UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Implementación de un GateWay para conectar sensores con transceiver de LoRa a servicios públicos y privados

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Senén Palanca Barrio

Tutor: Sara Blanc Clavero

Curso 2018/2019

Resumen

Bajo el concepto de IoT o Internet de las Cosas, podemos "observar" a través de sensores y procesar los datos en servicios deslocalizados. Un Sistema IoT enfocado a la agricultura podría proporcionar grandes avances en ese sector. Debido a los avances de tecnologías emergentes, como por ejemplo RF LoRa™, se pueden transmitir mensajes a mayor distancia y utilizando una potencia insignificante. La finalidad de este proyecto es desarrollar una pasarela de interconexión entre sensores con tecnología LoRa y diversos servicios fuera del campo, tanto públicos como privados, que harán viable una primera versión de lo que podría ser un sistema de Smart Farming. La pasarela se implementa en un sistema empotrado, una Raspberry Pi. A diferencia de otros proyectos, la solución propuesta implementa servicios bajo el paradigma de la Coreografía de servicios (arquitecturas SOA), algunos de los cuales ya estaban desarrollados, pero que ha sido necesario modificar y ampliar para completar nuestros nuevos objetivos de interactuación.

Palabras clave: LoRa™, IoT, Internet of Things, Smart Farming, Service Choreography

Resum

Amb el concepte de IoT o Internet de les Coses, podem “observar” a través de sensors i processar les dades en servicis deslocalitzats. Un Sistema IoT enfocat a l'agricultura podria proporcionar grans avanços en eixe sector. A causa dels avanços de tecnologies emergents, com per exemple RF LoRa™, es poden transmetre missatges a major distància i utilitzant una potència insignificant. La finalitat d'este projecte és desenrotllar una passarel·la d'interconnexió entre sensors amb tecnologia LoRa i diversos servicis fora del camp, tant públics com privats, que faran viable una primera versió del que podria ser un sistema de Smart Farming. La passarel·la s'implementa en un sistema empotrat, una Raspberry Pi. A diferència d'altres projectes, la solució proposada implementa servicis dins el paradigma de la Coreografia de servicis (arquitectures SOA), alguns dels quals ja estaven desenrotllats, però que ha sigut necessari modificar i àmpliar.

Palabras clave: LoRa™, IoT, Internet of Things, Smart Farming, Service Choreography

Abstract

Under the concept of IoT or Internet of Things, we can "observe" through sensors and process data in relocated services. An IoT System focused on agriculture could provide great progress in that sector. Due to advances in emerging technologies, such as RF LoRa™, messages are now transmitted throughout a greater distance and using negligible power. The main objective of this project is to develop a sensor interconnection gateway for LoRa technology sensors and with various out-of-the-field services, both public and private, that will make a first version of what could be a Smart Farming system viable. The gateway is implemented in a built-in system, a Raspberry Pi. Unlike other projects, the proposed solution implements services under the paradigm of the Choreography of services (SOA architectures), some of which were already developed, but it has been necessary to modify and expand to complete our new interaction objectives.

Palabras clave: LoRa™, IoT, Internet of Things, Smart Farming, Service Choreography

Tabla de contenidos

1. Introducción

Este proyecto implementa una arquitectura dedicada a servicios (en inglés SOA), basada en un coordinador encargado de administrar varios servicios e intercambiar mensajes entre estos, que se ejecutan de forma independiente.

La arquitectura SOA describe dos tipos de servicios, productor y consumidor. En este proyecto los servicios actúan adoptando los dos roles al mismo tiempo. Estos servicios, capaces de comunicarse entre sí a través de mensajes manejados por el coordinador, representan cada una de las funciones independientes de la solución presentada y permiten una gran flexibilidad cuando es necesario realizar modificaciones.

Esta arquitectura se ha implementado mediante un coreógrafo de servicios web (en inglés Coreographer). Este coreógrafo, diseñado en el instituto ITACA [1] de la UPV [2] por el equipo SABIEN [3], está orientado en su diseño para grandes espacios computacionales. Sin embargo, en nuestro proyecto se desarrolla una solución adaptada a sistemas empotrados, en concreto, a una Raspberry Pi [4]. Para esta solución se utilizan las librerías y recursos de coreografía para trabajar en un caso de estudio sobre Agricultura de Precisión.

Los servicios de nuestra solución IoT para Agricultura de Precisión van a implementar diversas funcionalidades que permitirán el flujo de servicios en el Coreographer tales como servicio de Base de datos, servicio de extracción en CSV [5], Clientes REST [6], Conectores TCP [7], intérpretes de mensajes multisensor, etc. En este proyecto, centramos el foco en el “Edge Computing”, donde nuestro sistema de coreografía empotrado actúa como intermediario entre los sensores instalados en el campo y los servicios en nube. Nuestra solución recibe datos de sensores con distintas tecnologías, tales como Wifi y LoRa [8], para retransmitir los datos pre-procesados a diferentes servicios externos en función de su tipo o de su tipología.

Motivación

Siempre me ha atraído el entorno de desarrollo Windows, y cuando vi que me permitían desarrollar sobre una Raspberry Pi3 una solución en Windows IoT[9], me gustó la idea porque son tecnologías emergentes. Siempre me ha gustado la programación en Windows, y vi una oportunidad de adquirir nuevos conocimientos y de aprender algo más de Hardware, necesario para implementar los distintos demostradores que he utilizado con RF LoRa™. A parte, nunca había trabajado con nada relacionado con IoT, y aunque no siempre me he visto con ganas, no sabía lo divertido que es hasta hoy.

Objetivos

La idea principal del proyecto es desarrollar un módulo que interconecte los sensores del campo con servicios en nube o deslocalizados (fuera del campo).

Actualmente, no existe una tecnología única en la implementación de la comunicación entre sensores y pasarelas. Entre las más frecuentes encontramos ZigBee [10], RF LoRa™, o la más frecuente, WiFi.

Esta diversidad nos obliga a adaptar nuestra solución a diferentes tecnologías. El objetivo del proyecto será desarrollar los conectores TCP necesarios para la recepción de mensajes de distintos componentes así como los intérpretes de mensajes teniendo en cuenta que nuestro módulo no sólo recibe mensajes de sensores sino también de clientes fuera del campo, con una aproximación al Fog Computing[11].

El proyecto se centra en desarrollar la compatibilidad del módulo con sensores RF LoRa™, así como hacer desarrollar un algoritmo de comunicación TCP con el cliente deslocalizado (donde residirá la interfaz gráfica o visualización de datos). Para desarrollar estos servicios (conectores, intérpretes e interactuación), se ha aplicado el concepto de Coreografía de servicios.

