UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Ben Abdelouahab Pereira, Omar

Tutor/a: Perles Ivars, Ángel Francisco

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización

---

Documentos:

1 - Memoria

2 - Planos

3 - Pliego de condiciones

4 - Presupuesto

5 - Anexos

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

D. Omar Ben Abdelouahab Pereira

TUTORIZADO POR

D. Ángel Perles Ivars

CURSO ACADÉMICO: 2021/2022

En primer lugar, me gustaría agradecer a mi familia por el apoyo que he recibido durante el desarrollo de este trabajo.

El segundo lugar, agradecer a mi tutor Ángel Perles Ivars por su ayuda.

Y, por último, me gustaría dedicarle este trabajo a mi bisabuela, que, a edad de 104 años, falleció el pasado 11 de agosto. Para mí es una persona a la que guardaré con gran cariño en mi corazón y que no puedo agradecerle lo suficiente por haberme acompañado durante estos 22 años.

Resumen

Con el crecimiento de la población en zonas urbanas, ha aumentado de forma considerable el consumo energético en las ciudades. Como consecuencia, aparece de esta forma el concepto de “Smart cities”, una idea que surge de aplicar las Tecnologías de la Información y Comunicación (TIC), el big data y las técnicas de IoT para la gestión eficiente y sostenible de edificios e infraestructura.

En este trabajo se pretende examinar la posibilidad de controlar y monitorizar sistemas HVAC. Para ello, se desarrolla un sistema de control para sistemas de climatización empleando el protocolo de comunicación para automatización de edificios BACnet, así como sensores IoT basados en LoRaWAN. Dicho sistema incluye a su vez una interfaz de usuario e implementa una serie de estrategias para la reducción del consumo.

En concreto, se diseña un prototipo del sistema con los elementos mínimos para el funcionamiento de la red BACnet y la red LoRaWAN, así como la simulación del funcionamiento del sistema real. También se diseña una aplicación encargada de realizar la lectura de los sensores de la red LoRaWAN y aplicar el control necesario en los actuadores de la red BACnet.

Palabras clave: IoT, Smart cities, Smart Buildings, LoRaWAN, BACnet, Sensor inalámbrico, Automatización de edificios, BAS, HVAC, Bacpypes, Raspberry Pi

Summary

With the growth of the population in the urban areas, the energy consumption has increased significantly in the cities. Consequently, has appeared the concept of Smart cities, an idea that emerges from the application of the information and communication technologies, the big data and the IoT techniques to achieve an efficient and sustainable building management.

This work pretends to examine the possibility of monitor and controlling HVAC systems. For that, a control system for air-conditioning is developed by using the building automatization protocol BACnet, as well as IoT sensors based on LoRaWAN tech. The system also includes a user interface and implements a set of consumption reduction strategies.

Specifically, a prototype system is designed with the minimum elements that assures the correct performance of the BACnet and LoRaWAN networks, as well as the simulation of how a real system would work. Besides, an application in charge of reading the LoRa sensors and apply the necessary control in the BACnet actuators is developed.

Palabras clave: IoT, Smart cities, Smart Buildings, LoRaWAN, BACnet, Wireless sensor, Building automatization, BAS, HVAC, Bacpypes, Raspberry Pi

Resum

Amb el creixement de la població en zones urbanes, ha augmentat de manera considerable el consum energètic a les ciutats. Com a conseqüència, apareix d'aquesta manera el concepte de “Smart cities”, una idea que sorgeix d'aplicar les Tecnologies de la Informació i Comunicació (TIC), el big data i les tècniques de IoT per a la gestió eficient i sostenible d'edificis i infraestructura.

En aquest treball es pretén examinar la possibilitat de controlar i monitorar sistemes HVAC. Per a això, es desenvolupa un sistema de control per a sistemes de climatització emprant el protocol de comunicació per a automatització d'edificis BACnet, així com sensors IoT basats en LoRaWAN. Aquest sistema inclou al seu torn una interfície d'usuari i implementa una sèrie d'estratègies per a la reducció del consum.

En concret, es dissenya un prototip del sistema amb els elements mínims per al funcionament de la xarxa BACnet i la xarxa LoRaWAN, així com la simulació del funcionament del sistema real. També es dissenya una aplicació encarregada de realitzar la lectura dels sensors de la xarxa LoRaWAN i aplicar el control necessari en els actuadors de la xarxa BACnet.

Paraules clau: IoT, Smart cities, Smart Buildings, LoRaWAN, BACnet, Sensor sense fil, Automatització de edificis, BAS, HVAC, Bacpypes, Raspberry Pi

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización

I. Memoria

Autor: D. Omar Ben Abdelouahab Pereira

Tutor: D. Ángel Perles Ivars

Índice

Índice de Figuras

Lista de siglas usadas

HVAC – Heating Ventilation Air Conditioning (Calefacción Ventilación y Refrigeración del aire)

BMS – Building Management System (Sistema de Gestión de Edificios)

BAS – Building Automation System (Sistema de Automatización de Edificios)

OSI – Open System Interconnection (modelo de interconexión de sistemas abiertos)

RF – Radio Frecuencies (Radio Frecuencias)

TIC – Tecnologías de la Información y Telecomunicación

1. Objeto

Este trabajo pretende la monitorización y control de sistemas de climatización combinando el uso de las tecnologías de comunicación, el protocolo de comunicación específico para la automatización de edificios BACnet y la aplicación de técnicas de IoT para mejorar la sensorización del sistema.

Además, se pretende construir un prototipo simplificado de una instalación de climatización que demuestre la integración buscada, así como una aplicación con interfaz gráfica para el control del prototipo. También la implementación de estrategias de regulación para la reducción del consumo energético.

2. Antecedentes

Cada vez aparecen más soluciones e información relacionadas con edificios inteligentes (*Smart buildings*). Y es que, la automatización de edificios mediante el uso de tecnologías de la información y la comunicación (TIC) se ha convertido en una pieza fundamental de cara a conseguir reducir la huella de carbono. Acorde a la Unión Europea, en la actualidad aproximadamente el 75% del parque inmobiliario es ineficiente desde el punto de vista energético. La renovación de edificios en uso podría reducir en un 5-6 % el consumo energético y reducir las emisiones de CO2 en otro 5%.

La UE ha puesto en marcha políticas ambiciosas para convertir las ciudades de los estados miembros en ciudades inteligentes (*Smart cities*) con el objetivo de impulsar la transición a energía limpia y cumplir el compromiso medioambiental de las ciudades, que es, además, el objetivo número 11 de los ODS aprobados por la ONU. Por otra parte, empresas como IBM, Microsoft o Google están invirtiendo miles de millones en esta tecnología.

Este trabajo nace siguiendo esa línea, usar la tecnología IoT y las TIC para la monitorización y control de sistemas complejos como el sistema de climatización de una vivienda o una empresa. Además, siguiendo la filosofía de IoT, se pretende lograr la comunicación dispositivos independientemente del fabricante. Se parte del protocolo BACnet, un protocolo libre especializado en la automatización de edificios y el control de sistemas de calefacción, ventilación y acondicionamiento del aire, que fue diseñado de forma que dispositivos de distinto fabricante y distintas funciones puedan comunicarse entre ellos y trabajar conjuntamente.

3. Estudio de necesidades y factores a considerar.

Este proyecto pretende la construcción de un prototipo a modo de demostrador de un sistema de climatización. Dicho prototipo debe simular el funcionamiento de un sistema HVAC real. Se ha prescindido de los elementos de calefacción, ventilación y refrigeración para la simplificación del mismo, y por ello, no son objetos de este trabajo. El prototipo debe contar con las siguientes características:

- Ha de poseer los sensores necesarios para medir las principales magnitudes de dichos sistemas. Al menos debe tener un sensor de temperatura y humedad, y otro para conocer el estado de puertas o ventanas.
- A su vez, debe implementar estrategias de ahorro energético.
- Ha de incluir una interfaz gráfica que muestre y permita modificar los valores de las variables del sistema en ejecución del programa para de esta forma poder simular distintas condiciones.
- Ha de ser lo más sencilla y económica posible, omitiendo de esta forma de elementos innecesarios para el alcance que se planea darle a este proyecto.
- Los componentes de la instalación deben ser compatibles con los de otros fabricantes, de forma que la instalación se pudiese ampliar de ser necesario.
- Debe ser fácil de instalar y que requiera el menor volumen posible.

4. Planteamiento de soluciones alternativas y justificación de la solución adoptada.

Para la realización de este proyecto, se ha dividido el estudio de soluciones alternativas en tres bloques siguiendo las pautas del apartado anterior, estos se corresponden con: Protocolo de Comunicación Específico, Elementos de la instalación (Hardware) y Programas Empleados (Software).

4.1. Protocolos de comunicación para automatización de edificios.

Existen una gran variedad de protocolos de comunicación enfocados en la automatización de edificios (BMS/BAS). Estos protocolos son un conjunto de reglas que van a tener que cumplir los dispositivos conectados a la red para poder comunicarse entre ellos. Dado que existen multitud de tipos de sistemas e instalaciones, cada uno con sus problemáticas, se han creado diversidad de protocolos enfocados en solucionar dichas problemáticas. Por ello, es necesario conocer las propiedades de los distintos protocolos BMS/BAS. A continuación, se explican los más usados en sistemas HVAC junto con sus ventajas e inconvenientes:

4.1.1 Modbus

Modbus es un protocolo de comunicaciones que se desarrolló en 1979 por la compañía Modicon. Se diseñó para el control de controladores lógicos programables (PLC). Este protocolo se popularizó mucho en el sector de la automatización industrial debido a que se trata de un protocolo libre y a que su configuración e implementación es sencilla. Actualmente se continúa utilizando en la industria y en instalaciones tanto domóticas como inmóticas con algunas adaptaciones.

Modbus se sitúa en los niveles 1, 2 y 7 del Modelo OSI (ISO/IEC 7498-1). Presenta una arquitectura de tipo maestro/esclavo donde el dispositivo maestro solicita información y los dispositivos esclavos se la proporcionan a partir de sus registros. Esto significa que el esclavo no puede ofrecer la información, mientras que el maestro puede leer y escribir en los registros de los esclavos. Una red Modbus estándar tiene 1 maestro y puede tener hasta un máximo de 247 esclavos.

Existen varias versiones de este protocolo en función del método de conexión (serie o ethernet), los más frecuentes son:

Modbus RTU

Esta versión se utiliza para comunicación serie usando la comunicación tradicional UART y las interfaces RS232, RS485 y RS422. Su característica distintiva es el uso de codificación binaria y un mecanismo de comprobación de redundancia cíclica (CRC) para la detección de errores y que garantiza la fiabilidad de estos. Los datos se transmiten en tramas de 1 byte con la siguiente estructura:

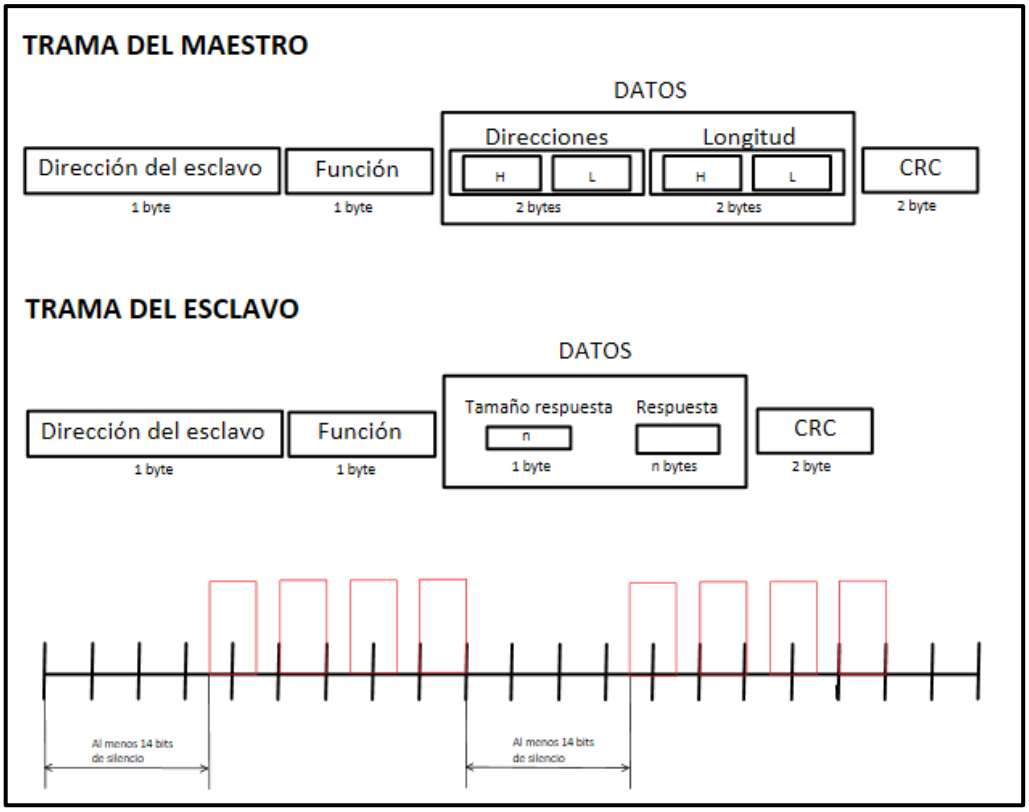


Figura 1. Estructura de la trama Modbus

Las tramas de datos deben estar separadas por periodos inactivos de al menos 3,5 caracteres hexadecimales, lo que equivale a 14 bits. Dicho tiempo se puede calcular a partir de la velocidad de transmisión de datos previamente fijada. Los dispositivos de red (esclavos) monitorizan la red constantemente hasta que se recibe el primer campo, el dispositivo lee el primer campo de dirección para saber si va dirigido a él, en caso de ser así, procesa la solicitud del maestro y prepara la respuesta. La transmisión del mensaje debe realizarse como flujo continuo, si hay un silencio de más de 1,5 veces un carácter, el mensaje se queda incompleto y es desechado por el receptor puesto que no coincidirá el valor del CRC.

Modbus TCP

Esta versión implementa el protocolo Modbus dentro un marco TCP en el puerto 502 con el fin de aprovechar las infraestructuras LAN actuales y se enfoca en la comunicación de PLC con sistemas SCADA, sensores y actuadores. Se utiliza la tecnología Ethernet para la comunicación y se basa modelo OSI, tal que, en el nivel 3 (nivel de red) se utiliza tecnología IP y a nivel 4 (nivel de transporte) se usan segmentos TCP.

Esta versión soluciona muchos de los problemas que tiene Modbus RTU debido a sus limitaciones. Permite conectar más dispositivos a la red, elimina la necesidad de verificar por CRC, puesto que la detección de errores está incluida en el protocolo TCP/IP. También incluye un método de informe por excepción, el cual no está permitido en los protocolos maestro-esclavo estándares.

No obstante, una red Ethernet permite la comunicación entre pares, eso provoca que la relación de maestro y esclavo se vea ligeramente alterada. Ahora tendremos múltiples esclavos servidores y múltiples maestros clientes. Y será necesario que el diseñador establezca las asociaciones lógicas de maestro y esclavo.

Análisis de ventajas e inconvenientes

VENTAJAS	INCONVENIENTES
• Arquitectura simple, pero efectiva en el manejo de datos pequeños. • Tamaño de instalación escalable con la versión TCP/IP. • Fácil implementación y mantenimiento. • Protocolo de comunicación libre. • Al ser muy estandarizado en el sector de la industria, y ser muy simple hay muchos dispositivos compatibles con este protocolo.	• Los tipos de datos que se pueden enviar es muy limitado. Y deben tener un tamaño reducido. • El protocolo no tiene un método estándar para interpretar correctamente el valor de un registro, es decir, no puede saber si el valor que recibe corresponde a una medida de 30ºC o 150ºC. • Las transmisiones de datos deben ser consecutivas, lo que limita las comunicaciones remotas. • Requiere de mucha configuración y programación.

4.1.2 KNX

KNX es un protocolo de comunicación enfocado en la automatización de edificios comerciales y domésticos. La tecnología KNX nace de la fusión de los protocolos EIB (European Installation Bus), EHS (European Home System) y BatiBUS, actualmente ostenta el estándar europeo EN 50090 y el internacional ISO/IEC 14543. Este protocolo es de uso libre y puede ser adquirida sin ningún tipo de cargo desde 2016.

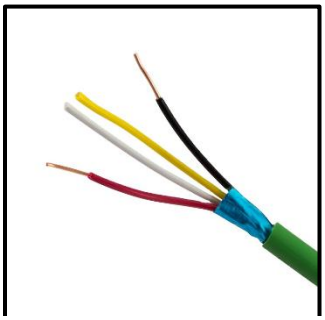


Figura 2. Cable de par trenzado KNX TP

Los sistemas de KNX pueden usar diversos medios para la comunicación entre dispositivos. La más utilizada es la trasmisión a través de un par de hilos trenzados (KNX TP). En las redes KNX TP, todos los dispositivos están conectados al bus de datos (cables rojo y negro) y a la fuente de alimentación (cables blanco y amarillo). La transmisión se realiza a 9600 baudios, de forma asíncrona y half-duplex (bidireccional, pero solo una dirección a la vez). Otra característica importante de la transmisión KNX TP es diferencial. Los datagramas o telegramas KNX TP consisten en 4 campos:

- Campo de control. Se define la prioridad del telegrama y también si este es repetido.
- Campo de dirección. Se define la dirección física (o de grupo) del emisor y del receptor.
- Campo de datos. Los datos de la transmisión pudiendo transmitir hasta un máximo de 16 Bytes.
- Campo de comprobación. El valor para poder verificar su contenido mediante un cálculo de paridad.

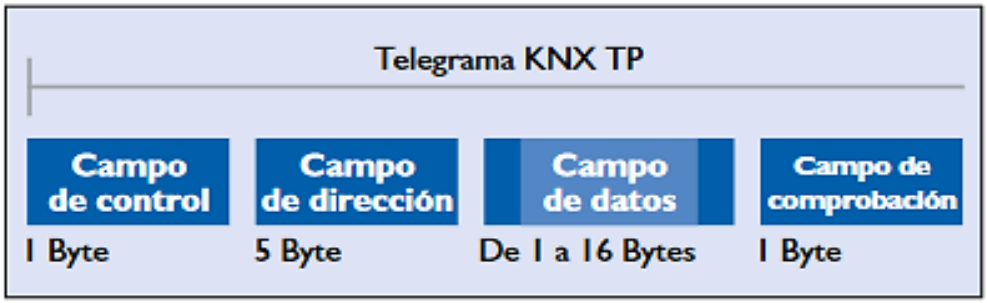


Figura 3. Estructura de trama KNX TP

Para acceder al bus KNX se usa un acceso aleatorio dependiendo de sucesos, es decir, un telegrama solo se puede enviar si no hay otra transmisión en curso. Para evitar colisiones, se establece un orden por prioridades.

KNX también tiene una versión basada en la tecnología TCP/IP (KNX IP), no obstante, el elevado coste de la instalación de cableado debido a que todos los terminales necesitarían una conexión a la red. Y a la gran cantidad de dispositivos necesarios para la transmisión de datos hacen que el coste energético se dispare haciendo que la eficiencia energética de la instalación se vea gravemente afectado. Por estos motivos esta versión no es muy utilizada.

Análisis de ventajas e inconvenientes

VENTAJAS	INCONVENIENTES
• Protocolo de comunicación libre. • KNX garantiza que dispositivos de diferentes fabricantes puedan. comunicarse y operar entre ellos. • Garantiza un alto grado de flexibilidad en la ampliación de instalaciones. • De bajo consumo energético. • Dispositivos preprogramados por lo que únicamente es necesaria la configuración.	• Es necesario realizar obras para colocar el cableado. • Los dispositivos para los sistemas KNX son más caros que sus similares. • Los instaladores deben ser especializados en el protocolo (Partner KNX). • Se puede dar la saturación de la red por repetición de mensajes, puesto que todos los dispositivos están conectados al mismo bus de comunicaciones.

4.1.3 BACnet

El protocolo de comunicación BACnet (Building Automation and Control Networks) es un protocolo diseñado para la automatización, control y gestión de sistemas HVAC, iluminación, seguridad… Permitiendo el intercambio de información e interoperabilidad de los dispositivos compatibles con el protocolo.

BACnet comenzó su desarrollo en 1987 por la Sociedad Estadounidense de Ingenieros de Calefacción, Refrigeración y Aire Acondicionado (ASHRAE). En 1995 se estableció como el estándar ANSI/ASHRAE y en 2003 estándar mundial ISO 16484-5. Actualmente es considerado como el protocolo más completo y potente en el sector de la automatización de edificios.

La arquitectura de BACnet se basa en las capas física, de enlace de datos, de red y de aplicación correspondientes al modelo OSI tal y como se muestra en la imagen.

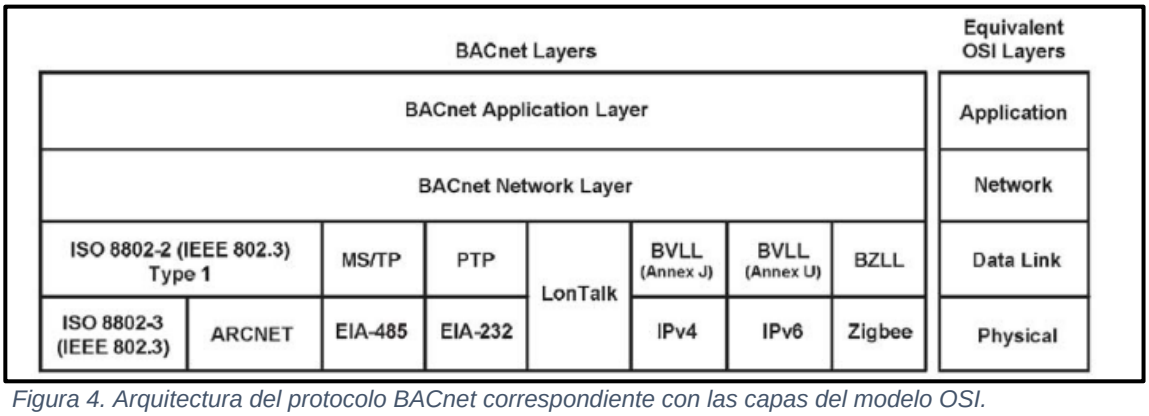


Figura 4. Arquitectura del protocolo BACnet correspondiente con las capas del modelo OSI.

En la capa física y de enlace destacan BACnet/IP usando la tecnología TCP/IP de forma similar a lo visto en KNX, BACnet/MSTP (master-slave) usando RS485 de forma similar a Modbus y por último BACnet/ethernet.

El protocolo BACnet define una serie servicios para la comunicación entre dispositivos, servicios de protocolo "*who-is, I-Am, who-has, I-have*" los cuales sirven para el descubrimiento de objetos y dispositivos, y servicios de lectura y escritura de datos. También se definen un conjunto de objetos estándar que representan los parámetros más utilizados en los sistemas de control. Dichos objetos pueden ser de tipo binario, tipo analógico o tipo alfanuméricos codificados entre otros.

Los mensajes enviados utilizan mensajes similares al UDP (User Datagram Protocol), de forma que los mensajes se envían a la red sin haber realizado una conexión previa con el dispositivo receptor.

Las tramas de BACnet MS/TP pueden tener de 0 a 501 bytes donde cada byte está formado por 8 bits y 2 bits adicionales para marcar inicio y final de la trama. La trama está dividida en 2 partes el encabezado (header) y por una parte de datos, tal y como se muestra en la imagen.

Figura 5. Estructura de trama BACnet MS/TP

Los primeros 2 bytes son los valores 55h y FFh y forman el preámbulo, son seguidos por un byte que indica el tipo de frame con un valor de 0 a 8. Los tipos de trama definidos en BACnet son:

- (0) Token: utilizado en el relacionamiento de estaciones maestras. La estación maestra que tiene el token puede iniciar la comunicación, al finalizar la comunicación le transfiere el Token a otra estación maestra. No presenta datos.
- (1) Poll for Master: se transmite periódicamente y sirve para localizar a otros maestros en la red y determinar la secuencia del token. Este mensaje debe ser respondido por otros maestros e ignorado por las estaciones esclavas. No presenta datos.
- (2) Reply to poll for Master: es la respuesta para el mensaje "Poll for Master". Tampoco presenta datos.
- (3) Test Request: utilizado para iniciar la comunicación en la red MS/TP y enviar información a una estación particular.
- (4) Test Request: contesta al "Test Request"
- (5) BACnet Data Expecting Reply: Utilizado por estaciones maestras para transmitir datos de parámetros. El mensaje presenta la dirección, los datos, prioridad y un código de mensaje. Una vez termina espera una respuesta de la estación destino.
- (6) BACnet Data not Expecting Reply: Igual que en un mensaje "BACnet Data Expecting Reply" salvo porque al terminar no espera respuesta del destino.
- (7) Reply Postend: utilizado por los maestros para indicar que la respuesta de una trama de tipo "BACnet Data Expecting Reply" será enviada más tarde. No presenta datos.

Las tramas de tipo 0, 1 y 2 solo son comprendidas por estaciones maestras y son ignoradas por las estaciones esclavas.

A continuación, se envían 2 bytes para indicar la dirección de destino y de fuente. Estas direcciones van del rango 0 a 254 (en BACnet MS/TP), donde las direcciones de 0 a 127 están reservadas para estaciones maestras y esclavos, mientras que el resto solo pueden ser usadas por estaciones esclavos. La dirección 255 está reservada para broadcast.

Por último, la trama se compone de 2 bytes para indicar el tamaño del mensaje, un byte para el mecanismo CRC-8 para el encabezado y otro mecanismo CRC-16 para el apartado de datos, y los datos que pueden tener de 0 a 501 bytes.

En BACnet, cabe destacar que cada dispositivo tiene asignado una ID configurable por el usurario. Esta se usa en las comunicaciones para reconocer el dispositivo con el que se quiere comunicar. También, hay que mencionar que las comunicaciones de tipo BACnet tienen asignados el puerto 47808 como puerto predeterminado, aunque en algunos casos, se pueden usar otros puertos. En la siguiente figura [Fig.6] se pueden apreciar toda la información que se transmite en una petición de servicio I-am.

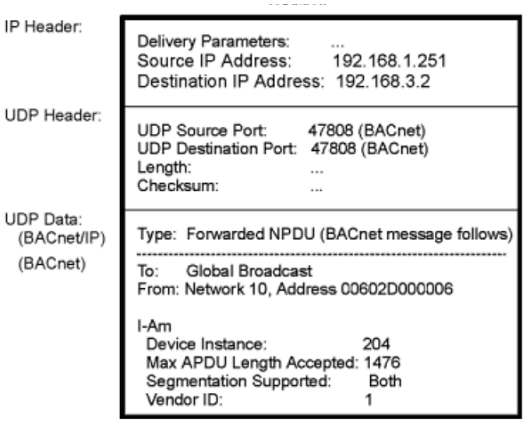


Figura 6. Descripción de petición de servicio I-am

Análisis de ventajas e inconvenientes

VENTAJAS	INCONVENIENTES
• Protocolo de comunicación libre. • No depende de la topología de red. • Tamaño de la instalación escalable de pequeños edificios a instalaciones con miles de dispositivos. Sin afectar negativamente al coste ni al rendimiento. • Garantiza la interoperabilidad de los dispositivos al implementar productos de otros fabricantes. • Permite la transmisión de una amplia variedad de tipos de datos. Lo que hace el análisis de la instalación más preciso. • Es muy robusto y soporta distintos tipos de LAN simultáneamente.	• Es necesaria una infraestructura Ethernet. • Los nuevos estándares de seguridad no están implementados en todos los dispositivos. • Numero de dispositivos limitada excepto en TCP/IP. • Requiere de una configuración, no es un “plug-and-play”. • Requiere de más memoria de almacenamiento que otros protocolos.

4.1.4 Protocolo de comunicación

Se ha optado por el uso de BACnet IP al tratarse de un protocolo de comunicación libre que garantiza la interoperabilidad entre dispositivos. También por tratarse de uno de los protocolos de comunicación del mercado más utilizados y ser considerado el más completo.

Además, este protocolo permite la comunicación de múltiples tipos de dados y es muy eficiente en términos de rendimiento. Por último, dado que se trata de un prototipo demostrador, la infraestructura necesaria es mínima y fácil de conectar. Y tampoco va a tener problemas de memoria.

4.2. Elementos de instalación

Las instalaciones HVAC están compuestas por diversos sistemas como unidades de tratamiento de aire o sistemas de refrigeración y calefacción. Dichos sistemas cuentan con multitud de dispositivos, tanto de actuadores y sensores, como de elementos de comunicación y de control necesarios para el correcto funcionamiento de la instalación. A continuación, se exponen los elementos necesarios y sus funciones:

4.2.1 Sensores

Los sensores son un elemento indispensable en cualquier instalación automatizada, puesto que permiten conocer el estado de las magnitudes físicas que se pretenden controlar, así como saber el estado de elementos de la instalación (Ej. Una ventana abierta o cerrada, o la temperatura de una habitación), de forma que de cuanta más información se disponga, mejor será el control del sistema y por ende un mejor resultado.

