_respuesta, para que ese procesamiento se realice en segundo plano y de forma transparente al usuario.
- `parar_hilo()`: Detiene el hilo creado anteriormente.
- `Desconectar()`: Desconecta la conexión serie que permite la comunicación con el dispositivo.

De esta forma se establecen las funciones básicas que ambos dispositivos y se proporciona una misma forma de acceso.

Además, las clases que implementen esta interfaz tendrán el atributo 'tipo' para indicar el dispositivo que están representando. Esto permitirá diferenciar acciones que solo deben ser realizadas por un módulo específico. Por

ejemplo, el escaneo de puntos de acceso cercanos solo deberá ser realizado por el módulo de WiFi.

3.3.3 Librería serial

Las implementaciones de la interfaz ComSerie utilizan la librería pySerial, la cual permite la comunicación con los puertos serie de un equipo. Esta librería incluye la clase Serial³¹, la cual facilita el control de un dispositivo serie específico. Para instanciar un objeto de esta clase, el constructor requiere los siguientes parámetros:

- Port: Nombre del puerto, por defecto es None.
- Baudrate, por defecto es 9600.
- Bytesize: numero de bits de datos, por defecto es 'EIGHTBITS'.
- Parity: Define si se debe realizar, o no, la comprobación de paridad. Por defecto está desactivado.
- Stopbits: Define el número de bits de parada, por defecto toma el valor: ""STOPBITS_ONE"".
- Timeout: tiempo de espera de lectura máximo. Por defecto no tiene definido ningún valor.

Existen otros parámetros opcionales que tienen un valor predeterminado y no serán utilizados en este proyecto. Al crear un objeto de la clase Serial, estos parámetros se asignan a los atributos correspondientes, lo que permite acceder y modificarlos posteriormente si se desea cambiar algún valor.

Es importante mencionar que la ejecución del constructor no realiza automáticamente una conexión con el dispositivo. Para establecer la conexión, es necesario llamar al método open(). Por lo tanto, se ha decidido instanciar la clase con los valores predeterminados de los parámetros y luego modificar los atributos necesarios para realizar la configuración del dispositivo, incluyendo aquellos relacionados con la velocidad de transmisión, el tipo de paridad y otros ajustes relevantes. Esto proporciona flexibilidad para adaptar la configuración a las necesidades específicas del dispositivo y del proyecto.

Una vez instanciado el objeto y configurados los atributos necesarios se puede ejecutar el método open() que realiza la conexión con el dispositivo. Las diferentes acciones que se pueden ejecutar sobre el dispositivo están definidas por los métodos que posee la clase Serial:

- readln(): Devuelve un objeto Bytes con los datos recibidos del dispositivo hasta encontrar un carácter de nueva línea ('\n') o hasta que termine el temporizados timeout.
- read(size): lee del dispositivo serie y devuelve un objeto Bytes con el número de bytes indicado por parámetro (por defecto un único byte). En el caso de tener activo el timeout, pueden recibirse menos bytes de los indicados.
- write(data): Recibe por parámetro los datos en un objeto Bytes a transmitir por el puerto serie. Devuelve el número de bytes transmitidos.
- close(): Cierra la conexión serie.

Algunos de los métodos de la clase Serial utilizan el objeto Bytes. Para que estos datos sean legibles y puedan ser utilizados como strings, es necesario codificarlos o decodificarlos utilizando los métodos decode() y encode(), los cuales están disponibles en las clases String y Bytes, respectivamente.

El método encode() es un método estático, por lo que se puede llamar directamente: str.encode(cadena_a_codificar). Por defecto, utiliza la codificación UTF-8.

Además de estos métodos también se dará uso al atributo in_waiting de los objetos Serial. Este atributo devuelve el número de bytes en espera de ser leídos. De esta forma sabremos si queda información por leer del dispositivo.

³¹ https://pyserial.readthedocs.io/en/latest/pyserial_api.html

La librería ofrece otros métodos y atributos además de los expuestos, pero estos serán los que se usará para crear las clases que implementará la interfaz ComSerie.

3.3.4 ComSerieESP

Esta clase se encarga de implementar la interfaz ComSerie utilizando los comandos AT específicos del módulo ESP-8266.

El constructor de esta clase recibe por parámetro el puerto (tal y como se definió en la interfaz) y realiza las siguientes acciones:

1. Crea un objeto de tipo Serial y lo establece como atributo para ser utilizado en el resto de métodos. Crea el atributo tipo con valor "ESP".

```
self.tipo="ESP"  
self.ser = serial.Serial()
```

2. Una vez creado el objeto Serial configura los atributos necesarios para poder realizar la conexión serie (puerto y tasa de baudios a la que funciona el módulo ESP8266) y ejecuta el método open.

```
self.ser.baudrate = 115200  
self.ser.port = puerto  
self.ser.timeout = int(config.get('ESP', 'timeout'))  
self.ser.open()
```

3. Envía el comando ATE0 para desactivar el funcionamiento en modo eco en la respuestas del módulo.

```
self.comando("ATE0")
```

Además de los métodos existentes en ComSerie se han añadido los siguientes para dar funcionalidad a acciones propias de este módulo:

- `set_modo(modo)`: Establece el modo de operación del módulo ejecutando el comando AT+CWMODE. Devuelve true si la ejecución es correcta y false en caso contrario.
- `listar_APs()`: Devuelve una lista con los puntos de acceso disponibles que detecta el módulo. Para ello se utiliza el comando AT+CWLAP
- `conectar_AP()`: Recibe por parámetros el nombre de la red wifi y la contraseña, utilizando el comando AT+CWJAP y comprobando que el módulo ha podido realizar tal conexión. Devuelve true o false si el módulo ha podido conectarse correctamente o no.

Para la conexión con servidores remotos, se utiliza una forma de comunicación llamada comunicación transparente. Esto implica que cualquier información que se escriba en el puerto serie será enviada directamente al servidor remoto, y cualquier dato recibido del servidor remoto será transmitido a través del puerto serie.

Esta elección de comunicación tiene una limitación importante: solo se puede establecer una conexión con un único servidor remoto. Sin embargo, en el contexto de este proyecto en particular, esta limitación no representa un inconveniente, ya que el objetivo es comunicarse con un servidor específico.

Teniendo en cuenta esta configuración de comunicación transparente, los métodos de la clase o interfaz se han implementado para facilitar esta forma de interacción. Esto significa que los métodos proporcionan la funcionalidad necesaria para enviar comandos y recibir datos desde el servidor remoto a través del puerto serie. Al utilizar estos métodos, se logra una comunicación efectiva y bidireccional con el servidor remoto utilizando el dispositivo conectado al puerto serie.

- `conectar(tipo, ip, puerto)`: Utiliza el comando AT+CIPSTART para conectarse al servidor

en la IP y puerto indicado. El tipo hace referencia a UDP o TCP. También se comprueba si la respuesta del dispositivo indica que la conexión se ha realizado satisfactoriamente o se ha producido algún error.

- `transmisión_continua()`: establecer una comunicación continua y transparente entre el dispositivo. Primero, se envía el comando "AT+CIPMODE=1" al dispositivo para activar el modo de transmisión transparente. Este modo permite una transmisión bidireccional directa entre el dispositivo y el servidor. Luego, se envía el comando "AT+CIPSEND" para entrar en el modo de acción y establecer la conexión con el servidor. Una vez que se recibe la respuesta, se habilita la comunicación directa con el servidor, lo que permite la transmisión continua de datos.
- `transmitir_cta(datos)`: Este comando envía directamente los datos al dispositivo mediante el puerto serie (Estos datos deben estar debidamente codificados).
- `transmisión_continua_off()`: Transmite la secuencia de caracteres necesaria para detener la transmisión continua. En el caso del ESP8266, se transmiten un retorno de carro y carácter de nueva línea (`'\r\n'`) se espera un segundo y posteriormente: `'+++'`. Por último hay que esperar dos segundos sin transmitir ningún byte.
- `desconectar()`: para desconectar la conexión con el servidor, se ejecuta el método `transmisión_continua_off` y posteriormente se transmite `AT+CIPCLOSE=5` para cerrar el socket correspondiente.

Para recibir información proveniente del servidor, es necesario que el usuario llame al método "iniciar_hilo". Este método se encarga de iniciar un hilo que ejecuta de forma constante el método "procesar_respuesta". Es importante que el usuario pase como parámetro el objeto "Sip" correspondiente al dispositivo.

El método "procesar_respuesta" recibe como parámetro el objeto "Sip" y se compone de un bucle infinito que se encarga de comprobar si existen bytes en espera de ser leídos desde el servidor. En caso de que se detecte la presencia de bytes en el buffer de entrada, se procede a realizar las siguientes acciones:

1. En primer lugar, se verifica si la cadena recibida no es igual a "ERROR". Esto se hace para asegurarse de que no haya ocurrido algún error en la comunicación con el servidor. En caso de que la cadena sea diferente de "ERROR", se procede con las siguientes operaciones.

```
cad = self.ser.readline()
line = cad.decode(encoding="utf-8")
if line != 'ERROR\r\n':
    recibido = recibido + line
```

2. Se busca en la cadena el valor del campo "content-length" en la cabecera del mensaje. Este campo almacena el tamaño en bytes del cuerpo del mensaje. Para localizar este campo, se busca una línea que contenga "content-length" y se extrae el valor asociado.

```
if line.split(':')[0]=='Content-Length':
    content= int([line.split(':')[1]])
```

3. Una vez obtenido el valor del campo "content-length", se puede determinar la longitud del cuerpo del mensaje. El cuerpo del mensaje se diferencia de la cabecera SIP por un retorno de carro y una nueva línea (`'\r\n'`).

Después de recibir el retorno de carro y nueva línea (`"\r\n"`), el método continúa para obtener el cuerpo del mensaje. Para ello, utiliza el tamaño indicado en el campo "content-length" de la cabecera:

4. Basándose en el tamaño obtenido del campo "content-length", se sabe cuántos bytes forman el cuerpo del mensaje.
5. A continuación, se lee la cantidad correspondiente de bytes del dispositivo ESP-8266.

```
if content > 0:  
    cad = self.ser.read(content)  
    line = cad.decode(encoding="utf-8")
```

6. Estos bytes leídos representan el cuerpo del mensaje y contienen la información adicional enviada por el servidor SIP.

Después de realizar las operaciones anteriores, se procede a llamar al método "sip_listener" del objeto "Sip", pasando como parámetro el mensaje SIP recibido, que incluye tanto la cabecera como el cuerpo:

7. El método "sip_listener" es responsable de procesar y analizar el mensaje SIP recibido. Esto implica interpretar los campos de la cabecera, realizar validaciones, generar respuestas o reformular la petición, en el caso de ser necesario.
8. Este método retorna la respuesta generada, que puede ser un mensaje SIP de respuesta o cualquier otra información relevante que se necesite transmitir al servidor.

```
respuesta = sip.sip_listener(recibido)
```

9. La respuesta obtenida se envía al servidor utilizando el método "transmitir_cta". Este método se encarga de enviar los datos al dispositivo mediante el puerto serie, asegurándose de que estén correctamente codificados.

```
self.transmitir_cta(respuesta)
```

En conclusión, la clase ComSerieESP actúa como controlador del dispositivo ESP-8266 permitiendo la conexión con un servidor externo y la transmisión de información a este. Además, obtendrá la información recibida de dicho servidor y procesarla según la implementación del cliente SIP desarrollado.

3.3.5 ComSerieGSM

Al igual que en la clase expuesta anteriormente, "ComSerieGSM" es otra clase que implementa la interfaz "ComSerie" y actúa como controlador del dispositivo GSM MC60. Esta clase se encarga de utilizar los comandos AT específicos del dispositivo y realizar la configuración necesaria para su correcto funcionamiento.

"ComSerieGSM" se utiliza para establecer la comunicación con el dispositivo GSM MC60 a través de la conexión serie. Proporciona métodos y funcionalidades para enviar comandos AT al dispositivo, recibir respuestas y configurar parámetros específicos, como la configuración de red, la gestión de llamadas y mensajes, entre otros.

Al implementar la interfaz "ComSerie", "ComSerieGSM" garantiza que cumple con los métodos y requisitos necesarios para interactuar con otros dispositivos que también implementen esta interfaz. Esto facilita la integración y la posibilidad de cambiar de dispositivo o controlador sin afectar la lógica del programa principal.

En resumen, "ComSerieGSM" es una clase que actúa como controlador del dispositivo GSM MC60, utilizando los comandos AT propios del dispositivo y realizando la configuración específica requerida. Proporciona una interfaz de comunicación con el dispositivo a través de la conexión serie, permitiendo enviar y recibir comandos y configurar parámetros necesarios para su funcionamiento adecuado.