A continuación, se desglosan los anteriores objetivos en unos más específicos:

- Compatibilizar el módulo de coreografía con sensores RF LoRa™ y comunicación bidireccional
- Compatibilizar el módulo de coreografía con sensores Wifi[1]
- Recibir e interpretar mensajes de sensores de distinta tipología (temperatura, humedad)
- Almacenamiento interno clasificado de datos en base de datos
- Desarrollar diferentes servicios capaces de procesar los datos recibidos de los distintos sensores (se utilizarán de muestra servicios públicos como ThingSpeak [12])
- Desarrollar a un servicio que permita la conexión HTTP [13] con un cliente ad-hoc para la descarga selectiva de los datos almacenados en la base de datos del módulo

Impacto esperado

Observar y registrar datos de distintas variables en el campo y procesarlas fuera de el mismo tiene un gran potencial en agricultura de precisión, especialmente con el objetivo de mejorar la eficiencia del riego y tomar decisiones tanto inmediatas como para los siguientes ciclos de cultivo.

[1] Este proyecto es continuación de otro anterior de la ETSINF, referenciado en [23], que parte de la adaptación de sensores con enlace WiFi.

En el caso de agricultura, la variable o variables que puede obtener un solo sensor no son significativas. Es la observación de un grupo de sensores y el procesado de todos los datos los que nos proporciona información útil para la toma de decisiones. Por este motivo, el uso de pasarelas que pre-procesan, almacenan y retransmiten los datos de forma selectiva, están justificadas.

Una pasarela tiene dos extremos. Por un lado, interactúa con los sensores del campo. Por otro lado, debe retransmitir los datos pre-procesados o en "crudo" a servicios ad-hoc o públicos para su tratamiento y uso en la toma de decisiones.

La posibilidad de recibir mensajes de los sensores a través de RF LoRa no sólo nos permite posicionar sensores a muchísima mayor distancia que hasta ahora utilizando Wifi, sino que además utiliza una cantidad de energía insignificante, con lo que podríamos tener sensores operativos por más tiempo y en lugares mucho más inaccesibles. En este proyecto se ha trabajado este punto.

Por otro lado, la comunicación en tiempo real para la retransmisión de los datos permitirá visualizarlos e interpretarlos en gráficas, envío de alertas, etc. Además, el hecho de que se puedan enviar datos a los distintos sensores desde el módulo añade una nueva característica al proyecto y permite tener sensores y actuadores que reciban dichos datos (comunicación bidireccional en el campo). Podemos establecer nuevas configuraciones para los sensores, o activar o desactivarlos por comandos desde el Cliente.

Por tanto, tenemos una solución flexible y compatible con RF LoRa, además de poseer la habilidad de comunicarse con un cliente, y muy abierta a cambios, por lo que se adapta perfectamente a un entorno IoT, en concreto a un proyecto de agricultura donde se recopilan, interpretan y visualizan los datos de forma selectiva.

Nuestra solución será compatible con servicios públicos IoT LoRaWan[14], como por ejemplo el conocido TTN [15], aunque también será posible conectar nuestro módulo para la descarga y visualización de datos con servicios ad-hoc deslocalizados, específicos para actuar sobre los cultivos.

Metodología

La metodología seguida en este proyecto ha sido bastante ágil y abierta a cambios, aunque se han determinado cuales son las metas a las que es necesario llegar.

Primero he realizado varias reuniones con la tutora, que me ha facilitado el contenido de la solución anterior, así como varias estructuras del Coreographer. Mi tarea al principio ha sido comprender cómo actúan los distintos servicios preexistentes de la aplicación, así como también entender la herramienta principal, el Coreographer, especialmente en el uso de los métodos y mensajes de coreografía.

Se han realizado también tutorías puntuales en el grupo SABIEN de ITACA-UPV para aprender a desarrollar servicios en el entorno de coreografía.

Posteriormente, y junto a otro compañero que realiza la parte gráfica, se han producido más reuniones donde se concretan los objetivos del proyecto. Después empiezo con el desarrollo, lo más cercano posible a los objetivos establecidos, añadiendo distintos servicios sin solaparse con nada de lo anterior. Estos servicios han variado mucho desde su primera forma para adaptarse mejor a los requisitos.

Se realizan sesiones semanalmente con la tutora, además de una completa disponibilidad por correo, para aclarar dudas y evaluar el trabajo realizado.

Finalmente se realizan un par de sesiones para acabar el proyecto, además de facilitarme unas guías de como redactar una memoria.

En el gráfico 1 se muestran las fases del proyecto con los paquetes de trabajo más generales y el cronograma empleado.

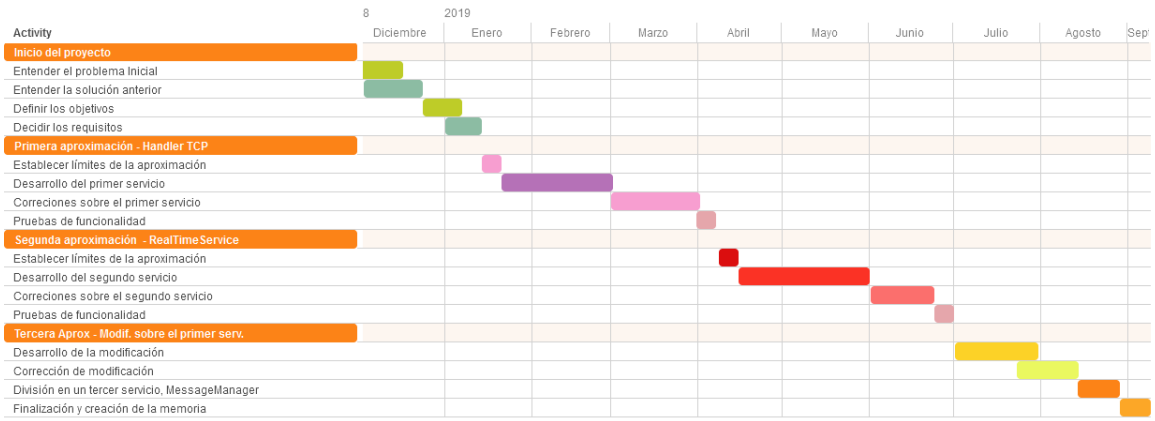

Gráfico 1 – Diagrama de Gantt del proyecto

Estructura

La memoria de la Solución propuesta se estructura de la siguiente forma:

En el punto 2, se introduce el estado del arte, la utilidad del proyecto y sus novedades.

En los puntos 3, 4 y 5, se realizará un análisis de funcionalidades que se han implementado, para entender de forma global qué hace el proyecto y cuáles son sus vías de comunicación. Luego poco a poco se irá profundizando en las secciones de diseño y arquitectura, donde los contenidos pasan a ser más específicos y divididos por Servicios.

En los puntos 6 y 7, se explican los componentes comerciales sobre los que se ha desplegado la solución y las pruebas de funcionamiento realizadas.

Finalmente, se proponen trabajos futuros y una conclusión generalizada.

Colaboración

Me he valido de la ayuda de mi compañero Juan Fran Tomás, quien me ha ayudado a interpretar la petición HTTP, además de elaborar una respuesta entendible por el navegador. Juan Fran se encuentra realizando un proyecto TFG para la ETSINF [16] de visualización de los datos que recibirá del módulo desarrollado en este proyecto. Por tanto, hemos tenido que acordar un algoritmo de intercambio de datos y una sintaxis apropiada para los mensajes.