En el caso de sistemas HVAC, esto es especialmente importante puesto que estos consumen mucha energía. Por ello, se llevan a cabo estrategias de regulación para reducir el consumo de estas. Algunas de las estrategias más utilizadas son:

- Arranque de los sistemas por temperatura exterior, de forma que en periodos cuyas temperaturas no sean extremas, los sistemas no arranquen si las condiciones son favorables.
- Control de parada debida a la apertura de puertas y ventanas.
- Control horario del sistema de climatización, el sistema se activa en franjas concretas del día o de la semana.
- Control en presencia de usuarios, por ejemplo, en un hogar inteligente si el sistema no detecta la presencia de los inquilinos, el sistema de climatización trabaja bajo una consigna diferente.

Como se puede observar, muchas de estas estrategias requieren no solo de sensores de temperatura, sino que también ópticos o magneticos para saber el estado de puertas y ventanas, así como la presencia de usuarios.

Debido a la naturaleza de una instalación HVAC, se requiere que los sensores estén distribuidos de forma dispersa. Eso supone un problema, puesto que es necesario comunicarlos con el controlador. Esto se puede realizar mediante el uso de infraestructuras TCP/IP y MSTP, o bien empleando tecnologías inalámbricas. En este trabajo se ha optado por el uso de LoRaWAN, un estándar de red inalámbrica de baja potencia y de gran alcance, con el fin de eliminar la necesidad de cualquier infraestructura cableada.

Para el prototipo de instalación, se ha decidido utilizar un sensor de temperatura y humedad y un sensor magnético para puertas.

4.2.2 LoRaWAN Gateway/Converter to BACnet

Como se ha explicado en el punto anterior, se ha optado por el uso de dispositivos inalámbricos LoRaWAN. Por ello, es necesario disponer de un Gateway (o punto de acceso). La función de una Gateway es comunicar los dispositivos inalámbricos sobre los que queremos leer/escribir con el resto de la red. Para ello, la Gateway debe generar una red inalámbrica a la que se conectan todos los dispositivos, de forma similar a un punto de acceso WIFI de la red.

Para comunicar la Gateway con el resto del sistema es necesario un convertidor que traduzca los mensajes LoRaWAN al protocolo BACnet, permitiendo así que un dispositivo BACnet pueda leer correctamente la información de un sensor. Existen Gateways que incluyen integrados un convertidor del formato LoRaWAN al formato de protocolos de comunicación para la automatización de edificios, por lo que generalmente la propia Gateway es el convertidor.

A la hora de escoger el convertidor hay una serie de factores a tener en cuenta. El primero es la frecuencia del dispositivo, esta debe ser de 868 MHz puesto que se trata de la banda ISM europea. La segunda cosa que debemos tener en cuenta es si queremos que el convertidor actúe como maestro o como esclavo. Puesto que la Gateway va a comunicar sensores y no controladores, este va a actuar como un esclavo. Por último, dependiendo del tipo de red de la instalación, el convertidor puede tener salida Ethernet (BACnet IP) o RS-485 (BACnet MSTP).

4.2.3 Controladores

Los controladores son los elementos más importantes de una instalación, puesto que se encargan de leer y procesar la información de los elementos pasivos (sensores e interruptores), calculan una respuesta y la transmiten a otros elementos activos. Se pueden categorizar dependiendo del modo de funcionamiento en:

- Controladores de propósito específico, estos controladores están diseñados para realizar una función determinada, como, por ejemplo: controlar la iluminación, las alarmas de incendio de un edificio o el sistema de calefacción. Estos dispositivos no suelen ser libremente programables siendo solo configurables.
- Controladores de propósito general o controlador central, estos controladores leen los datos de los elementos pasivos y gestionan los elementos activos de la instalación.

Los controladores de propósito general suelen ser ordenadores comunes, ordenadores industriales, PLCs o placas procesadoras como una placa Raspberry Pi.

Para este proyecto se ha optado por el uso de una Raspberry Pi puesto que solo se pretende controlar un sistema de ventilación:

- No se espera la necesidad de mucha potencia de cálculo
- Una Raspberry consume mucha menos energía que un ordenador o un PLC
- Es más fácil y flexible la implementación del sistema en una Raspberry que en una PLC.
- La Raspberry Pi es mucho más económica (aunque debido la escasez de estas debido a la pandemia su precio se ha visto incrementado llegando a costar hasta varias veces su precio normal).

4.2.4 Actuadores

Los actuadores son los dispositivos que realizan acciones en nuestro sistema al aplicar una acción de control sobre estos. Estos elementos son gestionados por el controlador mediante señales tanto analógicas como digitales. Existen muchos tipos de actuadores, pero las principales funciones que desempeñan en los sistemas HVAC son:

- **Calefacción**. Tanto eléctrica como de gas natural. Esta suele estar compuesta por una caldera, un sistema de tuberías para transportar el calor y se dispersa mediante radiadores u otros elementos. Aunque existen muchos tipos.
- **Refrigeración**. Los sistemas de refrigeración están formados por un elemento interior y uno exterior. Sometiendo un gas refrigerante a varios procesos termodinámicos se consigue absorber el calor de la habitación y expulsarlo fuera.
- **Ventilación**. Los sistemas de ventilación se componen de un sistema de regulación del caudal del aire, así como un mecanismo de extracción de aire y otro de impulsión de aire. De forma, adicional pueden tener filtros de partículas, sistemas de recuperación de calor, humidificadores y des humificadores.

Otras funciones importantes para cualquier sistema automático son los indicadores de estado y de magnitud. Como pueden ser indicadores LED o “displays” que muestren el estado de la instalación.

Dado que durante la elaboración del proyecto no se ha dispuesto de los elementos de calefacción, refrigeración y ventilación. Se ha optado por sustituir estos por indicadores LED simulando su estado de funcionamiento.

4.2.5 SCADA

Un supervisor SCADA (Supervisory Control And Data Acquisition) es un elemento que se utiliza para controlar y monitorizar instalaciones automáticas de forma remota. Permite leer el estado de tanto elementos pasivos como activos y almacenar su valor en memoria. Los SCADAS pueden ser un elemento físico de la instalación, aunque también se puede implementar por software.

En el caso de este proyecto, el SCADA estará implementado por software dentro de la aplicación que se ejecuta en el controlador.

4.2.6 Switch ethernet

Puesto que para la elaboración del proyecto se va a utilizar BACnet TCP/IP, es necesario conectar todos los elementos de la mediante cables Ethernet. Para ello, es necesario el uso de un Switch (o conmutador).

Figura 7. Switch ethernet de 8 puertos.

4.2.7 Módulo I / O

Un Módulo I/O (inputs y outputs), es un dispositivo que dispone de una serie de puertos de entrada y de salida configurables. Permite leer y enviar señales tanto analógicos como digitales a tensiones y corrientes mucho mayores de las que podría gestionar el GPIO de la Raspberry Pi. Este elemento es necesario para poder controlar todos los actuadores y poder leer sensores que no fuesen inalámbricos.

Este dispositivo debe ser compatible con BACnet TCP/IP para poder conectarlo al controlador mediante la red BACnet.

4.2.8 Fuentes de alimentación

Dado que la mayoría de los dispositivos operan con tensiones de 12/24 V, es necesario una fuente de alimentación o varias, que sean capaces de alimentar a todos los elementos de la instalación con la red eléctrica (230 VAC y 50 Hz).

Figura 8. Fuente de alimentación 12 V

4.3. Software

4.3.1 Servidor BACnet

Para el desarrollo del proyecto se requiere un servidor BACnet. Aunque existen una gran variedad de aplicaciones, la mayoría son software de pago o para Windows. Durante el desarrollo se priorizó que el software fuese de código libre. Entre las opciones más destacadas están:

- **Yabe,** un explorador grafico de dispositivos BACnet con las funciones más básicas del protocolo.
- **BACnet4Linux,** una aplicación BACnet basada en Linux como sistema operativo.
- **BACSharpk,** un stack para BACnet/IP escrito en C#.
- **BACnet stack,** un stack de BACnet/IP escrito en C.
- **BACpypes,** un stack de BACnet escrito en Python.

Dado que la aplicación se ejecuta en una Raspberry Pi, se ha optado por el uso de **Bacpypes**, una librería en Python de código abierto compatible con Raspberry. Diseñado para BACnet/IP, pero compatible con otras versiones del protocolo.

4.3.2 Librerías

Para el desarrollo de la interfaz, se valoraron diversas posibilidades como:

- **PYQt5**
- **PySimpleGUI**
- **PySide**
- **Tkinter**

Pero, se optó por utilizar la librería de Python **tkinter**, debido a que ofrece un alto grado de personalización, a que es una librería sencilla de usar y existen grandes cantidades de material didáctico de esta librería en internet.

4.3.3 Otros programas

Cabe destacar que durante el desarrollo del proyecto se emplearon otros programas que facilitaron la programación:

El primero y el más empleado fue **WireShark**, uno de los programas más famosos para la captura de paquetes y análisis de protocolos de comunicación. Se trata de una aplicación de código libre que permite la captura de paquetes BACnet de forma se puede comprobar el correcto funcionamiento de la red.

Figura 9. Logo del programa Wireshark

El segundo es el VNC viewer, un programa que permite controlar la Raspberry de forma remota mediante la interfaz VNC. Facilitando las tareas de programación, testeo y redacción.

Además, se usaron otros programas para la configuración de dispositivos. Dichos programas fueron proporcionados por los fabricantes y se hablará de ellos en el apartado 5.

5. Descripción detallada de la solución adoptada.

Para el desarrollo de este proyecto, se ha decidido construir una instalación HVAC básica a modo de demostrador utilizando el protocolo de automatización de edificios BACnet. Dicha instalación, debe leer los datos de unos sensores LoRaWAN, convertir la información a BACnet para que puedan ser leídos por los dispositivos de la red, procesar dicha información y escribir en los actuadores. En la siguiente figura [fig.10] se muestra el esquema de la instalación.

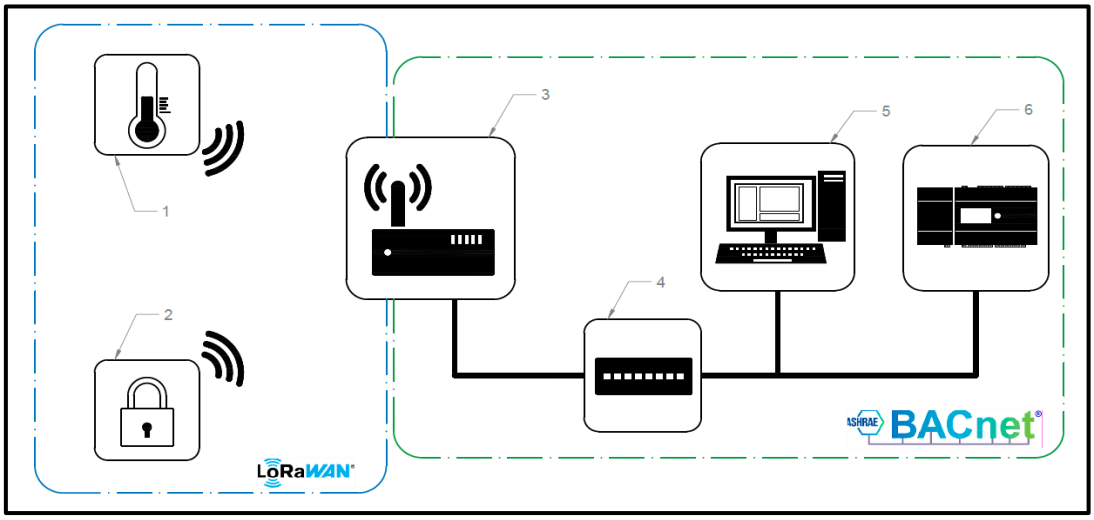


Figura 10. Esquema de la red

También se quiere implementar una interfaz gráfica que permita visualizar el estado del sistema y permita al usuario realizar cambios en el sistema. Así como poder ejecutar tareas especiales mediante la inserción de comandos en una consola. A continuación, se explican los distintos elementos de la solución adoptada:

5.1. La Red LoRaWAN

5.1.1 Elementos de la Red LoRaWAN

La red LoRaWAN se encarga de leer los sensores y traducirlos al protocolo BACnet. Para ello, es necesario un convertidor de LoRa a BACnet. Se ha escogido el HD67F17-B2-IP-868MHz [fig.11] de ADFweb, se trata de un convertidor BACnet Slave / LoRaWAN Gateway.

Figura 11. Convertidor HD67F17

Especificaciones

- Alimentación 12…35V DC
- Banda RF 868 MHz
- Tipo BACnet Slave BACnet/IP
- Conexión por Ethernet o Wifi

Para los sensores se han seleccionado el sensor de temperatura y humedad LHT52 de Dragino [fig.12], y el sensor de puerta magnético LDS01 también de Dragino [fig.13].

Especificaciones LHT52

Temperatura

- Rango -20 ~ 50 °C
- Resolución 0.01 °C
- Desviación medida ± 0.3 °C

Humedad

- Rango 0 ~ 99.0 %
- Resolución 0.1 %
- Desviación medida ± 3 %

Figura 12. Sensor de temperatura y humedad LHT52

Especificaciones LDS01

- Uplink periódico y al abrir/cerrar.
- Configurable mediante Downlink
- Compatible con múltiples RF
- Alimentado con pilas CR2032

Figura 13. Sensor de Apertura de puertas LDS01

5.1.2 Configuración general del convertidor HD67F17-B2-IP-868MHz

Para la configuración del convertidor es necesario usar el software SW67F17 que proporciona el propio fabricante. Al ejecutar el programa aparece el siguiente menú [fig.14]:

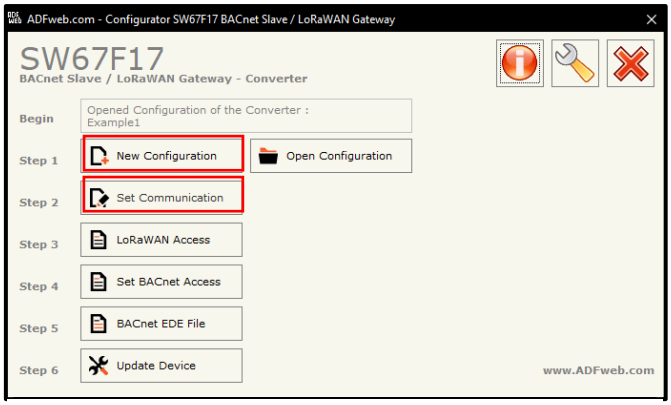


Figura 14. Menú del programa SW67F17 (5.1.2 Configuración del convertidor)

Se pulsa sobre "New Configuration" y se introduce el nombre que se desee. A continuación, se selecciona la opción de "Set Communication". Esto abre el panel de configuración de las redes tanto de LoRa como de BACnet [fig.15].

Dado que el dispositivo HD67F17-B2-IP-868MHz se trata de un convertidor de LoRaWAN - BACnet/IP. Y, además, en el prototipo planteado de la red no se dispone de un router BACnet, en el apartado 1 (*Select Device*) se selecciona la opción "Only Ethernet Cable".

Se ignora el apartado 2 y se salta directamente al apartado 3 (*LoRaWAN*) donde se va a configurar la red LoRaWAN. La banda RF, se deja en EU868 puesto que se trata de la banda de frecuencia reservada en la unión europea para aplicaciones IoT. A continuación, se cambia el Network Type (tipo de red) a Public y se deja RX2 Datarate (la tasa de transmisión-recepción de datos) a SF12BW125, el valor por defecto.

Por último, en el apartado 4 (*BACnet Slave*), se configura la red BACnet. En *Type*, se debe seleccionar BACnet/IP, puesto que es el protocolo con el que funciona nuestro dispositivo. El puerto debe ser 47808 que es el puerto que usa BACnet. Y se seleccionan un identificador, una dirección IP y un nombre de dispositivo BACnet.

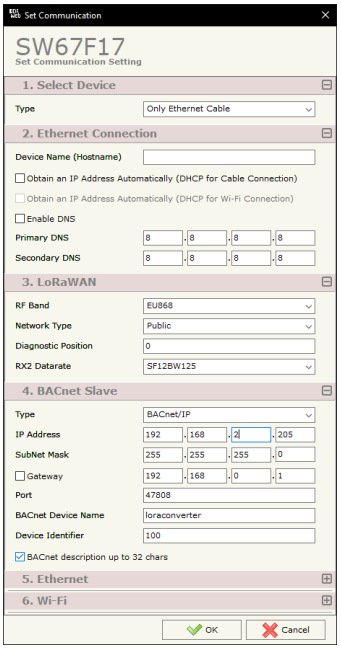


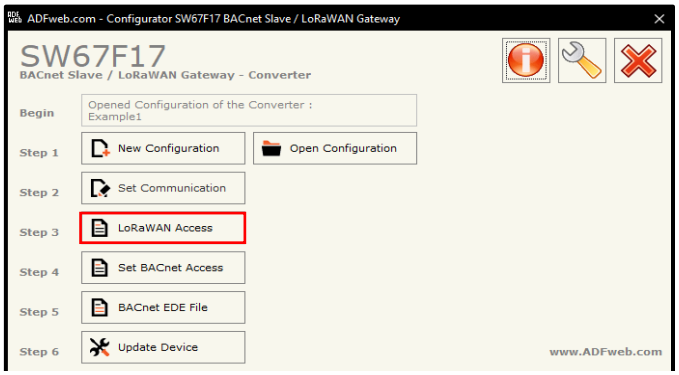
Figura 15. Configuración del convertidor

Para este proyecto se va a usar el identificador de red 100, el nombre de *loraconverter*, y la IP de default del dispositivo 192.168.2.205. Cabe destacar que la máscara de red que esta por defecto solo cubre las IP de 192.168.2.0 a la 192.168.2.255. Por lo que si se necesitara o se tuvieran dispositivos fuera de ese rango habría que modificar dicho valor.

Los apartados 5 y 6, no son configurables puesto que las opciones wifi se desactivan al seleccionar "Ethernet Only" en el apartado 1. Y las opciones Ethernet son para BACnet/MSTP.

5.1.3 Configuración de los sensores

Para que el convertidor pueda acceder a los sensores de la red LoRaWAN, hay que configurar la Gateway que el propio convertidor HD67F17-B2-X-868MHz tiene implementado. Para ello, hay que introducir las características de ambos sensores en la tabla de acceso del Gateway.



En el SW67F17, se selecciona la opción de “LoRaWAN Access”. Esto abre un panel con dos tablas. La primera tabla es una lista de dispositivos mientras que la segunda es una lista de las reglas de Uplink y Downlink.

Figura 16. Menú del programa SW67F17 (5.1.3 Configuración de gateway)

En la primera tabla [fig.17] se deben definir los tipos de dispositivos, así como las claves que permiten acceder a ellos. Los parámetros por definir son:

- **Device EUI.** Clave de 64 bits que identifica al sensor, esta clave es única para cada sensor.
- **Join EUI.** Clave de 64 bits que identifica al servidor de aplicación.
- **Device Name.** Nombre del dispositivo, puede ser cualquiera.
- **Activation Method.** Método de activación puede ser Over-the-air o una personalizada.
- **AppKey.** Es una clave de 128 bits y es la clave de encriptación entre la fuente del mensaje y el receptor. Esta clave es única para cada dispositivo.
- **Device Address.** Dirección del dispositivo dentro de la red.
- **Class.** Clase del dispositivo e indican el tipo de comunicación que soporta el dispositivo.
- **Diagnosis TimeOut.** Tiempo de espera el ms para la recepción de mensajes.
- **RX2 Datarate.** Define la frecuencia y velocidad de transmisión de datos.

Dichos valores son los siguientes para cada uno de los sensores:

LoRaWAN Gateway Devices List

N	Enable	Device EUI	Join EUI	Device Name	Activation Method	Network	Application Session	Application Key	Device Address	Class	Diag. TimeOut	RX2 Datarate
1	☑	A84041C26184594B	A840410000000100	LHT52	Over-the-Air Activation			4F58D526915133D1488EA665C68A551C	00000011	A	76000	SF12BW125
2	☑	A84041B8C18298EF	A000000000000107	LDS01	Over-the-Air Activation			C57B51438739BD3C6C25137389B53D17	00000012	A	76000	SF12BW125
3	☑											
4	☑											
5	☑											
6	☑											

Figura 17. Tabla de la lista de Dispositivos

A continuación, se configuran la lista de reglas de Uplink (lectura de datos en el dispositivo) y Downlink (escritura de datos). Puesto que para este proyecto solo se quiere leer de los sensores se omite la configuración del Downlink. La configuración se debe realizar para cada dispositivo de forma separada, y un mismo dispositivo puede leer de varios puertos simultáneamente. En este caso, la configuración requiere de un único puerto por dispositivo.

LHT52

Para el sensor LHT52 acorde con el Datasheet, los datos de humedad y temperatura se transmiten por el puerto 2 mientras el resto de los puertos están reservados para los registros (puerto 3) y propiedades del dispositivo (puerto 5). Los datos se trasmiten acorde con la estructura mostrada en [Tabla 1] forma cada 20 min (tiempo por defecto):

Size (bytes)	2	2	2	1	4
Value	Temperatura	Humedad	Temperatura Sensor Externo	Sensor Externo Activo.	Unix TimeStamp

Tabla 1. Estructura de los mensajes del LHT52

Dado que no se dispone de un sensor externo, solo se requiere la información de los primeros 4 bytes. Por lo que los valores a introducir en la tabla [fig.18] son los siguientes:

Figura 18. Tabla de reglas de Uplink para LHT52

N	Enable	Port	Start Test	Len Test	Hex	Value Test	Start Byte	Number Bytes	Position Byte
1	☑	2			☐		0	4	0

LDS01

Para el sensor LDS01 los datos se transmiten por el puerto 10 y presenta una estructura de mensaje tal y como muestra la [Tabla 2].

Size (bytes)	2	1	3	3	1
Value	Estado sensor y Batería	MOD (Siempre es 0X01)	Total de eventos puerta abierta	Duración del último evento pueta abierta	Alarma

Tabla 2. Estructura de los mensajes del LDS01

Para la aplicación deseada solo se requiere del valor del estado, es decir, los dos primeros bytes. Por lo que la tabla [fig.19] de reglas para este sensor debe asemejarse a la siguiente:

N	Enable	Port	Start Test	Len Test	Hex	Value Test	Start Byte	Number Bytes	Position Byte
1	☑	10			☐		0	2	6

Figura 19. Tabla de reglas de Uplink para LDS01

5.1.4 Configuración de la conversión a BACnet

Para que los dispositivos de la red puedan acceder e interpretar correctamente los datos de los sensores, debe indicarse les a los sensores en que posición de la memoria, el convertidor ha almacenado los datos y que tipo de datos representan.

Para ello, hay que configurar una serie de objetos de tipo BACnet con la opción de "Set BACnet Access" en el programa SW67F17. Estas estructuras de datos vienen previamente definidas en el protocolo, por lo que cualquier dispositivo BACnet puede leerlas.

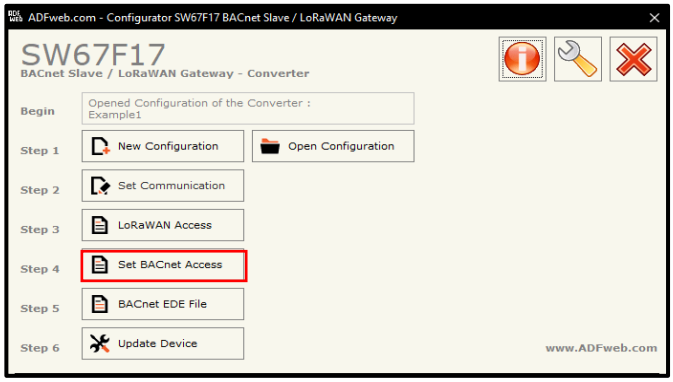


Figura 20. Menú del programa SW67F17 (5.1.4 Configuración del acceso BACnet)

La aplicación requerida necesitará de un objeto por variable a la que deba acceder:

- **Temperatura**. Puesto que acorde al datasheet el rango de este valor va de los -20 a los 50 °C. El objeto más adecuado para esta variable es integerValue (valor entero).
- **Humedad**. A diferencia del anterior el rango de esta variable va del 0 al 99.0 %, dado que siempre será un valor positivo se ha elegido el positiveIntegerValue (valor entero positivo).
- **Estado de la puerta**. Aunque lo lógico sería utilizar un objeto de tipo binario para esta variable, ya que solo puede tener dos estados posibles (abierto y cerrado). No es posible porque el objeto binaryInput lee un byte entero y comprueba el estado del bit menos significativo. Esto supone un problema porque acorde con el datasheet del LDS01 la variable del estado de la puerta se comparte con el estado de la batería del sensor. De forma que de los 2 bytes de la variable los 4 bits más significativos pertenecen al estado del sensor y el resto a la batería. Así que el objeto binaryInput no leería el valor del estado de la puerta sino el bit menos significativo de la batería.

Para poder leer el estado de la puerta lo que se ha decidido es leer el valor como un positiveIntegerValue. La razón es que se puede saber si está abierto viendo cómo cambia el valor. Cuando esté abierto dará un valor aproximado de 35000 mientras que cuando esté cerrado el valor bajará hasta los 3000.

A continuación, se muestra la tabla [fig.21] donde se define la asignación de las direcciones de memoria del convertidor a los objetos de tipo BACnet:

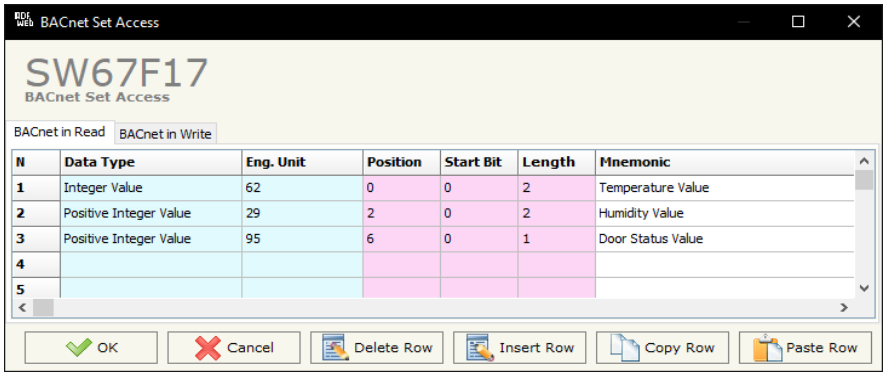


Figura 21. Tabla de asignación de posiciones de memoria a los objetos de tipo BACnet.

5.1.5 Actualización del firmware y configuración del HD67F17-B2-IP-868MHz

Una vez terminadas todas las configuraciones, hay que actualizar el dispositivo para que estas se apliquen. Esto se hace con la opción “Update Device” del SW67F17, al pulsar sobre esta, se abre el siguiente menú [fig.22].

Figura 23. Menú del programa SW67F17

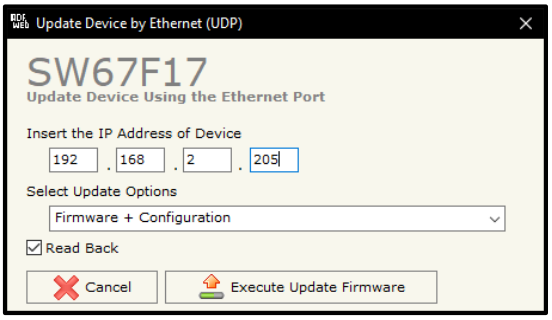


Figura 22. Menú de Actualización del dispositivo.

El menú da la opción de actualizar el firmware, la configuración u ambos simultáneamente. Para poder actualizar el convertidor se debe conectar mediante un cable ethernet al ordenador que este ejecutando el SW67F17. Por defecto, la IP del convertidor es 192.168.2.205, no obstante, se puede cambiar en las opciones mostradas en el apartado [5.1.2]. Cabe destacar que ambos dispositivos deben encontrarse en la misma red IP 192.168.2.XX, para que el SW67F17 sea capaz de encontrar el dispositivo. Al pulsar la opción “Execute Update Firmware” se actualizará automáticamente. Y con esto termina la configuración de la red LoRaWAN.

5.2. La Red BACnet

5.2.1 Elementos de la Red BACnet

Tal y como se explicó en el bloque de soluciones alternativas, se ha optado por el uso de una placa Raspberry como controlador principal. La función de la Raspberry es ejecutar la aplicación que controla el sistema. Concretamente se ha escogido una Raspberry Pi 4 modelo B de 4 Gb de RAM [fig.24].

Especificaciones Raspberry pi 4 Modelo B

- Broadcom BCM2711 Quad core Cortex-A72 (ARM v8)
  64-bit SoC @ 1.5GHz
- 4GB LPDDR4-3200 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless,
- Bluetooth 5.0, BLE Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- 5V DC via USB-C connector (mínimo de 3A)
- Raspberry Pi standard 40 pin GPIO header

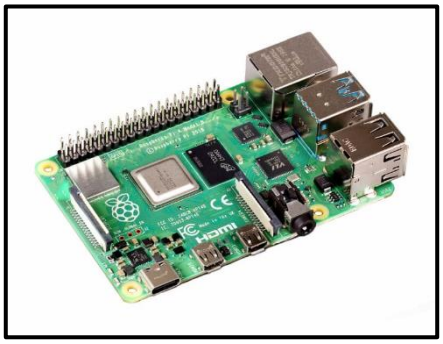


Figura 24. Raspberry pi 4 modelo B

Para comunicar todos los dispositivos es necesario un switch de red. Se ha optado por un switch de 8 puertos de TP-LINK, concretamente el modelo TL-SF1008D [fig.25], puesto que ya se disponía de este dispositivo previamente y no existen requisitos excepto que el número de puertos sea mayor o igual al de dispositivos (para este proyecto 3).