El constructor de la clase "ComSerieGSM" acepta como parámetro el puerto al que está conectado el dispositivo GSM, siguiendo la definición de la interfaz "ComSerie". Al instanciar un objeto de esta clase, se realizan las siguientes acciones:

1. Se crea un objeto de tipo "Serial" y se establece como atributo de la clase. Este objeto se utilizará en los demás métodos de la clase para realizar la comunicación serie con el dispositivo. Además, se crea el atributo "tipo" con el valor "GSM", que indica el tipo de dispositivo que representa esta instancia.

```
self.tipo="GSM"
self.ser = serial.Serial()
```

2. Una vez creado el objeto "Serial", se configuran los atributos necesarios para establecer la conexión serie con el dispositivo MC60. Esto incluye configurar el puerto y la tasa de baudios a la que el módulo MC60 opera. Luego, se ejecuta el método "open" para establecer la conexión con el dispositivo.

```
self.ser.baudrate = 115200
self.ser.port = puerto
self.ser.timeout = config.getint('GSM', 'timeout')
self.ser.open()
```

3. A continuación, se realizan las configuraciones iniciales necesarias para usar el dispositivo:
 - a. Se envía el comando "ATE0" al dispositivo. Este comando se utiliza para desactivar el modo de eco en las respuestas del módulo GSM. Al desactivar el eco, el dispositivo no enviará de vuelta los comandos recibidos, lo cual facilita la lectura y procesamiento de las respuestas del dispositivo.

```
self.comando("ATE0")
```

- b. Se desactiva el puerto DEBUG con el envío del comando AT+QEAUART=1,99. De esta forma, toda la comunicación serie con el dispositivo se realizará mediante su puerto principal.

```
self.comando("AT+QEAUART=1,99")
```

- c. El envío del comando "AT+CMEE=2", permite la configuración de los mensajes de errores en modo texto.

```
self.comando("AT+CMEE=2")
```

- d. Se configura la red GSM correspondiente a la operadora que estamos utilizando. El nombre de la APN proviene de la configuración asignada por el usuario en el fichero de propiedades.

```
self.comando('AT+QICSGP=1,+config.get('GSM', 'APN'))
```

- e. Mediante el envío del comando: "AT+QIREGAPP" se ejecuta la pila de protocolo TCP/IP.

```
self.comando('AT+QIREGAPP')
```

- f. Por último se envía "AT+QIACT" que activa el contexto GPRS finalizando la configuración del protocolo IP, permitiendo la realización de futuras conexiones.

```
self.comando('AT+QIACT')
```

Tanto la interfaz ComSerie como la implementación ComSerieESP utilizan la transmisión transparente para el intercambio de datos con un servidor externo. Aunque el módulo MC60 no admite este tipo de comunicación, se han implementado métodos simulados en la clase ComSerieGSM para que su funcionamiento sea similar al del ESP-8266. De esta manera, se logra una coherencia en la estructura y uso de los métodos en ambas implementaciones, a pesar de las diferencias en la tecnología subyacente de los módulos.

De esta forma, la implementación de los métodos existentes en ComSerie relativos a las conexiones de red se ha realizado de la siguiente forma:

- `conectar(tipo,ip,puerto)`: Antes de realizar la conexión con el comando `AT+QIOPEN` se realizan varias configuraciones:
 - o Mediante `AT+QIHEAD=1` se activa la cabecera de datos con el tamaño del paquete recibido.
 - o Por último se establece la conexión. Si el proceso es exitoso el método devuelve `true`. En caso contrario devuelve `false`.
- `transmision_continua()`: en este método se configura el módulo MC60 para que no envíe respuestas con los caracteres `>` o `SEND OK` al enviar datos. Esto se logra mediante el comando `AT+QIPROMPT=2`. Dado que el módulo MC60 no tiene un modo de funcionamiento similar a la transmisión continua del ESP-8266, esta configuración se utiliza para simular un comportamiento similar al enviar datos al servidor externo.
- `transmitir_cta(datos)`: mediante el comando `AT+QISEND` se transmiten los datos pasados por parámetros. El comando se ejecuta sin indicar el número de bytes a transmitir, por ello al finalizar la transmisión de datos se envía control-z (`\x1A`) para indicar la finalización de la transmisión.
- `transmision_continua_off()`: En este caso, este método no tiene sentido. Aunque se ha implementado este método por razones de compatibilidad con la interfaz, su ejecución no tiene ningún efecto en el funcionamiento del módulo.
- `desconectar()`: mediante la ejecución del comando `AT+QICLOSE` se cierran las conexiones de red abiertas.

En la implementación de la clase `ComSerieGSM`, al igual que en `ComSerieESP`, para recibir información del servidor es necesario llamar al método `iniciar_hilo` y pasar como parámetro el objeto `ClienteSIP`. Esto creará un hilo que ejecutará continuamente el método `procesar_respuesta`, permitiendo así la recepción constante de datos del servidor en el módulo MC60.

El método `procesar_respuesta` en la implementación de la clase `ComSerieGSM` funciona de manera similar a la implementación en `ComSerieESP`. Se mantiene un bucle infinito que verifica la disponibilidad de datos para leer. En este caso, se comprueba si los datos comienzan con el encabezado `IPD`, lo que indica que provienen del servidor. A partir de este encabezado, se extrae la cantidad de bytes que conforman el paquete recibido. Una vez leídos, se envían como parámetro al método `sip_listener` del objeto `ClienteSIP`. Si es necesario transmitir una respuesta al servidor, se ejecuta el método `transmitir_cta`.

En resumen, la clase `ComSerieGSM` desempeña el papel de controlador del dispositivo GSM MC60, permitiendo establecer la conexión con un servidor externo y transmitir información hacia él. Además, es capaz de recibir y procesar la información proveniente del servidor de acuerdo con la implementación del cliente SIP desarrollado. Al igual que en `ComSerieESP`, `ComSerieGSM` facilita la comunicación bidireccional entre el módulo GSM MC60 y el servidor externo, brindando una solución integral para la gestión de datos en entornos de comunicación remota.

3.3.6 Interfaz gráfica y estructura del proyecto

En el marco de esta investigación, se ha optado por utilizar la biblioteca `PySide2`³², disponible para el lenguaje de programación Python, con el fin de desarrollar la interfaz gráfica requerida. Para garantizar la estabilidad y la integridad del proyecto, se ha tomado la decisión de crear un entorno virtual de Python. Esta estrategia permite gestionar las dependencias de manera eficiente al incluir bibliotecas específicas y versiones controladas en el entorno virtual, evitando posibles conflictos con otras bibliotecas previamente instaladas en el sistema.

Un entorno virtual en Python es una herramienta que permite crear un espacio aislado y autónomo en el que se pueden instalar y gestionar las dependencias de un proyecto de software de forma independiente. En lugar de depender de las bibliotecas y versiones instaladas globalmente en el sistema, un entorno virtual proporciona una instalación aislada de Python y sus paquetes asociados.

³² https://wiki.qt.io/Qt_for_Python

Dentro de un entorno virtual, se puede instalar una versión específica de Python y las bibliotecas necesarias para el proyecto, sin afectar a otras aplicaciones o proyectos en el sistema. Esto permite mantener la compatibilidad y la consistencia del entorno de desarrollo, ya que cada proyecto puede tener sus propias dependencias y versiones de bibliotecas, sin interferir con otros proyectos.

Además, los entornos virtuales facilitan la gestión de dependencias al proporcionar herramientas para crear, activar, desactivar y eliminar entornos virtuales de manera sencilla. Esto permite tener un control preciso sobre las bibliotecas utilizadas en cada proyecto, evitando conflictos entre diferentes versiones de las mismas.

Una vez creado el entorno virtual hay que instalar las librerías que se usarán en el proyecto. Estas son:

- Pyside2
- Pyserial

A continuación se procede a la creación del árbol de directorio:

- En la raíz del entorno virtual se creará un directorio “ui_files”. Contendrá los ficheros ui en formato xml con la descripción de las distintas interfaces.
- Se creará la carpeta “views” que contendrá el código Python de las interfaces existentes en la carpeta anterior.
- “controllers”: en la carpeta creada con este nombre contendrá el código relativo al control de las interfaces.
- Se creará la carpeta “Sip” donde se encontrarán las clases expuestas en los puntos anteriores.
- En la raíz también se incluirá el archivo de código principal, que se nombrará app.py. La ejecución de dicho archivo pondrá en funcionamiento toda la aplicación, mostrando la interfaz de usuario.
- También se incluirá el archivo config.properties en la raíz, donde el usuario podrá establecer el valor de algunos parámetros. El esquema de dicho archivo es:

```
[GSM]
APN = airtelwap.es
timeout = 5
[ESP]
timeout = 5
```

Los elementos que aparecerán en la interfaz gráfica con los que el usuario podrá interactuar se dividirán en diferentes bloques dependiendo de su función.

El bloque relativo a la conexión serie tendrá los siguientes elementos:

- Un selector donde el usuario deberá introducir el dispositivo que va a utilizar. Tendrá dos opciones:
 - o ESP82-66
 - o MC60
- Un selector donde se pondrá a disposición del usuarios los distintos puertos serie del equipo para que elija el correspondiente al dispositivo.
- Un botón “Conectar”
- Un texto donde se indica el estado del dispositivo, al iniciar la aplicación aparecerá, por defecto “Desconectado”

Otro bloque será el relativo a la configuración de la conexión Wi-Fi. En este caso el bloque estará desactivado y únicamente se podrá hacer uso de él si el usuario ha elegido el dispositivo ESP-8266 en el bloque anterior. Este bloque incluye:

- Un selector donde el usuario podrá elegir las redes Wi-Fi disponibles.

- Una entrada de texto donde se introducirá la contraseña de la red wifi
- Un botón “Conectar”



The screenshot shows a 'Conexión Wi-Fi' (Wi-Fi Connection) dialog box. It contains a 'Red wifi:' (Wi-Fi network) dropdown menu, a 'Contraseña:' (Password) text input field, and a 'Conectar' (Connect) button at the bottom.


El último bloque de funcionalidad se refiere a la conexión con el servidor SIP. Sin embargo, este bloque estará inactivo hasta que se haya establecido la conexión Wi-Fi en el caso del dispositivo ESP-8266, o se haya conectado el dispositivo MC60. Una vez que se haya completado exitosamente la conexión correspondiente, el bloque de conexión con el servidor SIP estará disponible y se podrá iniciar la comunicación con el servidor para el intercambio de información. Para ello incluirá los siguientes elementos:

- Una entrada de texto donde el usuario indicará el dominio del servidor SIP.
- Una entrada de texto para indicar el usuario con el que iniciar sesión en el servidor SIP.
- Una entrada de texto donde se introducirá la contraseña.
- Un botón “Conectar”
- Un texto donde aparecerá el estado de la conexión. Por defecto aparecerá “Desconectado”.



The screenshot shows a 'Conexión SIP' (SIP Connection) dialog box. It contains three text input fields labeled 'Servidor SIP:', 'Usuario SIP:', and 'Contraseña SIP:'. Below these fields is an 'Iniciar Sesión' (Log In) button. At the bottom, it displays 'Estado: Desconectado' (Status: Disconnected).

Tras establecer la conexión, se habilitará al usuario la opción de enviar mensajes SIP. Esto se logrará mediante la inclusión de dos campos de texto adicionales, uno para ingresar el contenido del mensaje y otro para indicar el destinatario. Además, se proporcionará un botón que permitirá enviar el mensaje al servidor. De esta manera, el usuario tendrá la capacidad de enviar mensajes SIP de forma sencilla y directa a través de la interfaz.



The screenshot shows a message sending interface. It features a large text input field labeled 'Mensaje:' (Message) at the top. Below it, there is a 'Destinatario:' (Recipient) text input field and an 'Enviar' (Send) button.

Por último, la interfaz contará con un área de texto especialmente diseñada para mostrar notificaciones y mensajes SIP dirigidos al usuario, lo que garantizará una visualización clara y conveniente de la información

relevante del dispositivo. Esta función permitirá al usuario estar al tanto de aspectos importantes, como el resultado de la conexión serie (ya sea exitosa o con errores), el estado de la conexión WiFi o SIP, así como los mensajes SIP enviados por otros usuarios, fomentando así una comunicación bidireccional efectiva y proporcionando una experiencia interactiva y enriquecedora en la interfaz gráfica de usuario.

En conjunto, la interfaz de usuario quedaría:

The screenshot displays the 'Mensajería SIP' application window. It features three main connection sections on the left and a central message area on the right.