2. Estado del Arte

Podemos pensar en diversos usos para una solución genérica en sistemas de agricultura eficiente o de precisión (o Smart Agriculture). El proyecto que presento permite que se puedan supervisar distintas variables atmosféricas y de entorno para tomar mejores decisiones a diferentes escalas, tanto a corto como a medio plazo. Por ejemplo, un módulo inteligente, como el que pretende este proyecto, facilita la de creación tareas automatizadas a partir de los datos recibidos y la actuación inmediata en el mismo campo. Pero también permite la gestión remota, transportando los datos desde el campo hacia servicios de distinta índole, ya sean para visualizar los datos, para enviar alertas a dispositivos móviles, para actuar remotamente sobre los actuadores o ya sea para almacenarlos permitiendo su análisis posterior.

La idea no es nueva pero sí de actualidad ya que aún no existe una solución única. Por ejemplo, en la web de "Internet of Food and Farming 2020" (www.iof2020.eu) se explora el potencial de las soluciones IoT para sistemas de agricultura y ganadería. Este es un claro ejemplo del alcance que tiene la toma de decisiones a gran escala. Esta página muestra varios proyectos enfocados a las distintas áreas de la producción ganadera y en agricultura. En concreto, el proyecto "Digital Ecosystem Utilisation" [18] ofrece mejores resultados en comparación a métodos tradicionales de cultivo. Nos informa de que implementando este proyecto se ahorra un 10% de agua, la eficiencia aumenta hasta un 20%, y se utilizará un 10% menos de recursos en pesticidas y control de plagas.

Para desarrollar este tipo de proyectos existen tecnologías probadas y distintos servicios IoT en nube. Sin embargo, actualmente lo más complicado de encontrar en el mercado son módulos que interconecten el campo con dichos servicios agrupando datos, filtrándolos, e incluso realizando acciones automatizadas que revierten de manera inmediata en actuadores.

Entre los paradigmas que potencialmente se pueden adaptar mejor a una arquitectura de enlace que proporcione a su vez servicios específicos, destacamos el concepto de coreografía de servicios.

Por definición, la coreografía de servicios es una especificación establecida por el W3C [19]

en el año 2005, que define procesos en un modelo de negocio basado en XML. Los distintos procesos que lo componen adquieren el carácter de participantes, sin haber ningún punto único de control. La coreografía permite que una solución se adapte a las necesidades cada vez más versátiles. Debido a que los procesos funcionan de forma independiente, y sólo necesitan de mensajes para funcionar, es muy fácil exportar cualquiera de los procesos a otro proyecto, estableciendo de forma clara cuáles serán las interacciones necesarias con otros procesos.

Sin embargo, la adaptación de la coreografía a módulos de bajo consumo y pequeño tamaño para agricultura de precisión no es un área muy explotada, y menos comercialmente. Su potencial es importante y por eso me he decidido a incorporarlo a este proyecto como un valor añadido distinto a la típica pasarela de datos (LoRa Gateway).

Crítica a estado del arte y propuesta

Existen diversos proyectos existentes que han ayudado mucho al desarrollo de las funcionalidades objetivo de este proyecto. Por ejemplo, para recibir mensajes RF LoRa™, nos hemos valido de un componente comercial, LoPy de la marca PyCom, que posee antenas LoRa y Wifi internas. Utilizando el código del proyecto "LoRa MAC (RAW LoRa)" [17] y con modificaciones simples, recibimos mensajes de un emisor LoRa de forma rápida. Para el servidor TCP, he utilizado proyectos como "StreamSocketListener for Windows IOT" en la propia documentación de Microsoft.

Sin embargo, estos componentes comerciales, aunque facilitan el inicio y arranque del proyecto, luego han de adaptarse, programarse y, a veces, habrá que seleccionar los que mejor compatibilidad muestren con los distintos componentes que empleemos en la cadena, desde el sensor hasta el servicio.

No existe una solución comercial plug-and-play que incorpore la coreografía de servicios a nivel del propio campo y extensible a niveles superiores de la arquitectura sobre el concepto de "Fog Computing".

Aunque podemos relacionar cada uno de los servicios de la solución con un código secuencial, la coreografía nos va a permitir desarrollar una estructura portable y que puede crecer con nuevos servicios manteniendo los actuales sin necesidad de modificarlos. Este concepto es realmente interesante en los primeros estadios de la agricultura de precisión ya que son muchas las variables a controlar que se pueden ir incorporando a lo largo del tiempo tanto para actuar con los sistemas actuales como en previsión de los nuevos sistemas que puedan ir apareciendo a medida que crecen las necesidades.

Es importante destacar que, entre las ventajas que ofrece la coreografía, ante el fallo de cualquier servicio implementado en el módulo, todos los demás servicios continúan con su ejecución normal, lo que proporciona robustez al sistema.

En resumen, tendremos un sistema muy flexible y abierto a cambios, además de su modularidad, que permite que sea escalable además de permitir agregar nuevas funcionalidades sin problema.

Este proyecto es continuación de uno anterior (Integración IoT en servicios Web, por Josep Benedito Fuster, ETSINF). En el proyecto mencionado, se asentaron las bases de una estructura mínima de recepción TCP, y reenvío de datos a un servicio con un conector HTTP. Sin embargo, no contemplaba la recepción de datos por RF LoRa™. Lo que se ha realizado en el proyecto que presento es compatibilidad con cualquier tipo de socket, independientemente de la tecnología de comunicaciones del emisor de los datos. Además, se ha añadido un servicio que atiende a un algoritmo de intercambio de datos con un cliente en Windows. El objetivo de este cliente es que pueda conectar con un servicio web en el futuro para la visualización de los datos en gráficas interpretadas en tiempo real.

3. Análisis de la solución

La solución se resume a un sistema distribuido modulable, adaptado para IoT. Como se observa en la figura 1, el núcleo del proyecto se ubica en la Raspberry Pi, que se encargará de recibir, gestionar y almacenar sensores y sus datos, además de permitir la conexión con clientes a través de HTTP para la retransmisión de los datos recibidos de los sensores.