Figura 25. Switch TL-SF1008D

Por último, es necesario un módulo de entradas (inputs) y salidas (outputs) que sea compatible con el protocolo BACnet/IP. Se ha escogido el LIOB-550 de LOYTEC [fig.26], además para alimentar el dispositivo se ha escogido una fuente de alimentación L-POW-2415A del mismo fabricante.

Especificaciones LIOB-550

- Consumo de 4.5W (relees activados)
- 20 Puertos de entrada y salida configurables:
  - 8 Universal inputs
  - 2 Digital Inputs
  - 2 Analog output
  - 8 Digital output (4 de Relees a 6A Y 4 de Triacs a 0.5A)
- 1 Objeto de tipo BACnet por puerto.
- 2 Interfaces Ethernet
- Compatible con BACnet/IP
- Alimentado a 24 V DC o vía L-POW

Figura 26. Módulo de entrada y salidas LIOB-550

Especificaciones L-POW-2415A

- Alimentado 85 – 240 V AC, 50 – 60 Hz
- Suministra 24 V DC 15W

Figura 27. Fuente alimentación L-POW-2515A

5.2.2 Configuración del LIOB-550

A fin de que este dispositivo sea reconocido correctamente por la red BACnet, deben configurarse los objetos de tipo BACnet para cada puerto. De lo contrario, no se podría ni leer ni escribir en estos. Para ello, se utiliza el programa L-INX Configurator [fig.28], software proporcionado por el fabricante que permite la configuración de puertos de forma sencilla.

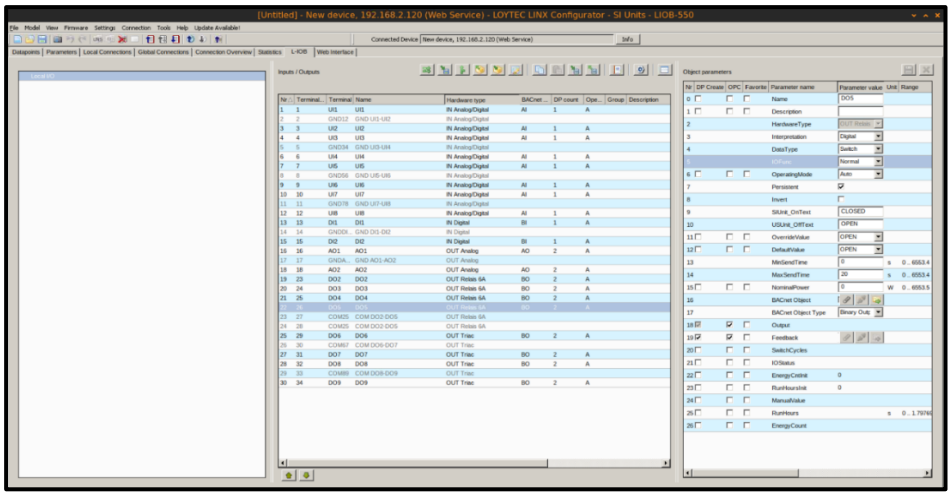


Figura 28. menú de configuración de L-INX Configurator

5.2.3 Búsqueda de dispositivos en la Red

Una vez actualizado el firmware y subida la configurado del LIOB-550. El siguiente paso es comprobar que la red se ha creado de forma satisfactoria. Para ello, se ha realizado un programa de Python para consola el cual no se va a explicar porque no es objeto de este proyecto, pero que se ha incluido junto al código como *BACnetConsole.py.* También se va a emplear el programa *wireshark* para leer los paquetes de la red.

Para ejecutar el programa *BACnetConsole.py*, es necesario un fichero BACpypes.ini que contiene la información del dispositivo y que se explicará más adelante junto con el módulo de BACnet de la aplicación.Tras ejecutar el programa en una consola, se introduce el comando 'Iam'. Dicho mensaje manda un mensaje con la información del dispositivo. Utilizando el wireshark podemos capturar el paquete y comprobar su contenido [fig.29]. En primer lugar, podemos ver como el dispositivo con la dirección 192.168.2.200 (siendo este la raspberry) envía un mensaje a 192.168.2.255 la dirección de retrasmisión en red (o broadcast). También podemos comprobar como la envía al puerto 47808, el puerto que utilizan las comunicaciones BACnet. Por último, cabe destacar como el dispositivo manda una petición i-Am con la información device 599. Esto significa que la raspberry, tiene asignado la deviceID 599 en nuestra red.

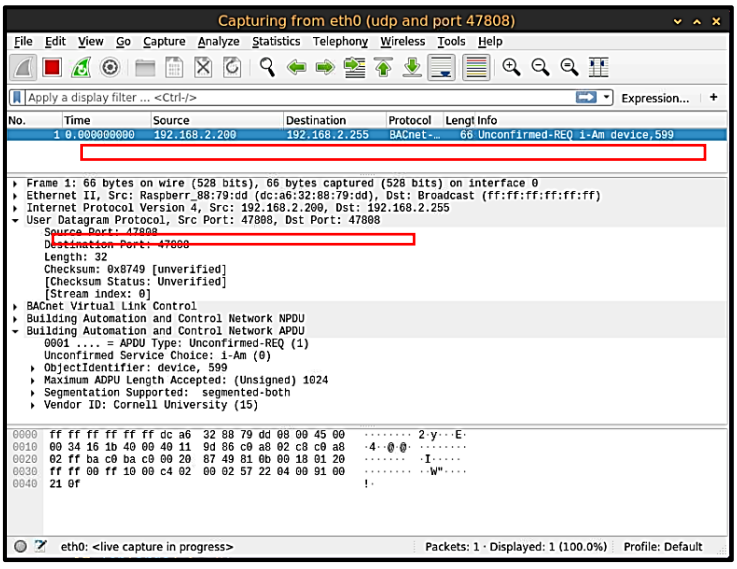


Figura 29. Captura de petición Iam con wireshark

A continuación, vamos a buscar que dispositivos están conectados en la red. Para ello, en la consola introducimos el comando 'whois'. Este comando hará que nuestro dispositivo mande una petición whois y todos los dispositivos en la misma red deberán responder con una petición iam. Además, el programa BACnetConsole, imprime la información de las peticiones Iam por pantalla. Tal y como se muestra en la [fig.30] se pueden apreciar 2 respuestas I-am. La primera pertenece al convertidor de LoRa – BACnet y la segunda al módulo de I/O LIOB-550. Nótese que la configuración del convertidor se corresponde con la elegida en el apartado 5.1.2.

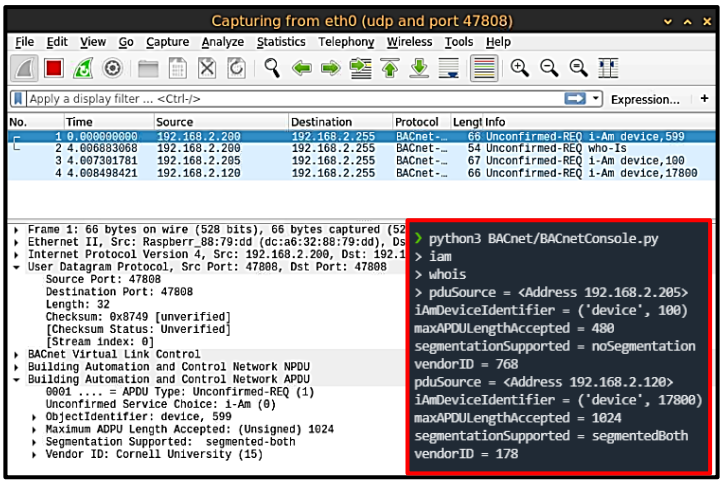


Figura 30. Captura de la petición who-is y resultado en consola.

5.2.4 Lectura de los Objetos BACnet de la Red

Una vez se ha confirmado que la detección de dispositivos funciona correctamente, se va a leer los objetos y propiedades de estos. Para ello se va a usar un programa del propio repositorio de github de la librería bacpypes: https://github.com/JoelBender/bacpypes.

Se ejecuta el archivo *ReadObjectsList.py*, introduciendo la deviceID y su dirección. De esta forma se obtiene la lista de los objetos [fig.31] disponibles, donde se especifica su tipo **objectType**, su id dentro del dispositivo para distinguirlo de objetos de mismo tipo **objectId** y por último una descripción o nombre.

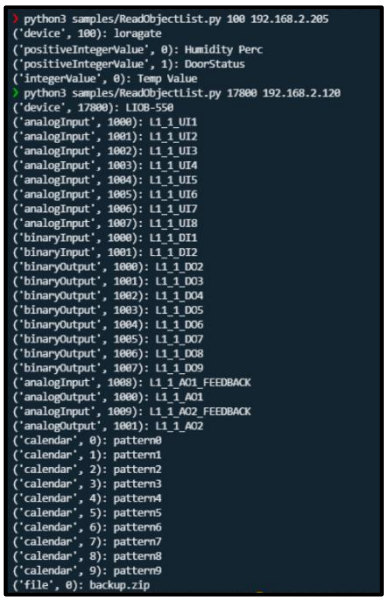


Figura 31. Lista de objetos de la red BACnet.

Una vez se dispone de la lista de objetos podemos ejecutar el archivo *ReadAllProperties.py*, para leer las propiedades. No obstante, para este trabajo la única que se va a emplear es **presentValue**. Esta propiedad almacena el valor de la variable siendo este *active* o *inactive* en los objetos digitales (binaryInput o binaryOutput). O siendo un valor numérico para los objetos analógicos (positiveIntegerValue, integerValue o analogOutput).

5.2.5 Configuración y montaje de los actuadores

Para terminar con la configuración de la red BACnet, es necesario conectar los actuadores al módulo I/O LIOB-550. Los actuadores consisten en cuatro indicadores LED y una salida analógica que es medida con un voltímetro. Los indicadores LEDs necesitan una salida digital (DO), tal y como se ve en la [fig.32], el LIOB-550 dispone de ocho salidas digitales, no obstante, las salidas de la 2 a la 5 inclusive son de relees, mientras que las salidas de la 6 a la 9 son de triacs. Puesto que se pretende demostrar que se podría utilizar para el control de elementos de un sistema de climatización. Se ha optado por el uso de los triacs como también se podría haber usado un releé.

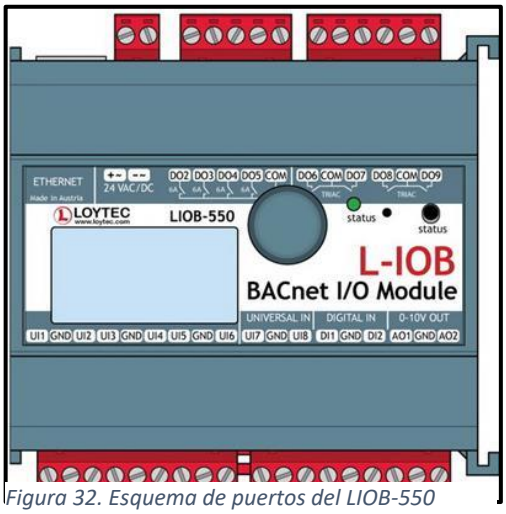


Figura 32. Esquema de puertos del LIOB-550

Por otra parte, para el multímetro se va a usar la salida analógica AO1. La cual puede suministrar hasta 10V.

Indicadores LEDs

Los indicadores LED que se han seleccionado para este proyecto son los 1904XXIP de CML Innovative Technologies [fig.33]. Estos LEDs tienen una tensión directa de aproximadamente 2.2V (Vf) y soportan una intensidad máxima de 20mA.

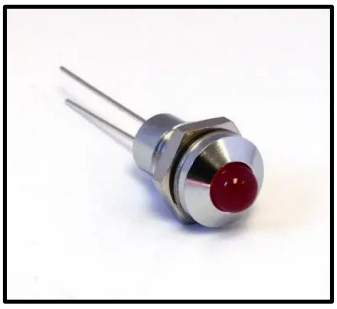


Figura 33. Indicador LED

Para alimentar los LEDs se va a usar una tensión de 12V, por lo que la resistencia necesaria debe ser:

$$R = \frac{V - Vf}{I} = \frac{12 - 2.2}{2mA} \geq 490\ \Omega$$

Puesto que el valor debe ser igual o mayor a 490 Ω se ha escogido un valor de 560 Ω para la elaboración de este proyecto. Para alimentar el indicador se debe suministrar los 12 V en el puerto común a la salida digital, tal y como se muestra en la [fig.34].

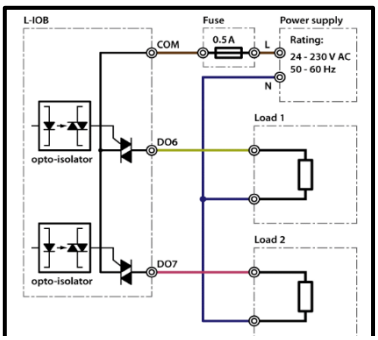


Figura 34. Esquema de las salidas digitales

Multímetro

El multímetro se trata de un circuito integrado con tres displays de 7 segmentos. Tiene tes terminales dos para alimentar el multímetro y marcar el fondo de escala (positivo y tierra) y un tercero que se debe conectar a la tensión que se desea medir.

Se ha diseñado una placa de circuito impreso (PCB) para alimentar los actuadores, reducir el volumen de cables necesarios, fijar la posición los dispositivos y facilitar el montaje. A continuación, se muestran el esquema eléctrico [fig.35] y un modelo 3D de la PCB [Fig.36] realizados con el programa Isis Proteus.

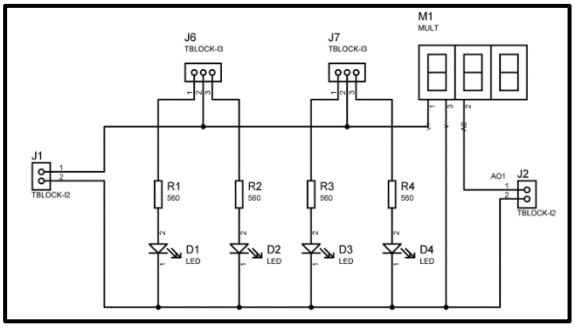


Figura 35. Esquema eléctrico de PCB

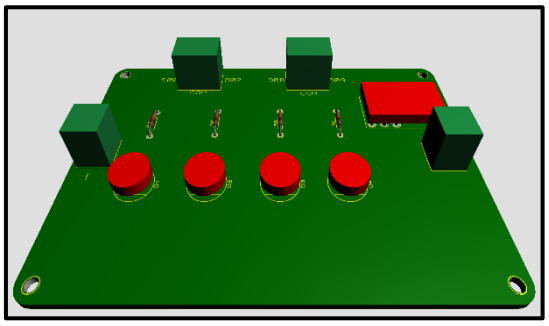


Figura 36. Modelo 3D de la PCB

Por último, se muestra el esquema de conexión entre la PCB y el módulo LIOB–550 [fig.37]:

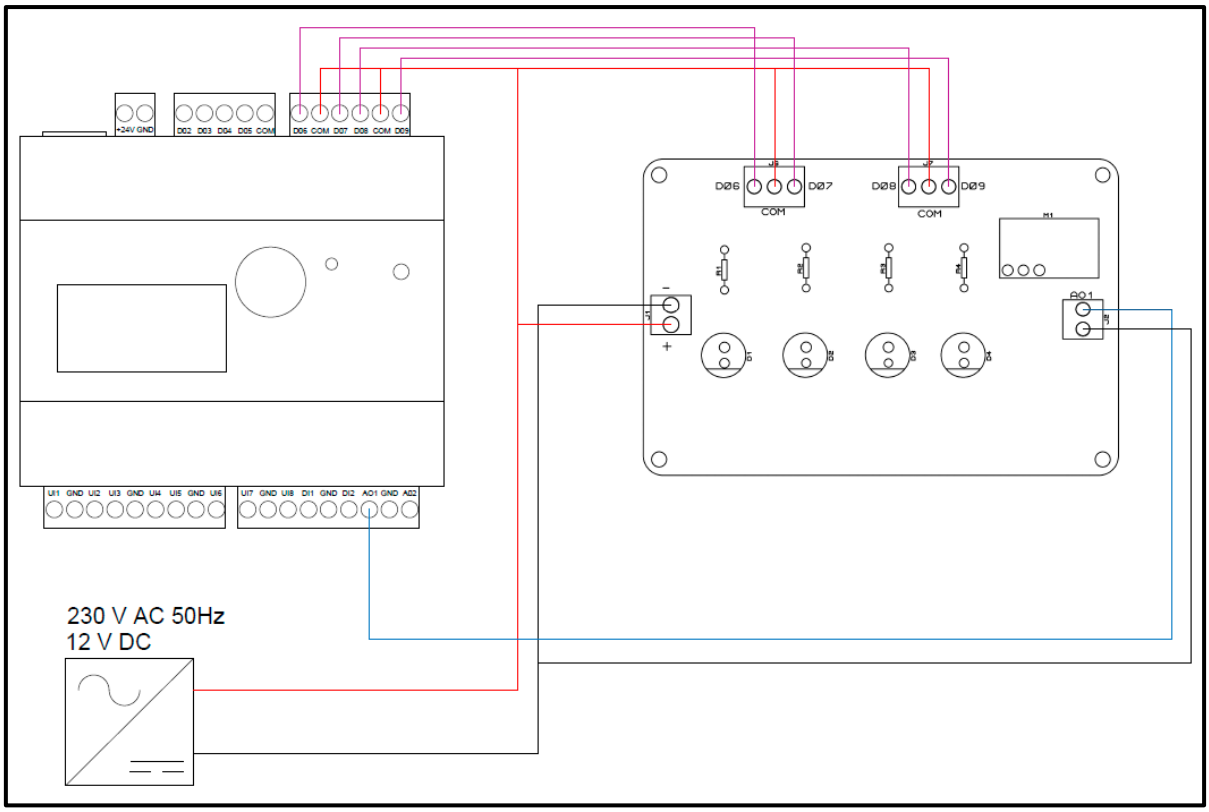


Figura 37. Esquema de conexión entre la PCB y el LIOB-550

5.3. La Aplicación

A continuación, se explican las funciones de los siguientes módulos en los que se ha dividido la aplicación:

5.3.1 Módulo de BACnet

El módulo de BACnet es el que crea el servidor BACnet y gestiona tanto las peticiones como las respuestas. Para su programación se ha utilizado la librería de Python **bacpypes,** la cual incluye una serie de funciones y objetos necesarios para crear la aplicación y funciones del protocolo.

El módulo BACnet se puede dividir en tres bloques atendiendo a su naturaleza. El primero es la definición del objeto *BacnetApplication,* el segundo la implementación de las funciones básicas del protocolo y el tercero la inicialización de la aplicación BACnet. A continuación, se detallan en profundidad las características de cada uno de estos bloques.

Bloque 1. BacnetAplication

Define un objeto de clase BacnetApplication que hereda los métodos y atributos del objeto BIPSimpleApplication definido por bacpypes, y se le implementan los métodos necesarios para poder gestionar las peticiones que le lleguen y responder con peticiones de confirmación.

Este objeto es el encargado de realizar las peticiones de lectura, escritura y búsqueda de dispositivos, así como interpretar las respuestas de la red y responder a peticiones de otros dispositivos.

Bloque 2. Funciones básicas del protocolo BACnet

BACnet define una serie de funciones para reconocer dispositivos y poder interactuar con ellos, para este proyecto se han implementado siguientes:

- **I am,** el dispositivo envía un mensaje indicando su dirección IP y el número ID de dispositivo que tiene asignado en la red.
- **Who Is,** el dispositivo envía un mensaje *whois* y todos los dispositivos de la red que reciben este mensaje responden con un mensaje *I-am.* Este mensaje se utiliza para detectar dispositivos en la red, pero dado el riesgo de desbordamiento por mensajes I-am no es el método más conveniente en redes grandes. El mensaje who-is se puede usar en globalBroadcast o también puedes preguntar a determinadas direcciones o incluso a un rango de IDs de dispositivos.
- **Read Property,** el dispositivo envía una petición para leer el valor de una propiedad del objeto BACnet de un dispositivo. Dicho dispositivo devuelve un mensaje con el valor o con un mensaje de error.
- **Write Property,** el dispositivo envía una petición para escribir sobre el valor de una propiedad de un objeto bacnet de un dispositivo. Dicho dispositivo devuelve un mensaje de reconocimiento *ack* si la operación se ha realizado correctamente.

A excepción de la función I-am, el resto de las funciones requieren de parámetros para poder funcionar correctamente. A continuación, se profundiza un poco más en las otras tres funciones puesto que tienen un poco más de complejidad:

Who-Is

Esta función tiene varios modos de funcionamiento dependiendo de los argumentos que se introduzcan [fig.38]:

- **Sin argumentos.** Petición a todas las direcciones.
- **Un solo argumento.** Petición a una única dirección siendo esta el argumento introducido.
- **Dos argumentos.** Petición a todas las direcciones acotadas en un rango concreto de id.
- **Tres argumentos.** Petición a una única dirección, pero en un rango de concreto de id.
- **Más de tres argumentos o algún argumento inválido.** Saltará la excepción y no se realizará la petición devolviendo un mensaje de error.

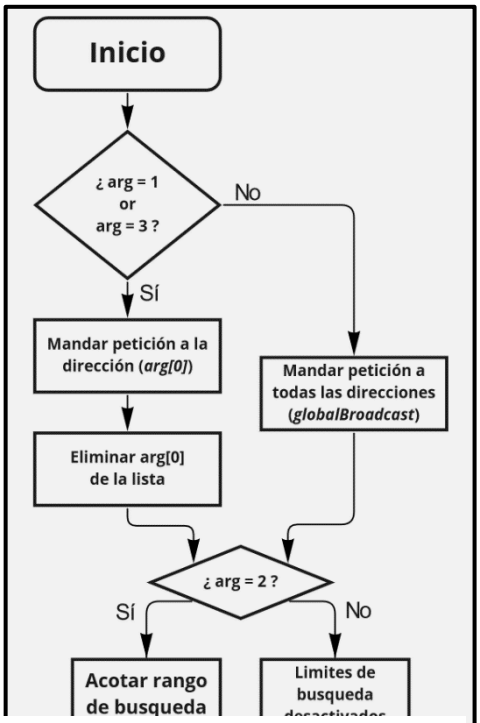


Figura 38. Diagrama de flujo de petición Who-Is

Read Property - Write Property

A diferencia de la función Who-is, las funciones ReadProperties y WriteProperties solo tienen un modo de funcionamiento. No obstante, requieren de una serie de parámetros para poder funcionar correctamente. Estos son:

- Dirección IP del dispositivo [addr].
- ID del Objeto que se quiere leer/escribir [obj_id] *.
- Propiedad que se quiere leer/escribir [prop_id].
- Valor que se quiere escribir (solo para escritura) [value].

(*) El valor de obj_id esta a su vez formado por dos valores separados por ":", siendo los valores el tipo de objeto objectType y la id del objeto para ese dispositivo objetID (pueden haber 2 id iguales para 2 tipos de objetos distintos por ello es necesario especificar el tipo).

En este proyecto, la dirección IP de cada dispositivo es fija. Siendo estas:

- 192.168.2.205 dirección del convertidor LoRa – BACnet.
- 192.168.2.200 dirección de la Raspberry Pi.
- 192.168.2.120 dirección del módulo I/O LIOB-550.

Aprovechando esto, se ha decidido implementar funciones para cada uno de los objetos BACnet sobre los cuales se quiere interactuar. Dichas funciones tienen los valores de los argumentos ya predefinidos y de esta forma se hace el código más organizado. A continuación, se listan las funciones junto con los argumentos:

Funciones de lectura:

- **Do_readTemp**: Lee la temperatura del sensor LHT52 y realiza la conversión del valor recibido al valor real dividiendo el recibido entre 100. Si la lectura falla devuelve ´ValueError´. Los argumentos son:
    - **Dirección** [‘192.168.2.205’]
    - **Object_id** [‘integerValue:0’]
    - **Prop_id** ['presentValue']
- **Do_readHumidity**: Lee la humedad del sensor LHT52. También realiza la conversión del valor recibido al valor real, pero dividiendo entre 10. Y Si la lectura falla también devuelve ´ValueError´. Los argumentos son:
    - **Dirección** [‘192.168.2.205’]
    - **Object_id** [‘positiveIntegerValue:0’]
    - **Prop_id** ['presentValue']
- **Do_readDoorStatus:** Tal y como se explicó en el punto 5.1.4, este sensor lee el primer byte del sensor LDS01. Si el valor es mayor a 30000 devuelve el valor ‘OPEN’ de lo contrario devuelve ‘CLOSED’. Los argumentos son:
    - **Dirección** [‘192.168.2.205’]
    - **Object_id** [‘positiveIntegerValue:1’]
    - **Prop_id** ['presentValue']

Funciones de escritura:

- **Do_writeHeaterStatus**: Cambia el valor del puerto DO6 por el introducido como argumento. Sus argumentos son:
    - **Dirección** [‘192.168.2.120’]
    - **Object_id** [‘binaryOutput:1004’]
    - **Prop_id** ['presentValue']
    - **Value** (introducido como argumento)
- **Do_writeCoolingStatus**: Cambia el valor del puerto DO7 por el introducido como argumento. Sus argumentos son:
    - **Dirección** [‘192.168.2.120’]
    - **Object_id** [‘binaryOutput:1005’]
    - **Prop_id** ['presentValue']
    - **Value** (introducido como argumento)
- **Do_writePauseMode**: Cambia el valor del puerto DO8 por el introducido como argumento. Sus argumentos son:
    - **Dirección** [‘192.168.2.120’]
    - **Object_id** [‘binaryOutput:1006’]
    - **Prop_id** ['presentValue']
    - **Value** (introducido como argumento)
- **Do_writeVentStatus**: Cambia el valor del puerto DO9 por el introducido como argumento. Sus argumentos son:
    - **Dirección** [‘192.168.2.120’]
    - **Object_id** [‘binaryOutput:1007’]
    - **Prop_id** ['presentValue']
    - **Value** (introducido como argumento)

- **Do_writeAnalogValue**: Cambia el valor del puerto AO1 por el introducido como argumento. Sus argumentos son:
    - **Dirección** [‘192.168.2.120’]
    - **Object_id** [‘analogOutput:1000’]
    - **Prop_id** ['presentValue']

- **Value** (introducido como argumento)

También cabe mencionar que, para que las funciones write y read se puedan llevar a cabo, es necesario utilizar un IOCB (Input Output Control Block). El IOCB es un objeto que almacena los parámetros para una operación, y los devuelve una vez se ha llevado a cabo. Bacpypes tiene definido uno en su librería.

Bloque 3. Inicialización de la aplicación BACnet

Se ha creado una función que realice automáticamente la creación de los objetos necesarios e inicie la tarea que mantiene la aplicación en funcionamiento. Primeramente, se declaran las variables globales BACnet_device y BACnet_app, estas variables contendrán los objetos necesarios para el funcionamiento de la aplicación.

A continuación, se declara una variable *args* que debe almacenar los parámetros de configuración de la red. Para ello, se crea un archivo bajo el nombre "*BACpypes.ini*" en el directorio donde se ejecuta el código principal (módulo aplicación). Este archivo contiene toda la información necesaria de la red BACnet [fig.39]. Luego se utiliza una función que lee el archivo y almacena los parámetros en la variable *args*.

```
[BACpypes]
objectName: Argos
address: 192.168.2.200/24
objectIdentifier: 599
maxApduLengthAccepted: 1024
segmentationSupported: segmentedBoth
maxSegmentsAccepted: 1024
vendorIdentifier: 15
```

Figura 39. Contenido del archivo BACpypes.ini

Después se crea el objeto *BACnet_device* a partir de los datos obtenidos del archivo *BACpypes.ini.* Este objeto definirá nuestro dispositivo dentro de la red BACnet. Luego, se crea el objeto *BACnet_app* a partir de *BACnet_device* y la dirección IP definida en nuestro fichero *BACpypes.ini*.

Por último, se realiza la gestión del hilo de ejecución (o thread). Se define una tarea que al ser iniciada va a ejecutar en segundo plano y de forma paralela la función BACpypesThread. Dicha función inicia la aplicación BACnet y la mantiene en ejecución hasta que todas las tareas principales del programa se hayan terminado.

5.3.2 Módulo de Datos

El módulo de Datos se encarga de gestionar el estado y el valor de las variables del sistema. Estas variables se declaran como atributos de la clase RoomData y el resto de los módulos acceden a ellas a través de setters y getters. Las propiedades de la clase se dividen en dos bloques, los valores de Input los cuales hacen referencia a los datos que entran al sistema mediante sensores. Y los datos de Output, que representan el estado de los actuadores después de haber calculado la respuesta.

Valores de los sensores (Inputs)

- ***Temp.*** Valor del sensor de temperatura °C, leído directamente del sensor LHT52.
- ***TempObjective.*** Valor de temperatura deseado por el usuario, con un valor por defecto de 28 °C.
- ***TempExt.*** Valor de temperatura del exterior de la instalación, con un valor por defecto de 28 °C.
- ***Humidity.*** Valor de la humedad relativo en %, leído directamente del sensor LHT52.
- ***DoorStatus.*** Estado de la puerta, 'OPEN' si abierta y 'CLOSED' si cerrada.
- ***WindowStatus.*** Estado de la ventana, 'OPEN' si abierta y 'CLOSED' si cerrada. Por defecto el valor de esta variable está fijado como CLOSED.
- **AnalogValue.** Valor analógico en V, valor por defecto 0.0 V

Variables de estado de los actuadores (Outputs)

- ***HeaterStatus.*** El estado de la calefacción, 'ON' para encendido y 'OFF' para apagado.
- ***CoolingStatus.*** El estado del sistema de refrigeración, 'ON' u 'OFF'.
- **PauseMode.** Modo de pausa del sistema, 'ON' activado y 'OFF' apagado.
- **VentStatus.** El estado de la ventilación, 'ON' u 'OFF'.