- Conexión serie:** Includes a 'Dispositivo' dropdown menu set to 'ESP8266', a 'Puerto serie' dropdown menu set to 'COM6', a 'Conectar' button, and an 'Estado' indicator showing 'Desconectado'.
- Conexión Wi-Fi:** Includes a 'Red wifi' dropdown menu, a 'Contraseña' text input field, a 'Conectar' button, and an 'Estado' indicator showing 'Desconectado'.
- Conexión SIP:** Includes 'Servidor SIP', 'Usuario SIP', and 'Contraseña SIP' text input fields, an 'Iniciar Sesión' button, and an 'Estado' indicator showing 'Desconectado'.

The central area contains a large empty box for displaying messages. Below it is a 'Mensaje:' text input field and a 'Destinatario:' dropdown menu, with an 'Enviar' button positioned to the right.

4 RESULTADOS Y ANÁLISIS

En esta sección, se presentan los resultados obtenidos durante la implementación del protocolo SIP en los módulos Wi-Fi y GSM. Se procederá a la realización de las diferentes pruebas propuestas en el punto 3.2 utilizando los escenarios especificados en ese mismo punto y el código implementado según el punto 3.3

4.1 Implementación del protocolo SIP en el módulo Wi-Fi. Bloque I de pruebas

Durante las pruebas del primer bloque, se verificará la implementación del cliente SIP utilizando el módulo Wi-Fi. Esto incluye la prueba de la comunicación serie con el módulo. Para llevar a cabo estas pruebas, se requerirá tener desarrollado previamente unos requisitos.

Requisitos previos.

Tal y como se indicó en el punto 3.2 hay que tener desarrollado ciertas partes del código, las cuáles se pondrán a prueba. Estas son:

- Clase ClienteSIP, referente a la implementación del cliente SIP.
- Interfaz ComSerie
- Clase ComSerieEsp. Esta clase será utilizada tanto para probar su correcto funcionamiento como para comprobar el funcionamiento adecuado de la clase Sip al conectarse a un servidor externo.

En lo relativo al escenario es necesario tener instalado:

- El servidor Kamailio: Es instalado en una máquina virtual Ubuntu 18. La IP asignada en la red local es 192.168.1.113. El servicio corre en el puerto estandarizado para las conexiones SIP, concretamente en el puerto 5060. Se han creado los diferentes usuarios para la realización de pruebas:
 - o Usuario: usersip. Este usuario será el que iniciará sesión desde el módulo Wifi. La contraseña asociada es passip
 - o Usuario userprueba: Este usuario iniciará sesión desde MicroSip y será utilizado para probar el intercambio de mensajes. Su contraseña es passprueba
- WireShark instalado en la máquina de Ubuntu junto con Kamailio para comprobar el intercambio de mensajes.
- MicroSIP, instalado en el mismo equipo que ejecuta el código a probar, u otro. Se ha iniciado sesión con el usuario UserPrueba.

Finalmente, se ha desarrollado un script dedicado específicamente a las pruebas, en el cual se utilizan todas las funcionalidades específicas de cada clase. A continuación, se describirá en detalle cada parte de este script,

indicando las pruebas que se realizan y los resultados obtenidos en cada una de ellas. Esto nos permitirá evaluar exhaustivamente el funcionamiento y rendimiento de las clases implementadas.

Comprobación de la conexión serie con ESP-8266

En la primera parte del Script se establece conexión con el dispositivo y se conecta a la red Wifi. Para comprobar el buen funcionamiento, se mostrarán los comandos enviados al dispositivo así como las respuestas recibidas de este.

En nuestro caso, el dispositivo está conectado al puerto COM3. Por ello el inicio del script de prueba es:

```
serial = ComSerieESP('COM3')
serial.comando("AT+CIPCLOSE=5")
serial.set_modo(1)
conexWifi = serial.conectar_AP("nombreWifi","passwifi132456789")
print("Variable conexWifi" + |conexWifi|)
if conexWifi:
```

conexWifi indicará si la conexión se ha producido correctamente. El resto del script está condicionado a que esta variable tome el valor True.

El resultado de ejecutar esta primera parte del script de prueba es:

```
ATE0
AT+CIPCLOSE=5
AT+CWMODE=1
OK
AT+CWJAP="nombreWifi","passwifi132456789"
AT+CWJAP="nombreWifi","passwifi132456789"
WIFI DISCONNECT
WIFI CONNECTED
WIFI GOT IP
OK
Variable conexWifi: True
```

Se puede comprobar que los comandos enviados son correctos y que, al enviar la petición de conexión a la red Wifi, el módulo obtiene una dirección IP. Además, la variable ConexWifi indica que la operación se ha realizado correctamente. Todo esto puede comprobarse en la configuración del punto acceso wifi donde podemos observar que el dispositivo está conectado y la IP que tiene asignada.



Configura tu dispositivo

Nombre

Tipo

Dirección IP

Una vez el módulo se ha conectado a la red local podemos establecer conexión con el servidor Kamailio. En el siguiente fragmento del script de pruebas se procede a abrir la conexión UDP, obtener la dirección IP y activar el modo de transmisión continua:

```
if conexwifi:
    print("Presione una tecla para continuar...")
    msvcrt.getch()
    print(serial.conectar("UDP", "192.168.1.113", 5060))
    ip_local=serial.obtener_IP()
    print(ip_local)
    serial.transmision_continua()
    print("Presione una tecla para continuar...")
    msvcrt.getch()
```

El resultado de la ejecución es:

```
Variable conexwifi: True
Presione una tecla para continuar...
AT+CIPSTART="UDP", "192.168.1.113", 5060

True
AT+CIFSR

+CIFSR:STAIP, "192.168.1.51"
+CIFSR:STAMAC, "84:0d:8e:88:f6:82"

OK

192.168.1.51
AT+CIPMODE=1

AT+CIPSEND

AT+CIPMODE=1

OK

AT+CIPSEND

OK

Presione una tecla para continuar...
```

Después de establecer la conexión (CIPSTART) y recibir una respuesta exitosa, se obtiene la dirección IP asignada al módulo. En este caso, se ha decidido obtener la IP después de realizar la conexión UDP para evitar

posibles problemas debido a retrasos en la negociación DHCP. De esta manera, nos aseguramos de que el módulo ha completado exitosamente la negociación DHCP y tiene una IP asignada. Además, al enviar los comandos de cambio al modo de transmisión transparente (CIPMODE=1) y de inicio de la transmisión (CIPSEND), se recibe la respuesta "OK". Esto indica que a partir de este momento, cualquier dato transmitido al módulo a través de la comunicación serie será retransmitido al servidor SIP, y cualquier dato recibido del dispositivo provendrá del servidor. Esta funcionalidad permite una comunicación bidireccional efectiva entre el cliente SIP y el servidor, asegurando la correcta transmisión de datos. Con estos resultados, se asegura que el módulo está correctamente configurado y preparado para establecer conexiones y realizar transmisiones en el contexto del protocolo SIP.

Para la creación del objeto Sip se facilita la IP obtenida anteriormente, así como el dominio (en este caso IP), puerto usuario y contraseña. Además es necesario facilitar una función encargada de procesar los mensajes SIP recibidos de otros usuarios. Para estas pruebas se ha propuesto la siguiente función:

```
def mensaje_nuevo(origen, mensaje):
    print("~~~~~")
    print("~~~~~NUEVO MENSAJE~~~~~")
    print("ORIGEN:      "+origen)
    print()
    print(mensaje)
    print("~~~~~")
    print("~~~~~")
```

La creación de un objeto SIP no supone la realización de ninguna petición al servidor, por ello una vez obtenemos la instancia se creará la petición de registro. El método registro devolverá la cabecera SIP de la petición de registro, que se mostrará por pantalla:

```
sesion_sip = sip.ClienteSIP(ip_local,"usersip","192.168.1.113","passip",False,mensaje_nuevo,aviso,serial.transmitir_cta,False,"5060")
reg = sesion_sip.sip_registro(60)
print(reg)
```

Esta parte del código muestra:

```
Presione una tecla para continuar...
REGISTER sip:192.168.1.113:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.51;rport
Max-Forwards: 70
Contact: <sip:usersip@192.168.1.51>
To: "-"<sip:usersip@192.168.1.113:5060>
From: "-"<sip:usersip@192.168.1.113:5060>
Call-ID: 3c842141a5a64e8d9de579033e5fbfcb
CSeq: 1 REGISTER
Expires: 60
Allow: NOTIFY, INVITE, ACK, CANCEL, BYE, REFER, INFO, OPTIONS, MESSAGE
User-Agent: fracrusan
Content-Length: 0
```

Que corresponde con la cabecera de la petición sip para el registro del usuario usersip en el servidor accesible desde 192.168.1.113:5060

Antes de enviar la petición al servidor, es necesario crear el hilo dedicado a recibir los mensajes entrantes provenientes del servidor. Esta configuración permite verificar el correcto funcionamiento del sistema, ya que se mostrarán todos los mensajes recibidos del servidor, así como las respuestas que se envíen automáticamente. Además, para un mayor análisis y control, se utilizará WireShark en el equipo servidor para capturar y examinar los paquetes de la comunicación SIP. Esta herramienta proporciona una visión detallada de la interacción entre el cliente y el servidor, permitiendo detectar posibles problemas o anomalías en la comunicación. La sección del script de prueba responsable de esta funcionalidad es la siguiente:

```

serial.crear_hilo(sesion_sip)
serial.transmitir_cta(reg)

while(sesion_sip.estado != "Conectado"):
    time.sleep(1)
    print(sesion_sip.estado)

print("Inicio de sesión realizado")

```

Después de enviar el registro, el script se bloquea hasta que el estado del objeto Sip cambie a "Conectado", lo cual ocurre cuando se recibe la respuesta "200 OK" a la solicitud de registro. Una vez que esto sucede, la ejecución del script continúa.

La ejecución de este fragmento provoca el envío de la petición y la respuesta del servidor, que puede observarse en Wireshark:

168	34.068012522	192.168.1.51	192.168.1.113	SIP	449 Request: REGISTER sip:192.168.1.113:5060 (1 binding)
169	34.069546030	192.168.1.113	192.168.1.51	SIP	472 Status: 401 Unauthorized
170	34.070817537	192.168.1.113	192.168.1.51	SIP	472 Status: 401 Unauthorized
173	35.408999994	192.168.1.51	192.168.1.113	SIP	644 Request: REGISTER sip:192.168.1.113:5060 (1 binding)
174	35.411674710	192.168.1.113	192.168.1.51	SIP	420 Status: 200 OK (1 binding)
177	36.663112614	192.168.1.51	192.168.1.113	SIP	644 Request: REGISTER sip:192.168.1.113:5060 (1 binding)
178	36.665289828	192.168.1.113	192.168.1.51	SIP	420 Status: 200 OK (1 binding)

El servidor recibe un paquete Register desde la IP correspondiente al módulo. La respuesta a dicho paquete es 401 Unauthorized. En la cabeza de dicho paquete se incluye la cabecera de autenticación Digest:

```

▶ Frame 169: 472 bytes on wire (3776 bits), 472 bytes captured (3776 bits) on interface 0
▶ Ethernet II, Src: Vmware_57:03:9e (00:0c:29:57:03:9e), Dst: Espressi_88:f6:82 (84:0d:8e:88:f6:82)
▶ Internet Protocol Version 4, Src: 192.168.1.113, Dst: 192.168.1.51
▶ User Datagram Protocol, Src Port: 5060, Dst Port: 25968
▶ Session Initiation Protocol (401)
  ▼ Status-Line: SIP/2.0 401 Unauthorized
    Status-Code: 401
    [Resent Packet: False]
    [Request Frame: 167]
    [Response Time (ms): 1]
  ▼ Message Header
    ▶ Via: SIP/2.0/UDP 192.168.1.51;rport=25968;received=192.168.1.51
    ▶ To: "-< sip:usersip@192.168.1.113:5060>;tag=9dd61ff61e802d8e2bef5f14621ef3c2.a350"
    ▶ From: "-< sip:usersip@192.168.1.113:5060>"
    Call-ID: 7558d97550e3a1e0f244ad6c56b46832
    ▶ CSeq: 1 REGISTER
    ▶ WWW-Authenticate: Digest realm="192.168.1.113", nonce="ZiZ01GSMzWg7GwbvtAoVwnzj63yS4gV2"
    Server: kamailio (5.1.2 (x86_64/linux))
    Content-Length: 0