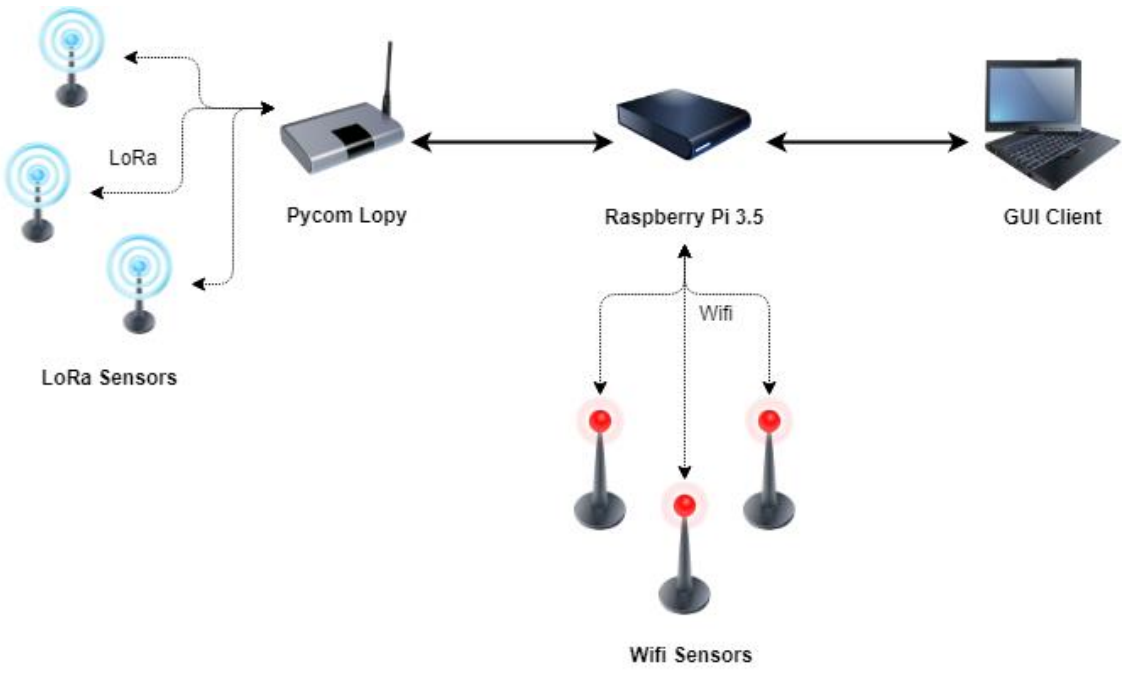

Figura 1 - Visión general de los componentes interconectados

En la figura 1 se muestra la conexión inalámbrica de 3 tipos de emisores sobre la Raspberry Pi donde se implementa la solución del proyecto. A la izquierda de la figura superior existe el módulo concentrador LoPy [19] (o similar) quien recibe datos de los sensores LoRa y los reenvía hacia el núcleo de la aplicación. La parte de debajo de la figura observamos sensores que envían directamente a través de wifi al núcleo mientras que en la parte derecha de la figura vemos la interconexión con un cliente que pretende recibir los datos almacenados en la base de datos del núcleo de forma selectiva para su potencial visualización.

Los requisitos desarrollados en el proyecto han sido:

- Crear conector, socket TCP e intérprete de mensajes (heterogéneos porque viene distintos emisores)
- Crear algoritmo para la extracción selectiva de datos de la base de datos
- Habilitar función para borrar toda la base de datos
- Crear servicio de datos en Tiempo Real capaz de enviar datos recibidos de los sensores. Este servicio tiene como peculiaridad su capacidad para enviar de forma continua o discontinua. Teniendo en cuenta las

posibilidades de pérdida de enlace (en entornos agrarios), los datos no enviados durante la desconexión podrán ser enviados en paquete tras la reconexión del enlace
- Habilitar la des/conexión del Servicio de Tiempo Real desde la conexión TCP
- Habilitar un sistema que permita enviar órdenes de actuación a los sensores (comunicación bidireccional)
- Interpretar el mensaje recibido por la conexión TCP, y devolver un mensaje en función de la petición
- Separar el Socket TCP de la gestión de mensajes en dos servicios diferentes que se comuniquen entre sí
- Habilitar la posibilidad de recibir peticiones HTTP desde el Servidor TCP y devolver un mensaje entendible
- Testear todas las configuraciones posibles que puede recibir el Servidor TCP

Análisis de Seguridad

La aplicación implementa un mínimo de seguridad, ya que para conectarse a ella es necesario un token que esté aceptado por el sistema a modo de contraseña. Esto se ha hecho no para evitar que los datos de la aplicación puedan ser leídos, si no para evitar que se produzcan acciones de control sobre los sensores sin tener una acreditación ya que puede perfectamente causar muchísimos problemas en un sistema de agricultura.

Presupuesto

Para calcular el presupuesto de este proyecto he utilizado los precios que ofrece Amazon. En segundo lugar, para calcular el coste de una hora, considero el mismo precio que el contrato que se me ha ofrecido para continuar este proyecto, que es 8 euros a la hora. Debido a que la realización del proyecto ha supuesto 350 horas, el coste total del proyecto se puede obtener de la siguiente tabla:

Concepto	Precio
Raspberry Pi 3 B+	37,00 €
PyCom LoPy	35,00 €
Desarrollo del producto	8.00€/h * 350 = 2800,00 €
Total	2.872 €

4. Diseño de la solución

Arquitectura del Sistema

Esta solución está compuesta por 3 proyectos distintos.

El primero es el de gestión de datos. Es la capa DAL [20], y se encarga de almacenar los datos en una Base de Datos [21] de tipo SQL [22]. Los datos se organizan en dos tablas: Una contiene almacenados todos los valores recopilados de sensores, la tabla DATA. Otra lleva un registro de los sensores existentes en la solución.

El segundo proyecto gestiona el inicio del Coreographer. Es el proyecto inicial, el primero que se ejecuta cuando lanzamos el proyecto. Es aquí donde se añaden los servicios al Coreographer para iniciarse después, también desde este origen.

Por último, el proyecto que contiene toda la lógica de negocio engloba a los Servicios. Se observan de forma clara todos los que se han implementado en la solución en la figura 2.

Debido a que esta solución parte de una existente [23], existen partes que no se modifican, o que directamente pasan a estar desfasadas (como el SocketCSV). Las modificaciones realizadas sobre el sistema preexistente se observan con más detalle en el esquema que se muestra a continuación.

Existen tres posibles vías de comunicación externa con la aplicación, dos de las cuales son bidireccionales. A continuación, se explican de forma detallada las tres posibles vía que están representadas en la figura 3.

La primera es la vía de los sensores Wifi, que envían datos a través de Wifi en un formato de conexión TCP (representado en la figura 4 a la izquierda), para que posteriormente esos datos sean interpretados por un servicio, el TCPListener. Este servicio es el encargado de enviar un mensaje de coreografía dirigido a la base de datos y que contiene el dato recibido del exterior para su posterior almacenaje (representado por la segunda flecha de la figura, de izquierda a derecha).

