DEPARTAMENTO DE SISTEMAS Y AUTOMÁTICA

Universidad
Carlos III de Madrid

UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Electrónica, Industrial y Automática

INFORMÁTICA INDUSTRIAL I

Trabajo final: Domótica

Alumnos:

100283996	Hurtado Fragoso, Daniel.
100282672	López Pina, Santiago.

Profesora:

Gonzalez-Quijano, Javier

1. INTRODUCCIÓN

En esta memoria vamos a explicar como hemos realizado mediante el lenguaje de programación de C++ el proyecto de una casa domótica. Para ello nos hemos ayudado del programa DIA para la realización del diagrama UML, del programa QT Creator para la realización del código del programa y también hemos usado un repositorio en Github para poder programar de manera más sencilla el código entre los dos miembros del equipo de forma paralela.

La casa domótica necesitará un sistema de comunicaciones que conectará con los distintos nodos, para así poder establecer que valores se leen de los sensores y cuales son los que deben cambiar y enviarle cuales serán los cambios.

Una vez establecido este sistema, procederemos a crear un PC Central que controlará toda la casa, este será el que tendrá los nodos y dentro de los nodos, los actuadores, los sensores y el controlador.

También hemos tenido en cuenta que tanto los sensores como los actuadores, dependiendo de la función que tengan, podrán ser discretos o analógicos, esto será añadido a ambas clases con el objetivo de que el programa pueda tener todas las opciones disponibles.

2. DIAGRAMA UML

Para la realización del diagrama de UML hemos utilizado la aplicación ya mencionada DIA, basándonos en la introducción nos hemos dispuesto a realizarlo.

Para comenzar, hemos valorado que el programa tiene que girar en torno a una clase de Comunicaciones, esta debe de tener las identificaciones ID pertinentes para después poder establecer los lazos con otra clase que estará dentro de PC Central.

La Clase Comunicaciones deberá tener un atributo al que nosotros hemos llamado Mensaje y será de tipo String, este mensaje será el que contenga la ID del sensor, la ID del actuador, la acción que debe realizar y el valor que tiene que tener.

Para ello hemos creado un método dentro de esta clase llamado createMessage (ID1, ID2, acción y valor) será de tipo string. Este método se encargará de crear el mensaje que después mandaremos a Nodo, que estará agregado a PC Central.

Dentro de esta clase haremos una agregación para que la comunicación se pueda hacer a través de ficheros, que será una nueva clase, o a través de Ethernet, esta última será un trabajo futuro, mientras que la clase de Ficheros tendrá un atributo de tipo string al que hemos denominado fichero. También contará con 2 métodos que completarán lo que se agrega de su clase Padre, sendMessage(mensaje: string), este método se encargará de mandar el atributo mensaje y receiveMessage(mensaje: String) que se encargará de leerlo.

Una vez realizada la columna vertebral del programa que es la Clase Comunicaciones, nos disponemos a realizar el UML de la parte Principal del programa, a esta le llamaremos PC Central que tendrá como atributo un vector nodo[] del tipo nodo que será una clase agregada a esta posteriormente.

PC Central tendrá un método que será básico para el programa, que será la de añadir un nuevo nodo, cada vez que queramos agregar algún elemento a nuestra casa, este método nos permitirá hacerlo. También tendrá como método el estado de los actuadores que será el que lee cuando el valor que le tiene que dar al actuador. Mientras que el estado valoresSensores será de tipo float y leerá el valor que tiene el sensor.

Agregado a PC Central nos encontraremos con la Clase Nodo, esta tendrá como atributo a comunicación que es del tipo comunicaciones, y nos servirá para enlazar el PC Central con Comunicaciones. También tendrá el atributo ID que le dará a los objetos de las clases herederas su identificación para cada sensor y para cada actuador.

Dentro de Nodo encontraremos los Métodos añadirsensor y añadiractuador, los dos harán la misma función pero cada uno refiriéndose a una clase distinta heredera de Nodo.

Esta clase será la clase Padre de las clases controlador, actuador, y sensor, de la que heredarán sus métodos y sus atributos. De las dos últimas también habrá una nueva herencia para distinguir si los actuadores son discretos o analógicos.