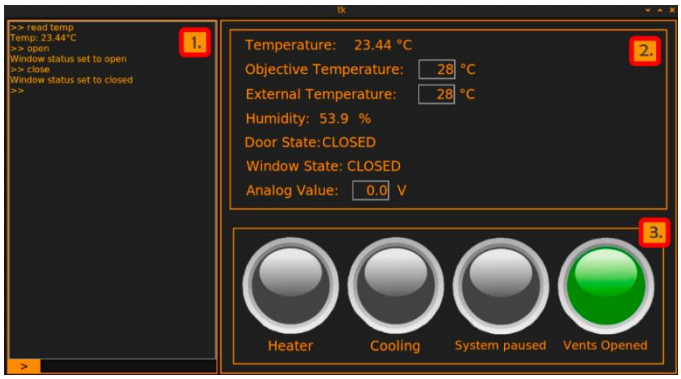
También implementa una función para leer directamente el valor de los sensores usando las funciones del módulo BACnet y actualizar su valor en el objeto RoomData

5.3.3 Módulo de Interfaz

El módulo de interfaz tiene varios propósitos. El primero es mostrarle al usuario la información del sistema de forma gráfica. El segundo es permitir al usuario realizar cambios en las variables del sistema, mediante widgets de tipo entry. Por último, tiene una última finalidad de ejecutar comandos durante la ejecución del programa permitiendo así la realización de pruebas.

Este módulo se ha programado utilizando la librería de Python tkinter. Se crea un objeto interfaz que en su construcción crea un objeto tk, esto crea la ventana general. Luego, sobre esta, se van situando los distintos elementos o widgets. La interfaz se divide en tres partes tal y como se muestra en la [fig.40]:

1. **La consola.** Permite introducir los comandos, muestra el historial de comandos y respuestas del sistema.
2. **El panel de Datos.** Muestra el valor de cada una de las variables de entrada (inputs) y permite modificar algunas.
3. **El panel de LEDs.** Muestra el estado de las variables de salida (outpus), cuando una salida esta encendida el led se "enciende".



Además de crear la interfaz gráfica, como bien se ha mencionado antes, el módulo de interfaz implementa funciones y atributos para controlarla. Estos se pueden dividir dos grupos:

Sistema de Flags

El sistema de Flags es un mecanismo de alertas cuya finalidad es permitir que el módulo de la interfaz pueda realizar peticiones al módulo supervisor. Ante determinados eventos como cambio de valor de un *entry widget* (las entradas) o la inserción de un comando en la consola, se eleva el Flag (se fija el valor en True). Una vez realizada la acción pertinente, se limpia la alerta (se cambia el valor a False).

Los Flags se declaran como atributos del objeto interface. También se implementan los métodos *get* para obtener su valor, *raise* para activar la alerta (valor a True) y *clear* para limpiar la alerta (valor a False). Los Flags que se han empleado son:

- **openWindowRequest.** Cambiar el valor de windowStatus a Open.
- **closeWindowRequest.** Cambiar el valor de windowStatus a Closed.
- **readAnalogValueRequest.** Actualizar el valor de AnalogValue en la base de datos.
- **readExtTempRequest.** Actualizar el valor de tempExt en la base de datos.
- **readObjTempRequest.** Actualizar el valor de tempObj en la base de datos.
- **updateInterfaceRequest.** Actualizar la interfaz.
- **exitRequest.** Petición de salida.

Funciones de interfaz

Una serie de funciones que gestionan los cambios en los distintos elementos de la interfaz y se encargan de que estos funcionan correctamente. A continuación, se enumeran todas las funciones junto con una explicación de su funcionamiento:

SetLedOn/Off. Son funciones encargadas de “encender” o “apagar” los LEDs de la interfaz. Lo que realmente hace es cambiar la imagen del widget por otra. Hay una función de encender y apagar por cada led.

CleatText. Elimina todas las líneas de texto de la consola.

UpdateText. Activa la escritura en la consola, escribe el comando introducido en la consola, acto seguido escribe la respuesta y vuelve a desactivar la escritura en la consola.

ReadEntry. Lee el valor del terminal de la consola, y llama a la función ReadCommand con el valor de la terminal como atributo.

ReadCommand. Primero divide la cadena de texto introducida por la función anterior en dos elementos usando el primer espacio como separador. Dichos elementos son introducidos en una tupla [*command*]. Acto seguido si la tupla tiene rango mayor que uno separa el comando de los argumentos y los introduce en una segunda tupla [*args*]. Después compara el primer valor de *command* con una serie de valores establecidos. Si el valor de *command* coincide con alguno de los valores definidos se ejecutan las funciones para realizar la acción.

Los valores definidos por como comandos son:

- **Test.** Se escribe "*This is a test :)*" en la consola. Este comando sirve para comprobar que la función updateText funciona correctamente.
- **Clear.** Se ejecuta la función ClearText, también se ejecuta con el comando **'cls'.**
- **Exit.** Se activa la petición de salida de la interfaz (del sistema de flags) y tras dos segundos destruye el objeto interfaz.
- **WhoIs.** Envía un mensaje de tipo WhoIs en la red BACnet.
- **IAm.** Envía un mensaje IAm en la red BACnet.
- **Read.** Se realiza una petición del tipo ReadProperty en la red BACnet, si el argumento de la tupla *args* se corresponde a **temp**, se leerá el valor de temperatura del propio sensor. Si se corresponde a **humidity**, leerá el valor de humedad. Y si es **door**, leerá el estado de la puerta. Por el contrario, si no recibe argumentos o recibe más de uno, dará un error.
- **Open/Close.** Cambia el valor de la variable WindowStatus (estado de la ventana), abierto si **open** y cerrado si **close**. Puesto que se quiere cambiar en la base de datos, se realiza activando la petición de actualizar el estado (abierto o cerrado) con el sistema de flags.
- **Update.** Realiza la petición de actualizar elementos de la interfaz del sistema de flags
- **Led.** Este comando sirve para "encender" o "apagar" los leds. Envia una petición writeProperty para cambiar el estado del led seleccionado al valor deseado. Luego utiliza la función setLed para modificar la interfaz. Cabe destacar que realmente no se cambia el valor en la base de datos. Por lo que su principal finalidad es poder asegurar que los leds están operativos. La sintaxis de este comando es led + "color del led" + "estado (on/off)".

Cabe destacar que los comandos se pueden introducir en mayúsculas completamente o minúsculas completamente puesto que se usan el método lower para convertir todos los caracteres a minúscula. También que si el comando no ha sido reconocido devolverá un mensaje de "comando no definido".

5.3.4 Módulo del Supervisor

El módulo del Supervisor es el que realiza la gestión de los objetos explicados en los módulos anteriores y ejerce las acciones para controlar el sistema. Al inicializarse [fig.41], carga los datos de la base de datos en la interfaz y crea dos tareas. Cuando estas tareas se inician, se ejecutan una serie de acciones cíclicamente. Dichas tareas son la tarea **Supervisor** y la tarea **Aplicación**. También se definen las funciones encargadas de leer las peticiones del módulo de interfaz, actualizar el estado de la base de datos y actualizar la interfaz utilizando la base de datos. A continuación, se explican las funciones y las tareas a realizar:

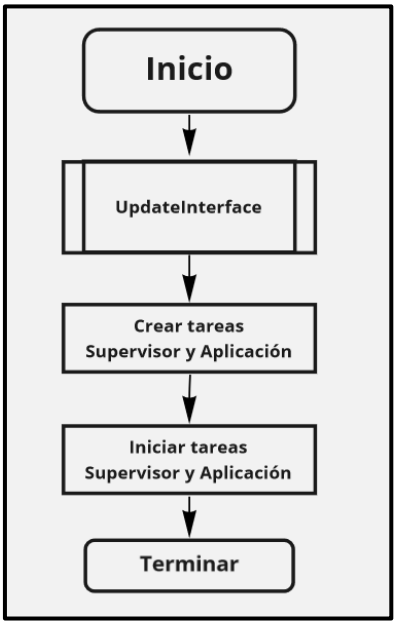


Figura 41. Diagrama de flujo de inicialización del objeto supervisor

ReadFlags

Esta función tiene como objetivo leer los flagRequests del objeto interfaz y realizar las operaciones correspondientes tal y como se muestra en el siguiente diagrama de flujo [fig.42]:

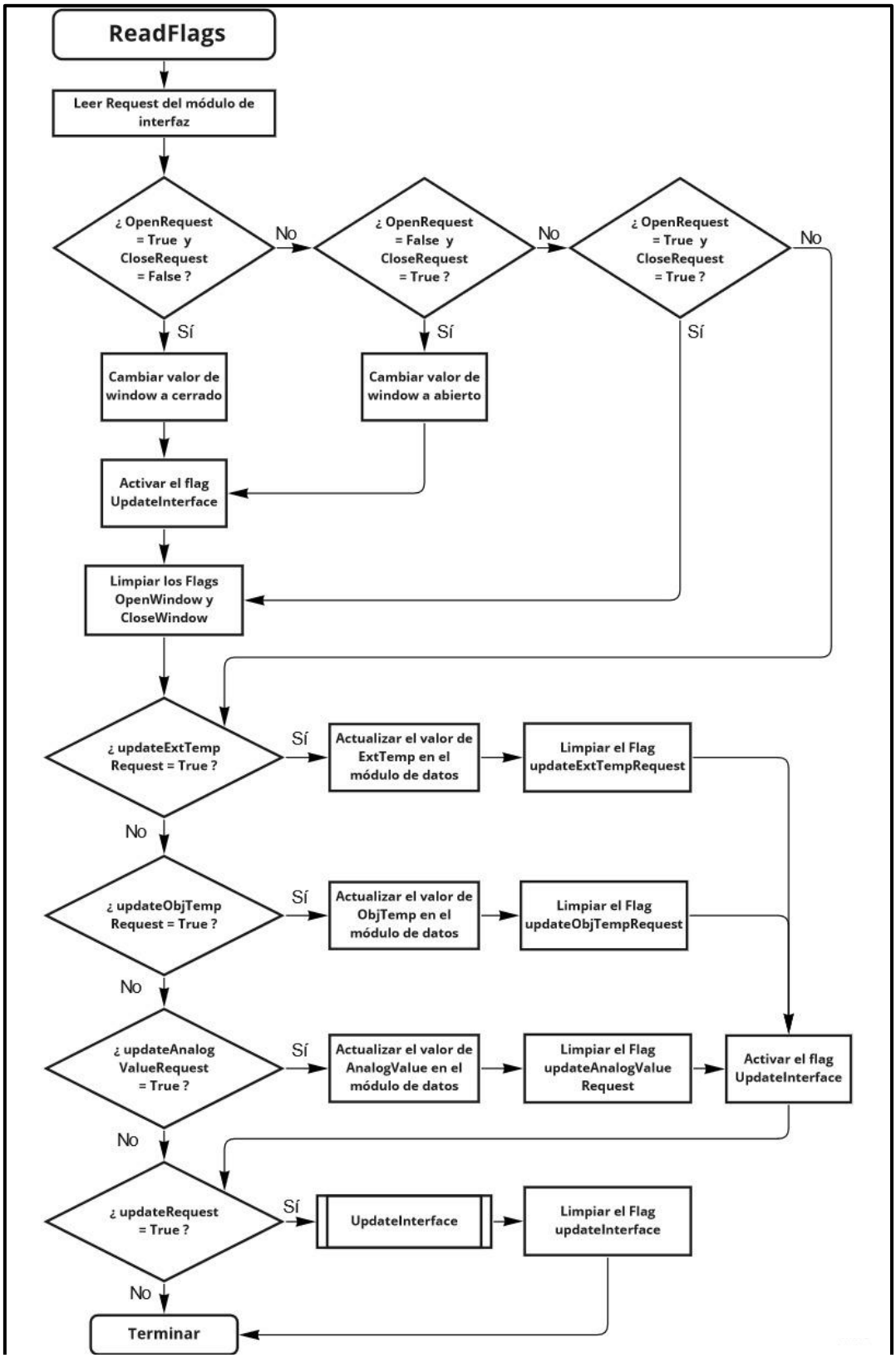


Figura 42. Diagrama de flujo de la función ReadFlags.

UpdateStatus

La siguiente función lee los valores de la base de datos y calcula el estado de cada actuador en base a unas reglas:

- Si la ventana o la puerta está abierta el sistema se pausa.
- Si el sistema no está en pausa y la temperatura actual es distinta a la temperatura objetivo el sistema se inicia. De forma que si la temperatura objetivo es menor se enciende el calefactor (heater) y si es mayor se activa la refrigeración (cooling). Excepto si la temperatura externa es favorable y hay una diferencia de al menos 2 grados con la temperatura actual.

El proceso de cálculo se realiza siguiendo el siguiente diagrama [fig.43]:

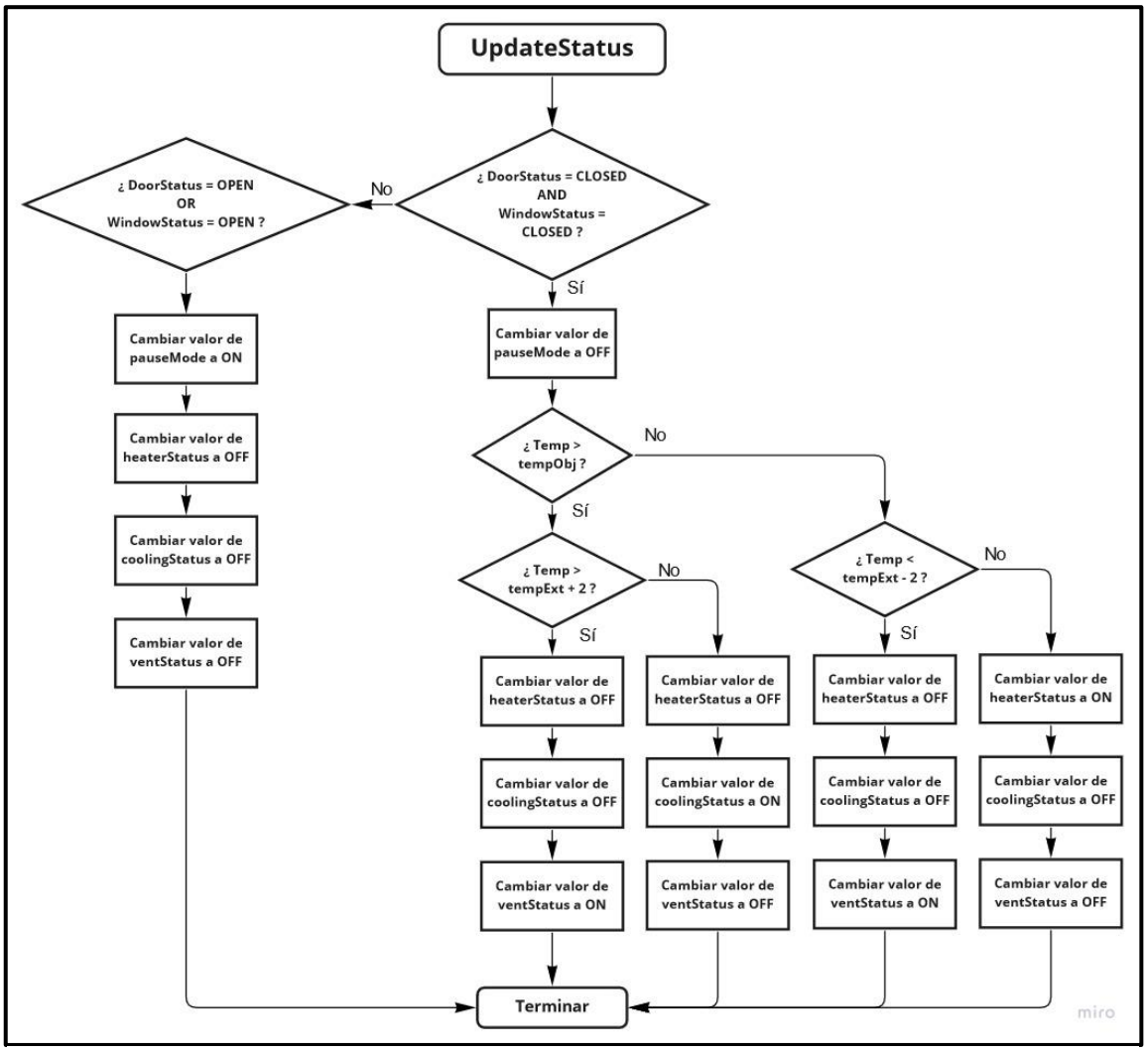


Figura 43. Diagrama de flujo de la función updateStatus.

UpdateInterface

La finalidad de esta función es actualizar los datos de la base de datos para luego cargarlos dentro de la interfaz [fig.44]. Primero lee los valores de los sensores usando la función del módulo de datos. A continuación, llama a la función updateStatus para actualizar el valor de las salidas acorde a las reglas establecidas. Por último, se modifican los valores de las salidas digitales del LIOB-550 y se actualiza el valor de los widgets de la interfaz.

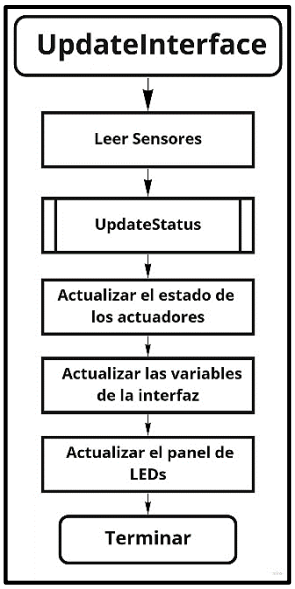


Figura 44. Diagrama de flujo de la función updateInterface.

Tarea Supervisor

Esta tarea ejecuta cada 0.2 segundos la tarea readFlags, la razón por la que el intervalo de tiempo es tan breve es porque de no ser así, las variables de temperatura externa, temperatura objetivo, valor analógico entre otras tardarían mucho tiempo en actualizarse en la base de datos incluso podrían llegar a borrarse. Esta tarea se ejecuta mientras el Flag ExitRequest sea False.

Tarea Aplicación

En cuanto a la tarea aplicación, su trabajo es ejecutar la función updateInterface cada 5 min. Al contrario que la tarea supervisor, esta tarea se ejecuta mientras haya otras tareas principales en ejecución.

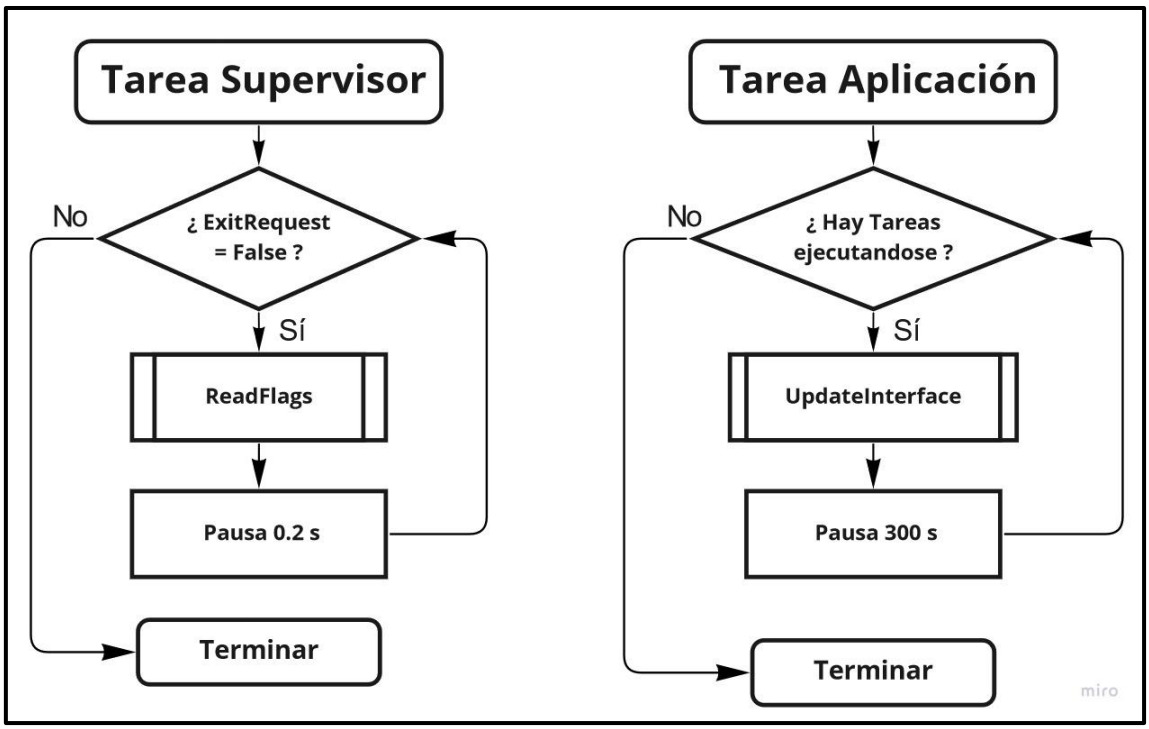


Figura 45. Diagramas de flujo de la tarea supervisor y tarea aplicación.

5.3.5 Módulo de Aplicación

Para terminar, el módulo aplicación es el que se ejecuta al iniciar el programa y es el que se encarga de crear los objetos. Primero crea el objeto de interfaz y lo almacena en la variable app. Luego inicia la aplicación BACnet. Seguidamente crea la base de datos y la almacena en la variable dataModule. Después crea un objeto de tipo Supervisor que es el que crea las tareas supervisor y aplicación. Por último, se ejecuta un bucle infinito para mantener la interfaz operativa con la función de tkinter mainloop().

En total la aplicación tiene cuatro hilos de ejecución paralelos. El principal es el que crea los objetos y gestiona los eventos de la interfaz. El segundo es la aplicación BACnet que lee y envía las peticiones a la red. Los dos últimos son la tarea Aplicación y Supervisor del módulo del supervisor. Cabe destacar que al utilizar el comando 'exit' en la consola, se envía la petición que termina la tarea supervisor y se destruye el objeto interface. Los otros dos hilos debido a sus características no se pueden destruir de esta forma, para poder terminarlos se han definido como tareas de tipo Daemon las cuales terminan si no hay ninguna tarea que no sea de tipo Daemon en ejecución.

6. Conclusiones

Durante la elaboración del presente trabajo, se han pretendido lograr una serie de objetivos. Inicialmente, se realizó un trabajo de investigación sobre los protocolos de comunicación existentes para la automatización de edificios. La razón de este estudio surge a raíz del creciente interés en las "*Smart Cities*" como método para combatir la crisis energética. Este interés proviene de la necesidad de crear sistemas de control de edificios que sean más eficientes e inteligentes empleando las TIC junto a las técnicas de IoT.

El estudio realizado pretende analizar las principales características de los diversos protocolos, así como sus diferencias y sus posibles aplicaciones en los "*Smart buildings".* Una vez terminado el estudio, se trató de crear un sistema de control funcional para un sistema HVAC integrando sensores IoT basados en LoRaWAN, así como el uso del protocolo de comunicación para la automatización BACnet. Además, se procuró que el sistema fuese lo más completo posible. Por ello, se incluyeron una interfaz de usuario y una serie de estrategias para el ahorro de energía.

El proyecto se ha enfocado como prueba de concepto de la integración de LoRaWAN con el protocolo BACnet en sistemas reales. También se ha querido que este trabajo sirviese de guía para el diseño de este tipo de sistemas.

Este trabajo me ha servido para desarrollarme tanto de ingeniero como de programador. En el desarrollo de este proyecto he tenido que aprender el lenguaje de programación Python, he aprendido a realizar interfaces gráficas para usuarios, he aprendido a ejecutar líneas de código de forma paralela, he aprendido a diseñar y configurar redes BACnet, entre muchas otras cosas.

La principal dificultad que he encontrado en este trabajo ha sido la falta de información acerca de los protocolos, especialmente de como leer los sensores de la red y la implementación de la aplicación BACnet al sistema de control. En muchas ocasiones me he visto obligado a pasarme horas probando ingeniería inversa hasta entender cómo funcionaba.

Para terminar, es interesante hablar de las futuras líneas de este proyecto puesto que ayudan a comprender el alcance que puede llegar a tener este trabajo:

- Implementación de una base de datos para almacenar los valores de todas las variables en uno o varios ficheros a lo largo del tiempo (5 – 10 min). De forma que se puedan realizar representaciones gráficas para su posterior análisis. Permitiendo así integrarle al sistema un cierto grado de previsión.
- Añadir un sistema que permita reiniciar la aplicación ante un apagón o tras la interrupción del programa.
- Integrar una función para subir y bajar los valores de la instalación a una nube, de forma que se pueda monitorizar y controlar el sistema de forma remota.
- Por último, hay que mencionar que durante la configuración del LIOB-550 se descubrió que este tenía una opción que permitía medir el consumo del actuador conectado a cada puerto. Dicho valor se puede leer de la red BACnet y se podría almacenar en la base de datos del primer punto. Esto permitiría calcular y graficar el consumo medio en cada mes del año, comparar el consumo entre años y como consecuencia gestionar de forma más eficiente el consumo energético.