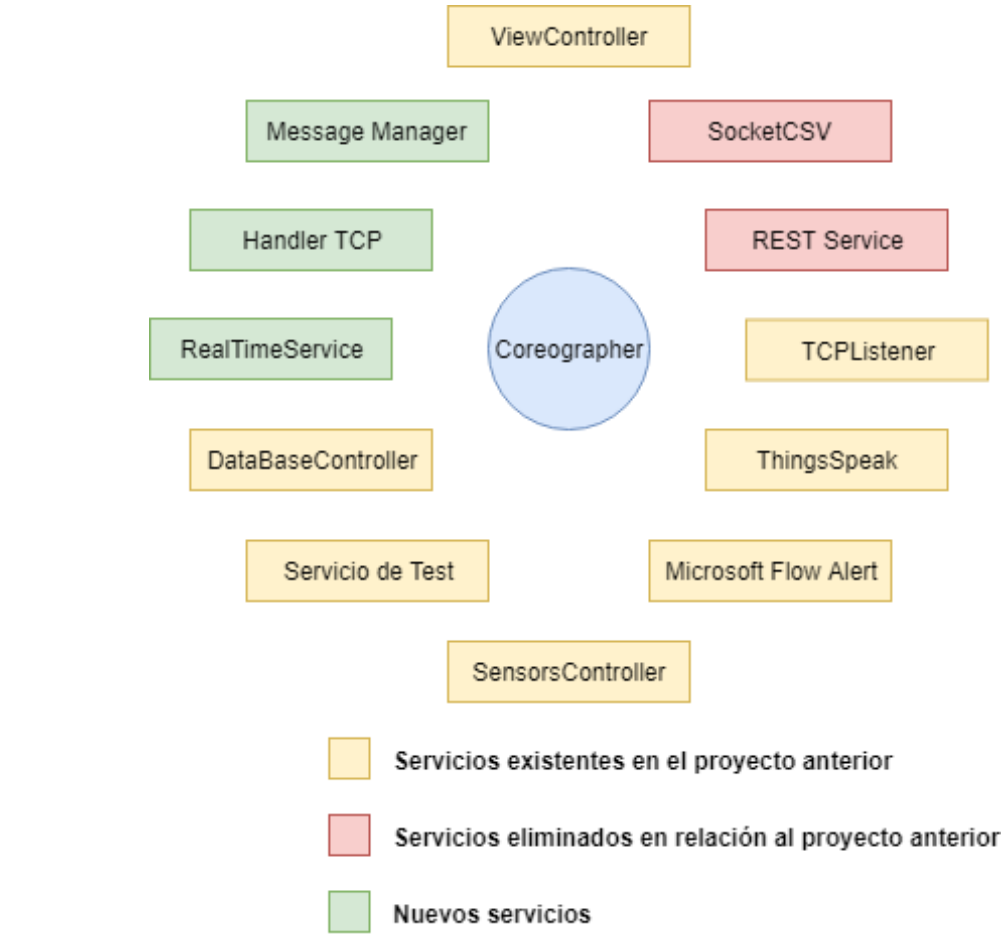

Figura 2 - Esquema global de los servicios que componen la aplicación

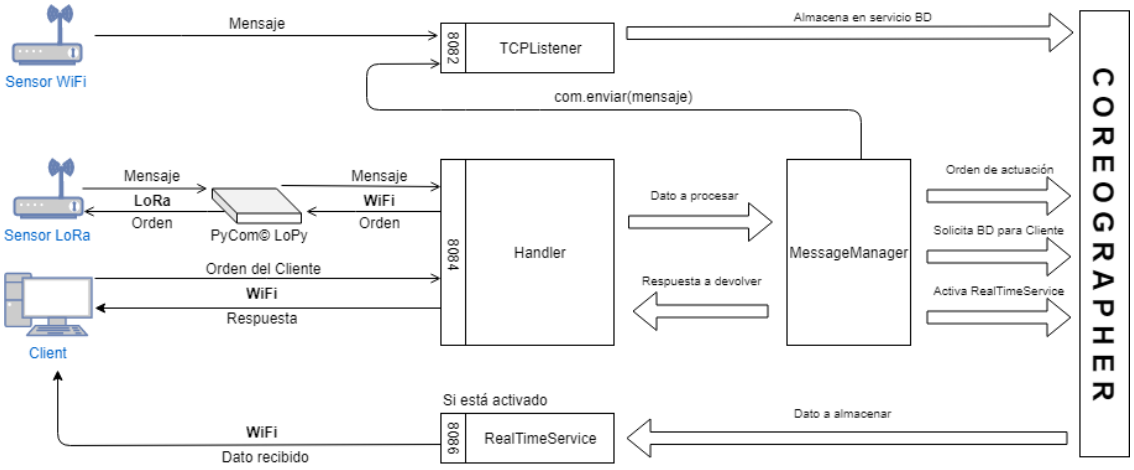

Figura 3 - Visión general de las modificaciones del proyecto

Figura 4 - Esquema del funcionamiento del TCPListener

La segunda vía de comunicación es la más extensa, ya que se interpreta el mensaje recibido y se realiza la acción correspondiente que generará un mensaje de respuesta. Existen dos elementos en esta vía: la tarea de recibir los mensajes es función del HandlerTCP, mientras que la de generar una respuesta relacionada es del MessageManager. Este último es quien recibe los datos enviados por el Handler, que a su vez han sido enviados desde el exterior, y envía al Handler de vuelta un mensaje coherente. Posteriormente se procederá a explicar con más detalle la división entre estos dos procesos, ya que en el dibujo no se representa la división para facilitar la comprensión. A través de esta vía (con los dos servicios implicados) y con los comandos correspondientes:

1. Se reciben los datos de los sensores LoRa, que han sido previamente enviados por el Pycom LoPy (que se encarga de recibir los datos enviados por sensores a través de LoRa, y reenviar los datos de los sensores a la Raspberry a través de una conexión TCP similar a la del TCPListener). Los datos se envían al TCPListener como si de un sensor Wifi se tratase a través de una nueva conexión TCP. Para evitar modificar el código existente, y debido a que el TCPListener utiliza un socket desarrollado por el equipo SABIEN que a su vez usa una estructura XML compleja, ha sido necesario utilizar el Handler de intermediario con un socket común sin estructura XML [25], quien reciba de LoPy y envíe el dato con el formato utilizado anteriormente al TCPListener. Otra solución podría haber sido implementar la misma estructura XML en el LoPy y enviar los datos directamente al TCPListener, pero no conseguí en su momento adaptar el mismo formato que el Socket desarrollado por SABIEN.
2. Se gestionan los datos. Tanto como para acceder a la base de datos, como para activar o desactivar el servicio de Tiempo Real. Podemos obtener toda la base de datos completa, y también podemos borrarlo todo. El servicio de tiempo real permanece inactivo hasta que la orden correspondiente recibida por el Handler lo activa. Estas funciones se observan de forma concisa en la imagen posterior.
3. Se añaden nuevas órdenes de actuación, para que posteriormente sean enviadas a los sensores. Estas órdenes se almacenan en una variable para que cuando alguna vía de comunicación con los sensores se abra, se envíe a través de esta vía a los sensores la orden de actuación. En el esquema posterior se observa esta función en la segunda entrada del Cliente. Por ejemplo, cuando LoPy, encargado de comunicarse con sensores LoRa, abra una nueva conexión para enviar el dato recibido por LoRa, la solución aprovechará la conexión para enviar las ordenes recibidas hasta el momento y la variable se vaciará. Es necesario

implementar un TCPClient [26] que envíe las órdenes de actuación al socket del sensor, puesto que el TCPListener no está habilitado para enviar, solo para recibir. Esta limitación redunda en que no podamos enviar datos a los sensores wifi, pero sí a los Sensores LoRa. Por lo que queda pendiente crear un nuevo servicio que implemente una conexión hacia el sensor para poder actuar sobre él.

4. Todo lo anterior se hace a través de TCP, pero subimos un escalón y ahora es posible conectarnos a través del navegador. Se ha habilitado una capa de HTTP que permite realizar todo lo anterior, pero ahora además en lugar de enviar comandos sólo a través de una conexión TCP, lo hace a través de una HTTP con la siguiente estructura: http://127.0.0.1:8084/Comando...