La clase controlador tendrá un atributo al que hemos denominado ref que será el setPoint del regulador que establezcamos de tipo float,.También tendrá como atributos las IDs de los sensores y de los actuadores. Estos tres atributos se unirán en un método al que hemos denominado controlActuador que se encargara de las funciones de un regulador de tipo P.

Dentro de la clase Actuador, hemos hecho herencia de esta clase, dando lugar a las clases Discreto y Analógico, la primera con dos métodos, que serán abrir y cerrar, mientras que el analógico sólo tendrá un método que devolverá los valores de la ID de tipo int y el porcentaje de abierto de tipo float.

La clase Sensor será la clase Padre de analógico, que tendrá como atributo: valor, que será del tipo float y un método que leerá el valor, getvalor(), el discreto tendrá un atributo valor de tipo bool y tendrá como método de nuevo getvalor().

Con esto se concluye la explicación del diagrama UML que hemos establecido para la realización de este proyecto, ahora debemos implementar el UML en el código ayudándonos del programa QT Creator.

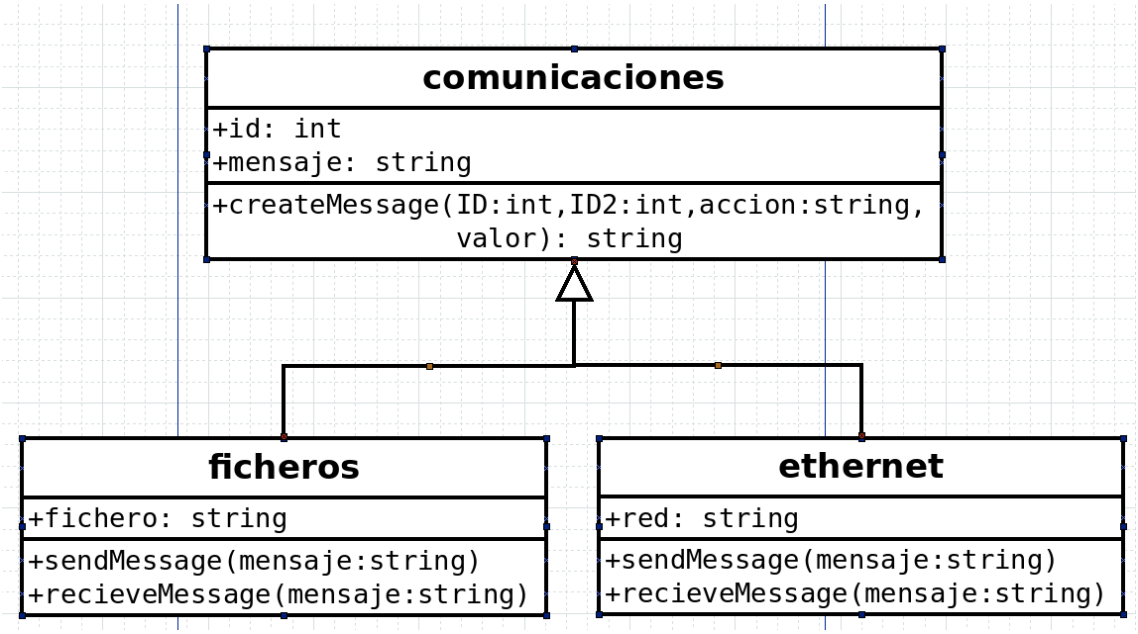

Figura 1. Diagrama UML clase comunicaciones

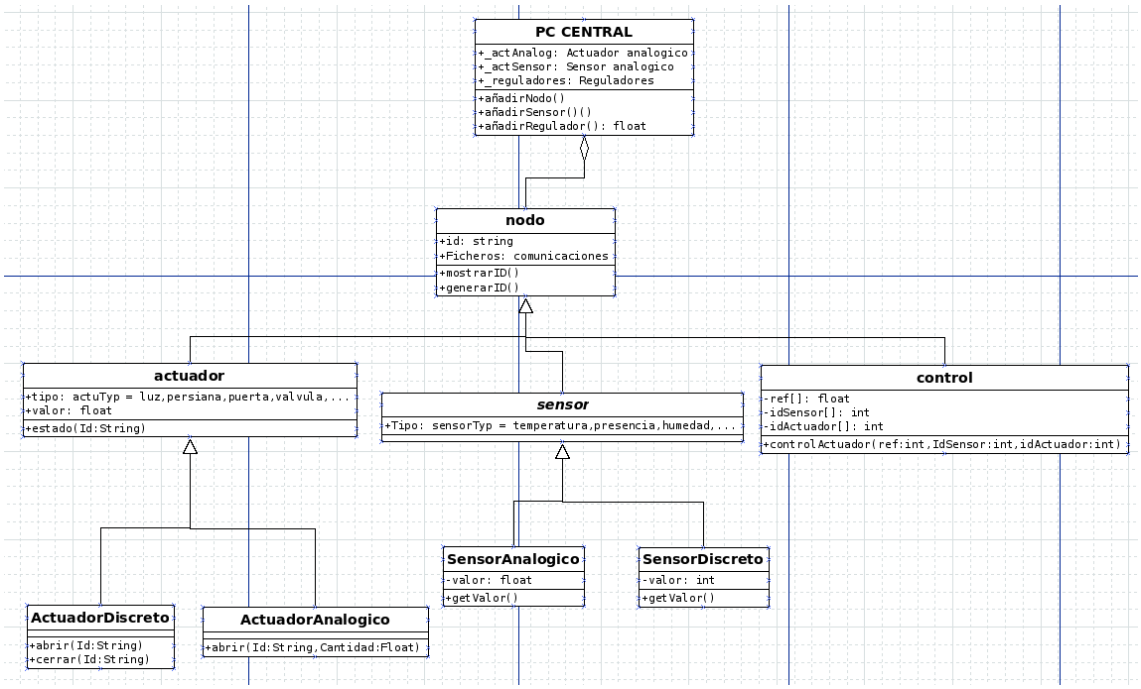

Figura 2. Diagrama UML clase NodoCentral

3. EXPLICACIÓN DEL PROGRAMA

La primera vez que se inicia el programa, comprueba si están creados los ficheros de configuración, si es la primera vez que se ejecuta, los generará. Si por el contrario no es la primera vez que se ejecuta, no será necesario que los vuelva a crear.

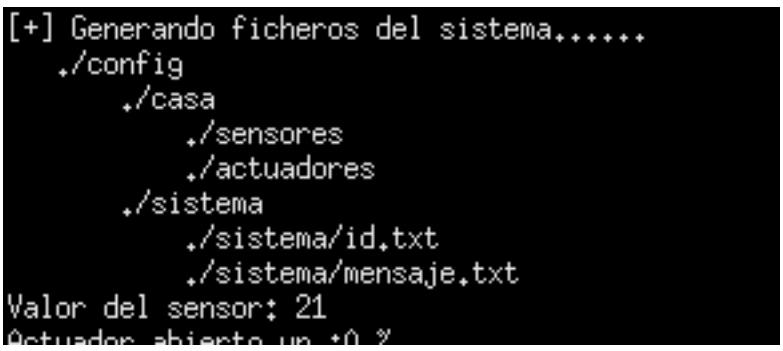

Figura 3. Ejemplo de primera ejecución

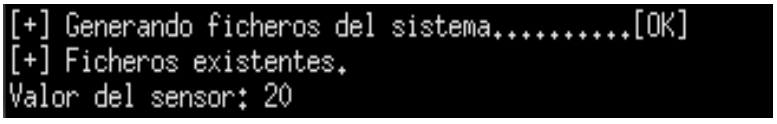

Figura 4. Ejemplo de segunda ejecución

La versión que se adjunta con el programa es capaz de crear un objeto NodoCentral y añadir un sensor y actuador analógico, a su vez se configura un regulador de tipo P para que mantenga la temperatura de la habitación constante.

Al iniciar el programa se le pedirá al usuario que ingrese por teclado la temperatura que quiere mantener en la habitación.