7. Bibliografía

https://www.cursosaula21.com/modbus-que-es-y-como-funciona/

https://es.wikipedia.org/wiki/Modbus

http://www.sapiensman.com/tecnoficio/computacion/redes_MODBUS.php

https://www.eeymuc.co/31-protocolo-modbus/

https://es.wikipedia.org/wiki/Modelo_OSI#Capa_f%C3%ADsica_-_Capa_1

https://domoticayhogar.com/domotica-knx/

https://www.knx.org/wAssets/docs/downloads/Marketing/Flyers/KNX-Basics/KNX-Basics_es.pdf

https://www.domoticadomestica.com/que-es-la-domotica-knx-y-en-que-consiste/org

http://www.bacnet.org/

https://automation-networks.es/glossary/bacnet

https://static.weg.net/medias/downloadcenter/hf6/hca/WEG-CFW100-bacnet-manual-del-usuario-10007481066-es.pdf

https://bacpypes.readthedocs.io/en/latest/

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

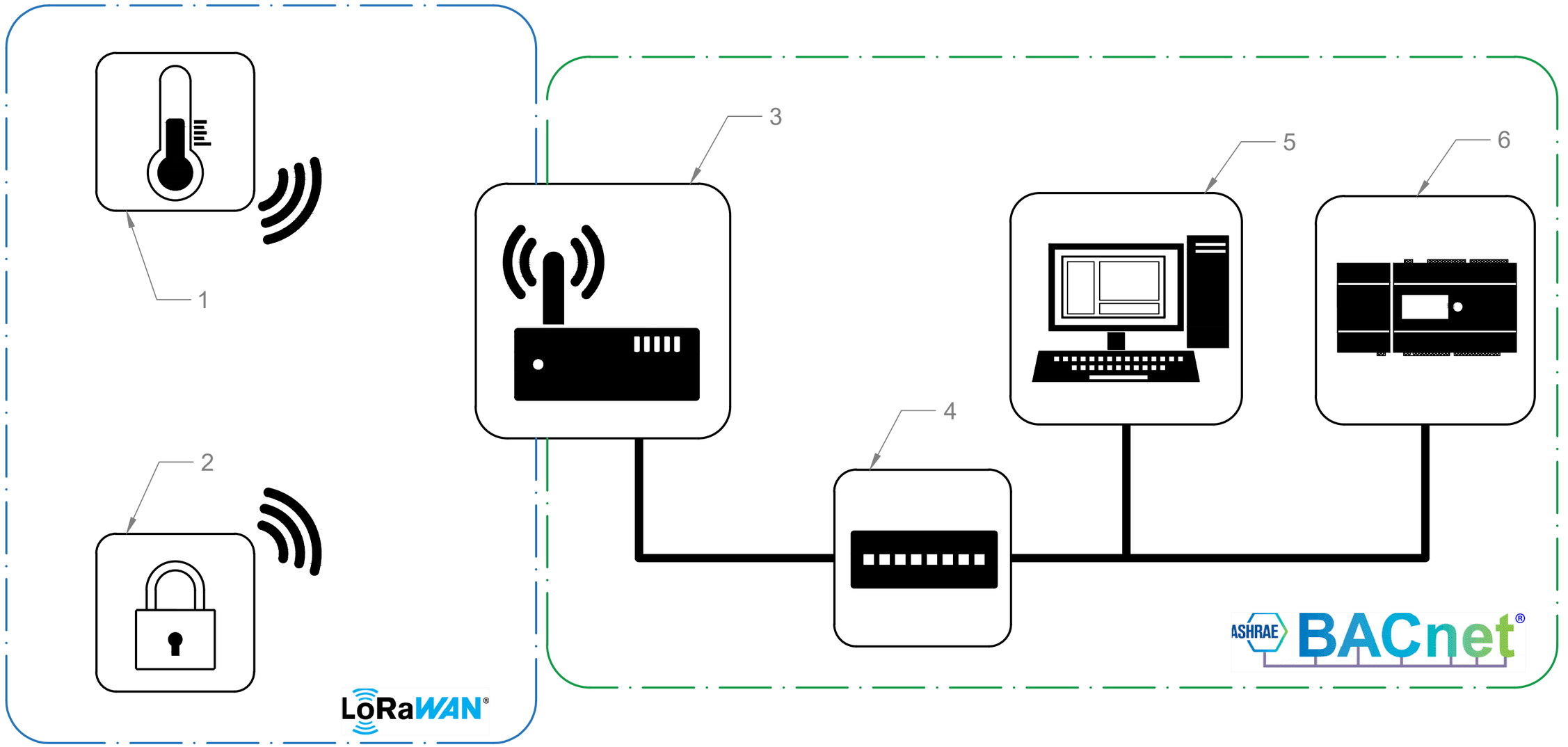
Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización

II. Planos

Autor: D. Omar Ben Abdelouahab Pereira

Tutor: D. Ángel Perles Ivars

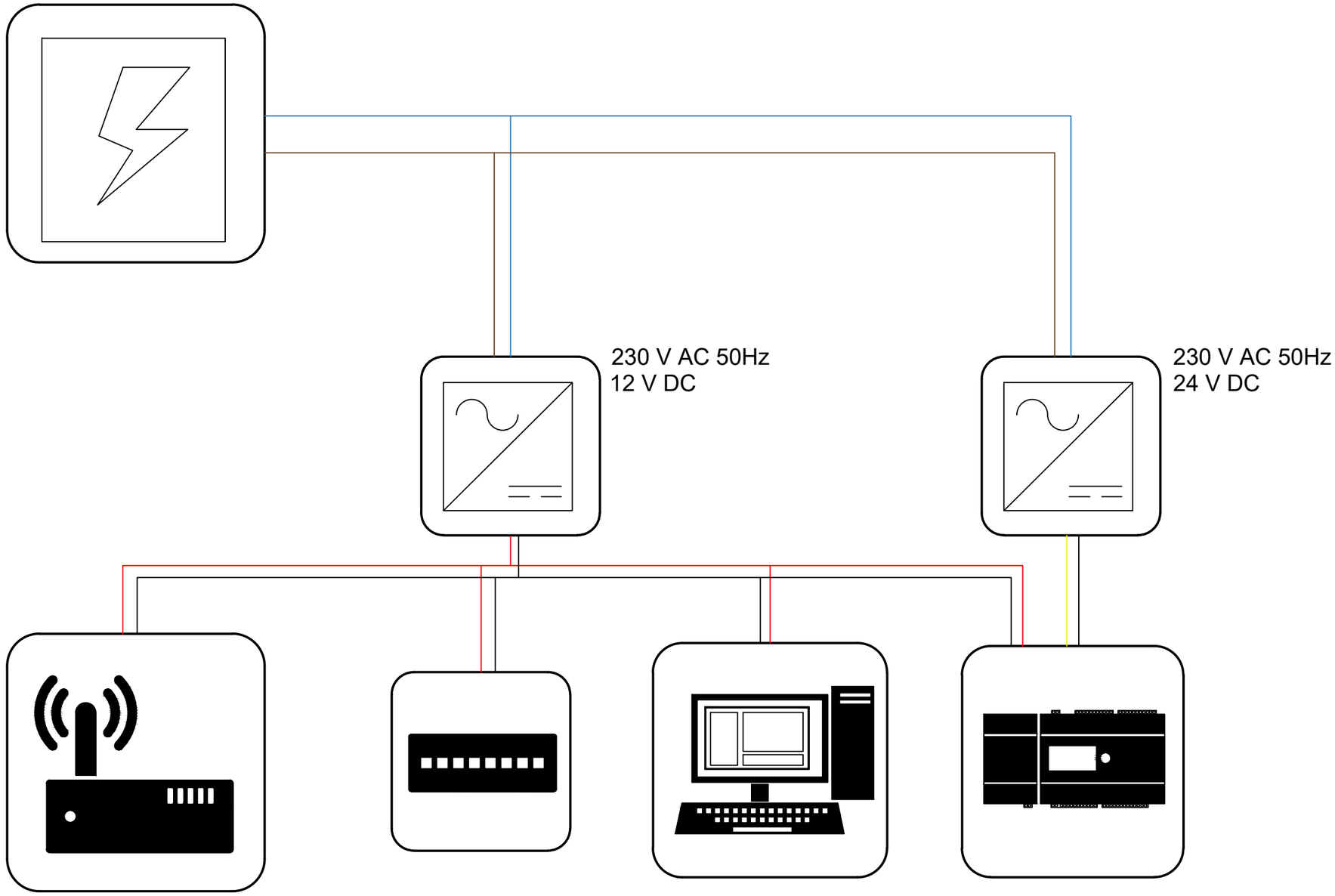
Índice



Marca	Cantidad	Denominación	Referencia
1	1	Sensor LHT52 de Dragino	Comercio
2	1	Sensor LSD01 de Dragino	Comercio
3	1	Convertidor LoRa–Bacnet HD67F17–B2–IP–8C8MHz	Comercio
4	1	Switch de red 8 puertos	Comercio
5	1	Raspberry Pi 4 modelo B	Comercio
6	1	Módulo de I/O LIOB–550	Comercio

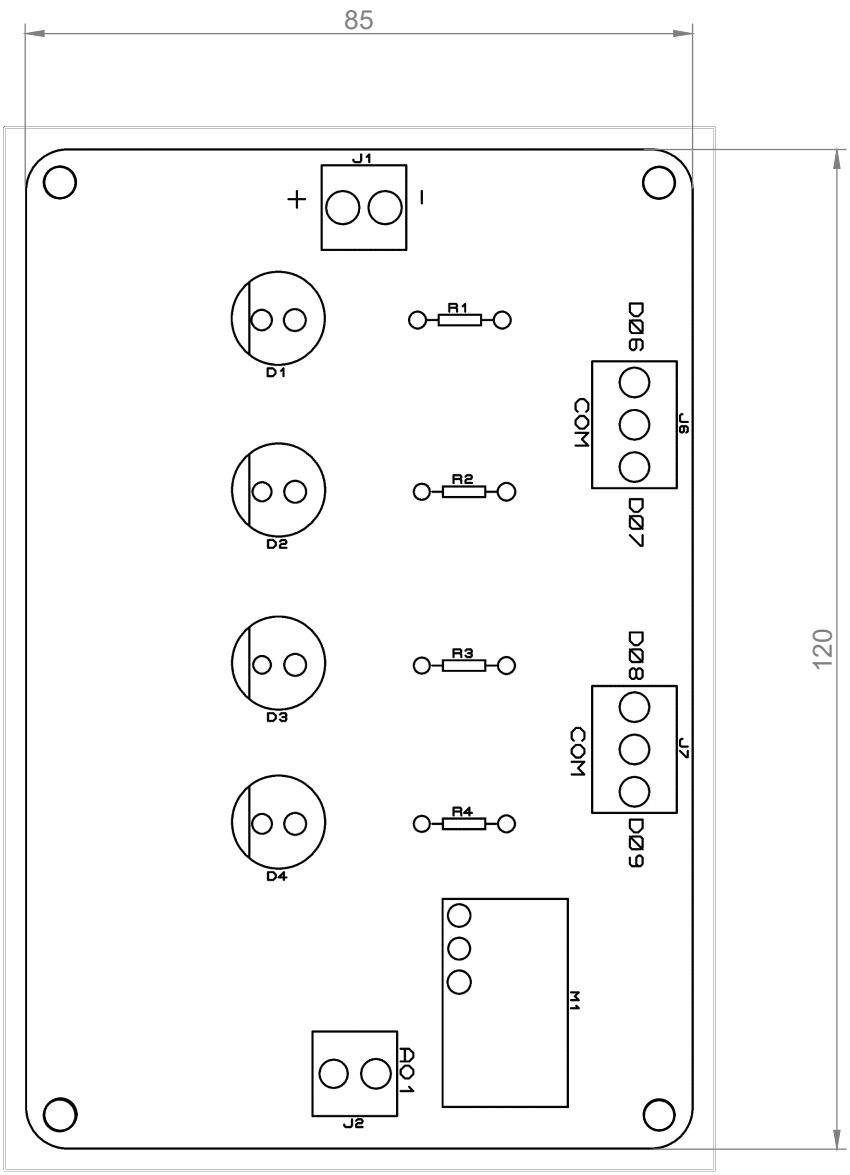
	NOMBRE	FECHA	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
PROYECTADO	OMAR BEN ABDELOUAHAB PEREIRA	21/08/2021	TÍTULO: Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización.
DIBUJADO	OMAR BEN ABDELOUAHAB PEREIRA	21/08/2021	
CONFORMADO	OMAR BEN ABDELOUAHAB PEREIRA	21/08/2021	

ESCALA:	DENOMINACIÓN del PLANO:	Nº de PLANO:
No procede	ESQUEMA DE LA RED	001



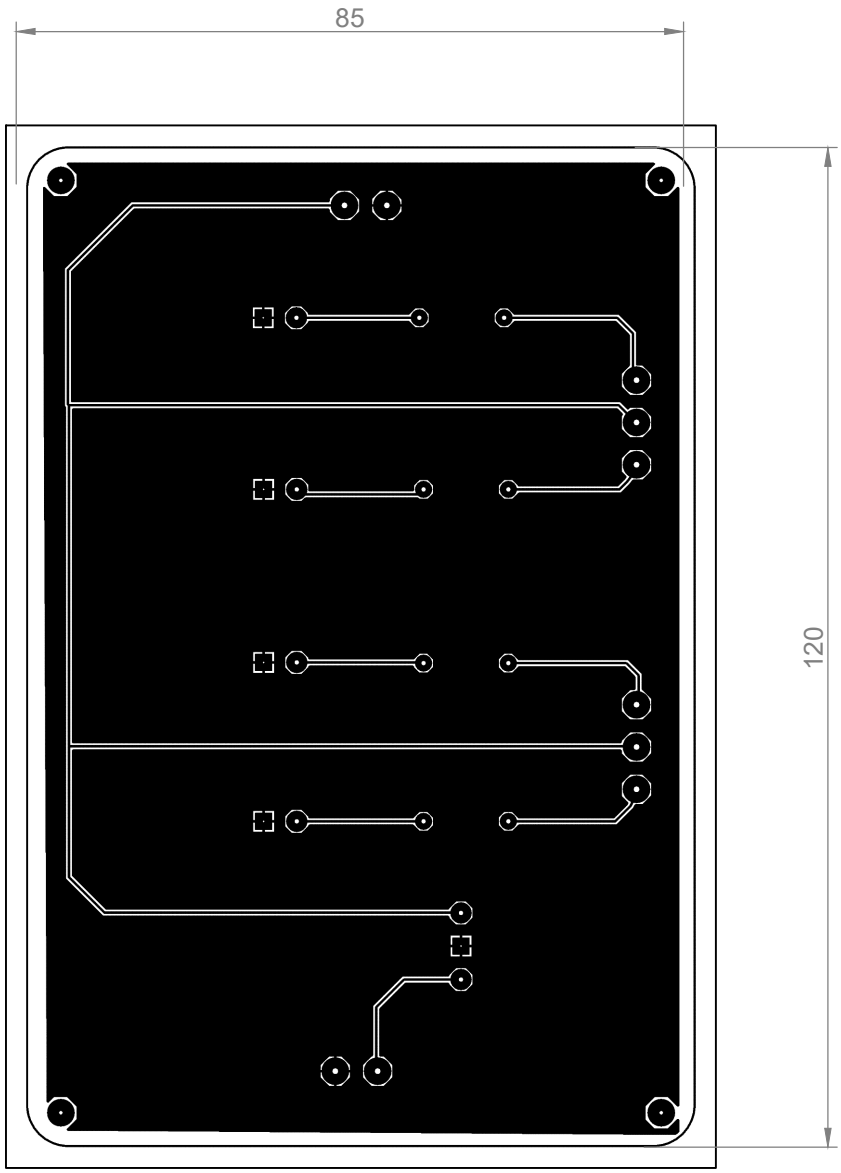
	NOMBRE	FECHA	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
PROYECTADO	OMAR BEN ABDELOUAHAB PEREIRA	24/08/2021	TÍTULO: Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización.
DIBUJADO	OMAR BEN ABDELOUAHAB PEREIRA	24/08/2021	
CONFORMADO	OMAR BEN ABDELOUAHAB PEREIRA	24/08/2021	

ESCALA:	DENOMINACIÓN del PLANO:	Nº de PLANO:
No procede	ESQUEMA ELÉCTRICO DE LA RED	002



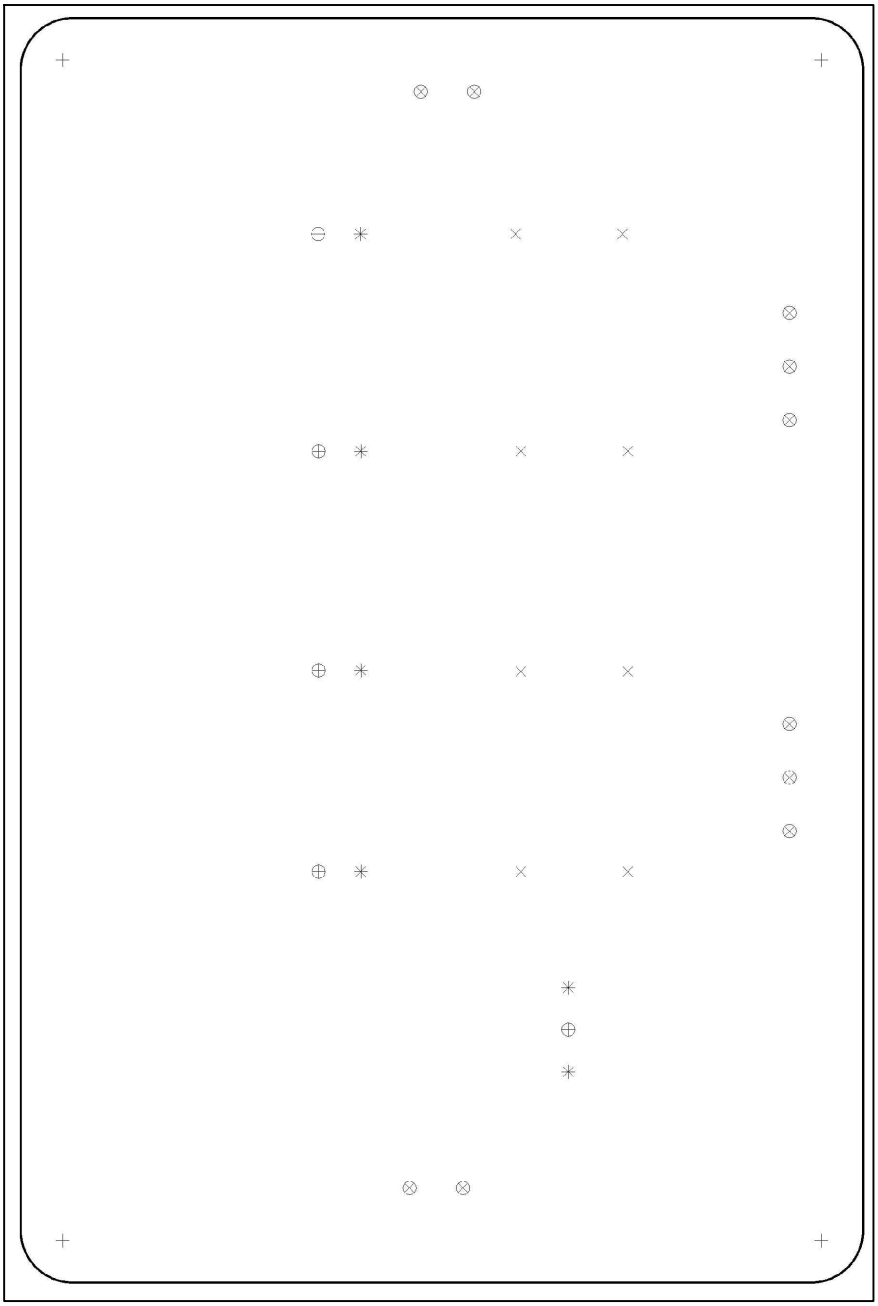
	NOMBRE	FECHA	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
PROYECTADO	OMAR BEN ABDELOUAHAB PEREIRA	23/08/2022	TÍTULO: Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización.
DIBUJADO	OMAR BEN ABDELOUAHAB PEREIRA	23/08/2022	
CONFORMADO	OMAR BEN ABDELOUAHAB PEREIRA	23/08/2022	

ESCALA:	DENOMINACIÓN del PLANO:	Nº de PLANO:
1/20	PLANO DE SERIGRAFÍA DE COMPONENTES DE LA PCB	003



	NOMBRE	FECHA	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
PROYECTADO	OMAR BEN ABDELOUAHAB PEREIRA	23/08/2022	TÍTULO: Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización.
DIBUJADO	OMAR BEN ABDELOUAHAB PEREIRA	23/08/2022	
CONFORMADO	OMAR BEN ABDELOUAHAB PEREIRA	23/08/2022	

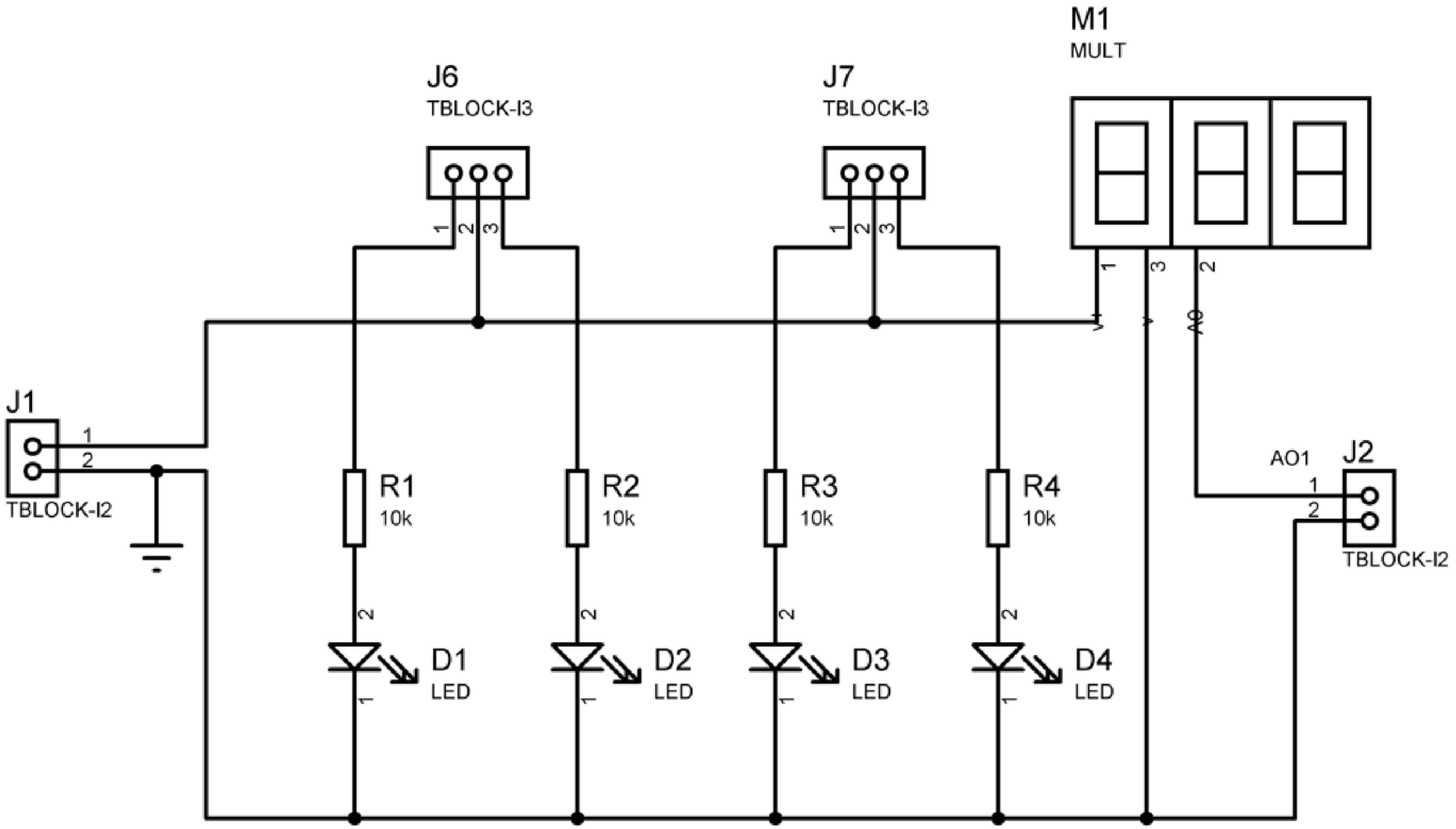
ESCALA:	DENOMINACIÓN del PLANO:	Nº de PLANO:
1/20	PLANO DE LAS PISTAS DE LA PCB	004



SYM	SIZE	PLATED	QTY
+	3.2MM	YES	4
×	Ø.8MM	YES	8
✳	1.ØMM	YES	6
⊕	1.3MM	YES	5
⊗	1.5MM	YES	1Ø

	NOMBRE	FECHA	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
PROYECTADO	OMAR BEN ABDELOUAHAB PEREIRA	23/08/2022	TÍTULO: Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización.
DIBUJADO	OMAR BEN ABDELOUAHAB PEREIRA	23/08/2022	
CONFORMADO	OMAR BEN ABDELOUAHAB PEREIRA	23/08/2022	

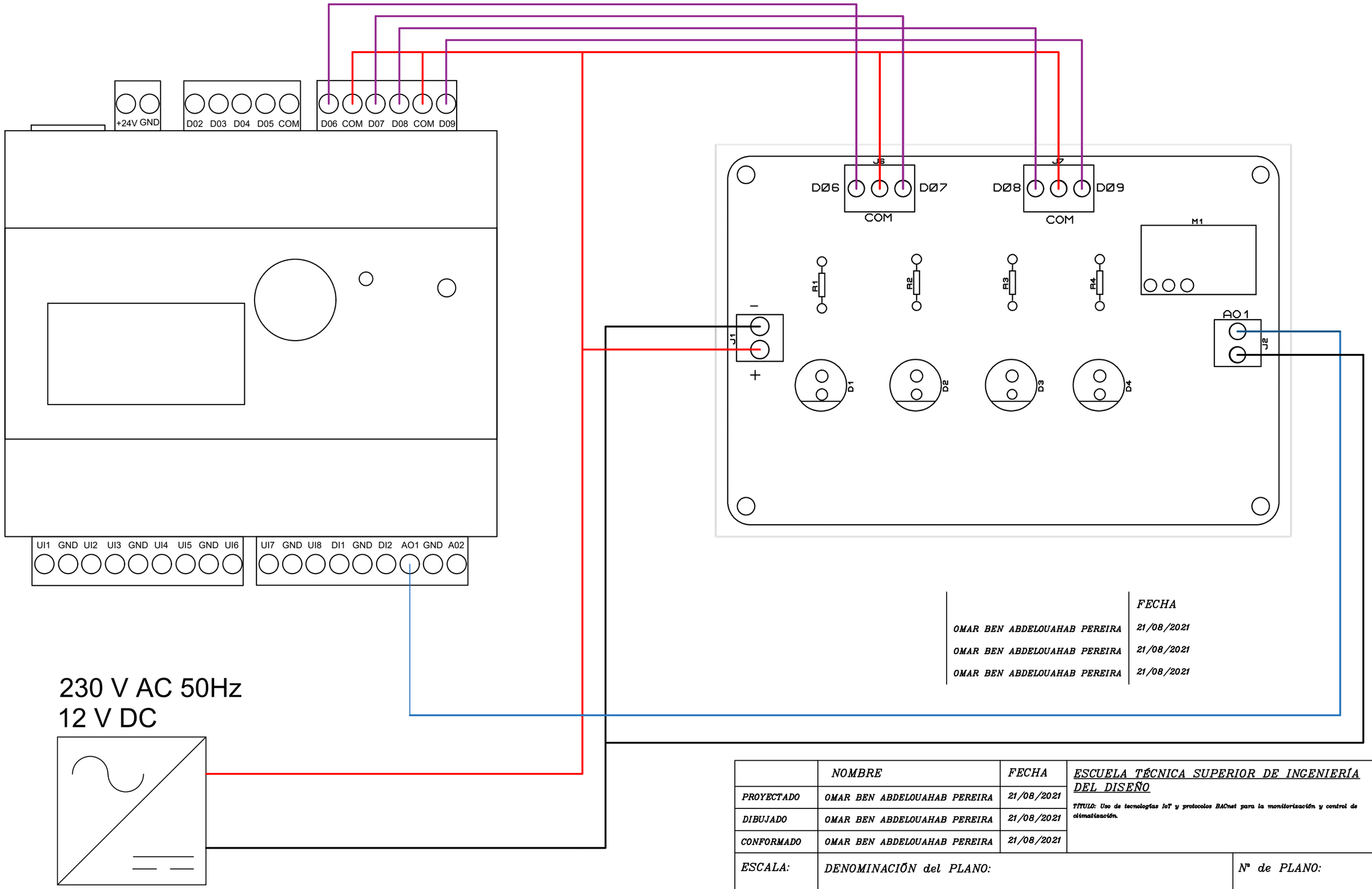
ESCALA:	DENOMINACIÓN del PLANO:	Nº de PLANO:
No procede	PLANO DE TALADROS DE LA PCB	005



Marca	Cantidad	Denominación	Referencia
J1, J2	2	Conector T Block de 2 pines	Comercio
J6, J7	2	Conector T Block de 3 pines	Comercio
R1,R2, R3,R4	4	Resistencia de 560 ohms	Comercio
M1	1	Multimetro de 3 displays	Comercio
D1	1	Indicador LED 1904X3IP	Comercio
D2	1	Indicador LED 1904X7IP	Comercio
D3	1	Indicador LED 1904X2IP	Comercio
D4	1	Indicador LED 1904X1IP	Comercio

	NOMBRE	FECHA	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
PROYECTADO	OMAR BEN ABDELOUAHAB PEREIRA	26/08/2021	TÍTULO: Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización.
DIBUJADO	OMAR BEN ABDELOUAHAB PEREIRA	26/08/2021	
CONFORMADO	OMAR BEN ABDELOUAHAB PEREIRA	26/08/2021	

ESCALA:	DENOMINACIÓN del PLANO:	Nº de PLANO:
No procede	ESQUEMA ELECTRÓNICO DE LA PCB	006



		FECHA
OMAR BEN ABDELOUAHAB PEREIRA		21/08/2021
OMAR BEN ABDELOUAHAB PEREIRA		21/08/2021
OMAR BEN ABDELOUAHAB PEREIRA		21/08/2021

	NOMBRE	FECHA	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO
PROYECTADO	OMAR BEN ABDELOUAHAB PEREIRA	21/08/2021	TÍTULO: Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización.
DIBUJADO	OMAR BEN ABDELOUAHAB PEREIRA	21/08/2021	
CONFORMADO	OMAR BEN ABDELOUAHAB PEREIRA	21/08/2021	

ESCALA:	DENOMINACIÓN del PLANO:	Nº de PLANO:
No procede	ESQUEMA DE CONEXIÓN DE LOS ACTUADORES	007

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización

---

III. Pliego de Condiciones

Autor: D. Omar Ben Abdelouahab Pereira

Tutor: D. Ángel Perles Ivars

Índice

1. Definición y alcance del pliego

El presente trabajo consiste en el diseño e instalación de un sistema de control, que por medio del uso de técnicas de IOT pueda monitorizar los valores de humedad, temperatura y estado de puertas/ventanas de los sensores LoRaWAN. Y mediante el protocolo de comunicación específico de la automatización de edificios BACnet permita controlar un sistema de climatización.

Este pliego pretende describir acorde a lo explicado en la memoria y en los planos, las condiciones bajo las cuales se debe llevar a cabo la instalación y configuración de los dispositivos, así como servir de guía de uso y mantenimiento.

También, cabe destacar que este proyecto se ha diseñado teniendo en cuenta el Real Decreto 842/2002, de 2 de agosto, por el que se aprueba el Reglamento Electrotécnico para Baja Tensión (BOE nº: 224, de 18/09/2002).

2. Condiciones de carácter general

Este proyecto se ha realizado bajo ciertas condiciones de funcionamiento específicas y que deben ser cumplidas durante la instalación y el funcionamiento del mismo.

2.1. Condiciones de los materiales

Se especifican a continuación las características técnicas que deben tener cada uno de los elementos de la instalación:

2.1.1. Controlador

El controlador debe tratarse de un ordenador de placa reducida de características similares o superiores a una Raspberry Pi 3 Modelo B de 4 GB. No obstante, se recomienda el uso de una Raspberry Pi 4 Modelo B.

Dicho controlador debe tener alguna distribución Linux como sistema operativo y debe tener la versión 3.7 o superior de Python instalada junto con las librerías Tkinter y bacpypes.

Por último, debe ser alimentado a una tensión 12V con una corriente de 3A, y se recomienda instalarle un ventilador para evitar el sobrecalentamiento de la placa.

2.1.2. Switch ethernet

El switch debe tener una cantidad de puertos superior al numero de dispositivos a conectar. También debe ser compatible con el estándar IEEE802.3 y tener una velocidad de transmisión de datos de al menos 100 Mbps.