Para entender mejor cuál es el funcionamiento interno de esta vía de comunicación, será de ayuda este esquema:

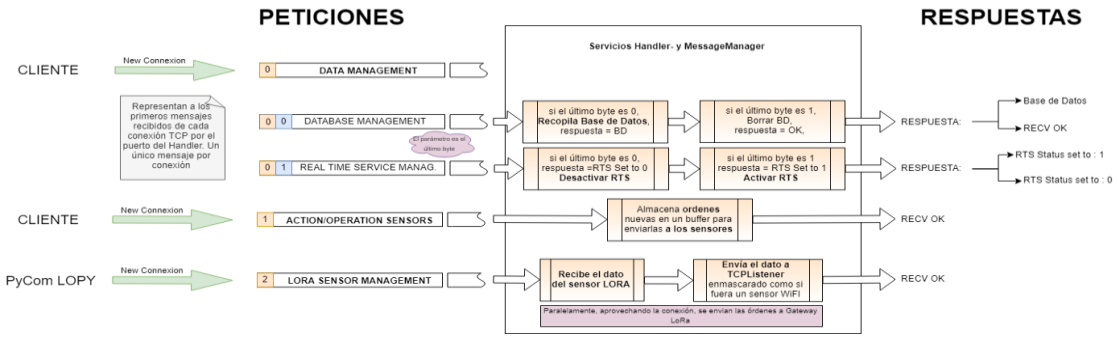

Figura 5 - Esquema de funcionamiento interno de los servicios Handler y Message Manager, que actúan de forma conjunta.

Por último, la única vía unidireccional existente es la del Servicio de Tiempo Real. Si el servicio permanece activo, representado en el esquema por la franja verde, cuando se realice una conexión a este por el puerto 8086 (en la imagen, representada con una flecha verde), empezará a enviar todos los datos que se almacenen en la BD a partir de ese momento obviando los anteriores, permitiendo así tener un control de lo que ocurre en un periodo de tiempo más breve y en tiempo real.

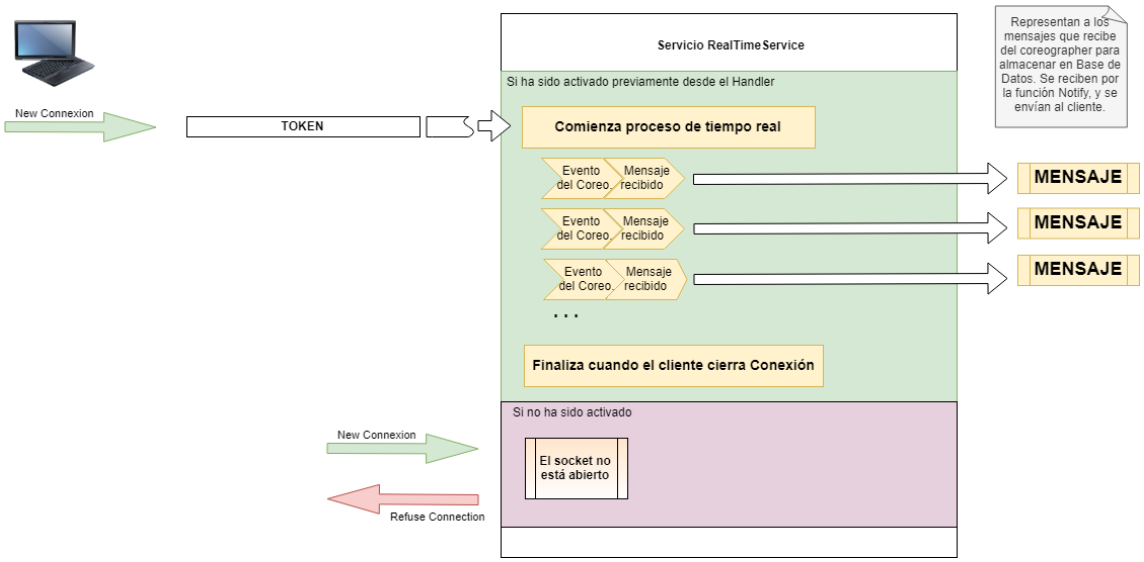

Figura 6 - Esquema de funcionamiento interno y de las salidas del servicio RTSManagement

Únicamente se enviarán los datos recibidos desde el inicio de la conexión con el cliente que interprete los datos.

En relación a la comunicación entre procesos, gestionada por el Coreographer, es importante saber que para enviar un mensaje a un proceso primero hay que crear y componer un objeto, también facilitado por la herramienta, al que hay que incluir el emisor, receptor, un objeto Diccionario con los datos que queremos enviar y un protocolo de envío. En esta solución, el protocolo de envío es el de menor complejidad (he preferido no entrar en detalles ya que todos los mensajes son de tipo Evento, es decir, no se espera una respuesta en el Servicio). Este protocolo se integra con un envío del mensaje que produce en el receptor una llamada al método Notify (XFIPA msg) (con el mensaje en el parámetro recibido de la función). Dentro del método podemos gestionar el mensaje. Para enviar un mensaje, se utilizan la llamada al evento OnPropagateCommand con el mensaje como argumento, que envía el mensaje al proceso o procesos destino. La localización del destino puede ser única (mensaje dirigido a un solo proceso), o múltiple, si deseamos enviar a más de un proceso el mismo mensaje. Esta última característica se consigue a través de máscaras, que permiten dividir las distintas secciones de la aplicación. Las máscaras se componen de texto y un delimitador, el punto, y se registran en un servicio, siendo de carácter único y permanente. Se introducen como si fuera el nombre del receptor del mensaje. Unos ejemplos de máscara son los siguientes:

“GestionDeDatos.SinProceso” – Servicio 1

“GestionDeDatos.ConProceso” – Servicio 2

Si enviamos un mensaje con el receptor GestionDeDatos.ConProceso, el mensaje será recibido en la función Notify (explicado con más detalle en la parte de diseño)

del Servicio 2. El resultado sería el mismo que enviando el mensaje al destino "Servicio 2", mientras que si el receptor es únicamente "GestionDeDatos.*", recibirán ambos servicios el mensaje. Esto permite mucha flexibilidad, ya que no será necesario enviar múltiples mensajes, sino realizar correctamente el etiquetado.

Tecnología utilizada

Para desarrollar este proyecto, he utilizado:

- Windows IOT
- .NET
- C#
- LoRa
- Protocolo HTTP
- MicroPython

Windows IoT

Es un sistema operativo desarrollado por Microsoft orientado a los dispositivos embebidos y procesadores pequeños, y que permite desarrollar y ejecutar código en .NET y se pueden implementar aplicaciones UWP [27] con él. Además, permite su ejecución en procesadores ARM [28], que es el que tiene la raspberry Pi.

.NET

Se trata de un framework de Microsoft que permite desarrollar aplicaciones, y que le da mucho valor a la transparencia de redes. Uno de los lenguajes del que se vale este software para crear aplicaciones es C# [29], el utilizado en cuestión.

C#

Es un lenguaje orientado a objetos fuertemente tipado, desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis deriva de lenguajes anteriores como C y C++ [30]

LoRa

Como sus siglas propiamente indican (LoRa – Long Range), se trata de una tecnología de comunicaciones emergente similar al WiFi para grandes distancias. Sus dos características principales son la gran amplitud y el bajo uso de potencia que necesita para funcionar. Es la tecnología más importante en este proyecto, pues la finalidad de este es implementar comunicaciones sin utilizar casi recursos.