```

Ante esta petición (recogida por el hilo creado) se responde de forma automática con otra petición Register, pero incluyendo esta vez el campo authorization con el valor correspondiente:

```

▶ Frame 173: 644 bytes on wire (5152 bits), 644 bytes captured (5152 bits) on interface 0
▶ Ethernet II, Src: Espressi_88:f6:82 (84:0d:8e:88:f6:82), Dst: Vmware_57:03:9e (00:0c:29:57:03:9e)
▶ Internet Protocol Version 4, Src: 192.168.1.51, Dst: 192.168.1.113
▶ User Datagram Protocol, Src Port: 25968, Dst Port: 5060
▶ Session Initiation Protocol (REGISTER)
  ▼ Request-Line: REGISTER sip:192.168.1.113:5060 SIP/2.0
    Method: REGISTER
    ▶ Request-URI: sip:192.168.1.113:5060
    [Resent Packet: False]
  ▼ Message Header
    ▶ Via: SIP/2.0/UDP 192.168.1.51;rport
    Max-Forwards: 70
    ▶ Contact: < sip:usersip@192.168.1.51>
    ▶ To: "-< sip:usersip@192.168.1.113:5060>"
    ▶ From: "-< sip:usersip@192.168.1.113:5060>"
    Call-ID: 7558d97550e3a1e0f244ad6c56b46832
    ▶ CSeq: 2 REGISTER
    Expires: 60
    Allow: NOTIFY, INVITE, ACK, CANCEL, BYE, REFER, INFO, OPTIONS, MESSAGE
    ▶ Authorization: Digest username="usersip", realm="192.168.1.113", nonce="ZiZ01GSMzWg7GwbvtAoVwnzj63yS4gV2", uri="sip:usersip@192.168.1.113", response="4c25845aa060e65c22a219682adeaba0", algorithm=MD5
    User-Agent: Tracusan
    Content-Length: 0

```

Ante esta segunda petición, el servidor envía la respuesta "200 OK", que hace cambiar el estado del objeto Sip a "Conectado":

```
Content: 0
~~~~~RECEPCION~~~~~
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.51;rport=25968;received=192.168.1.51
To: ""<sip:usersip@192.168.1.113:5060>;tag=9dd61ff61e802d8e2bef5f14621ef3c2.a350
From: ""<sip:usersip@192.168.1.113:5060>
Call-ID: 7558d97550e3a1e0f244ad6c56b46832
CSeq: 2 REGISTER
Contact: <sip:usersip@192.168.1.51>;expires=60
Server: kamailio (5.1.2 (x86_64/linux))
Content-Length: 0

Conectado
Inicio de sesión realizado
Presione una tecla para proceder al envío de mensajes SIP...
```

Cada 60 segundos (tal y como se ha indicado en el argumento del método de registro) el sistema enviará una petición de registro al servidor para evitar que la sesión caduque. Puede comprobarse en la siguiente captura de paquetes:

59	13.952918913	192.168.1.51	192.168.1.113	SIP	856 Request: REGISTER sip:192.168.1.113:5060 (1 bir
275	68.674789978	192.168.1.51	192.168.1.113	SIP	449 Request: REGISTER sip:192.168.1.113:5060 (1 bir
276	68.783902184	192.168.1.113	192.168.1.51	SIP	472 Status: 401 Unauthorized
279	69.391297193	192.168.1.51	192.168.1.113	SIP	644 Request: REGISTER sip:192.168.1.113:5060 (1 bir
280	69.417552616	192.168.1.113	192.168.1.51	SIP	420 Status: 200 OK (1 binding)
430	123.6371445...	192.168.1.51	192.168.1.113	SIP	449 Request: REGISTER sip:192.168.1.113:5060 (1 bir
431	123.6429970...	192.168.1.113	192.168.1.51	SIP	472 Status: 401 Unauthorized
432	124.2494912...	192.168.1.51	192.168.1.113	SIP	644 Request: REGISTER sip:192.168.1.113:5060 (1 bir

El siguiente paso es el envío de mensajes SIP. Los pasos a seguir son los mismos, primero hay que crear el contenido del paquete SIP mediante una llamada al método correspondiente de la instancia SIP y, a continuación, transmitir dichos datos mediante el módulo. Puede verse en el script de prueba:

```
print("Presione una tecla para proceder al envío de mensajes SIP...")
msvcrt.getch()
print("Enviamos mensjae")
encoded_string = sesion_sip.sip_mensaje("userprueba","Mensaje de prueba")
serial.transmitir_cta(encoded_string)
while True:
    time.sleep(60)
    encoded_string = sesion_sip.sip_mensaje("userprueba","Mensaje de prueba de ESP")
    serial.transmitir_cta(encoded_string)
```

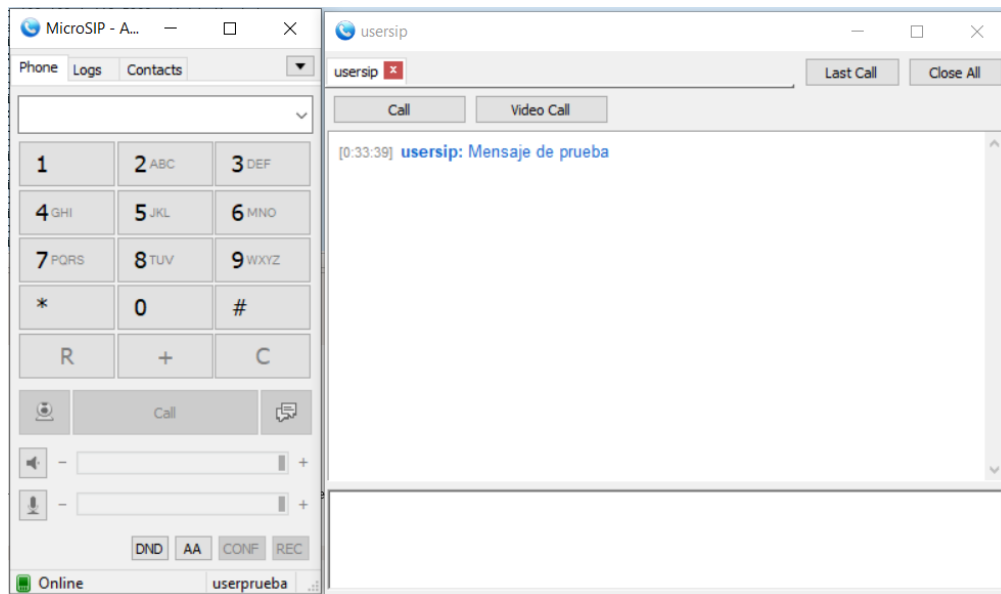
La ejecución genera el siguiente traspaso de mensajes:

379	96.150381812	192.168.1.51	192.168.1.113	SIP	454 Request: MESSAGE sip:userprueba@192.168.1.113 (text/plain)
380	96.150953622	192.168.1.113	192.168.1.51	SIP	486 Status: 407 Proxy Authentication Required
381	96.661878356	192.168.1.51	192.168.1.113	SIP	644 Request: MESSAGE sip:userprueba@192.168.1.113 (text/plain)
384	96.665242019	192.168.1.113	192.168.1.51	SIP	332 Status: 200 OK

Al igual que con la petición Register es necesaria la autorización Digest, por ello el sistema desarrollado reenvía la solicitud con los campos necesarios. El segundo paquete MESSAGE contiene:

```
Session Initiation Protocol (MESSAGE)
Request-Line: MESSAGE sip:userprueba@192.168.1.113 SIP/2.0
Message Header
  Via: SIP/2.0/UDP 192.168.1.51;rport
  Max-Forwards: 70
  To: <sip:userprueba>;message-type=IM
  From: ""<sip:usersip@192.168.1.113:5060>
  Call-ID: 9d08632d121661b37901579baf2aa32a
  CSeq: 2 MESSAGE
  Allow: SUBSCRIBE, NOTIFY, INVITE, ACK, CANCEL, BYE, REFER, INFO, OPTIONS, MESSAGE
  Content-Type: text/plain
  Proxy-Authorization: Digest username="usersip", realm="192.168.1.113", nonce="ZiZkmsM40L7uIo1PnLR6ibb6R+LqIn", uri="sip:userprueba", response="e350ed7b45d0aa517792697f29287bea", algorithm=MD5
  User-Agent: fracrusan
  Content-Length: 17
Message Body
  Line-based text data: text/plain (1 lines)
  Mensaje de prueba
```

El mensaje ha sido recibido correctamente en el software MicroSIP donde se ha iniciado sesión con el usuario userprueba:



Por último queda comprobar la recepción de mensajes, para ello, desde microSIP haremos el envío de un mensaje de prueba. El traspaso de mensajes es:

227	103.0759331...	192.168.1.113	192.168.1.51	SIP	578 Request: MESSAGE sip:usersip@192.168.1.51 (text/plain)
228	103.0972194...	192.168.1.51	192.168.1.113	ICMP	70 Destination unreachable (Port unreachable)
239	107.0765680...	192.168.1.113	192.168.1.51	SIP	578 Request: MESSAGE sip:usersip@192.168.1.51 (text/plain)
241	107.0891144...	192.168.1.51	192.168.1.113	ICMP	70 Destination unreachable (Port unreachable)
247	111.0766085...	192.168.1.113	192.168.1.51	SIP	578 Request: MESSAGE sip:usersip@192.168.1.51 (text/plain)

Se puede observar que ha habido errores, además en la salida del script de pruebas no se ha obtenido ningún mensaje. El error que envía el módulo ante la petición de mensajes del servidor es “Port unreachable”, por ello vamos a observar la cabecera de la petición del servidor:

```

▶ Frame 227: 578 bytes on wire (4624 bits), 578 bytes captured (4624 bits) on interface 0
▶ Ethernet II, Src: Vmware_57:03:9e (00:0c:29:57:03:9e), Dst: Espressi_88:f6:82 (84:0d:8e:88:f6:82)
▶ Internet Protocol Version 4, Src: 192.168.1.113, Dst: 192.168.1.51
▶ User Datagram Protocol, Src Port: 5060, Dst Port: 5060
▼ Session Initiation Protocol (MESSAGE)
  ▶ Request-Line: MESSAGE sip:usersip@192.168.1.51 SIP/2.0
  ▼ Message Header
    ▶ Via: SIP/2.0/UDP 192.168.1.113;branch=z9hG4bK5647.eb921556d0522b5b6d560d71d60238ed.0
    ▶ Via: SIP/2.0/UDP 192.168.1.100:54358;received=192.168.1.100;rport=54358;branch=z9hG4bKPj051420841d45484da294400630a5d2af
    Max-Forwards: 69
    ▶ From: <sip:userprueba@192.168.1.113>;tag=c5cad646e1164d7595ef924660dd2cb8
    ▶ To: <sip:usersip@192.168.1.113>
    Call-ID: 833f60502739489ba9e770aa56fde473
    ▶ CSeq: 11929 MESSAGE
    User-Agent: MicroSIP/3.20.7
    Content-Type: text/plain
    Content-Length: 16
  ▼ Message Body
    ▼ Line-based text data: text/plain (1 lines)
      Envío de mensaje

```

En la cabecera UDP vemos que el puerto destino es 5060. Esto responde al funcionamiento normal del protocolo SIP, cada cliente debe estar a la escucha en el puerto 5060 (por defecto) para la recepción de peticiones. En nuestro caso hay dos opciones:

- Escuchar en el puerto 5060 para recibir peticiones.
- Modificar la petición de registro para incluir el puerto en el que se está escuchando las solicitudes y respuestas. Este puerto corresponde al que se ha abierto durante la conexión con el servidor.

La clase SIP está diseñada para poder realizar la segunda opción mencionada, ya que el último parámetro de la petición de registro corresponde al puerto específico en el que se recibirán las solicitudes. Sin embargo, es necesario determinar mediante comandos AT el puerto utilizado por el módulo para establecer la conexión. Es importante tener en cuenta que este enfoque solo sería válido para este escenario en particular. En el caso de que el módulo Wi-Fi se conecte a internet a través de un NAT (Traducción de Direcciones de Red), no sería posible determinar el puerto asignado por el NAT cuando el servidor no se encuentre en la red local. Además, tampoco sería factible abrir un puerto específico en el NAT para la recepción de solicitudes. Por esta razón, en el escenario final se utiliza Asterisk, una solución que permite configurar el puerto de envío de solicitudes de manera que coincida con el puerto que el cliente utiliza para la conexión principal, utilizando una única conexión.

Por ello, la recepción de mensajes será probada en el siguiente bloque de pruebas donde se utilizará un servidor Asterisk fuera de nuestra red local.

4.1 Implementación del protocolo SIP en ambos módulos. Bloque II de pruebas.

En el bloque de pruebas anterior se comprobó el buen funcionamiento del cliente SIP desarrollado en un entorno de pruebas que difiere del que nos encontramos en la realidad. Por ello, en este nuevo bloque de pruebas crearemos un escenario más realista que difiere del escenario anterior. El servidor SIP se alojará en un equipo externo, no en nuestra red local externa. Para acceder a él podremos usar su IP pública, o bien, mediante el dominio spinar.es. Para este caso usaremos Asterisk que lo configuraremos en los siguientes aspectos:

- Debido a que algunos ISP utilizan cortafuegos en sus redes que impiden el tráfico en determinados puertos, el servicio de Asterisk atenderá peticiones en el puerto 50060. De esta forma, para la comunicación habrá que conectarse al puerto 50060 UDP del dominio spinar.es
- Los clientes podrán conectarse a la red a través de NAT. De esta forma el servidor utilizará únicamente la conexión que ha abierto el cliente para la comunicación y no usará otros puertos. Además también se bloquea la comunicación entre clientes.
- Se utilizará numeración para identificar clientes, con el fin de dar realismo al escenario. Se han creado los siguientes usuarios:
 - o 2108: Este usuario será usado por el módulo wifi.
 - o 0510: Este usuario será utilizado por el módulo GSM.
 - o 0602: MicroSip iniciará sesión con este usuario.
- Asterisk estará alojado un servidor Debian sin entorno gráfico. Por ello no utilizaremos Wireshark para comprobar el intercambio de paquetes. Este intercambio ya se ha probado en el módulo anterior. Para poder obtener una mayor información del sistema se ejecutará Asterisk en modo debug.

Este bloque de pruebas se dividirá en la comprobación de los diferentes aspectos del sistema. Cada uno de estos aspectos son:

- Implementación del cliente Sip en el módulo ESP 8266. Se realizará de nuevo las pruebas del primer bloque, pero utilizando el nuevo servidor. Además, se confirmará el buen funcionamiento de la recepción de mensajes.
- Implementación del cliente Sip en el dispositivo MC60. Se utilizará un script de pruebas muy similar al utilizado en el caso anterior para comprobar que todas las funciones se realizan de forma correcta.
- Interfaz gráfica: Una vez se ha probado que las funcionalidades del cliente sip se ejecutan correctamente en ambos dispositivos, se procederá a utilizarlas a través de la interfaz gráfica.

Implementación del cliente SIP en el módulo ESP8266

Para la realización de este conjunto de pruebas deberá estar desarrollado:

- La clase Sip, correspondiente a la implementación del cliente Sip,
- La clase ComSerieESP, utilizada para la comunicación con el módulo wifi.
- El script utilizado en las pruebas anteriores, modificando el dominio y el puerto de acceso.

Procedemos a ejecutar el script modificando el dominio, puerto y credenciales de inicio de sesión. Para este caso se utilizará el usuario 2108. Los mensajes tendrán como destinatario el usuario 0602.

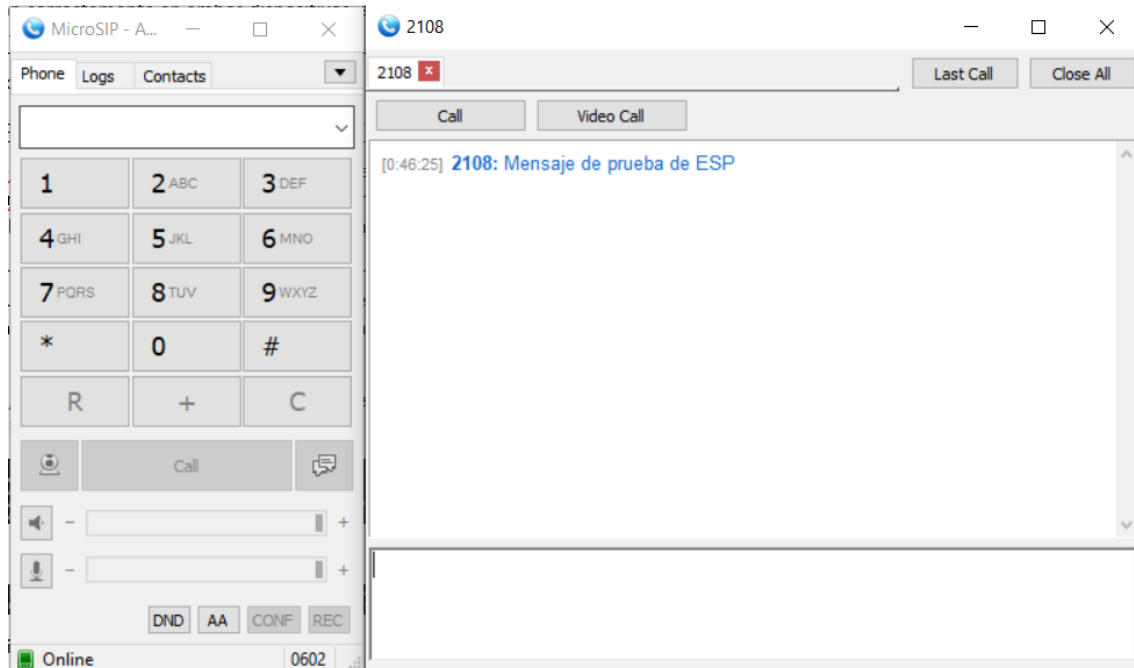
Al ejecutar el script, Asterisk notifica el inicio de sesión del usuario 2108 con 60 segundos de caducidad de la sesión:

```
*CLI> -- Added contact 'sip:2108@00.31.91.131:41319;x-ast-orig-host=192.168.1.51:0' to AOR '2108' with expiration of 60 seconds
== Endpoint 2108 is now Reachable
-- Executing [0602@mensaje:1] NoOp("Message/ast_msg_queue", "Mensaje de "-" <sip:2108@spinar.es>") in new stack
-- Executing [0602@mensaje:2] NoOp("Message/ast_msg_queue", "Mensaje para pjsip:0602") in new stack
-- Executing [0602@mensaje:3] MessageSend("Message/ast_msg_queue", "pjsip:0602,-" <sip:2108@spinar.es>") in new stack
```

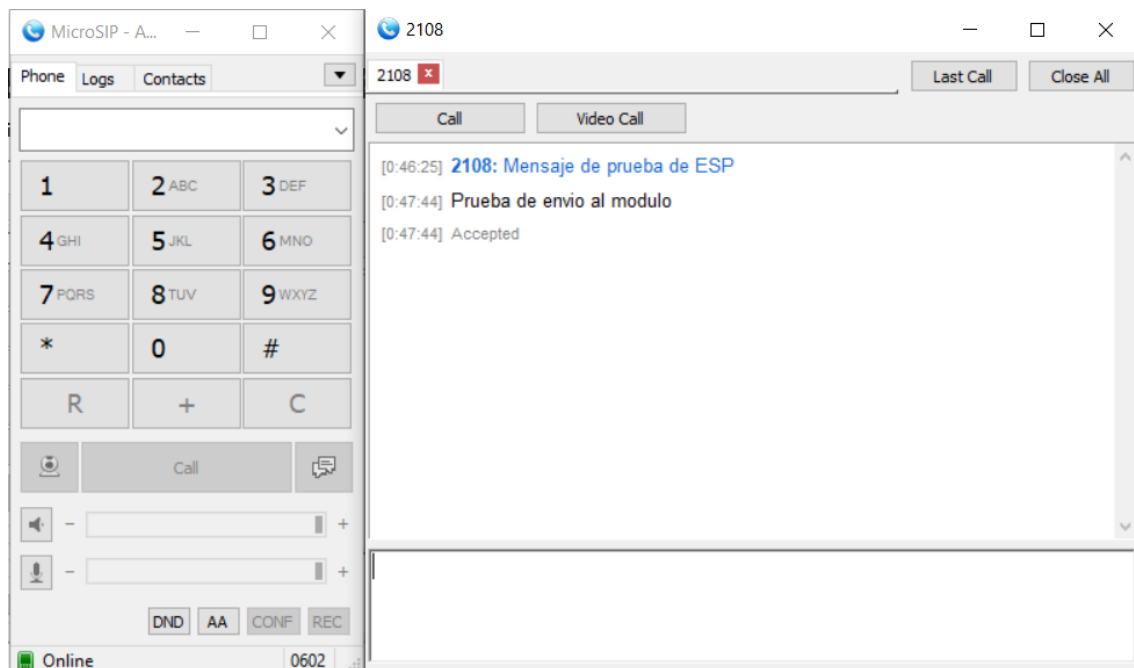
Cuando se realiza el envío del mensaje SIP, Asterisk también lo notifica de forma correcta:

```
[Jun 18 22:42:15] WARNING[227809]: res_pjsip_messaging.c:719 update_to_uri: To address '0602' is not a valid SIP/SIPS URI
-- Executing [0602@mensaje:4] Hangup("Message/ast_msg_queue", "16") in new stack
```

El mensaje es recibido de forma correcta en microSIP:



Por último, se comprobará el envío de un mensaje a 2108, funcionalidad que no se pudo probar en el bloque anterior. Para ello desde MicroSip se procede al envío del mensaje:



Al hacerlo, en la consola de Asterisk se recibe el siguiente aviso:

```
[Jun 18 22:47:44] WARNING[227809]: res_pjsip_messaging.c:719 update_to_uri: To address '2108@spinar.es' is not a valid SIP/SIPS URI
-- Executing [2108@mensaje:4] Hangup("Message/ast_msg_queue", "16") in new stack
```

Indicando que se ha recibido la petición de envío de mensaje y que se ha tramitado.

Por último, la ejecución del script de pruebas muestra por pantalla:

```
~~~~~
~~~~~NUEVO MENSAJE~~~~~
ORIGEN: <sip:0602@spinar.es>

Prueba de envio al modulo
~~~~~
~~~~~
```

Mensaje correspondiente al codificado en la función creada para la recepción de mensajes. Con esta comprobación toda la implementación del cliente SIP en ESP-8266 queda probada de forma satisfactoria.

Implementación del cliente SIP en el módulo MC60

Esta prueba consistirá en la realización de las mismas acciones que en el caso anterior, pero utilizando el dispositivo MC60. Para la realización de estas pruebas deberá estar desarrollado:

- Implementación del cliente Sip.
- Clase ComSerieGSM, utilizada para controlar la comunicación con el dispositivo
- Scripts de pruebas. Se utilizará un script de pruebas muy similar al caso anterior. Debido a que tanto ComSerieGSM como ComSerieGSM implementan la interfaz ComSerie los cambios a realizar serán mínimos. Únicamente se suprimirá las partes relativas a la configuración de la conexión Wi-Fi.

Al instanciar el objeto serie y conectarse al dispositivo se ejecutan automáticamente los comandos AT necesarios para la configuración del módulo. Por ello, se conectará al servidor SIP directamente. Una vez conectado, obtendremos la dirección IP asignada al módulo.

Para el inicio de sesión SIP se utilizará el usuario 0510. El procedimiento es igual que en el caso anterior, ya que aunque no se use el modo de transmisión continua se ha simulado su funcionamiento. También se mantiene la función encargada de mostrar por pantalla los mensajes recibidos.

Los resultados obtenidos durante la ejecución del script de pruebas son los siguientes:

Tras la configuración y conexión al servidor se obtiene la IP del módulo:

```
b'OK\r\n'
AT+QIDNSIP=1

b'OK\r\n'
AT+QIOPEN="UDP","spinar.es",50060"

b'OK\r\n'
b'CONNECT OK\r\n'
True
AT+QILOCIP

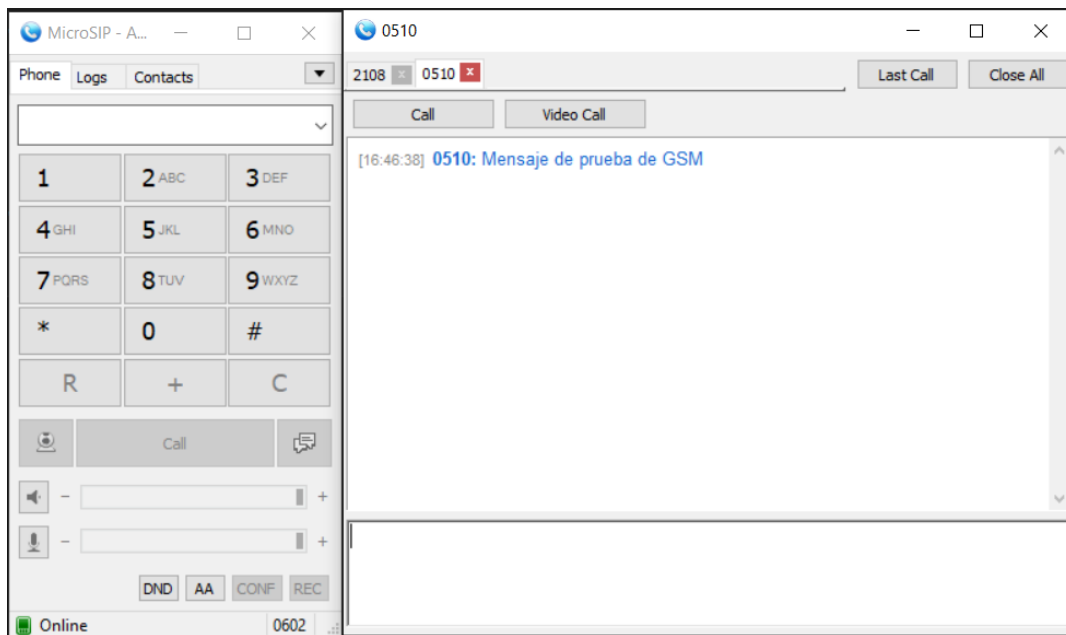
10.211.11.35
AT+QIPROMPT=2
```

A continuación, se procede con el registro del usuario y el envío de mensajes. La consola de Asterisk muestra lo siguiente:

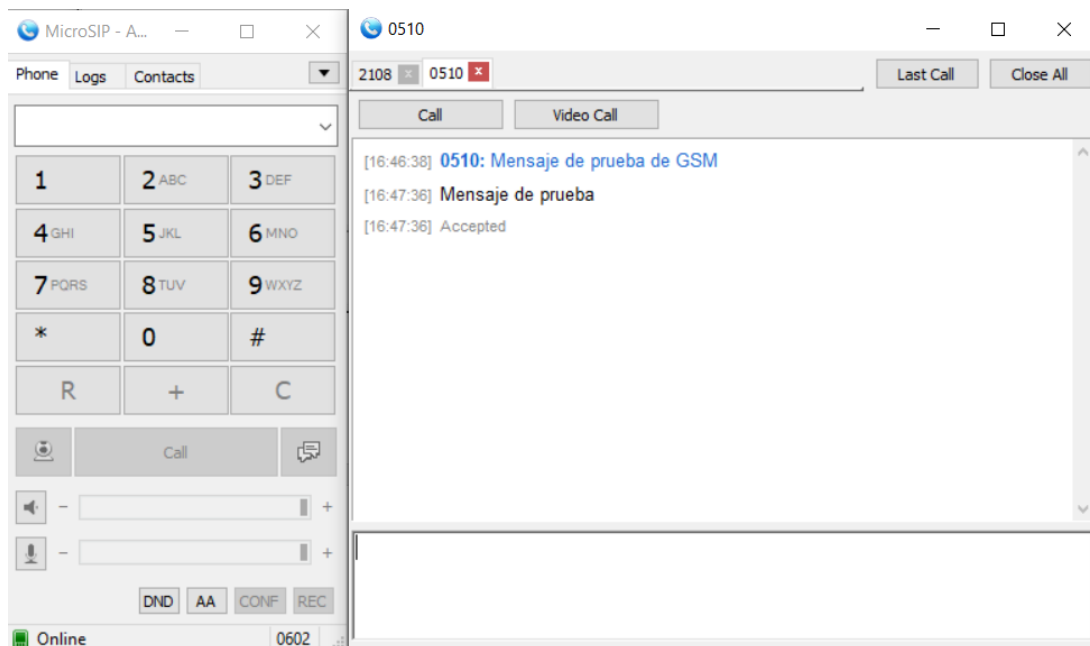
```
-- Added contact 'sip:0510@31.4.223.31:48424;x-ast-orig-host=10.211.11.35:0' to AOR '0510' with expiration of 60 seconds
== Endpoint 0510 is now Reachable
-- Executing [0602@mensaje:1] NoOp("Message/ast_msg_queue", "Mensaje de "-" <sip:0510@spinar.es>") in new stack
-- Executing [0602@mensaje:2] NoOp("Message/ast_msg_queue", "Mensaje para pjsip:0602") in new stack
-- Executing [0602@mensaje:3] MessageSend("Message/ast_msg_queue", "pjsip:0602,-" <sip:0510@spinar.es>") in new stack
[Jun 19 14:46:38] WARNING[244364]: res_pjsip_messaging.c:719 update_to_uri: To address '0602' is not a valid SIP/SIPS URI
-- Executing [0602@mensaje:4] Hangup("Message/ast_msg_queue", "16") in new stack
```

Donde puede observarse que se ha iniciado sesión con el usuario 0510 desde la IP que ha sido asignada al módulo. A continuación, aparece la petición de envío de mensaje, que es procesada correctamente. Además, el

mensaje ha sido correctamente recibido por 0602:



Por último, desde esta propia interfaz de MicroSip se procede al envío de un mensaje:



La petición es recibida y procesada por Asterisk:

```
[Jun 19 14:47:36] WARNING[244364]: res_pjsip_messaging.c:719 update_to_uri: To address '0510@spinar.es' is not a valid SIP/SIPS URI
-- Executing [0510@mensaje:4] Hangup("Message/ast_msg_queue", "16") in new stack
```

Al llegar al módulo, se ejecuta la función que se desarrolló para manejar la recepción de mensajes, mostrando por la consola:

```
~~~~~
~~~~~NUEVO MENSAJE~~~~~
ORIGEN: <sip:0602@spinar.es>
Mensaje de prueba
~~~~~
~~~~~
```

Se ha experimentado un error: la recepción duplicada del mismo mensaje. El primer paquete que recibe el

módulo es:

```
COMSERIE 165 ~~~~~
MESSAGE sip:0510@31.4.223.31:48424 SIP/2.0
Via: SIP/2.0/UDP 212.227.41.212:50060;rport;branch=z9hG4bKPj77971f7b-5693-470c-8d3d-fd649e1d73d7
From: <sip:0602@spinar.es>;tag=bc6b0d95-6cfe-453b-b8f5-803c0e757c7f
To: <sip:0510@31.4.223.31>
Contact: <sip:0510@212.227.41.212:50060>
Call-ID: bda8a009-68a6-4865-bb63-1e4b746161ed
CSeq: 27864 MESSAGE
Max-Forwards: 70
User-Agent: Asterisk PBX 20.1.0
Content-Type: text/plain
Content-Length: 17
```

Este mensaje se procesa correctamente, enviando la respuesta asertiva:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.211.11.35;received=192.168.1.111;rport;branch=z9hG4bKPj77971f7b-5693-470c-8d3d-fd649e1d73d7
Max-Forwards: 70
Contact: <sip:0510@10.211.11.35>
To: <sip:0510@31.4.223.31>
From: <sip:0602@spinar.es>;tag=bc6b0d95-6cfe-453b-b8f5-803c0e757c7f
Call-ID: bda8a009-68a6-4865-bb63-1e4b746161ed
CSeq: 27864 MESSAGE
User-Agent: fracrusan
Content-Length: 0
```

Aun así, tras unos segundos se recibe de nuevo el paquete MESSAGE:

```
MESSAGE sip:0510@31.4.223.31:48424 SIP/2.0
Via: SIP/2.0/UDP 212.227.41.212:50060;rport;branch=z9hG4bKPj77971f7b-5693-470c-8d3d-fd649e1d73d7
From: <sip:0602@spinar.es>;tag=bc6b0d95-6cfe-453b-b8f5-803c0e757c7f
To: <sip:0510@31.4.223.31>
Contact: <sip:0510@212.227.41.212:50060>
Call-ID: bda8a009-68a6-4865-bb63-1e4b746161ed
CSeq: 27864 MESSAGE
Max-Forwards: 70
User-Agent: Asterisk PBX 20.1.0
Content-Type: text/plain
Content-Length: 17

Mensaje de prueba
```

Este mensaje se procesa de igual forma que el anterior. Puede observarse que es el mismo mensaje pues coincide el número de secuencia. Este número de secuencia debe ser único para cada transacción³³. En la consola de Asterisk no aparece que se haya tenido que hacer reenvíos, el último mensaje que aparece es el mostrado anteriormente, sin duplicidades:

```
[Jun 19 14:47:36] WARNING[244364]: res_pjsip_messaging.c:719 update_to_urt: To address '0510@spinar.es' is not a valid SIP/SIPS URI
-- Executing [0510@mensaje:4] Hangup("Message/ast_msg_queue", "16") in new stack
```

Este mensaje, es igual que en el caso del módulo ESP-8266 donde no se produce la duplicidad.

Además, se puede comprobar que el mensaje de respuesta 200 elaborado es correcto e idéntico al enviado en las pruebas anteriores con el módulo Wi-Fi

Por todo ello, Para solucionar este error puntual, que se produce algunas veces y únicamente con este módulo se ha decidido filtrar los mensajes según el número de secuencia. Si el mensaje con ese número ha sido procesado anteriormente no se procesará.

Después de realizar el cambio mencionado, el mensaje se mostrará una única vez. Esto se debe a que la función encargada de mostrar el mensaje solo se llama la primera vez que se recibe un número de secuencia. Si el número de secuencia se recibe varias veces, solo se enviará la respuesta "200 OK" al servidor. De esta manera, se evita la repetición innecesaria del mensaje y se asegura que solo se envíe una confirmación al servidor para indicar que el mensaje ha sido recibido correctamente.

Para lograr esto, se realizó una modificación en la clase ClienteSIP. Se agregó un atributo llamado

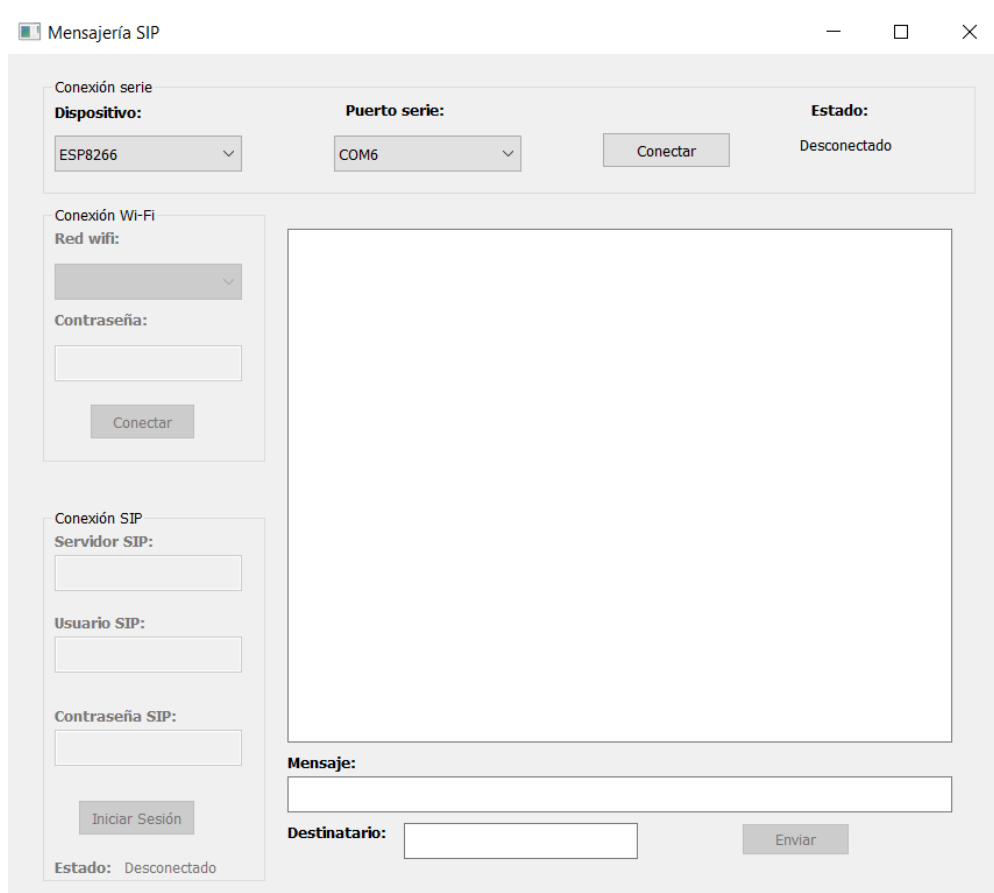
³³ RFC 3291. Apartado 20.16

cseq_history, que es una lista. Cada vez que se recibe un mensaje, se extrae el número de secuencia y se verifica si ya existe en la lista. Si no existe, se llama a la función encargada de procesar el mensaje y se agrega el número de secuencia a la lista cseq_history. En cualquier caso, se prepara y envía la respuesta correspondiente al servidor.

Pruebas con sistema de interfaz gráfica

Una vez se ha verificado el correcto funcionamiento de las diferentes funcionalidades del sistema, se dará paso a la prueba de la interfaz gráfica. Esta etapa permitirá evaluar la usabilidad y la interacción del usuario con la aplicación. Se probará la capacidad de enviar mensajes SIP a través de las entradas de texto designadas, así como la recepción y visualización de notificaciones y mensajes SIP en el área correspondiente de la interfaz. Estas pruebas permitirán asegurar que la interfaz gráfica cumple con las expectativas del usuario y proporciona una experiencia interactiva satisfactoria.