2.1.3. Elementos de la red

Los sensores de tanto temperatura y humedad como de estado de puertas/ventanas, el convertidor de LoRaWAN al protocolo BACnet y el módulo I/O (inputs/outputs) deben ser los especificados en la memoria.

Cualquier modificación en la lista de materiales debe ser comunicada al autor del proyecto o al ingeniero responsable pertinente.

2.1.4. Fuente de alimentación

La fuente de alimentación requerida debe convertir una tensión de entrada de 230V AC a una tensión de salida de 12 V y debe suministrar una corriente de al menos 5 A. Dicha fuente debe ser conectada a tierra y debe contar con una protección contra sobretensión, sobrecorriente y cortocircuito.

2.1.5. Panel de actuadores

La placa PCB debe fabricarse y debe ser ensamblada acorde con la norma IPC-2221. Además, la PCB tendrá que cumplir las siguientes especificaciones:

- El dieléctrico debe ser FR4 y debe contar con un grosor de 1.6 mm.
- La tarjeta debe haber sido comprobada bajo una prueba electrónica que garantice la continuidad y el correcto aislamiento entre pistas.

Cualquier anomalía en la placa detectada durante la inspección de los materiales debe ser reportada formalmente al autor o ingeniero responsable

Los componentes electrónicos de la placa serán los indicados en el plano del esquema electrónico. Su correcto funcionamiento deberá ser comprobado su correcto funcionamiento de forma previa al ensamblaje a la PCB por soldadura.

2.1.6. Otros elementos

Los cables de red ethernet deben ser Cat 5e o superior y compatibles con el switch ethernet elegido.

2.2. Instalación

Puesto que este proyecto se trata de un prototipo que aún esta en fase de desarrollo y de pruebas, la instalación debe ser llevada a cabo por personal con conocimiento en la materia. Esto se debe principalmente a que durante la instalación es necesario configurar los distintos dispositivos para que puedan funcionar correctamente en la red.

La instalación será realizada dentro de un cuadro eléctrico de acero o de aluminio por un técnico qualificado a excepción de los sensores inalámbricos. El convertidor de LoRaWAn a BACnet, el LIOB-550 y el L-POWER 2415A se fijan sobre rieles de acero atornillados al fondo del cuadro. Mientras que el resto de los dispositivos se fijan mediante pegatinas adhesivas. Una vez situados los elementos se conectan los cables ethernet y los cables a la alimentación.

Una vez instalado, se comprobará que todos los elementos están correctamente alimentados y que en ningún punto del circuito se superen las tensiones y corrientes permitidas en el Reglamento de Baja Tensión. También se asegurará que ningún cable pueda hacer contacto con ningún elemento y que la instalación esta correctamente puesta a tierra.

Para garantizar la fiabilidad de las mediciones de temperatura y humedad, debe evitarse la colocación del sensor LHT52 cerca de fuentes de emisión de calor, de frío y/o de humedad, así como de elementos que puedan generar una interferencia electromagnética en la banda de 868MHz. Además, debe certificarse que la comunicación en la red LoRaWAN se realiza sin problemas y no interfiera con otros dispositivos de radio.

Por último, se instalarán la versión 3.7 o superior de Python (en caso de no estar instalado previamente) y la aplicación en la placa Raspberry PI o en su sustituto.

2.3. Prueba de servicio

Tras la instalación, se deberán realizar las correspondientes pruebas para poder certificar que la instalación se ha llevado a cabo de forma correcta.

2.3.1. Prueba de los sensores

Se comprobará que los valores recibidos por la aplicación sean correctos y se correspondan con la realidad. Para el sensor LHT52 se realizarán cinco medidas de tanto humedad como de temperatura y se obtendrá la media correspondiente de cada una. Seguidamente se realizarán cinco mediciones con un termohigrómetro y se obtendrá la media de temperatura y humedad. Finalmente se compararán las medias del LHT52 con las medias obtenidas para el termohigrómetro y se asegurará que dicho valor entra dentro del rango especificado en el datasheet del sensor.

Por otra parte, para garantizar el funcionamiento del sensor LDS01 se realizarán 20 mediciones alternando el estado de abierto a cerrado sin seguir ninguna secuencia. Solo se certificará como correcto funcionamiento al pasar con un 90% de acierto (18 de 20).

Por último, se comprobará que el sensor esta correctamente fijado a la pared (para el LHT52) o a la ventana/puerta (para el LDS01). Para ello, se ejercerán empujes con distintos niveles de fuerza y en distintos ángulos.

2.3.2. Prueba del cuadro del sistema

Se comprobará que la compuerta del cuadro se abre sin ejercer ninguna resistencia y que el cuadro esta correctamente fijada a la pared mediante tornillos. Para ello, se ejercerá una fuerza de empuje hacia abajo en la parte superior del cuadro y se abrirá la puerta del cuadro repetidas veces.

Una vez asegurado el buen montaje del cuadro, se hace lo mismo con los dispositivos en su interior para comprobar que los dispositivos se han adherido bien a las pegatinas o que están bien fijos al riel. Se verifica que los cables ethernet están conectados y que la luz del switch ethernet esta encendida para el puerto correspondiente.

2.3.3. Prueba de la aplicación

Se comprueba que el funcionamiento de la aplicación es el esperado. Para ello, primero se ejecuta la aplicación y se confirma que los valores de la interfaz se han cargado correctamente y se adecuan con los reales. Seguidamente se introducen comandos en la consola de la interfaz para comprobar que la interfaz se actualiza

A continuación, se procede a hacer peticiones BACnet y se verifica que se realizan correctamente mediante el uso de *wireshark* o un programa similar de captura de paquetes de red.

Por último, se modifican los valores de la interfaz para asegurarse de que las rutinas funcionan correctamente para cada uno de los casos posibles. También se aprovecha para asegurarse que los indicadores del panel se encienden cuando les corresponde.

3. Guía de uso

Una vez encendido el sistema puede tardar hasta 15 minutos en ser completamente operativo, esto se debe a que al cargar la aplicación no dispone de las mediciones de temperatura y humedad. Este tiempo se puede reducir modificando la frecuencia de medida del sensor LHT52.

Al tratarse de un prototipo para la realización de pruebas, una vez en funcionamiento el usuario debe introducir los parámetros manualmente para la correcta simulación del sistema.

Para cerrar el programa se recomienda utilizar el comando exit en la consola de la interfaz, para garantizar que el proceso termine correctamente. Ante la interrupción de la ejecución del programa es imprescindible terminar el proceso manualmente o no se podrá volver a ejecutar la aplicación.

4. Plan de Mantenimiento

Al tratarse de un prototipo, el mantenimiento del sistema que debe realizarse es mínimo. Se recomienda que se efectuarlo de forma anual. Dicho mantenimiento consiste en realizar una serie de tareas:

- Comprobación de los sensores, se realizan una serie de mediciones para garantizar el correcto funcionamiento de los sensores tal y como se explica en el apartado [2.3.1] de este mismo documento.
- Se realiza el cambio de baterías para los sensores inalámbricos. Las baterías usadas serán desechadas acorde a la normativa vigente pertinente en materia de gestión ambiental de residuos.
- Limpieza de polvo en el cuadro eléctrico.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización

---

IV. PRESUPUESTO

Autor: D. Omar Ben Abdelouahab Pereira

Tutor: D. Ángel Perles Ivars

Índice

Cuadro de Precios Unitarios

Cuadro de mano de obra

Nº	Designación	Importe		
		Precio (Euros)	Cantidad (Horas)	Total (Euros)
1	Oficial 1ª electricidad	20,34	2,0 h	40,70
2	Programador de Ordenador	21,81	200,0 h	4.362,00
3	Oficial 1º telecomunicaciones	19,28	4,5 h	86,80
			Importe total:	4.489,50

Valencia, agosto de 2022
Ingeniero electrónico

Omar Ben Abdelouahab Pereira

Cuadro de materiales

Nº	Designación	Importe		
		Precio (Euros)	Cantidad Empleada	Total (Euros)
1	Ordenador de placa reducida Raspberry Pi 4 Modelo B 4Gb de RAM o similar	100,00	1,0 u	100,00
2	Indicador LED rojo 1904X3IP	3,78	1,0 u	3,80
3	Indicador LED rojo 1904X3IP	3,78	1,0 u	3,80
4	Indicador LED rojo 1904X3IP	3,78	1,0 u	3,80
5	Voltímetro en módulo de pantalla LED digital de 7 segmentos	2,11	1,0 u	2,10
6	Resistencia de 560 ohmios 1/4 W	0,40	1,0 u	0,40
7	Placa de circuito impreso	0,50	1,0 u	0,50
8	Fuente de alimentación 12V 5A	9,20	1,0 u	9,20
9	Sensor de temperatura LHT52 de Dragino	39,86	1,0 u	39,90
10	Sensor magnético de estado de puerta LDS01 de Dragino.	20,10	1,0 u	20,10
11	Convertidor de LoRaWAN a protocolo de comunicación BACnet HD67F17-B2-IP-868MHz	381,45	1,0 u	381,50
12	Switch ethernet de 8 puertos	16,81	1,0 u	16,80
13	Cables ethernet Cat 5e	2,45	6,0 m	14,70
14	Módulo de puertos de entradas y salidas LIOB-550	503,00	1,0 u	503,00
15	Fuente alimentación L-POW 2415A	80,00	1,0 u	80,00
16	Indicador LED rojo 1904X3IP	3,78	1,0 u	3,80
			Importe total:	1.183,40

Valencia, agosto de 2022
Ingeniero electrónico

Omar Ben Abdelouahab Pereira

Cuadro de Precios Descompuestos

Nº	Código	Ud	Descripción			Total
			1 Sistema de Control			
1.1	d1	**u**	**Controlador Raspberry Pi 4 Modelo B de 4 Gb con programa informático para la lectura de los valores de temperatura, humedad relativa y estado de puertas y ventanas mediante el uso de redes LoRaWAN, así como la monitorización y control del estado de los actuadores mediante el uso de una red BACnet/Ip. Con interfaz gráfica que permite monitorizar e interactuar con la aplicación. Completamente instalada, comprobada y en correcto funcionamiento. Incluye el transporte de los materiales.**			
	m1		1,0 u	Ordenador de placa reducida Raspberry Pi 4 Modelo B 4Gb de RAM o similar	100,00	100,00
	MOOI.9a		200,0 h	Programador de Ordenador	21,81	4.362,00
	%		2,0 %	Costes Directos Complementarios	4.462,00	89,24
				Precio total por u		**4.551,24**
1.2	d2	**u**	**Red de sensores LoRaWAN. Incluye el sensor de temperatura y humedad LHT52 de Dragino, el sensor magnético de estado de puerta LDS01 también de dragino y el convertidor de LoRaWAN a protocolo de comunicación BACnet HD67F17-B2-IP-868MHz. Completamente instalada, comprobada, configurada y en perfecto funcionamiento. Incluye el transporte de materiales.**			
	m2		1,0 u	Sensor de temperatura LHT52 de Dragino	39,86	39,86
	m3		1,0 u	Sensor magnético de estado de puerta LDS01 de Dragino.	20,10	20,10
	m4		1,0 u	Convertidor de LoRaWAN a protocolo de comunicación BACnet HD67F17-B2-IP-868MHz	381,45	381,45
	MOOL.8a		3,5 h	Oficial 1º telecomunicaciones	19,28	67,48
	%		2,0 %	Costes Directos Complementarios	508,89	10,18
				Precio total por u		**519,07**
1.3	d3	**u**	**Red ethernet con cables Cat 5e o superior. Incluye la instalación de un switch ethernet de al menos 5 puertos, compatible con IEEE802.3X, así como la alimentación del mismo a la red eléctrica. Completamente instalada, comprobada y en perfecto funcionamiento. Incluye el precio de transporte.**			
	m5		1,0 u	Switch ethernet de 8 puertos	16,81	16,81
	m6		6,0 m	Cables ethernet Cat 5e	2,45	14,70
	MOOL.8a		0,5 h	Oficial 1º telecomunicaciones	19,28	9,64
	%		2,0 %	Costes Directos Complementarios	41,15	0,82
				Precio total por u		**41,97**
1.4	d4	**u**	**Módulo de puertos de entrada y salida LIOB-550 instalda, completamente configurado, en perfecto funcionamiento y conectado al panel de actuadores.**			
	m7		1,0 u	Módulo de puertos de entradas y salidas LIOB-550	503,00	503,00
	m8		1,0 u	Fuente alimentación L-POW 2415A	80,00	80,00
	m9		1,0 u	Indicador LED rojo 1904X3IP	3,78	3,78
	m10		1,0 u	Indicador LED azul 1904X7IP	3,78	3,78
	m11		1,0 u	Indicador LED amarillo 1904X2IP	3,78	3,78
	m12		1,0 u	Indicador LED verde 1904X1IP	3,78	3,78
	m13		1,0 u	Voltímetro en módulo de pantalla LED digital de 7 segmentos	2,11	2,11
	m14		1,0 u	Resistencias de 560 ohmios 1/4 W	0,40	0,40
	m15		1,0 u	Placa de circuito impreso	0,50	0,50
	MOOL.8a		0,5 h	Oficial 1º telecomunicaciones	19,28	9,64
	MOOE.8a		1,5 h	Oficial 1ª electricidad	20,34	30,51
	%		2,0 %	Costes Directos Complementarios	641,28	12,83
				Precio total por u		**654,11**

Nº	Código	Ud	Descripción				Total
1.5 d5		**u**	**Fuente de alimentación de 12V y 5A, incluye la conexión de la fuente a la red eléctrica y a los dispositivos de la red actuadores incluidos. Se excluye alimentación del LIOB-550 y Raspberry Pi 4 (alimentación aparte). Completamente instalada, comprobada y en correcto funcionamiento.**				
	m16		1,0	u	Fuente de alimentación 12V 5A	9,20	9,20
	MOOE.8a		0,5	h	Oficial 1ª electricidad	20,34	10,17
	%		2,0	%	Costes Directos Complementarios	19,37	0,39
					Precio total por u		**19,76**

Cuadro de precios nº 1

Nº	Designación	Importe	
		En cifra (Euros)	En letra (Euros)
1	u Controlador Raspberry Pi 4 Modelo B de 4 Gb con programa informático para la lectura de los valores de temperatura, humedad relativa y estado de puertas y ventanas mediante el uso de redes LoRaWAN, así como la monitorización y control del estado de los actuadores mediante el uso de una red BACnet/Ip. Con interfaz gráfica que permite monitorizar e interactuar con la aplicación. Completamente instalada, comprobada y en correcto funcionamiento. Incluye el transporte de los materiales.	4.551,24	CUATRO MIL QUINIENTOS CINCUENTA Y UN EUROS CON VEINTICUATRO CÉNTIMOS
2	u Red de sensores LoRaWAN. Incluye el sensor de temperatura y humedad LHT52 de Dragino, el sensor magnético de estado de puerta LDS01 también de dragino y el convertidor de LoRaWAN a protocolo de comunicación BACnet HD67F17-B2-IP-868MHz. Completamente instalada, comprobada, configurada y en perfecto funcionamiento. Incluye el transporte de materiales.	519,07	QUINIENTOS DIECINUEVE EUROS CON SIETE CÉNTIMOS
3	u Red ethernet con cables Cat 5e o superior. Incluye la instalación de un switch ethernet de al menos 5 puertos, compatible con IEEE802.3X, así como la alimentación del mismo a la red eléctrica. Completamente instalada, comprobada y en perfecto funcionamiento. Incluye el precio de transporte.	41,97	CUARENTA Y UN EUROS CON NOVENTA Y SIETE CÉNTIMOS
4	u Módulo de puertos de entrada y salida LIOB-550 instalda, completamente configurado, en perfecto funcionamiento y conectado al panel de actuadores.	654,11	SEISCIENTOS CINCUENTA Y CUATRO EUROS CON ONCE CÉNTIMOS
5	u Fuente de alimentación de 12V y 5A, incluye la conexión de la fuente a la red eléctrica y a los dispositivos de la red actuadores incluidos. Se excluye alimentación del LIOB-550 y Raspberry Pi 4 (alimentación aparte). Completamente instalada, comprobada y en correcto funcionamiento.	19,76	DIECINUEVE EUROS CON SETENTA Y SEIS CÉNTIMOS

Valencia, agosto de 2022
Ingeniero electrónico

Omar Ben Abdelouahab Pereira

Cuadro de Mediciones

Presupuesto parcial nº 1 Sistema de Control

Nº	Ud	Descripción	Medición
1.1	**U**	**Controlador Raspberry Pi 4 Modelo B de 4 Gb con programa informático para la lectura de los valores de temperatura, humedad relativa y estado de puertas y ventanas mediante el uso de redes LoRaWAN, así como la monitorización y control del estado de los actuadores mediante el uso de una red BACnet/Ip. Con interfaz gráfica que permite monitorizar e interactuar con la aplicación. Completamente instalada, comprobada y en correcto funcionamiento. Incluye el transporte de los materiales.**	
		Total u .....:	**1,0**
1.2	**U**	**Red de sensores LoRaWAN. Incluye el sensor de temperatura y humedad LHT52 de Dragino, el sensor magnético de estado de puerta LDS01 también de dragino y el convertidor de LoRaWAN a protocolo de comunicación BACnet HD67F17-B2-IP-868MHz. Completamente instalada, comprobada, configurada y en perfecto funcionamiento. Incluye el transporte de materiales.**	
		Total u .....:	**1,0**
1.3	**U**	**Red ethernet con cables Cat 5e o superior. Incluye la instalación de un switch ethernet de al menos 5 puertos, compatible con IEEE802.3X, así como la alimentación del mismo a la red eléctrica. Completamente instalada, comprobada y en perfecto funcionamiento. Incluye el precio de transporte.**	
		Total u .....:	**1,0**
1.4	**U**	**Módulo de puertos de entrada y salida LIOB-550 instalda, completamente configurado, en perfecto funcionamiento y conectado al panel de actuadores.**	
		Total u .....:	**1,0**
1.5	**U**	**Fuente de alimentación de 12V y 5A, incluye la conexión de la fuente a la red eléctrica y a los dispositivos de la red actuadores incluidos. Se excluye alimentación del LIOB-550 y Raspberry Pi 4 (alimentación aparte). Completamente instalada, comprobada y en correcto funcionamiento.**	
		Total u .....:	**1,0**

Valencia, agosto de 2022
Ingeniero electrónico

Omar Ben Abdelouahab Pereira

Presupuesto

Presupuesto parcial nº 1 Sistema de Control

Nº	Ud	Descripción	Medición	Precio	Importe
1.1	U	**Controlador Raspberry Pi 4 Modelo B de 4 Gb con programa informático para la lectura de los valores de temperatura, humedad relativa y estado de puertas y ventanas mediante el uso de redes LoRaWAN, así como la monitorización y control del estado de los actuadores mediante el uso de una red BACnet/Ip. Con interfaz gráfica que permite monitorizar e interactuar con la aplicación. Completamente instalada, comprobada y en correcto funcionamiento. Incluye el transporte de los materiales.**			
		Total u .....:	**1,0**	**4.551,24**	**4.551,24**
1.2	U	**Red de sensores LoRaWAN. Incluye el sensor de temperatura y humedad LHT52 de Dragino, el sensor magnético de estado de puerta LDS01 también de dragino y el convertidor de LoRaWAN a protocolo de comunicación BACnet HD67F17-B2-IP-868MHz. Completamente instalada, comprobada, configurada y en perfecto funcionamiento. Incluye el transporte de materiales.**			
		Total u .....:	**1,0**	**519,07**	**519,07**
1.3	U	**Red ethernet con cables Cat 5e o superior. Incluye la instalación de un switch ethernet de al menos 5 puertos, compatible con IEEE802.3X, así como la alimentación del mismo a la red eléctrica. Completamente instalada, comprobada y en perfecto funcionamiento. Incluye el precio de transporte.**			
		Total u .....:	**1,0**	**41,97**	**41,97**
1.4	U	**Módulo de puertos de entrada y salida LIOB-550 instalda, completamente configurado, en perfecto funcionamiento y conectado al panel de actuadores.**			
		Total u .....:	**1,0**	**654,11**	**654,11**
1.5	U	**Fuente de alimentación de 12V y 5A, incluye la conexión de la fuente a la red eléctrica y a los dispositivos de la red actuadores incluidos. Se excluye alimentación del LIOB-550 y Raspberry Pi 4 (alimentación aparte). Completamente instalada, comprobada y en correcto funcionamiento.**			
		Total u .....:	**1,0**	**19,76**	**19,76**
		Total presupuesto parcial nº 1 Sistema de Control :			**5.786,15**

Presupuesto de ejecución material

1 Sistema de Control		**5.786,15**
	Total .....:	**5.786,15**

Asciende el presupuesto de ejecución material a la expresada cantidad de CINCO MIL SETECIENTOS OCHENTA Y SEIS EUROS CON QUINCE CÉNTIMOS.

Valencia, agosto de 2022
Ingeniero electrónico

Omar Ben Abdelouahab Pereira

Resumen del Presupuesto

Proyecto: Sistema de control para instalación de climatización

Capítulo	Importe
Capítulo 1 Sistema de Control	5.786,15
Presupuesto de ejecución material	5.786,15
13% de gastos generales	752,20
6% de beneficio industrial	347,17
Suma	6.885,52
21% IVA	1.445,96
Presupuesto de ejecución por contrata	8.331,48

Asciende el presupuesto de ejecución por contrata a la expresada cantidad de OCHO MIL TRESCIENTOS TREINTA Y UN EUROS CON CUARENTA Y OCHO CÉNTIMOS.

Valencia, agosto de 2022
Ingeniero electrónico

Omar Ben Abdelouahab Pereira

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Uso de tecnologías IoT y protocolos BACnet para la monitorización y control de climatización

V. Anexos

Autor: D. Omar Ben Abdelouahab Pereira

Tutor: D. Ángel Perles Ivars

Índice

1. Estructura del programa

Para que el programa funcione correctamente, los ficheros deben situarse en un directorio siguiendo la siguiente estructura:

```
My_App
    |_BACnet
    |       |_BACnetApp.py
    |       |_BACnetConsole.py
    |_GUI
    |   |_resources
    |   |       |_ledBlue.png
    |   |       |_ledGreen.png
    |   |       |_ledOff.png
    |   |       |_ledRed.png
    |   |       |_ledYellow.png
    |   |_iface.py
    |_App.py
    |_BACpypes.ini
    |_DataModule.py
    |_SupervisorModule.py
```

2. BACpypes.ini

```
[BACpypes]
objectName: Argos
address: 192.168.2.200/24
objectIdentifier: 599
maxApduLengthAccepted: 1024
segmentationSupported: segmentedBoth
maxSegmentsAccepted: 1024
vendorIdentifier: 15

```

3. BACnetApp.py

```
1.  import sys
2.  from threading import Thread
3.
4.  from bacpypes.core import run, enable_sleeping, deferred
5.  from bacpypes.consolelogging import ConfigArgumentParser
6.  from bacpypes.task import recurring_function
7.
8.  from bacpypes.pdu import Address, GlobalBroadcast
9.  from bacpypes.object import get_datatype
10. from bacpypes.iocb import IOCB
11.
12. from bacpypes.apdu import SimpleAckPDU, \
13.     ReadPropertyRequest, ReadPropertyACK, WritePropertyRequest, WhoIsRequest, IAmRequest
14. from bacpypes.primitivedata import Null, Atomic, Boolean, Unsigned, Integer, \
15.     Real, Double, OctetString, CharacterString, BitString, Date, Time, ObjectIdentifier
16. from bacpypes.constructeddata import Array, Any, AnyAtomic
17.
18. from bacpypes.app import BIPSimpleApplication
19. from bacpypes.local.device import LocalDeviceObject
20.
21.
22. # globals
23. BACnet_device = None
24. BACnet_app = None
25.
26.
27. def BACpypesThread():
28.         run()
29.
30. # ------------------------------------------------------------------------------------------------------
31. #  APLICACION BACNET  ------------------------------------------------------------------------------------
32. # ------------------------------------------------------------------------------------------------------
33.
34. class BacnetApplication (BIPSimpleApplication):
35.
36.     def __init__(self, *args):
37.         BIPSimpleApplication.__init__(self,*args)
38.
39.         # keep track of requests to line up responses
40.         self._request = None
41.
42.     def request(self, apdu):
43.         # save a copy of the request
44.         if isinstance(apdu, WhoIsRequest):
45.             self._request = apdu
46.         BIPSimpleApplication.request(self, apdu)
47.
48.     def indication(self, apdu):
49.         BIPSimpleApplication.indication(self, apdu)
50.
51.     def confirmation(self, apdu):
52.         BIPSimpleApplication.confirmation(self,apdu)
53.
54. # ------------------------------------------------------------------------------------------------------
55. #  DEVICES/NETWORK RECOGNITION ---------------------------------------------------------------------------
56. # ------------------------------------------------------------------------------------------------------
57.
58. def do_whois(args):
59.
60.     try:
61.         # gather the parameters
62.         if (len(args) == 1) or (len(args) == 3):
63.             addr = Address(args[0])
64.             del args[0]
65.         else:
66.             addr = GlobalBroadcast()
67.
68.         if len(args) == 2:
69.             lolimit = int(args[0])
70.             hilimit = int(args[1])
71.        else:
72.             lolimit = hilimit = None
```

```
73.
74.         # code lives in the device service
75.         BACnet_app.who_is(lolimit, hilimit, addr)
76.
77.     except Exception as error:
78.             print("Error: who is request failed")
79.
80. def do_iam():
81.
82.     # code lives in the device service
83.     BACnet_app.i_am()
84.
85. # ------------------------------------------------------------------------------------------------------------
86. #  READ FUNCTIONS ----------------------------------------------------------------------------------------------
87. # ------------------------------------------------------------------------------------------------------------
88.
89. def do_read(args):
90.         """read <addr> <objid> <prop> [ <indx> ]"""
91.
92.         try:
93.             addr, obj_id, prop_id = args[:3]
94.             obj_id = ObjectIdentifier(obj_id).value
95.             if prop_id.isdigit():
96.                 prop_id = int(prop_id)
97.
98.             datatype = get_datatype(obj_id[0], prop_id)
99.             if not datatype:
100.                 raise ValueError("invalid property for object type")
101.
102.              # build a request
103.              request = ReadPropertyRequest(
104.                  objectIdentifier=obj_id,
105.                  propertyIdentifier=prop_id,
106.                  )
107.              request.pduDestination = Address(addr)
108.
109.              if len(args) == 4:
110.                  request.propertyArrayIndex = int(args[3])
111.
112.              # make an IOCB
113.              iocb = IOCB(request)
114.
115.              # give it to the application
116.              deferred(BACnet_app.request_io, iocb)
117.
118.              # wait for it to complete
119.              iocb.wait()
120.
121.              # do something for success
122.              if iocb.ioResponse:
123.                  apdu = iocb.ioResponse
124.
125.                  # should be an ack
126.                  if not isinstance(apdu, ReadPropertyACK):
127.                      return
128.
129.                  # find the datatype
130.                  datatype = get_datatype(apdu.objectIdentifier[0], apdu.propertyIdentifier)
131.                  if not datatype:
132.                      raise TypeError("unknown datatype")
133.
134.                  # special case for array parts, others are managed by cast_out
135.                  if issubclass(datatype, Array) and (apdu.propertyArrayIndex is not None):
136.                      if apdu.propertyArrayIndex == 0:
137.                          value = apdu.propertyValue.cast_out(Unsigned)
138.                      else:
139.                          value = apdu.propertyValue.cast_out(datatype.subtype)
140.                  else:
141.                      value = apdu.propertyValue.cast_out(datatype)
142.
143.                  return (value)
144.
145.              # do something for error/reject/abort
146.              if iocb.ioError:
147.                  sys.stdout.write(str(iocb.ioError) + '\n')
148.
149.
```

```
150.
151.             except Exception as error:
152.                 return 'reading failed'
153.
154.  def do_readTemp():
155.      args = ['192.168.2.205','integerValue:0','presentValue']
156.      value = do_read(args)
157.      if (value != 'reading error'):
158.          return value/100
159.      else:
160.           return 'ValueError'
161.
162.  def do_readHumidity():
163.      args = ['192.168.2.205','positiveIntegerValue:0','presentValue']
164.      value = do_read(args)
165.      if (value != 'reading error'):
166.          return value/10
167.      else:
168.           return 'ValueError'
169.
170.  def do_readDoorStatus():
171.      args = ['192.168.2.205','positiveIntegerValue:1','presentValue']
172.      value = do_read(args)
173.      if (value != 'reading error'):
174.          if value > 30000:
175.              return 'OPEN'
176.          else:
177.              return 'CLOSED'
178.      else:
179.           return 'ValueError'
180.
181.
182.  # ------------------------------------------------------------------------------------------------------
183.  #  WRITE FUNCTIONS ---------------------------------------------------------------------------------
184.  # ------------------------------------------------------------------------------------------------------
185.
186.  def do_write(args):
187.      """write <addr> <objid> <prop> <value> [ <indx> ] [ <priority> ]"""
188.
189.      try:
190.          addr, obj_id, prop_id = args[:3]
191.          obj_id = ObjectIdentifier(obj_id).value
192.          value = args[3]
193.
194.          indx = None
195.          if len(args) >= 5:
196.              if args[4] != "-":
197.                  indx = int(args[4])
198.
199.          priority = None
200.          if len(args) >= 6:
201.              priority = int(args[5])
202.
203.          # get the datatype
204.          datatype = get_datatype(obj_id[0], prop_id)
205.
206.          # change atomic values into something encodeable, null is a special case
207.          if (value == 'null'):
208.              value = Null()
209.          elif issubclass(datatype, AnyAtomic):
210.              dtype, dvalue = value.split(':', 1)
211.
212.              datatype = {
213.                  'b': Boolean,
214.                  'u': lambda x: Unsigned(int(x)),
215.                  'i': lambda x: Integer(int(x)),
216.                  'r': lambda x: Real(float(x)),
217.                  'd': lambda x: Double(float(x)),
218.                  'o': OctetString,
219.                  'c': CharacterString,
220.                  'bs': BitString,
221.                  'date': Date,
222.                  'time': Time,
223.                  'id': ObjectIdentifier,
224.                  }[dtype]
225.              value = datatype(dvalue)
226.
```

```
227.          elif issubclass(datatype, Atomic):
228.              if datatype is Integer:
229.                  value = int(value)
230.              elif datatype is Real:
231.                  value = float(value)
232.              elif datatype is Unsigned:
233.                  value = int(value)
234.              value = datatype(value)
235.          elif issubclass(datatype, Array) and (indx is not None):
236.              if indx == 0:
237.                  value = Integer(value)
238.              elif issubclass(datatype.subtype, Atomic):
239.                  value = datatype.subtype(value)
240.              elif not isinstance(value, datatype.subtype):
241.                  raise TypeError("invalid result datatype, expecting %s" %(datatype.subtype.__name__,))
242.          elif not isinstance(value, datatype):
243.              raise TypeError("invalid result datatype, expecting %s" % (datatype.__name__,))
244.
245.          # build a request
246.          request = WritePropertyRequest(
247.              objectIdentifier=obj_id,
248.              propertyIdentifier=prop_id
249.              )
250.          request.pduDestination = Address(addr)
251.
252.          # save the value
253.          request.propertyValue = Any()
254.          try:
255.              request.propertyValue.cast_in(value)
256.          except Exception as error:
257.              return
258.
259.          # optional array index
260.          if indx is not None:
261.              request.propertyArrayIndex = indx
262.
263.          # optional priority
264.          if priority is not None:
265.              request.priority = priority
266.
267.          # make an IOCB
268.          iocb = IOCB(request)
269.
270.          # give it to the application
271.          deferred(BACnet_app.request_io, iocb)
272.
273.          # wait for it to complete
274.          iocb.wait()
275.
276.          # do something for success
277.          if iocb.ioResponse:
278.              # should be an ack
279.              if not isinstance(iocb.ioResponse, SimpleAckPDU):
280.                  return
281.              sys.stdout.write("ack\n")
282.
283.          # do something for error/reject/abort
284.          if iocb.ioError:
285.              sys.stdout.write(str(iocb.ioError) + '\n')
286.      except Exception as error:
287.          return
288.
289.  def do_writeHeaterStatus(value):
290.      args = ['192.168.2.120','binaryOutput:1004','presentValue']
291.      args.append(value)
292.      do_write(args)
293.
294.  def do_writeCoolingStatus(value):
295.      args = ['192.168.2.120','binaryOutput:1005','presentValue']
296.      args.append(value)
297.      do_write(args)
298.
299.  def do_writePauseMode(value):
300.      args = ['192.168.2.120','binaryOutput:1006','presentValue']
301.      args.append(value)
302.      do_write(args)
303.
```

```
304.
305.  def do_writeVentStatus(value):
306.      args = ['192.168.2.120','binaryOutput:1007','presentValue']
307.      args.append(value)
308.      do_write(args)
309.
310.  def do_writeAnalogValue(value):
311.      args = ['192.168.2.120','analogOutput:1000','presentValue']
312.      args.append(value)
313.      do_write(args)
314.
315.  # ------------------------------------------------------------------------------------------------------
316.  #  BACNET INITIALIZATION ---------------------------------------------------------------------------
317.  # ------------------------------------------------------------------------------------------------------
318.
319.  def initBacnet():
320.      global BACnet_device
321.      global BACnet_app
322.
323.      # parse the command line arguments
324.      args = ConfigArgumentParser(description=__doc__).parse_args()
325.
326.      # make a device object
327.      BACnet_device = LocalDeviceObject(
328.          objectName=args.ini.objectname,
329.          objectIdentifier=int(args.ini.objectidentifier),
330.          maxApduLengthAccepted=int(args.ini.maxapdulengthaccepted),
331.          segmentationSupported=args.ini.segmentationsupported,
332.          vendorIdentifier=int(args.ini.vendoridentifier),
333.          )
334.
335.      # make a simple application
336.      BACnet_app = BacnetApplication(BACnet_device, args.ini.address)
337.
338.      # enable sleeping will help with threads
339.      enable_sleeping()
340.
341.      # make the thread object and start it
342.      bacpypes_thread = Thread(target= BACpypesThread,name='bacnetApp',daemon=True)
343.      bacpypes_thread.start()
344.
```