HTTP

Se trata de un protocolo de comunicación que utilizan las páginas web para transferir información a través de la World Wide Web. Es destacable que es un protocolo sin estado, es decir, no guarda datos de las conexiones anteriores.

MicroPython

Es un lenguaje de programación orientado al desarrollo en pequeños dispositivos. Utiliza la semántica clásica de Python debido a que implementa su versión 3ra optimizada para microcontroladores y entornos reducidos.

5. Desarrollo de la solución

Si observamos la figura numero 2, la primera del punto anterior, es fácil detectar los servicios agregados en este proyecto, además de los eliminados. La memoria de este proyecto se enfoca a los servicios añadidos, aunque haré un resumen de cada uno de los otros servicios, explicados ya en la memoria del proyecto anterior [23]. Todo servicio que queramos implementar en la aplicación debe heredar una clase proporcionada por la herramienta. Si un proceso hereda de esta clase, se convierte en un servicio, por lo que pasa a ejecutarse de forma independiente. Para que se ejecute, debe haberse registrado en el coreógrafo, ya que este es el encargado de poner el servicio en marcha.

A continuación, en la siguiente imagen, se muestran las interacciones de coreografía que se producen en nuestra solución.

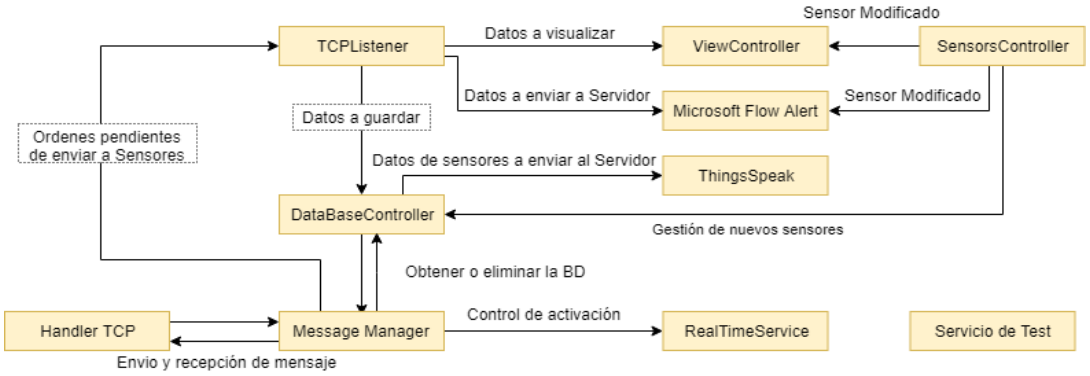

Figura 7 – Mensajes de coreografía intercambiados entre los servicios

Estas interacciones representan toda la comunicación existente entre los servicios, y que es necesaria para que puedan funcionar. En resumen, las flechas representan a los mensajes de coreografía enviados y recibidos.

La herramienta utilizada permite seleccionar un protocolo de envío de mensajes, que se asigna al propio mensaje. Los protocolos utilizados en este proyecto son los dos siguientes:

- **Mensaje de tipo Event**

Este tipo de mensaje indica que no se espera ninguna respuesta a ese mensaje. Es un evento que se produce. El Servicio receptor tiene acceso al mensaje desde la función Notificar. Es el más utilizado en esta solución, y el único que he utilizado en mi implementación.

- **Mensaje de tipo Request**

Se trata de un tipo de mensaje más complejo, pues cuando el receptor lo recibe, hay un campo en el mensaje en el que está el nombre de uno de los métodos del receptor. Ese método se ejecutará, sin hacer falta que pase por el Notificar.

El único servicio autónomo, es decir, que no envía ni recibe ningún mensaje de coreografía es el servicio de Test, que como se ha explicado, se comunica como si fuera un emisor TCP con destino al TCPListener, y como no se trata de un mensaje de coreografía considero que es mejor no indicarlo en el esquema.

Haciendo hincapié en el esquema superior, el comportamiento externo del TCPListener es enviar los datos recibidos por una conexión TCP con el Envoltorio [31] en XML hacia los servicios que necesiten de esos datos, que son el ViewController, MicrosoftFlowAlert y DataBaseController, este último para guardar los datos en una BD Local. Los datos de la BD tienen como destino dos servicios. El servidor de ThingSpeak y el MessageManager. Y para gestionar esos datos se utiliza el servicio DataBaseController. El Handler TCP únicamente se comunica con el MessageManager, reduciendo así la complejidad de las dos piezas. El servicio MessageManager se comunicará con el Servicio de base de datos y con otro servicio adyacente, el RealTimeService. Este último no envía ningún mensaje de coreografía. Por último, el proceso SensorsController, gestiona los sensores, y si se produce un cambio en la configuración de estos, se enviará la información de cambio a los servicios pertinentes.

Absolutamente todos los servicios necesitan implementar una serie de métodos que explicaré a continuación para permitirse estar en un entorno de coreografía. Es importante saber que estos métodos se encuentran en todos y cada uno de los servicios existentes.

Start()

El código incluido en este método se ejecutará una única vez al inicio de la ejecución del servicio.

Stop()

Aunque se puede deducir, su función es ejecutarse cuando finalice el Servicio.

Notificar(XFIPA msg)

Este método se ejecutará en el servicio cuando este reciba un mensaje de otro servicio. El mensaje lo recibimos a través del argumento de la llamada del método. Dependiendo del tipo de mensaje y valiéndonos de una estructura Switch-Case, seleccionamos el método a ejecutar. La estructura interna de la función notificar es la siguiente:

```
public bool Notificar(XFIPAMSG msg)
{
    switch (msg.Contenido.metodo)
    {
        case "ReceiveData":
            ReceiveData(msg);
            break;

    }
            return true;
}
```

Figura 8 - Función Notificar del proceso HandlerTCP

Aquí se observa que al recibir el mensaje se ejecuta la función correspondiente al método del mensaje. A continuación, se explicarán los servicios ya existentes previamente en el proyecto:

TCPListener

Se trata de un receptor de mensajes TCP. Actúa solo como receptor, ya que no permite emitir ningún mensaje, para ello hace falta un TCPConnector. Queda pendiente para la siguiente versión. Únicamente reenvía el mensaje al servicio DBController, quien se encargará de detectar el tipo de dato, si es un dato de sensor o un sensor agregado y posteriormente almacenarlo en la BD. El Servicio se ejecuta al inicio del de la ejecución del proyecto. Se trata de un proceso limpio y sencillo que redirecciona el dato del sensor para que pueda ser almacenado. Es importante volver a mencionar que para que se produzca la conexión con este servicio, el emisor debe poseer la misma estructura XML del mensaje, lo que la complicación que supone enviar un dato a este Socket en lugar de al nuevo HandlerTCP es bastante mayor.