Una vez se ejecuta la aplicación, la interfaz gráfica que aparece por defecto es:



Una vez se vayan realizando conexiones, los bloques que están deshabilitando se pondrán a disposición del usuario. Para llevar a cabo las pruebas de la interfaz gráfica, se utilizarán los casos de uso propuestos como referencia.

Caso de uso 1: Elección del dispositivo.

El desplegable “Dispositivo” pone a disposición del usuario ambos módulos utilizados en este proyecto. La selección del dispositivo no modifica el resto de UI, cumpliendo con los requisitos del sistema. A partir de este punto, para el primer conjunto de pruebas se utilizará el dispositivo ESP8266.

Caso de uso 2: Elección del puerto y conexión serie.

En nuestro caso, el módulo está conectado al puerto COM3. Una vez conectado, el estado cambiará el texto:

The screenshot shows a software window titled "Mensajería SIP". It features three main configuration panels on the left and a central status area. The top panel, "Conexión serie", includes a "Dispositivo:" dropdown menu set to "ESP8266", a "Puerto serie:" dropdown menu set to "COM3", a "Desconectar" button, and a status indicator "Estado: Conectado". The middle panel, "Conexión Wi-Fi", includes a "Red wifi:" dropdown menu set to "nombreWifi", a "Contraseña:" text input field, and a "Conectar" button. The bottom panel, "Conexión SIP", includes "Servidor SIP:", "Usuario SIP:", and "Contraseña SIP:" text input fields, an "Iniciar Sesión" button, and a status indicator "Estado: Desconectado". The central area contains a large text box with the message "Conexión serie realizada a ESP8266". Below this, there is a "Mensaje:" text input field and a "Destinatario:" dropdown menu, with an "Enviar" button positioned to the right.

Caso de uso 3: Conexión a la red Wi-Fi

Al haberse conectado un módulo Wifi habilitará el bloque de configuración de la conexión Wi-Fi. Para este caso se utilizará la misma red que en las pruebas del primer bloque. La red es "nombreWifi" y la contraseña passwifi123456789.

The screenshot shows the 'Mensajería SIP' application window. It is divided into three main sections for connection configuration:

- Conexión serie:** Includes a dropdown for 'Dispositivo' (ESP8266), a dropdown for 'Puerto serie' (COM3), a 'Desconectar' button, and a status indicator 'Estado: Conectado'.
- Conexión Wi-Fi:** Includes a dropdown for 'Red wifi' (nombreWifi), a password field (Contraseña), and a 'Conectar' button.
- Conexión SIP:** Includes input fields for 'Servidor SIP', 'Usuario SIP', and 'Contraseña SIP', an 'Iniciar Sesión' button, and a status indicator 'Estado: Desconectado'.

A central log area displays the following messages:

```
Conexión serie realizada a ESP8266
Realizando conexión a red: nombreWifi
Realizada conexión a red: nombreWifi
```

At the bottom, there is a 'Mensaje:' input field, a 'Destinatario:' input field, and an 'Enviar' button.

Al realizarse la conexión se avisa al usuario y se habilita el bloque correspondiente a la conexión SIP.

Caso de uso 4: Conexión e inicio de sesión en el servidor SIP

El servidor utilizado será Asterisk, alojado en el servidor al que se accede mediante el dominio spinar.es. El puerto de acceso debe indicarse también en el campo Servidor SIP, ya que difiere del estandarizado. Se utilizará el usuario creado para las pruebas anteriores.

The screenshot displays the 'Mensajería SIP' application window. It is divided into several sections:

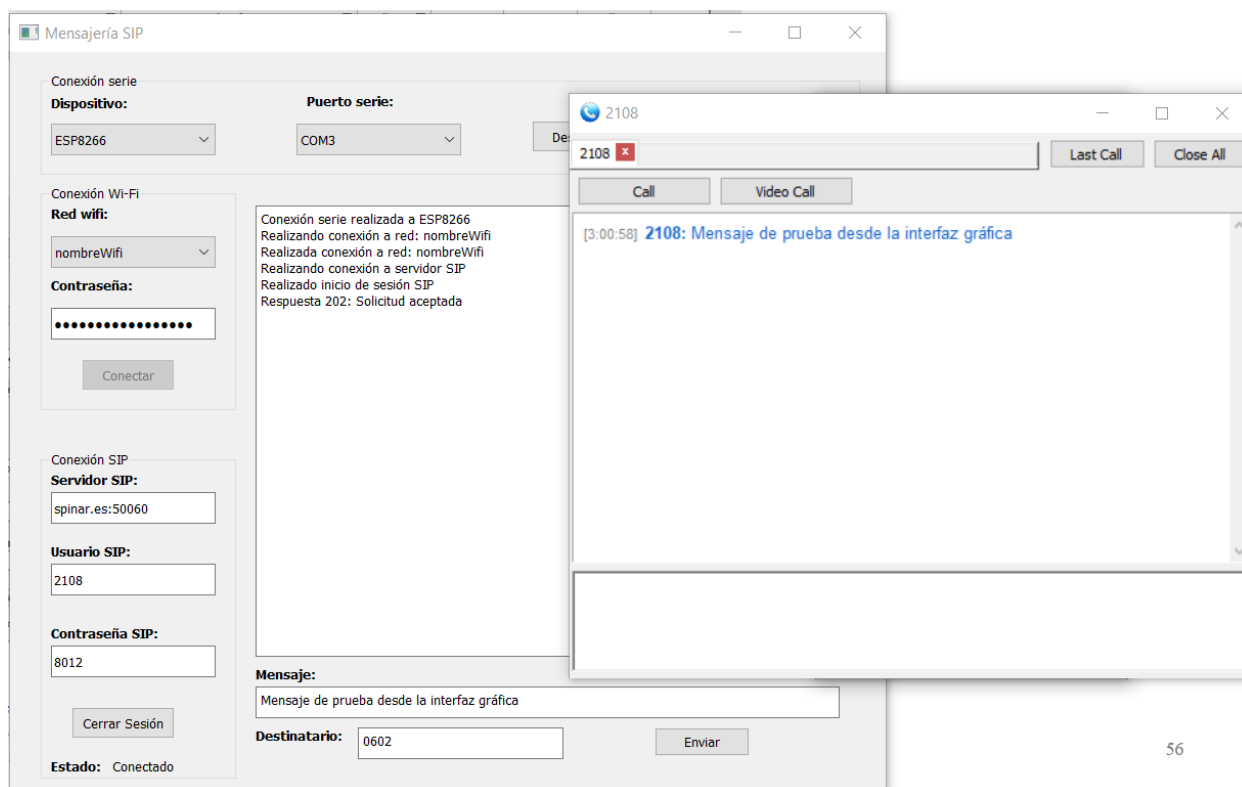
- Conexión serie:** Includes a dropdown for 'Dispositivo' (ESP8266), a dropdown for 'Puerto serie' (COM3), a 'Desconectar' button, and a status indicator 'Estado: Conectado'.
- Conexión Wi-Fi:** Includes a dropdown for 'Red wifi' (nombreWifi), a password field (masked with dots), and a 'Conectar' button.
- Conexión SIP:** Includes text input fields for 'Servidor SIP' (spinar.es:50060), 'Usuario SIP' (2108), and 'Contraseña SIP' (8012), along with a 'Cerrar Sesión' button and a status indicator 'Estado: Conectado'.
- Log/Status:** A large text area containing the following text:

```
Conexión serie realizada a ESP8266
Realizando conexión a red: nombreWifi
Realizada conexión a red: nombreWifi
Realizando conexión a servidor SIP
Realizado inicio de sesión SIP
```
- Mensaje:** A text input field for entering a message.
- Destinatario:** A text input field for the recipient, with an 'Enviar' button to the right.

Se observará que el botón a pasado a ser “Cerrar Sesión” y que el estado a “Conectado”. Además se ha habilitado el botón “Enviar” para que el usuario pueda proceder con el envío de mensajes.

Caso de uso 5: Envío de mensajes.

Los mensajes serán enviados al usuario 0602 registrado mediante microSip:



56

4.1.1.1 Caso de uso 6: Recepción de mensajes.

Desde la propia interfaz de MicroSIP se procede al envío de un mensaje al usuario 2108, que será recibido por el módulo y mostrado en el panel de notificaciones de la interfaz gráfica:

The screenshot shows the 'Mensajería SIP' application window. It is divided into several sections:

- Conexión serie:** Includes a dropdown for 'Dispositivo' (ESP8266), a dropdown for 'Puerto serie' (COM3), a 'Desconectar' button, and an 'Estado' indicator showing 'Conectado'.
- Conexión Wi-Fi:** Includes a dropdown for 'Red wifi' (nombreWifi), a password field (Contraseña) with masked characters, and a 'Conectar' button.
- Conexión SIP:** Includes input fields for 'Servidor SIP' (spinar.es:50060), 'Usuario SIP' (2108), and 'Contraseña SIP' (8012), along with a 'Cerrar Sesión' button and an 'Estado' indicator showing 'Conectado'.
- Message Log:** A large text area displaying the following log:


```

Conexión serie realizada a ESP8266
Realizando conexión a red: nombreWifi
Realizada conexión a red: nombreWifi
Realizando conexión a servidor SIP
Realizado inicio de sesión SIP
Respuesta 202: Solicitud aceptada
<sip:0602@spinar.es>: Mensaje de prueba de la interfaz gráfica
      
```
- Message Input:** A 'Mensaje:' field containing 'Mensaje de prueba desde la interfaz gráfica' and a 'Destinatario:' field containing '0602', with an 'Enviar' button.

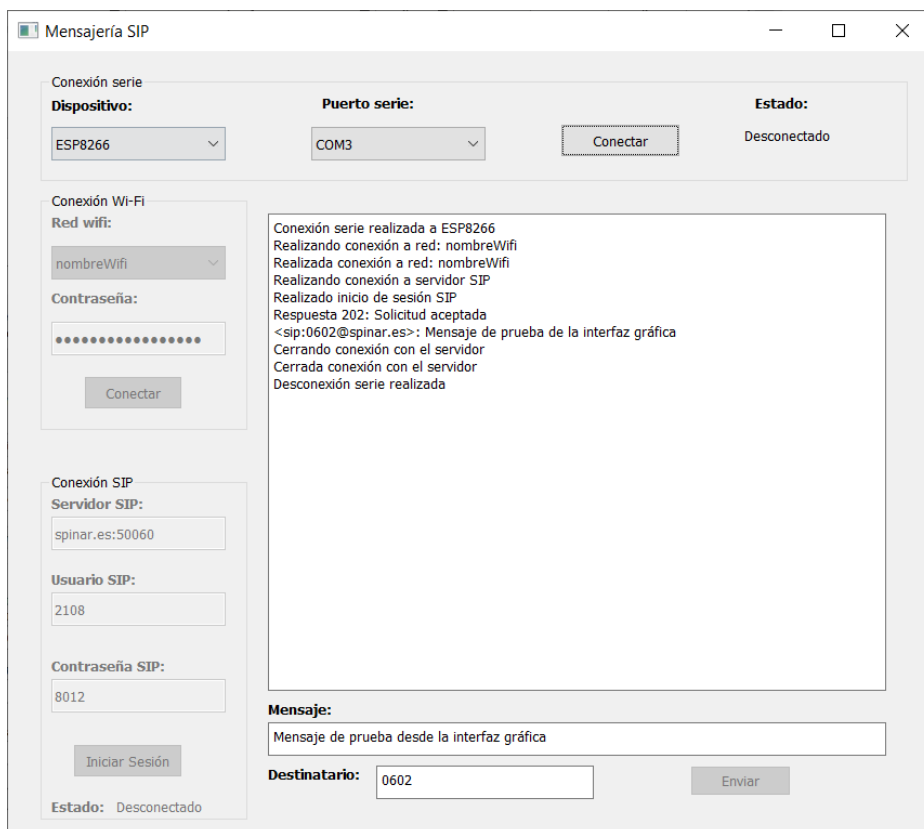
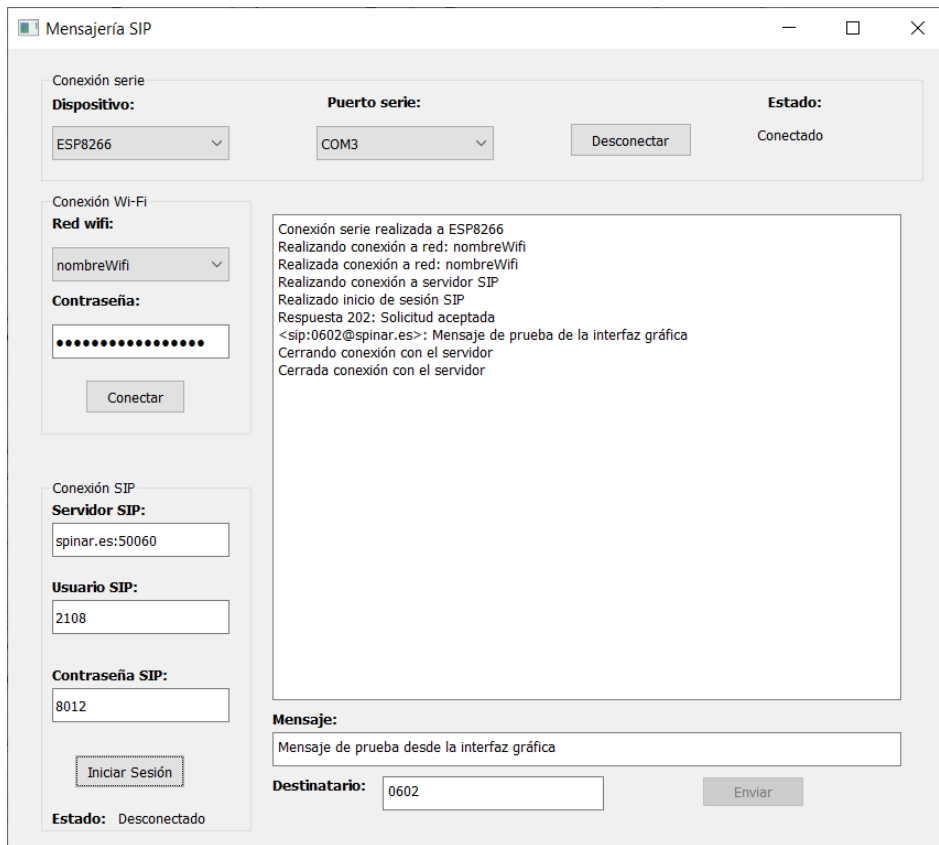
El mensaje aparece correctamente, mostrando tanto el usuario como el cuerpo del mensaje que se ha recibido.