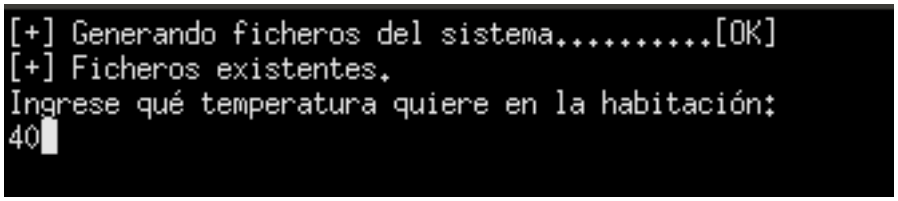

Figura 5. Ajuste del setpoint de temperatura.

De esta manera, el actuador se puede asimilar como una válvula que dejará pasar más o menos aire caliente. Para simular el efecto del aire caliente en la habitación se ha añadido un simulador de eventos en el que va variando la temperatura del sensor de forma aleatoria dentro de un rango de ± 5ºC la temperatura de setpoint.

Para poder ver una muestra del funcionamiento, se ha decidido hacer veinte iteraciones del proceso, número suficiente para ver que el sistema es estable.

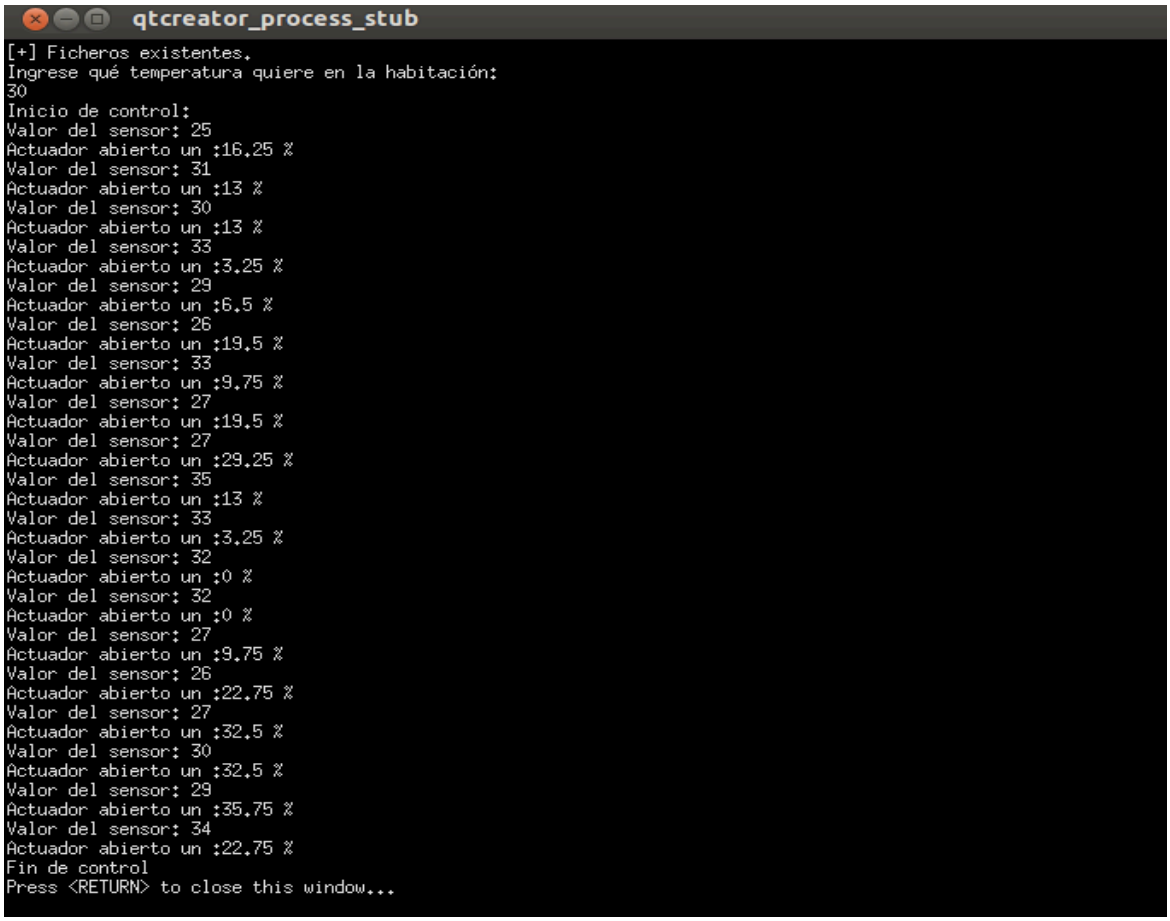

Figura 6 Ejemplo de veinte iteraciones del regulador

Como se ve en la figura 6, la temperatura de inicio es 25ºC como está por debajo del setpoint (30ºC) la válvula se abre un 16.25% para que deje entrar calor, en la siguiente iteración la temperatura es de 31 como supera el setpoint, la válvula se abre un porcentaje menor (13%) para que no entre tanto aire caliente.

Otra parte del programa que no está implementada, debido a falta de tiempo, es la parte de comunicaciones, que genera ficheros de texto para que los nodos se puedan comunicar entre sí.

El protocolo del mensaje es el que está explicado en la documentación. El primer término es la ID de origen del mensaje; el segundo la ID a la que va dirigido; a continuación la acción que tiene que ejecutar y por último el valor de la acción que tiene que tomar.

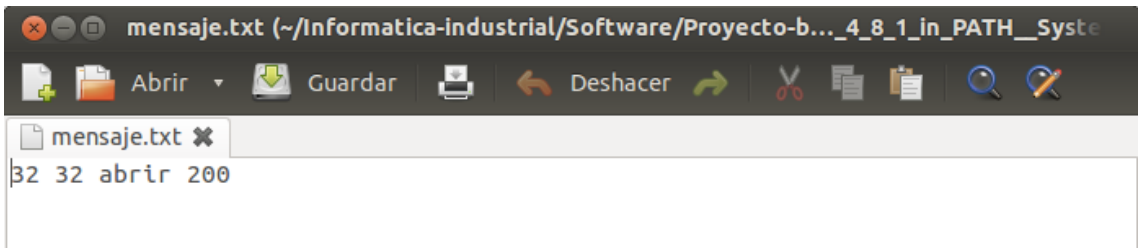

Figura 7. Mensaje de comunicaciones

El programa , está escrito usando el lenguaje C++ y usando únicamente librerías estándar incluidas dentro del entorno de desarrollo QT. Gracias al buen diseño del UML, el paso a lenguaje de programación ha sido más sencillo.