2. DataModule.py

```
1.  from tkinter import *
2.  from BACnet.BACnetApp import do_readTemp, do_readHumidity, do_readDoorStatus
3.
4.
5.  class RoomData():
6.
7.      temp = None
8.      tempObjective = 28
9.      tempExt = 28
10.     humidity = None
11.     doorStatus = None
12.     windowStatus = 'CLOSED'
13.     analogValue = 0.0
14.
15.     heaterStatus = 'OFF'
16.     coolingStatus = 'OFF'
17.     pauseMode = 'OFF'
18.     ventStatus = 'OFF'
19.
20.
21.     def __init__(self):
22.         self.readSensors()
23.
24.     def readSensors(self):
25.         ''' Read sensor values and update the variables '''
26.         self.tempSet()
27.         self.humiditySet()
28.         self.doorStatusSet()
```

```
29.
30.     # Setters --------------------------------------------------------------------------------------
31.     '''From sensors'''
32.     def tempSet(self):
33.         self.temp = do_readTemp()
34.     def humiditySet(self):
35.         self.humidity = do_readHumidity()
36.     def doorStatusSet(self):
37.         self.doorStatus = do_readDoorStatus()
38.
39.     '''From the interface'''
40.     def tempObjectiveSet(self,value):
41.         self.tempObjective = value
42.     def tempExtSet(self,value):
43.         self.tempExt = value
44.     def analogValueSet(self,value):
45.         if (float(value) > 10):
46.             value = '10'
47.         self.analogValue = value
48.     def windowStatusSet(self,value):
49.         self.windowStatus = value
50.
51.     '''From the led panel'''
52.     def heaterStatusSet(self,value):
53.         self.heaterStatus = value
54.     def coolingStatusSet(self,value):
55.         self.coolingStatus = value
56.     def pauseModeSet(self,value):
57.         self.pauseMode = value
58.     def ventStatusSet(self,value):
59.         self.ventStatus = value
60.
61.     # Getters --------------------------------------------------------------------------------------
62.     ''' Inputs'''
63.     def tempGet(self):
64.         return self.temp
65.     def tempExtGet(self):
66.         return self.tempExt
67.     def tempObjectiveGet(self):
68.         return self.tempObjective
69.     def humidityGet(self):
70.         return self.humidity
71.     def doorStatusGet(self):
72.         return self.doorStatus
73.     def windowStatusGet(self):
74.         return self.windowStatus
75.     def analogValueGet(self):
76.         return self.analogValue
77.
78.     ''' Outputs'''
79.     def heaterStatusGet(self):
80.         return self.heaterStatus
81.     def coolingStatusGet(self):
82.         return self.coolingStatus
83.     def pauseModeGet(self):
84.         return self.pauseMode
85.     def ventStatusGet(self):
86.         return self.ventStatus
87.
```

3. iface.py

```
1.  from tkinter import *
2.  import time
3.  from BACnet.BACnetApp import do_iam, do_whois, do_readHumidity, do_readTemp, do_readDoorStatus, \
4.   do_writeHeaterStatus, do_writeCoolingStatus, do_writePauseMode, do_writeVentStatus, do_writeAnalogValue
5.
6.  #Define colors codes
7.  C_BG =           '#1a1a1a'
8.  C_TERMINAL =    '#242425'
9.  C_ORANGE =      '#ffaa00'
10. C_DARK_ORANGE = '#ff8a00'
11.
12.
```

```
13.
14. class interface():
15.
16.     # Declare flags to call events in the supervisor module
17.     openWindowRequest = False
18.     closeWindowRequest = False
19.     readAnalogValueRequest = False
20.     readExtTempRequest = False
21.     readObjTempRequest = False
22.     updateInterfaceRequest = False
23.     exitRequest = False
24.
25.     def getOpenWindowRequest(self):
26.         return self.openWindowRequest
27.     def getCloseWindowRequest(self):
28.         return self.closeWindowRequest
29.     def getreadAnalogValueRequest(self):
30.         return self.readAnalogValueRequest
31.     def getreadExtTempRequest(self):
32.         return self.readExtTempRequest
33.     def getreadObjTempRequest(self):
34.         return self.readObjTempRequest
35.     def getUpdateInterfaceRequest(self):
36.         return self.updateInterfaceRequest
37.     def getExitRequest(self):
38.         return self.exitRequest
39.
40.     def raiseOpenWindowRequest(self):
41.         self.openWindowRequest = True
42.     def raiseCloseWindowRequest(self):
43.         self.closeWindowRequest = True
44.     def raiseReadAnalogValueRequest(self,e):
45.         self.readAnalogValueRequest = True
46.     def raiseReadExtTempRequest(self,e):
47.         self.readExtTempRequest = True
48.     def raiseReadObjTempRequest(self,e):
49.         self.readObjTempRequest = True
50.     def raiseUpdateInterfaceRequest(self):
51.         self.updateInterfaceRequest = True
52.
53.     def clearOpenWindowRequest(self):
54.         self.openWindowRequest = False
55.     def clearCloseWindowRequest(self):
56.         self.closeWindowRequest = False
57.     def clearReadAnalogValueRequest(self):
58.         self.readAnalogValueRequest = False
59.     def clearReadExtTempRequest(self):
60.         self.readExtTempRequest = False
61.     def clearReadObjTempRequest(self):
62.         self.readObjTempRequest = False
63.     def clearUpdateInterfaceRequest(self):
64.         self.updateInterfaceRequest = False
65.
66.     def __init__(self):
67.
68.         #-----------------------------------------------------------------------------------------------
69.         # WINDOW CONFIGURATION ---------------------------------------------------------------------------
70.         #-----------------------------------------------------------------------------------------------
71.
72.         self.root = Tk()
73.         self.root.geometry("1220x650+500+200")
74.         self.root.configure(bg=C_BG, bd=0,highlightthickness=0)
75.         self.root.attributes('-alpha', 0.4)
76.         self.root.overrideredirect(False)
77.
78.         #-----------------------------------------------------------------------------------------------
79.         # CONSOLE CONFIGURATION --------------------------------------------------------------------------
80.         #-----------------------------------------------------------------------------------------------
81.
82.         self.Console = Frame(self.root,width=250,height=650 ,bg=C_TERMINAL, relief="raised",bd=0)
83.         self.Console.config(highlightbackground=C_DARK_ORANGE,highlightcolor=C_DARK_ORANGE)
84.         self.Console.config(highlightthickness=2)
85.         self.Console.pack(side=LEFT,expand=1,fill=Y,padx=(5,0),pady=(5,5))
86.
87.         self.Console2 = Frame(self.Console,height=50, width=200 ,bg=C_TERMINAL, relief="raised",bd=0)
88.         self.Console2.pack(side=TOP)
89.
```

```
90.
91.         self.Console3 = Frame(self.Console,height=400, relief="raised",bd=0)
92.         self.Console3.pack(side=BOTTOM)
93.
94.         self.console_terminal = Text(self.Console2,bg=C_TERMINAL, fg=C_ORANGE, borderwidth=0)
95.         self.console_terminal.config(height='32',width='37',wrap=WORD, font='Console 12' )
96.         self.console_terminal.pack(expand=1)
97.         self.console_terminal.insert(INSERT,">> ")
98.         self.console_terminal.config(state=DISABLED)
99.
100.          self.console_icon = Label(self.Console3,text='>',width=4,bg=C_DARK_ORANGE)
101.          self.console_icon.config(font='Console 14 bold',fg=C_BG, borderwidth=2)
102.        self.console_icon.pack(side=LEFT)
103.
104.        self.console_input = Entry(self.Console3, width=26, bg=C_BG, fg=C_ORANGE, borderwidth=0)
105.        self.console_input.config(highlightbackground=C_DARK_ORANGE, borderwidth=0,highlightthickness=2)
106.        self.console_input.pack(side=RIGHT)
107.        self.console_input.bind('<Return>',self.read_Entry)
108.
109.          #-----------------------------------------------------------------------------------------
110.          # DATA MONITOR DISPLAY -------------------------------------------------------------------
111.          #-----------------------------------------------------------------------------------------
112.
113.          self.Display = Frame(self.root, width=1200, height=650, bg=C_TERMINAL, relief="raised", bd=0)
114.          self.Display.config(highlightbackground=C_DARK_ORANGE, highlightthickness=2)
115.          self.Display.pack(side=RIGHT,expand=1,fill='both',padx=(5,5),pady=(5,5),ipadx=10,ipady=10)
116.
117.          # Variables Frames -----------------------------------------------------------------------
118.          self.VariablesFrame = Frame(self.Display, width=1200, height=650,bg=C_TERMINAL, bd=0)
119.          self.VariablesFrame.config(highlightbackground=C_DARK_ORANGE, highlightthickness=2)
120.          self.VariablesFrame.pack(side=TOP,padx=(15,15),pady=(15,10))
121.
122.          self.tempVariables = Frame(self.VariablesFrame, width=1200, height=650,bg=C_TERMINAL, bd=0)
123.          self.tempVariables.config(highlightthickness=0)
124.          self.tempVariables.pack(side=TOP,anchor="w",pady=(10,5))
125.
126.          self.tempVariables2 = Frame(self.VariablesFrame, width=1200, height=650,bg=C_TERMINAL, bd=0)
127.          self.tempVariables2.config(highlightthickness=0)
128.          self.tempVariables2.pack(side=TOP,anchor="w",pady=(5,5))
129.
130.          self.tempVariables3 = Frame(self.VariablesFrame, width=1200, height=600,bg=C_TERMINAL, bd=0)
131.          self.tempVariables3.config(highlightthickness=0)
132.          self.tempVariables3.pack(side=TOP,anchor="w",pady=(5,5))
133.
134.          self.HumidityVariable = Frame(self.VariablesFrame, width=1200, height=600,bg=C_TERMINAL, bd=0)
135.          self.HumidityVariable.config(highlightthickness=0)
136.          self.HumidityVariable.pack(side=TOP,anchor="w",pady=(5,5))
137.
138.          self.doorStatusVariable = Frame(self.VariablesFrame, width=1200, height=600,bg=C_TERMINAL)
139.          self.doorStatusVariable.config(bd=0, highlightthickness=0)
140.          self.doorStatusVariable.pack(side=TOP,anchor="w",pady=(5,5))
141.
142.          self.windowStatusVariable = Frame(self.VariablesFrame, width=1200, height=600,bg=C_TERMINAL)
143.          self.windowStatusVariable.config(bd=0, highlightthickness=0)
144.          self.windowStatusVariable.pack(side=TOP,anchor="w",pady=(5,5))
145.
146.          self.analogValueVariable = Frame(self.VariablesFrame, width=1200, height=600,bg=C_TERMINAL)
147.          self.analogValueVariable.config(bd=0, highlightthickness=0)
148.          self.analogValueVariable.pack(side=TOP,anchor="w",pady=(5,5))
149.
150.          self.spacing = Frame(self.VariablesFrame, width=1200, height=10,bg=C_TERMINAL, bd=0)
151.          self.spacing.config(highlightthickness=0)
152.          self.spacing.pack(side=TOP,anchor="w")
153.
154.          # Temp Variable --------------------------------------------------------------------------
155.          self.tempLabel = Label(self.tempVariables, text='Temperature:', width=14, bg=C_TERMINAL)
156.          self.tempLabel.config(font='Console 18',fg=C_DARK_ORANGE)
157.          self.tempLabel.pack(side=LEFT,padx=(2,0),anchor="w")
158.
159.          self.tempValue = Frame(self.tempVariables, width=1200, height=600,bg=C_TERMINAL, bd=0)
160.          self.tempValue.config(highlightthickness=0)
161.          self.tempValue.pack(side=RIGHT,padx=(2,0),anchor="w")
162.
163.
164.
165.
166.
```

```
167.
168.            self.tempValueLabel = Label(self.tempValue, text='30.2',width=5,bg=C_TERMINAL)
169.            self.tempValueLabel.config(font='Console 18',fg=C_DARK_ORANGE)
170.            self.tempValueLabel.pack(side=LEFT,anchor="w")
171.
172.            self.tempUnitsLabel = Label(self.tempValue, text='°C',width=2,bg=C_TERMINAL)
173.            self.tempUnitsLabel.config(font='Console 18',fg=C_DARK_ORANGE)
174.            self.tempUnitsLabel.pack(side=RIGHT,anchor="w")
175.
176.            # Objective Temp Variable ---------------------------------------------------------------------
177.            self.objTempLabel = Label(self.tempVariables2, text='Objective Temperature:', width=22)
178.            self.objTempLabel.pack(bg=C_TERMINAL,font='Console 18',fg=C_DARK_ORANGE)
179.            self.objTempLabel.pack(side=LEFT,padx=(3,0),anchor="w")
180.
181.            self.objTempValue = Frame(self.tempVariables2, width=1200, height=600,bg=C_TERMINAL, bd=0)
182.            self.objTempValue.config(highlightthickness=0)
183.            self.objTempValue.pack(side=RIGHT,padx=(3,0),anchor="w")
184.
185.            self.objTempValueLabel = Entry(self.objTempValue,width=4,bg=C_TERMINAL)
186.            self.objTempValueLabel.config(justify='right', font='Console 18', fg=C_DARK_ORANGE)
187.            self.objTempValueLabel.pack(side=LEFT,anchor="w")
188.            self.objTempValueLabel.bind('<Return>',self.raiseReadObjTempRequest)
189.            self.objTempValueLabel.bind('<FocusOut>',self.raiseReadObjTempRequest)
190.
191.            self.objTempUnitsLabel = Label(self.objTempValue, text='°C', width=2, bg=C_TERMINAL)
192.            self.objTempUnitsLabel.config(font='Console 18',fg=C_DARK_ORANGE)
193.            self.objTempUnitsLabel.pack(side=RIGHT,padx=(6,0),anchor="w")
194.
195.            # External Temp Variable
196.            self.extTempLabel = Label(self.tempVariables3, text='External Temperature:', width=21)
197.            self.extTempLabel.config(bg=C_TERMINAL,font='Console 18',fg=C_DARK_ORANGE)
198.            self.extTempLabel.pack(side=LEFT,padx=(4,0),anchor="w")
199.
200.            self.extTempValue = Frame(self.tempVariables3, width=1200, height=600,bg=C_TERMINAL, bd=0)
201.            self.extTempValue.config(highlightthickness=0)
202.            self.extTempValue.pack(side=RIGHT,anchor="w")
203.
204.            self.extTempValueLabel = Entry(self.extTempValue,width=4,bg=C_TERMINAL)
205.            self.extTempValueLabel.config(justify='right', font='Console 18', fg=C_DARK_ORANGE)
206.            self.extTempValueLabel.pack(side=LEFT,padx=(16,0),anchor="w")
207.            self.extTempValueLabel.bind('<Return>',self.raiseReadExtTempRequest)
208.            self.extTempValueLabel.bind('<FocusOut>',self.raiseReadExtTempRequest)
209.
210.            self.extTempUnitsLabel = Label(self.extTempValue, text='°C', width=2, bg=C_TERMINAL)
211.            self.extTempUnitsLabel.config(font='Console 18',fg=C_DARK_ORANGE)
212.            self.extTempUnitsLabel.pack(side=RIGHT,padx=(6,0),anchor="w")
213.
214.            # Humidity Variable
215.            self.HumidityLabel = Label(self.HumidityVariable, text='Humidity:', width=8, bg=C_TERMINAL)
216.            self.HumidityLabel.config(font='Console 18',fg=C_DARK_ORANGE)
217.            self.HumidityLabel.pack(side=LEFT,padx=(23,0),anchor="w")
218.
219.            self.HumidityValue = Frame(self.HumidityVariable, width=1200, height=600,bg=C_TERMINAL, bd=0)
220.            self.HumidityValue.config(highlightthickness=0)
221.            self.HumidityValue.pack(side=RIGHT,anchor="w")
222.
223.            self.HumidityValueLabel = Label(self.HumidityValue, text='40', width=5, bg=C_TERMINAL)
224.            self.HumidityValueLabel.config(font='Console 18',fg=C_DARK_ORANGE)
225.            self.HumidityValueLabel.pack(side=LEFT,anchor="w")
226.
227.            self.HumidityUnitsLabel = Label(self.HumidityValue, text='%', width=2, bg=C_TERMINAL)
228.            self.HumidityUnitsLabel.config(font='Console 18',fg=C_DARK_ORANGE)
229.            self.HumidityUnitsLabel.pack(side=RIGHT,anchor="w")
230.
231.            # Door Status Variable
232.            self.doorStatusLabel = Label(self.doorStatusVariable, text='Door State:', width=9)
233.            self.doorStatusLabel.config(bg=C_TERMINAL,font='Console 18',fg=C_DARK_ORANGE)
234.            self.doorStatusLabel.pack(side=LEFT,padx=(24,0),anchor="w")
235.
236.            self.doorStatusValue = Frame(self.doorStatusVariable, width=1200, height=600,bg=C_TERMINAL)
237.            self.doorStatusValue.config(bd=0, highlightthickness=0)
238.            self.doorStatusValue.pack(side=RIGHT,anchor="w")
239.
240.            self.doorStatusValueLabel = Label(self.doorStatusValue, text='OPEN', width=6, bg=C_TERMINAL)
241.            self.doorStatusValueLabel.config(font='Console 18',fg=C_DARK_ORANGE)
242.            self.doorStatusValueLabel.pack(side=LEFT,anchor="w",padx=(4,0))
243.
```

```
244.
245.          # Window State Variable
246.          self.windowStatusLabel = Label(self.windowStatusVariable, text='Window State:', width=12)
247.          self.windowStatusLabel.config(bg=C_TERMINAL,font='Console 18',fg=C_DARK_ORANGE)
248.          self.windowStatusLabel.pack(side=LEFT,padx=(24,0),anchor="w")
249.
250.          self.windowStatusValue = Frame(self.windowStatusVariable, width=1200, height=600)
251.          self.windowStatusValue.config(bg=C_TERMINAL, bd=0, highlightthickness=0)
252.          self.windowStatusValue.pack(side=RIGHT,anchor="w")
253.
254.          self.windowStatusValueLabel = Label(self.windowStatusValue, text='CLOSE', width=6)
255.          self.windowStatusValueLabel.config(bg=C_TERMINAL, font='Console 18', fg=C_DARK_ORANGE)
256.          self.windowStatusValueLabel.pack(side=LEFT,anchor="w",padx=(4,0))
257.
258.          # Analog Value Variable
259.          self.analogLabel = Label(self.analogValueVariable, text='Analog Value:', width=12)
260.          self.analogLabel.config(bg=C_TERMINAL, font='Console 18', fg=C_DARK_ORANGE)
261.          self.analogLabel.pack(side=LEFT,padx=(20,0),anchor="w")
262.
263.          self.analogValue = Frame(self.analogValueVariable, width=1200, height=600,bg=C_TERMINAL)
264.          self.analogValue.config(bd=0, highlightthickness=0)
265.          self.analogValue.pack(side=RIGHT,anchor="w")
266.
267.          self.analogValueLabel = Entry(self.analogValue,width=4,bg=C_TERMINAL)
268.          self.analogValueLabel.config(justify='right', font='Console 18',fg=C_DARK_ORANGE))
269.          self.analogValueLabel.pack(side=LEFT,padx=(16,0),anchor="w")
270.          self.analogValueLabel.bind('<Return>',self.raiseReadAnalogValueRequest)
271.          self.analogValueLabel.bind('<FocusOut>',self.raiseReadAnalogValueRequest)
272.
273.          self.analogValueUnitsLabel = Label(self.analogValue, text='V', width=2, bg=C_TERMINAL)
274.          self.analogValueUnitsLabel.config(font='Console 18',fg=C_DARK_ORANGE)
275.          self.analogValueUnitsLabel.pack(side=RIGHT,padx=(6,0),anchor="w")
276.
277.          #-----------------------------------------------------------------------------------------
278.          # LED PANEL--------------------------------------------------------------------------------
279.          #-----------------------------------------------------------------------------------------
280.
281.          self.ledPanelFrame = Frame(self.Display, width=1200, height=600, bg=C_TERMINAL, bd=0)
282.          self.ledPanelFrame.config(highlightthickness=2, highlightbackground=C_DARK_ORANGE)
283.          self.ledPanelFrame.pack(side=BOTTOM, padx=(10,10), pady=(15, 15))
284.
285.          self.ledPanelFrameLeft = Frame(self.ledPanelFrame, width=1200, height=600, bg=C_TERMINAL,bd=0)
286.          self.ledPanelFrameLeft.pack(side=LEFT, padx=(15,15), pady=(15, 10))
287.
288.          self.ledPanelFrameRight = Frame(self.ledPanelFrame, width=1200, height=600,bg=C_TERMINAL,bd=0)
289.          self.ledPanelFrameRight.pack(side=RIGHT, padx=(0,15), pady=(15, 10))
290.
291.          # Led images definition ------------------------------------------------------------------
    -
292.          self.ledOFF = PhotoImage(file='GUI/resources/ledOff.png')
293.          self.ledRED = PhotoImage(file='GUI/resources/ledRed.png')
294.          self.ledGREEN = PhotoImage(file='GUI/resources/ledGreen.png')
295.          self.ledYELLOW = PhotoImage(file='GUI/resources/ledYellow.png')
296.          self.ledBLUE = PhotoImage(file='GUI/resources/ledBlue.png')
297.
298.          # RED led --------------------------------------------------------------------------------
299.          self.led1Frame = Frame(self.ledPanelFrameLeft, width= 175, height=600, bg=C_TERMINAL, bd=0)
300.          self.led1Frame.pack(side=LEFT, padx=(0,15))
301.
302.          self.led1 = Label(self.led1Frame, image=self.ledOFF, bg=C_TERMINAL, borderwidth=0)
303.          self.led1.pack(side=TOP)
304.
305.          self.led1Label = Label(self.led1Frame, text='Heater', width=6,fg=C_DARK_ORANGE)
306.          self.led1Label.config(font='Console 18',bg=C_TERMINAL)
307.          self.led1Label.pack(side=BOTTOM,pady=(5,5))
308.
309.          # BLUE led -------------------------------------------------------------------------------
310.          self.led2Frame = Frame(self.ledPanelFrameLeft, width= 175, height=600, bg=C_TERMINAL, bd=0)
311.          self.led2Frame.pack(side=RIGHT)
312.
313.          self.led2 = Label(self.led2Frame, image=self.ledOFF, bg=C_TERMINAL, borderwidth=0)
314.          self.led2.pack(side=TOP)
315.
316.          self.led2Label = Label(self.led2Frame, text='Cooling', width=6,fg=C_DARK_ORANGE)
317.          self.led2Label.pack(font='Console 18',bg=C_TERMINAL)
318.          self.led2Label.config(side=BOTTOM,pady=(5,5))
319.
```

```
320.
321.          # YELLOW led ------------------------------------------------------------------------------
322.          self.led3Frame = Frame(self.ledPanelFrameRight, width= 175, height=600, bg=C_TERMINAL, bd=0)
323.          self.led3Frame.pack(side=LEFT,  padx=(0,15))
324.
325.          self.led3 = Label(self.led3Frame, image=self.ledOFF, bg=C_TERMINAL, borderwidth=0)
326.          self.led3.pack(side=TOP)
327.
328.          self.led3Label = Label(self.led3Frame, text='System paused', width=12, fg=C_DARK_ORANGE)
329.          self.led3Label.config(font='Console 16',bg=C_TERMINAL))
330.          self.led3Label.pack(side=BOTTOM,pady=(5,5))
331.
332.          # GREEN led
333.          self.led4Frame = Frame(self.ledPanelFrameRight, width= 175, height=600, bg=C_TERMINAL, bd=0)
334.          self.led4Frame.pack(side=RIGHT)
335.
336.          self.led4 = Label(self.led4Frame, image=self.ledOFF, bg=C_TERMINAL, borderwidth=0)
337.          self.led4.pack(side=TOP)
338.
339.          self.led4Label = Label(self.led4Frame, text='Vents Opened', width=12, fg=C_DARK_ORANGE)
340.          self.led4Label.config(font='Console 16',bg=C_TERMINAL))
341.          self.led4Label.pack(side=BOTTOM,pady=(5,5))
342.
343.      #-------------------------------------------------------------------------------------------------
344.      # INTERFACE FUNCTIONS ----------------------------------------------------------------------------
345.      #-------------------------------------------------------------------------------------------------
346.
347.      # Led ON/OFF Functions
348.      def setRedLedON(self):
349.          self.led1.config(image=self.ledRED)
350.      def setRedLedOFF(self):
351.          self.led1.config(image=self.ledOFF)
352.
353.      def setBlueLedON(self):
354.          self.led2.config(image=self.ledBLUE)
355.      def setBlueLedOFF(self):
356.          self.led2.config(image=self.ledOFF)
357.
358.      def setYellowLedON(self):
359.          self.led3.config(image=self.ledYELLOW)
360.      def setYellowLedOFF(self):
361.          self.led3.config(image=self.ledOFF)
362.
363.      def setGreenLedON(self):
364.          self.led4.config(image=self.ledGREEN)
365.      def setGreenLedOFF(self):
366.          self.led4.config(image=self.ledOFF)
367.
368.      #-------------------------------------------------------------------------------------------------
369.      # CONSOLE FUNCTIONS ------------------------------------------------------------------------------
370.      #-------------------------------------------------------------------------------------------------
371.
372.      def read_Entry(self,e):
373.          ''' Gets the text value of the entry widget then calls to the read_command function'''
374.          try:
375.              input = self.console_input.get()
376.              self.read_command(input)
377.          except:
378.              return
379.
380.      def clear_text(self):
381.          ''' Cleans the console text widget'''
382.          self.console_terminal.configure(state='normal')
383.          self.console_terminal.delete('1.0', END)
384.          self.console_terminal.insert(INSERT, ">> ")
385.          self.console_terminal.see('end')
386.          self.console_terminal.config(state=DISABLED)
387.
388.      def update_text(self,order,response):
389.          ''' Displays the response and the previous command'''
390.          self.console_terminal.configure(state='normal')
391.          self.console_terminal.insert(END,order)
392.          self.console_terminal.insert(END,"\n" + response)
393.          self.console_terminal.insert(END,"\n>> ")
394.          self.console_terminal.see('end')
395.          self.console_terminal.configure(state=DISABLED)
396.
```

```
397.
398.      def read_command(self,input):
399.          ''' Parses the input to get the command then checks if correspond with a known command'''
400.          # Parse the console input
401.          command = input.strip().lower().split(' ',1)
402.
403.          if (len(command) > 1):
404.              args = command[1].split()
405.          else:
406.              args = []
407.
408.          if (command[0] == 'test'):
409.              self.update_text(command[0],'This is a test :)')
410.
411.          elif (command[0] == 'clear' or command[0] == 'cls'):
412.             self.clear_text()
413.
414.          elif (command[0] == 'exit'):
415.              self.exitRequest = True
416.              time.sleep(2)
417.              self.root.quit()
418.
419.          elif (command[0] == 'whois'):
420.              do_whois(args)
421.              self.update_text(command[0],'WhoIs sended')
422.
423.          elif (command[0] == 'iam'):
424.              do_iam()
425.              self.update_text(command[0],'Iam  sended')
426.
427.          elif (command[0] == 'read'):
428.              if (len(args) == 0 or len(args) > 1):
429.                  self.update_text(command[0],'Error. Invalid arguments')
430.
431.              elif (args[0] == 'temp'):
432.                  value = do_readTemp()
433.                  response = 'Temp: ' + str(value) + '°C'
434.                  self.update_text(command[0] + ' ' + args[0],response)
435.
436.              elif (args[0] == 'humidity'):
437.                  value = do_readHumidity()
438.                  response = 'Humidity: ' + str(value) + '%'
439.                  self.update_text(command[0] +  ' ' + args[0],response)
440.
441.              elif (args[0] == 'door'):
442.                  value = do_readDoorStatus()
443.                  response = 'DoorStatus: ' + str(value)
444.                  self.update_text(command[0] +  ' ' + args[0],response)
445.
446.          elif (command[0] == 'open'):
447.              self.raiseOpenWindowRequest()
448.              response = 'Window status set to open'
449.              self.update_text(command[0],response)
450.
451.          elif (command[0] == 'close'):
452.              self.raiseCloseWindowRequest()
453.              response = 'Window status set to closed'
454.              self.update_text(command[0],response)
455.
456.          elif (command[0] == 'update'):
457.              self.raiseUpdateInterfaceRequest()
458.              response = 'Variables values updated'
459.              self.update_text(command[0],response)
460.
461.          elif (command[0] == 'led'):
462.              if (len(args) == 1 ):
463.                  self.update_text(command[0] + ' ' + args[0],'Error. Invalid arguments')
464.
465.              elif (len(args) == 0 or len(args) > 2):
466.                  self.update_text(command[0],'Error. Invalid arguments')
467.
468.              elif (args[0] == 'red'):
469.                  if (args[1] == 'on'):
470.                      self.setRedLedON()
471.                      do_writeVentStatus('active')
472.                      response = 'Red led state on'
473.                      self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],response)
```

```
474.
475.                 elif (args[1] == 'off'):
476.                     self.setRedLedOFF()
477.                     do_writeVentStatus('inactive')
478.                     response = 'Red led state off'
479.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],response)
480.                 else:
481.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],'Error. ' + args[1] +
    ' Invalid argument')
482.
483.             elif (args[0] == 'blue'):
484.                 if (args[1] == 'on'):
485.                     self.setBlueLedON()
486.                     do_writeCoolingStatus('active')
487.                     response = 'Blue led state on'
488.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],response)
489.                 elif (args[1] == 'off'):
490.                     self.setBlueLedOFF()
491.                     do_writeCoolingStatus('inactive')
492.                     response = 'Blue led state off'
493.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],response)
494.                 else:
495.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],'Error. ' + args[1] +
    ' Invalid argument')
496.
497.             elif (args[0] == 'yellow'):
498.                 if (args[1] == 'on'):
499.                     self.setYellowLedON()
500.                     do_writePauseMode('active')
501.                     response = 'Yellow led state on'
502.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],response)
503.                 elif (args[1] == 'off'):
504.                     self.setYellowLedOFF()
505.                     do_writePauseMode('inactive')
506.                     response = 'Yellow led state off'
507.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],response)
508.                 else:
509.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],'Error. ' + args[1] +
    ' Invalid argument')
510.
511.             elif (args[0] == 'green'):
512.                 if (args[1] == 'on'):
513.                     self.setGreenLedON()
514.                     do_writeVentStatus('active')
515.                     response = 'Green led state on'
516.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],response)
517.                 elif (args[1] == 'off'):
518.                     self.setGreenLedOFF()
519.                     do_writeVentStatus('inactive')
520.                     response = 'Green led state off'
521.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],response)
522.                 else:
523.                     self.update_text(command[0] + ' ' + args[0] + ' ' + args[1],'Error. ' + args[1] +
    ' Invalid argument')
524.
525.             else:
526.                 self.update_text(command[0] + ' ' + args[0],'Error. ' + args[0] + ' Invalid argument')
527.
528.         else:
529.             self.update_text('Command Error: ' , command[0] + " command not defined")
530.
531.         self.console_input.delete(0,'end')
532.
533.
```