Caso de uso 7: envío de re-register.

Tal y como se detalló en el caso de uso, el actor no es el usuario si no el propio sistema que debe evitar el cierre de sesión automático por caducidad. Aún así se puede comprobar que, aunque pasen varios minutos, la sesión sigue activa y está disponible el envío y recepción de mensajes.

Caso de uso 8: Cierre de sesión y desconexión.

Por último, se procederá al cierre de sesión y de la conexión serie:



Se puede observar que, al producirse la desconexión, se inhabilitan los bloques correspondientes, volviendo la interfaz a su estado original. Tras confirmar el buen funcionamiento de la interfaz utilizando el dispositivo ESP-8266, se realizarán las mismas acciones con el MC60.

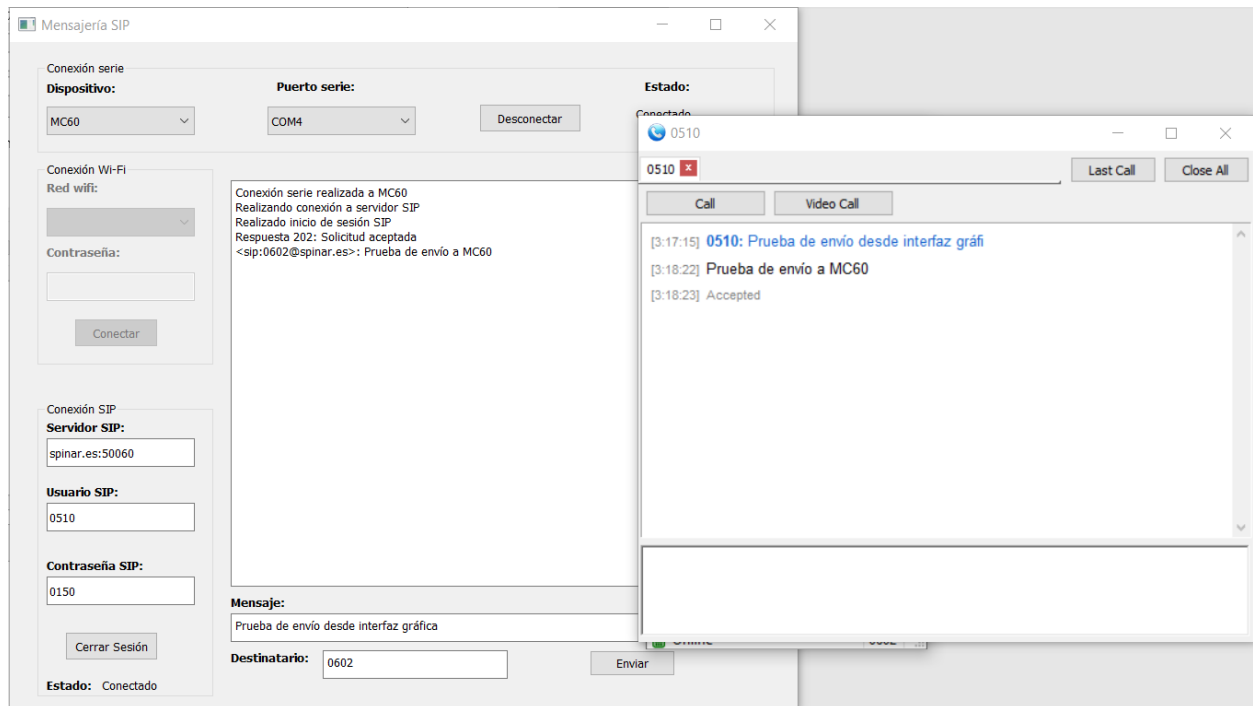
En primer lugar se cambia el tipo de dispositivo y el puerto al que está conectado el dispositivo (COM4) para realizar la conexión:

The screenshot shows the 'Mensajería SIP' application window. It features three main connection sections on the left and a central message area.

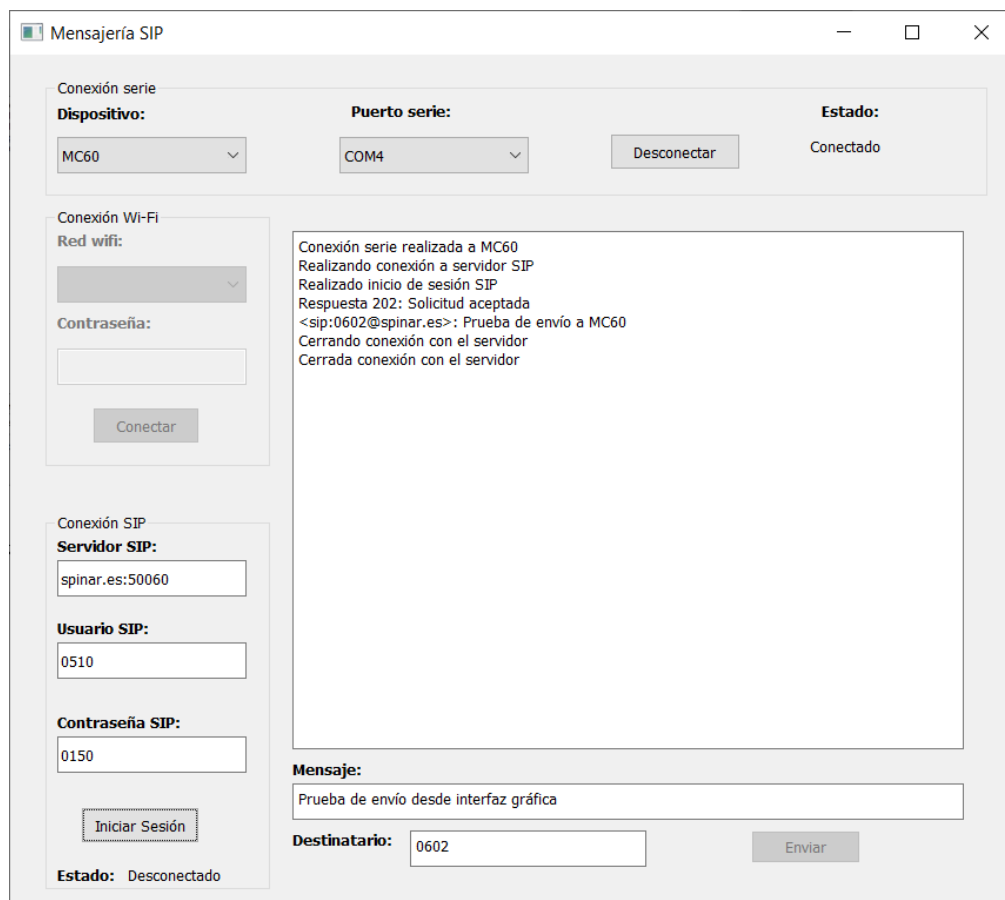
- Conexión serie:** A section with three fields: 'Dispositivo:' set to 'MC60', 'Puerto serie:' set to 'COM4', and 'Estado:' set to 'Conectado'. A 'Desconectar' button is located to the right of the 'Puerto serie:' field.
- Conexión Wi-Fi:** A section with 'Red wifi:' (a dropdown menu), 'Contraseña:' (a text input field), and a 'Conectar' button.
- Conexión SIP:** A section with 'Servidor SIP:', 'Usuario SIP:', and 'Contraseña SIP:' (all text input fields), an 'Iniciar Sesión' button, and an 'Estado: Desconectado' label.

The central area contains a large text box with the message 'Conexión serie realizada a MC60'. Below this is a 'Mensaje:' text input field and a 'Destinatario:' text input field, with an 'Enviar' button to the right of the 'Destinatario:' field.

En este caso, una vez realizada la conexión no se habilita el módulo de "Conexión Wi-Fi". A continuación, se realizará el inicio de sesión SIP y el traspaso de mensajes:



Al cerrar sesión y desconectar el dispositivo la interfaz gráfica vuelve a su estado original:



Las pruebas realizadas demuestran que el sistema funciona correctamente. Se ha verificado que todas las funcionalidades de la interfaz gráfica, incluyendo el envío de mensajes SIP, la recepción de notificaciones y mensajes de otros usuarios, y la visualización adecuada de la información, cumplen con los requisitos

establecidos. Además, se ha confirmado que el sistema se adapta correctamente a diferentes casos de usos y requisitos propuestos, garantizando un flujo de comunicación eficiente y una experiencia de usuario satisfactoria. Los resultados obtenidos validan el diseño y la implementación del sistema, asegurando su operatividad y confiabilidad.

5 CONCLUSIÓN

En conclusión, este proyecto ha logrado implementar con éxito un cliente SIP en los módulos ESP-8266 y MC60, brindando a los usuarios la capacidad de enviar y recibir mensajes SIP para establecer comunicaciones basadas en el protocolo SIP. La utilización del servidor SIP Asterisk ha sido fundamental en la configuración y gestión de las comunicaciones, proporcionando una infraestructura confiable y versátil. Además, se ha desarrollado una interfaz gráfica intuitiva que facilita la interacción de los usuarios con el cliente SIP.

A lo largo del proyecto, se ha demostrado la viabilidad y utilidad de utilizar los módulos ESP-8266 y MC60 para implementar un cliente SIP, ofreciendo una solución práctica y funcional para la transmisión de mensajes SIP. Si bien en esta etapa del proyecto se ha enfocado en el envío y recepción de mensajes SIP, este trabajo sienta las bases para futuras expansiones y mejoras.

El desarrollo de este cliente SIP representa un avance significativo en el campo de las comunicaciones basadas en protocolos IP. Proporciona una alternativa flexible y de bajo costo para establecer comunicaciones entre diferentes dispositivos, abriendo nuevas oportunidades. Algunas de estas posibles mejoras que se pueden aplicar utilizando como base este proyecto son:

- Gracias a la entrada y salida de audio del MC60 podría desarrollarse el establecimiento de llamadas de VoIP mediante SIP.
- SCAIP es un protocolo que funciona utilizando como base la mensajería instantánea SIP. Este protocolo establece unos formatos determinados de mensaje para intercambiar información, alertas, etc. Dicho protocolo podría implementarse procesando el contenido de los mensajes recibidos y generar nuevos mensajes SIP con contenido SCAIP.

En cuanto a funcionalidad de la propia aplicación se podría:

- Configurar distintas opciones de conexión con el servidor SIP, como son: uso de proxy, otros protocolos de transporte como TCP, o incluso TLS.
- La aplicación podría identificar el dispositivo que se haya conectado, siendo innecesario que el usuario tenga que elegir el módulo que utilice.