A la hora de implementar código y tener dudas, hemos acudido a nuestro tutor asignado en el proyecto y la mayoría de las veces, hemos buscado la solución en internet. Siendo las páginas más usadas las siguientes:

- http://c.conclase.net
- http://www.cplusplus.com

Algunas de las funciones a destacar es la inclusión de un regulador de tipo P y el simulador de eventos que pasamos a detallar:

```
void Regulador::Actualizar()
{
    /** Obtenemos el valor del sensor */
    double valorSensor = _vSensores_ptr->at(_index).getValor();
    /** Calculamos el error del sistema */
    double error = _setpoint - valorSensor;

    double controlAction;
    /** el nuevo valor del sensor es la suma del error por el valor proporiconal,
      si el error es negativoes por que hemos superado el setpoin. */
    controlAction = _vActuadores_ptr->at(_index).getValor() + (error*_p);

    //set actuator value
    if (controlAction < 0)  controlAction = 0;/** Si el error es negativo */
    _vActuadores_ptr->at(_index).setValor(controlAction);
    cout << "Actuador abierto un :"  << _vActuadores_ptr->at(_index).getValor()<< " %" <<endl;

}
```

Figura 8. Ejemplo de código del regulador P

```
void Simulador::cambiarValorSensor(int index, int min, int max)
{
    /** Método cambiarvalorSensor() Modifica el valor del sensor que le indiquemos con un
        numero aleatorio dentro del intervalo introducido */
    _valorSim=min+rand()%((max+1)-min);  /** Generamos el numero aleatorio dentro del rango
    _ssc_ptr->_sensAnalog.at(index).setValor(_valorSim); /** Cambiamos el valor del sensor
    cout << "Valor del sensor: " << _ssc_ptr->_sensAnalog.at(index).getValor() << endl; /**
}
```

Figura 9. Ejemplo de simulador de eventos.