4. SupervisorModule.py

```
1.  from tkinter import *
2.  import time
3.
4.  from threading import Thread
5.  from DataModule import RoomData
6.  from GUI.iface import interface
7.  from BACnet.BACnetApp import do_writeHeaterStatus, do_writeCoolingStatus, do_writePauseMode, \
    do_writeVentStatus, do_writeAnalogValue
8.
9.
10. class Supervisor(RoomData,Thread,interface):
11.
12.     def __init__(self,RoomData,Thread,interface):
13.
14.         # Update the interface
15.         self.updateInterface(interface,RoomData)
16.
17.         # make the threads objects and start them
18.         supervisor_thread = Thread(target=
    self.supervisorTask,name='supervisor',args=(RoomData,interface,))
19.         application_thread = Thread(target=
    self.applicationTask,name='application',args=(RoomData,interface,),daemon=True)
20.
21.         supervisor_thread.start()
22.         application_thread.start()
23.
24.
25.     #-------------------------------------------------------------------------------------------------
26.     # TASKS ------------------------------------------------------------------------------------------
27.     #-------------------------------------------------------------------------------------------------
28.     def supervisorTask(self,RoomData,interface):
29.         ''' Monitors the value of the flags in the interface module'''
30.
31.         while(interface.getExitRequest() == False):
32.             self.readFlags(RoomData,interface)
33.             time.sleep(0.2)
34.
35.     def applicationTask(self,RoomData,interface):
36.         ''' Update the values of the sensorS in the data module then updates the interface labels'''
37.
38.         self.updateInterface(interface,RoomData)
39.         time.sleep(300)
40.
41.     #-------------------------------------------------------------------------------------------------
42.     #  SUPERVISOR FUNCTIONS ---------------------------------------------------------------------------
43.     #-------------------------------------------------------------------------------------------------
44.
45.     def updateInterface(self,interface,RoomData):
46.         ''' Updates the sensors values then updates the labels for each variable'''
47.
48.         # Update the data Base
49.         RoomData.readSensors()
50.
51.         # Update the system status
52.         self.updateStatus(RoomData,interface)
53.
54.         # Update the actuators values
55.         self.updateActuators(RoomData)
56.
57.         # Update the Labels
58.         interface.tempValueLabel.config(text=str(RoomData.tempGet()))
59.         interface.HumidityValueLabel.config(text=str(RoomData.humidityGet()))
60.         interface.doorStatusValueLabel.config(text=str(RoomData.doorStatusGet()))
61.         interface.windowStatusValueLabel.config(text=str(RoomData.windowStatusGet()))
62.
63.         interface.analogValueLabel.delete(0,END)
64.         interface.analogValueLabel.insert(0,str(RoomData.analogValueGet()))
65.         interface.extTempValueLabel.delete(0,END)
66.         interface.extTempValueLabel.insert(0,str(RoomData.tempExtGet()))
67.         interface.objTempValueLabel.delete(0,END)
68.         interface.objTempValueLabel.insert(0,str(RoomData.tempObjectiveGet()))
69.
70.
```

```
71.
72.         # Update the Led Panel
73.         if (RoomData.heaterStatusGet() == 'ON'):
74.             interface.setRedLedON()
75.         else:
76.             interface.setRedLedOFF()
77.
78.         if (RoomData.coolingStatusGet() == 'ON'):
79.             interface.setBlueLedON()
80.         else:
81.             interface.setBlueLedOFF()
82.
83.         if (RoomData.pauseModeGet() == 'ON'):
84.             interface.setYellowLedON()
85.         else:
86.             interface.setYellowLedOFF()
87.
88.         if (RoomData.ventStatusGet() == 'ON'):
89.             interface.setGreenLedON()
90.         else:
91.             interface.setGreenLedOFF()
92.
93.     def updateStatus(self,RoomData,interface):
94.         ''' Computes the current the status for each output'''
95.
96.         if (RoomData.doorStatusGet() == 'OPEN' or RoomData.windowStatusGet() == 'OPEN'):
97.             RoomData.pauseModeSet('ON')
98.             RoomData.heaterStatusSet('OFF')
99.             RoomData.coolingStatusSet('OFF')
100.            RoomData.ventStatusSet('OFF')
101.
102.         elif (RoomData.doorStatusGet() == 'CLOSED' and RoomData.windowStatusGet() == 'CLOSED'):
103.
104.             RoomData.pauseModeSet('OFF')
105.
106.             if(RoomData.tempGet() > float(RoomData.tempObjectiveGet())):
107.                 if(RoomData.tempGet() > (float(RoomData.tempExtGet()) + 2)):
108.                     RoomData.ventStatusSet('ON')
109.                     RoomData.coolingStatusSet('OFF')
110.                     RoomData.heaterStatusSet('OFF')
111.                 else:
112.                     RoomData.ventStatusSet('OFF')
113.                     RoomData.coolingStatusSet('ON')
114.                     RoomData.heaterStatusSet('OFF')
115.
116.             if(RoomData.tempGet() < float(RoomData.tempObjectiveGet())):
117.                 if(RoomData.tempGet() < (float(RoomData.tempExtGet()) - 2)):
118.                     RoomData.ventStatusSet('ON')
119.                     RoomData.coolingStatusSet('OFF')
120.                     RoomData.heaterStatusSet('OFF')
121.                 else:
122.                     RoomData.ventStatusSet('OFF')
123.                     RoomData.coolingStatusSet('ON')
124.                     RoomData.heaterStatusSet('OFF')
125.
126.     def updateActuators(self, RoomData):
127.
128.         if (RoomData.heaterStatusGet() == 'ON'):
129.             do_writeHeaterStatus('active')
130.         else:
131.             do_writeHeaterStatus('inactive')
132.
133.         if (RoomData.coolingStatusGet() == 'ON'):
134.             do_writeCoolingStatus('active')
135.         else:
136.             do_writeCoolingStatus('inactive')
137.
138.         if (RoomData.pauseModeGet() == 'ON'):
139.             do_writePauseMode('active')
140.         else:
141.             do_writePauseMode('inactive')
142.
143.         if (RoomData.ventStatusGet() == 'ON'):
144.             do_writeVentStatus('active')
145.         else:
146.             do_writeVentStatus('inactive')
147.             do_writeAnalogValue(float(RoomData.analogValueGet()))
```

```
148.
149.      def openWindow(self,RoomData):
150.          RoomData.windowStatusSet('OPEN')
151.      def closeWindow(self,RoomData):
152.          RoomData.windowStatusSet('CLOSED')
153.
154.      def readFlags(self,RoomData,interface):
155.          ''' Reads the interface requests flags value '''
156.
157.          OpenRequest = interface.getOpenWindowRequest()
158.          CloseRequest = interface.getCloseWindowRequest()
159.          UpdateRequest = interface.getUpdateInterfaceRequest()
160.          updateExtTempRequest = interface.getreadExtTempRequest()
161.          updateObjTempRequest = interface.getreadObjTempRequest()
162.          updateAnalogValueRequest = interface.getreadAnalogValueRequest()
163.
164.          # Change the Door Status if a Request is actived
165.          if (OpenRequest == True and CloseRequest == False):
166.              self.openWindow(RoomData)
167.              interface.clearOpenWindowRequest()
168.              interface.raiseUpdateInterfaceRequest()
169.          elif (CloseRequest == True and OpenRequest == False):
170.              self.closeWindow(RoomData)
171.              interface.clearCloseWindowRequest()
172.              interface.raiseUpdateInterfaceRequest()
173.          elif (OpenRequest == True and CloseRequest == True):
174.              interface.clearOpenWindowRequest()
175.              interface.clearCloseWindowRequest()
176.
177.          # Read the update external and objective temperature Flags
178.          if (updateExtTempRequest):
179.              RoomData.tempExtSet(interface.extTempValueLabel.get())
180.              interface.clearReadExtTempRequest()
181.              interface.raiseUpdateInterfaceRequest()
182.
183.          if (updateObjTempRequest):
184.              RoomData.tempObjectiveSet(interface.objTempValueLabel.get())
185.              interface.clearReadObjTempRequest()
186.              interface.raiseUpdateInterfaceRequest()
187.
188.          if (updateAnalogValueRequest):
189.              RoomData.analogValueSet(interface.analogValueLabel.get())
190.              interface.clearReadAnalogValueRequest()
191.              interface.raiseUpdateInterfaceRequest()
192.
193.          # Read Interface Update Request Flags
194.          if (UpdateRequest == True):
195.              self.updateInterface(interface, RoomData)
196.              interface.clearUpdateInterfaceRequest()
197.
```

5. App.py

```
1.  from threading import Thread
2.  from GUI.iface import interface
3.  from BACnet.BACnetApp import BacnetApplication, initBacnet, BACnet_app
4.  from DataModule import RoomData
5.  from SupervisorModule import Supervisor
6.
7.
8.  def main():
9.
10.     app = interface()
11.     initBacnet()
12.     dataModule = RoomData()
13.     supervisorProcess = Supervisor(dataModule,Thread,app)
14.     app.root.mainloop()
15.
16. if __name__ == '__main__':
17.     main()
18.
```

6. BACnetConsole.py

```
1.  import sys
2.
3.  from bacpypes.debugging import bacpypes_debugging, ModuleLogger
4.  from bacpypes.consolelogging import ConfigArgumentParser
5.  from bacpypes.consolecmd import ConsoleCmd
6.
7.  from bacpypes.core import run, deferred, enable_sleeping
8.  from bacpypes.iocb import IOCB
9.
10. from bacpypes.pdu import Address, GlobalBroadcast
11. from bacpypes.object import get_datatype
12.
13. from bacpypes.apdu import SimpleAckPDU, \
14.     ReadPropertyRequest, ReadPropertyACK, WritePropertyRequest, WhoIsRequest, IAmRequest
15. from bacpypes.primitivedata import Null, Atomic, Boolean, Unsigned, Integer, \
16.     Real, Double, OctetString, CharacterString, BitString, Date, Time, ObjectIdentifier
17. from bacpypes.constructeddata import Array, Any, AnyAtomic
18. from bacpypes.errors import DecodingError
19.
20. from bacpypes.app import BIPSimpleApplication
21. from bacpypes.local.device import LocalDeviceObject
22.
23. # some debugging
24. _debug = 1
25. _log = ModuleLogger(globals())
26.
27. # globals
28. this_device = None
29. this_application = None
30.
31. class BacnetApplication(BIPSimpleApplication):
32.
33.     def __init__(self, *args):
34.         if _debug: BacnetApplication._debug("__init__ %r", args)
35.         BIPSimpleApplication.__init__(self, *args)
36.
37.         # keep track of requests to line up responses
38.         self._request = None
39.
40.     def request(self, apdu):
41.         if _debug: BacnetApplication._debug("request %r", apdu)
42.
43.         # save a copy of the request
44.         if isinstance(apdu, WhoIsRequest):
45.             self._request = apdu
46.         # forward it along
47.         BIPSimpleApplication.request(self, apdu)
48.
49.     def indication(self, apdu):
50.         if _debug: BacnetApplication._debug("indication %r", apdu)
51.
52.         if not self._request:
53.             if _debug: BacnetApplication._debug("    - no pending request")
54.
55.         elif isinstance(apdu, IAmRequest):
56.             device_type, device_instance = apdu.iAmDeviceIdentifier
57.             if device_type != 'device':
58.                 raise DecodingError("invalid object type")
59.
60.             if (self._request.deviceInstanceRangeLowLimit is not None) and \
61.                 (device_instance < self._request.deviceInstanceRangeLowLimit):
62.                 pass
63.             elif (self._request.deviceInstanceRangeHighLimit is not None) and \
64.                 (device_instance > self._request.deviceInstanceRangeHighLimit):
65.                 pass
66.             else:
67.                 # print out the contents
68.                 sys.stdout.write('pduSource = ' + repr(apdu.pduSource) + '\n')
69.                 sys.stdout.write('iAmDeviceIdentifier = ' + str(apdu.iAmDeviceIdentifier) + '\n')
70.                 sys.stdout.write('maxAPDULengthAccepted = ' + str(apdu.maxAPDULengthAccepted) + '\n')
71.                 sys.stdout.write('segmentationSupported = ' + str(apdu.segmentationSupported) + '\n')
72.                 sys.stdout.write('vendorID = ' + str(apdu.vendorID) + '\n')
73.                 sys.stdout.flush()
```

```
74.
75.         # forward it along
76.         BIPSimpleApplication.indication(self, apdu)
77.
78.     def confirmation(self, apdu):
79.         if _debug: BacnetApplication._debug("confirmation %r", apdu)
80.
81.         # forward it along
82.         BIPSimpleApplication.confirmation(self, apdu)
83.
84. class BacnetConsoleCmd(ConsoleCmd):
85.
86.     def do_whois(self, args):
87.         """whois [ <addr> ] [ <lolimit> <hilimit> ]"""
88.         args = args.split()
89.         if _debug: BacnetConsoleCmd._debug("do_whois %r", args)
90.
91.         try:
92.             # gather the parameters
93.             if (len(args) == 1) or (len(args) == 3):
94.                 addr = Address(args[0])
95.                 del args[0]
96.             else:
97.                 addr = GlobalBroadcast()
98.
99.             if len(args) == 2:
100.                lolimit = int(args[0])
101.                hilimit = int(args[1])
102.            else:
103.                lolimit = hilimit = None
104.
105.             # code lives in the device service
106.             this_application.who_is(lolimit, hilimit, addr)
107.
108.         except Exception as error:
109.             BacnetConsoleCmd._exception("exception: %r", error)
110.
111.     def do_any(self, args):
112.         """any
113.         Print all of the I-Am's received as if an unconstrained Who-Is was
114.         sent out, without actually sending an unconstrained Who-Is.
115.         """
116.         this_application._request = WhoIsRequest()
117.         this_application._request.deviceInstanceRangeLowLimit = 0
118.         this_application._request.deviceInstanceRangeHighLimit = 4194303
119.
120.     def do_iam(self, args):
121.         """iam"""
122.         args = args.split()
123.         if _debug: BacnetConsoleCmd._debug("do_iam %r", args)
124.
125.         # code lives in the device service
126.         this_application.i_am()
127.
128.     def do_read(self, args):
129.         """read <addr> <objid> <prop> [ <indx> ]"""
130.         args = args.split()
131.         if _debug: BacnetConsoleCmd._debug("do_read %r", args)
132.
133.         try:
134.             addr, obj_id, prop_id = args[:3]
135.             obj_id = ObjectIdentifier(obj_id).value
136.             if prop_id.isdigit():
137.                 prop_id = int(prop_id)
138.
139.             datatype = get_datatype(obj_id[0], prop_id)
140.             if not datatype:
141.                 raise ValueError("invalid property for object type")
142.
143.             # build a request
144.             request = ReadPropertyRequest(
145.                 objectIdentifier=obj_id,
146.                 propertyIdentifier=prop_id,
147.                 )
148.             request.pduDestination = Address(addr)
149.
150.
```

```
151.
152.             if len(args) == 4:
153.                 request.propertyArrayIndex = int(args[3])
154.             if _debug: BacnetConsoleCmd._debug("    - request: %r", request)
155.
156.             # make an IOCB
157.             iocb = IOCB(request)
158.             if _debug: BacnetConsoleCmd._debug("    - iocb: %r", iocb)
159.
160.             # give it to the application
161.             deferred(this_application.request_io, iocb)
162.
163.             # wait for it to complete
164.             iocb.wait()
165.
166.             # do something for success
167.             if iocb.ioResponse:
168.                 apdu = iocb.ioResponse
169.
170.                 # should be an ack
171.                 if not isinstance(apdu, ReadPropertyACK):
172.                     if _debug: BacnetConsoleCmd._debug("    - not an ack")
173.                     return
174.
175.                 # find the datatype
176.                 datatype = get_datatype(apdu.objectIdentifier[0], apdu.propertyIdentifier)
177.                 if _debug: BacnetConsoleCmd._debug("    - datatype: %r", datatype)
178.                 if not datatype:
179.                     raise TypeError("unknown datatype")
180.
181.                 # special case for array parts, others are managed by cast_out
182.                 if issubclass(datatype, Array) and (apdu.propertyArrayIndex is not None):
183.                     if apdu.propertyArrayIndex == 0:
184.                         value = apdu.propertyValue.cast_out(Unsigned)
185.                     else:
186.                         value = apdu.propertyValue.cast_out(datatype.subtype)
187.                 else:
188.                     value = apdu.propertyValue.cast_out(datatype)
189.                 if _debug: BacnetConsoleCmd._debug("    - value: %r", value)
190.
191.                 sys.stdout.write(str(value) + '\n')
192.                 if hasattr(value, 'debug_contents'):
193.                     value.debug_contents(file=sys.stdout)
194.                 sys.stdout.flush()
195.
196.             # do something for error/reject/abort
197.             if iocb.ioError:
198.                 sys.stdout.write(str(iocb.ioError) + '\n')
199.
200.         except Exception as error:
201.             BacnetConsoleCmd._exception("exception: %r", error)
202.
203.     def do_write(self, args):
204.         """write <addr> <objid> <prop> <value> [ <indx> ] [ <priority> ]"""
205.         args = args.split()
206.         BacnetConsoleCmd._debug("do_write %r", args)
207.
208.         try:
209.             addr, obj_id, prop_id = args[:3]
210.             obj_id = ObjectIdentifier(obj_id).value
211.             value = args[3]
212.
213.             indx = None
214.             if len(args) >= 5:
215.                 if args[4] != "-":
216.                     indx = int(args[4])
217.             if _debug: BacnetConsoleCmd._debug("    - indx: %r", indx)
218.
219.             priority = None
220.             if len(args) >= 6:
221.                 priority = int(args[5])
222.             if _debug: BacnetConsoleCmd._debug("    - priority: %r", priority)
223.
224.             # get the datatype
225.             datatype = get_datatype(obj_id[0], prop_id)
226.             if _debug: BacnetConsoleCmd._debug("    - datatype: %r", datatype)
227.
```

```
228.
229.             # change atomic values into something encodeable, null is a special case
230.             if (value == 'null'):
231.                 value = Null()
232.             elif issubclass(datatype, AnyAtomic):
233.                 dtype, dvalue = value.split(':', 1)
234.                 if _debug: BacnetConsoleCmd._debug("    - dtype, dvalue: %r, %r", dtype, dvalue)
235.
236.                 datatype = {
237.                     'b': Boolean,
238.                     'u': lambda x: Unsigned(int(x)),
239.                     'i': lambda x: Integer(int(x)),
240.                     'r': lambda x: Real(float(x)),
241.                     'd': lambda x: Double(float(x)),
242.                     'o': OctetString,
243.                     'c': CharacterString,
244.                     'bs': BitString,
245.                     'date': Date,
246.                     'time': Time,
247.                     'id': ObjectIdentifier,
248.                     }[dtype]
249.                 if _debug: BacnetConsoleCmd._debug("    - datatype: %r", datatype)
250.
251.                 value = datatype(dvalue)
252.                 if _debug: BacnetConsoleCmd._debug("    - value: %r", value)
253.
254.             elif issubclass(datatype, Atomic):
255.                 if datatype is Integer:
256.                     value = int(value)
257.                 elif datatype is Real:
258.                     value = float(value)
259.                 elif datatype is Unsigned:
260.                     value = int(value)
261.                 value = datatype(value)
262.             elif issubclass(datatype, Array) and (indx is not None):
263.                 if indx == 0:
264.                     value = Integer(value)
265.                 elif issubclass(datatype.subtype, Atomic):
266.                     value = datatype.subtype(value)
267.                 elif not isinstance(value, datatype.subtype):
268.                     raise TypeError("invalid result datatype, expecting %s" %
    (datatype.subtype.__name__,))
269.             elif not isinstance(value, datatype):
270.                 raise TypeError("invalid result datatype, expecting %s" % (datatype.__name__,))
271.             if _debug: BacnetConsoleCmd._debug("    - encodeable value: %r %s", value, type(value))
272.
273.             # build a request
274.             request = WritePropertyRequest(
275.                 objectIdentifier=obj_id,
276.                 propertyIdentifier=prop_id
277.                 )
278.             request.pduDestination = Address(addr)
279.
280.             # save the value
281.             request.propertyValue = Any()
282.             try:
283.                 request.propertyValue.cast_in(value)
284.             except Exception as error:
285.                 BacnetConsoleCmd._exception("WriteProperty cast error: %r", error)
286.
287.             # optional array index
288.             if indx is not None:
289.                 request.propertyArrayIndex = indx
290.
291.             # optional priority
292.             if priority is not None:
293.                 request.priority = priority
294.
295.             if _debug: BacnetConsoleCmd._debug("    - request: %r", request)
296.
297.             # make an IOCB
298.             iocb = IOCB(request)
299.             if _debug: BacnetConsoleCmd._debug("    - iocb: %r", iocb)
300.
301.             # give it to the application
302.             deferred(this_application.request_io, iocb)
303.
```

```
304.
305.             # wait for it to complete
306.             iocb.wait()
307.
308.             # do something for success
309.             if iocb.ioResponse:
310.                 # should be an ack
311.                 if not isinstance(iocb.ioResponse, SimpleAckPDU):
312.                     if _debug: BacnetConsoleCmd._debug("    - not an ack")
313.                     return
314.
315.                 sys.stdout.write("ack\n")
316.
317.             # do something for error/reject/abort
318.             if iocb.ioError:
319.                 sys.stdout.write(str(iocb.ioError) + '\n')
320.
321.         except Exception as error:
322.             BacnetConsoleCmd._exception("exception: %r", error)
323.
324.
325.     def do_rtn(self, args):
326.         """rtn <addr> <net> ... """
327.         args = args.split()
328.         if _debug: BacnetConsoleCmd._debug("do_rtn %r", args)
329.
330.         # provide the address and a list of network numbers
331.         router_address = Address(args[0])
332.         network_list = [int(arg) for arg in args[1:]]
333.
334.         # pass along to the service access point
335.         this_application.nsap.update_router_references(None, router_address, network_list)
336.
337. def main():
338.     global this_device
339.     global this_application
340.
341.     # parse the command line arguments
342.     args = ConfigArgumentParser(description=__doc__).parse_args()
343.
344.     if _debug: _log.debug("initialization")
345.     if _debug: _log.debug("    - args: %r", args)
346.
347.     # make a device object
348.     this_device = LocalDeviceObject(
349.         objectName=args.ini.objectname,
350.         objectIdentifier=int(args.ini.objectidentifier),
351.         maxApduLengthAccepted=int(args.ini.maxapdulengthaccepted),
352.         segmentationSupported=args.ini.segmentationsupported,
353.         vendorIdentifier=int(args.ini.vendoridentifier),
354.         )
355.
356.     # make a simple application
357.     this_application = BacnetApplication(this_device, args.ini.address)
358.
359.     # make a console
360.     this_console = BacnetConsoleCmd()
361.     if _debug: _log.debug("    - this_console: %r", this_console)
362.
363.     # enable sleeping will help with threads
364.     enable_sleeping()
365.
366.     _log.debug("running")
367.
368.     run()
369.
370.     _log.debug("fini")
371.
372. if __name__ == "__main__":
373.     main()
374.
```