Para la realización de este código nos hemos basado en el proporcionado por nuestro tutor del proyecto Javier González-Quijano.
Como vemos en la figura 6 se implementa un regulador P clásico. Este regulador, coge el valor del sensor y calcula el desvío respecto al setpoint. El error lo multiplicamos por la constante de proporcionalidad (P) establecida a la hora de instanciar el objeto. Este valor tomará el valor negativo en caso de que hayamos superado el setpoint y positivo si todavía no lo hemos alcanzado, en el caso ideal de que estemos en el setpoint, tomará el valor de 0. De esta manera si el valor obtenido se lo sumamos al valor de apertura del actuador, automáticamente el actuador trabajará de forma automática alrededor del setpoint.

En la figura 9 vemos el ejemplo de simulación. En este caso al método cambiarValorSensor() se le pasan tres parámetros, índice de sensor y actuador que queremos controlar, valor mínimo y máximo sobre el que queremos generar un número aleatorio para que cambie la temperatura. De esta manera, el proceso de demostración se ejecuta de manera más automática.

Para ambos integrantes del grupo es el primer programa que hemos realizado con la metodología de objetos y los inconvenientes que hemos tenido a la hora de programar han sido debido al aprendizaje del lenguaje.

4. TRABAJOS FUTUROS

- **Añadir un menú de opciones**

La implementación final de la demo es un simple regulador, se podría añadir un menú inicial, en el que el sistema ejecutase un asistente en el que se fueran añadiendo los sensores, actuadores y reguladores e ir entrando en los distintos métodos. Debido a falta de tiempo se ha decidido hacer de la manera que se ha explicado en la presente memoria.

- **Implementar la clase comunicaciones con los reguladores**

Una de las especificaciones del proyecto era la comunicación por ficheros, está programada a falta de la implementación para que funcione de manera automática.

- **Añadir tipo de los sensores y actuadores (método enumerado)**

Se podría añadir un atributo a los sensores y actuadores para reconocer el tipo que es, y de esa manera mostrar mensajes por pantalla de una manera un poco más específica.

- **Implementar usuarios.**

Añadir usuarios al sistema permitiría unas opciones avanzadas de control, de esa manera se podría configurar acciones acordes con los niveles de usuario.

- **Entorno gráfico**

De manera final, se podría haber añadido un entorno gráfico en el que se mostrase la casa y un acceso inmediato a todos los sensores y actuadores de la casa, para poder modificarlos de una manera más rápida y así simular un sistema SCADA.

- **Simulación con Raspbery Pi**

Con ayuda de una Raspbery Pi, se podría haber hecho una pequeña maqueta con un led y un sensor de temperatura, en el que el programa leyera de forma automática el valor real del sensor. Si hubiéramos tenido dedicación exclusiva al proyecto se hubiera podido realizar.

5. CONCLUSIONES

Dentro de los propuestos por el departamento de la asignatura, consideramos que este trabajo es el más ambicioso y el que más tiempo necesita. Como indicamos en el apartado de Trabajos futuros, nuestra idea era presentar un proyecto más completo que el actual, pero debido a falta de tiempo y sobre todo carga lectiva de la propia asignatura y debido a otras, no hemos podido llevarlo acabo.

Este trabajo, requiere niveles de conocimiento del lenguaje C++ que a final de curso hemos aprendido, pero a principio de curso cuando se nos asignó el proyecto no disponíamos, debido a ello también hemos notado que cuando teníamos los conocimientos necesarios para realizar el trabajo llevábamos un cierto retraso incapaz de subsanar.

Las sensaciones finales del proyecto no son satisfactorias por todo lo expuesto anteriormente, es un proyecto muy completo que si se le dedicará el tiempo que merece, se podrían haber obtenido unos resultados bastante ingeniosos ya que es un proyecto que está relacionado directamente con nuestro campo de electrónica y automatización.