DataBaseController

Este proceso es el encargado de recibir todos los mensajes de coreografía dedicados a salvar datos. Otra de sus funciones es el enviar, vía coreógrafo también, datos que soliciten otros servicios. Realmente no es este servicio quien tiene acceso directo a la BD. Para ello, como he mencionado anteriormente, existe otro proyecto dentro de la solución con el que se vale para poder almacenar el dato y que es el encargado de crear la BD en SQL.

ThingsSpeak

El servicio ThingsSpeak es el que se conecta al servidor de ThingSpeak con la finalidad de almacenar los datos recibidos para su posterior visualización. La transmisión se hace aplicando la API que proporciona el servidor, a través de

HTTP. Una variable controla la frecuencia con la que se envían los datos al ThingSpeak, que se asigna en el inicio del proyecto.

MicrosoftFlowAlerts

Proceso que se encarga de enviar las correspondientes alertas a la web de Microsoft Flow [32]. Se envía una alerta únicamente en estos dos casos:

- Un sensor ha fallado
- Un valor fuera del rango esperado

En un futuro se podrían añadir más alertas, o implementar un sistema de creación de avisos.

Servicio Test

Este servicio no aporta ninguna funcionalidad extra a la solución. Únicamente sirve para emular un sensor externo. Lleva consigo una interfaz gráfica donde se ubican varios botones y al apretar un botón, se envía un mensaje TCP (Diferente para cada botón), que deberá ser recibido por el TCPListene r. Este servicio se diseño en la anterior implementación del proyecto debido a que no disponían de ningún sensor físico para emitir datos. Considero que es necesario para las pruebas, así que, aunque no tenga utilizad funcional, nos ayuda a detectar posibles errores de funcionamiento de la solución.

Servicio ViewController

El servicio en cuestión se encarga únicamente de mostrar por pantalla los datos recibidos por los sensores en una caja de texto.

Servicio SensorsController

Desde este servicio, que también contiene una parte gráfica, se modifica la configuración relacionada con los sensores. Esta modificación viaja a través de mensajes de coreografía hacia los servicios siguientes: MicrosoftFlowAlert, DBController y ViewController.

Ahora se presentarán los servicios que son nuevos en este proyecto:

HandlerTCP

Es el encargado de gestionar conexiones TCP entrantes. Estas conexiones, como se observa en el dibujo del inicio de este punto, se retransmiten directamente al servicio MessageManager. El servicio recibirá un mensaje proveniente de MessageManager con la respuesta solicitada por el cliente, que transmitirá hacia el Servicio actual. El HandlerTCP además contiene una funcionalidad adyacente,

que permite que también se pueda realizar la petición en lugar de con un Socket TCP, a través del navegador. La solución para esto ha sido extraer los parámetros necesarios de la petición HTTP del navegador que ha sido recibida, y componer un mensaje legible por el navegador. Este punto ha traído diversas complicaciones, ya que el cuerpo del mensaje a devolver hacia el Navegador tenía un formato extraño. Ha sido en este punto donde he recibido la ayuda de Juan Fran, quien ha trabajado con anterioridad en redes y conoce bien la estructura de las peticiones y respuestas de un servidor web. Por otra parte, han existido varios problemas en el cierre de la conexión, pues ambos sockets (emisor y receptor) debían ser síncronos, pero habiendo adoptado esta configuración el problema queda resuelto.

MessageManager

Al principio del desarrollo, la funcionalidad de este Servicio se encontraba en el HandlerTCP, pero debido a que aumentó mucho su complejidad, mi tutora me recomendó separarlo en dos servicios, uno encargado de la gestión de las conexiones, el Handler, y otro para recibir y gestionar esos mensajes.

Este proceso contiene la lógica de negocio necesaria para realizar la segunda vía de comunicación, explicada en el punto anterior y engloba todas las funcionalidades descritas en ese punto. Recibe el mensaje y lo procesa, en función del primer dígito, que puede ser:

- Si el primer dígito es un 0, tratamos los datos que ya han sido recibidos. Se puede borrar u obtener la base de datos a partir de comandos encapsulados siempre con un 0 al principio. Es necesario mencionar que hace falta una autentificación mínima, es decir, El parámetro siguiente al 0 debe de ser un token aceptado por el sistema, a modo de contraseña. Si el token no está aceptado, se deniega y se devuelve un mensaje de error. Si se acepta el token, dependiendo de los dos siguientes parámetros podemos, con un 0 en el primer parámetro
    - Si el último parámetro es un 0, obtiene toda la BD.
    - Si es un 1, se borra al completo.

- Si hay un 1, entonces podemos activar o desactivar el Sistema de tiempo real con el último parámetro. Con 0 y 1 respectivamente. En segundo lugar, si el primer parámetro es un 1, nos encontramos en la gestión del control de nuestra aplicación. Localizamos el token en el segundo parámetro. El tercer parámetro, muy importante, serán las órdenes que tienen que recibir los sensores. La forma en la que el sensor recibe el mensaje ya se ha explicado con detenimiento en el punto 4.

- En último lugar, si el primer parámetro es un 2, nos encontramos ante la recepción de datos de sensores LoRa, que han sido Reenviados utilizando el

módulo LoPy, o la recepción de datos provenientes de sensores wifi que tengan un socket genérico y sin estructura XML. El hecho de que se haya implementado un socket genérico facilita mucho la tarea de conectarse a la solución ya que no tendremos que adaptar el mensaje dentro de una estructura XML.

RealTimeServiceManagement

Se trata de un servicio simple que también adopta un socket TCP. Debía haber alguna forma de extraer los datos en tiempo real, es decir, los datos que están siendo recibidos por la aplicación en este momento, obviando todo lo almacenando en la base de datos. Para cumplir esta funcionalidad, se ha configurado el servicio para que reciba, de igual manera que el servicio DBController, los mensajes que han sido enviados a la aplicación por los sensores. DBController los almacenará mientras que este servicio en cuestión los enviará a través de la conexión TCP activa. Se ha añadido una condición que se aprueba desde el Servicio de MessageManager, cuando el cliente solicita activar este servicio. Si el cliente lo solicita, la condición se cumplirá, y entonces será posible recibir nuevas conexiones TCP. Si no hay conexión activa, lógicamente no se enviará el mensaje.

Después de clarificar los anteriores componentes, presentes en el proyecto, procedo a explicar por qué he eliminado dos de los anteriores procesos.

En primer lugar, el proceso SocketCSV, que recopilaba los datos de la BD, y luego los enviaba a través de una Api muy simple, ha dejado de ser funcional pues ahora tenemos el servicio HandlerTCP, que ha sido explicado anteriormente, pero que, en resumen, permite extraer los datos de la BD, que era la única funcionalidad del servicio SocketCSV.

El servicio REST es otra forma de extraer los datos, pero que queda desfasada al estar operativo el HandlerTCP. Ocurrió exactamente lo mismo que el SocketCSV. Me resultó mucho más fácil manejar conexiones TCP con sockets que protocolos en HTTP, que al fin y al cabo son conexiones TCP con una capa más.

6. Implantación del sistema

En todo momento la solución principal se ha implementado sobre una Raspberry Pi3 con Windows IOT instalado. El sistema operativo para dispositivos empotrados nos ofrece el siguiente entorno:

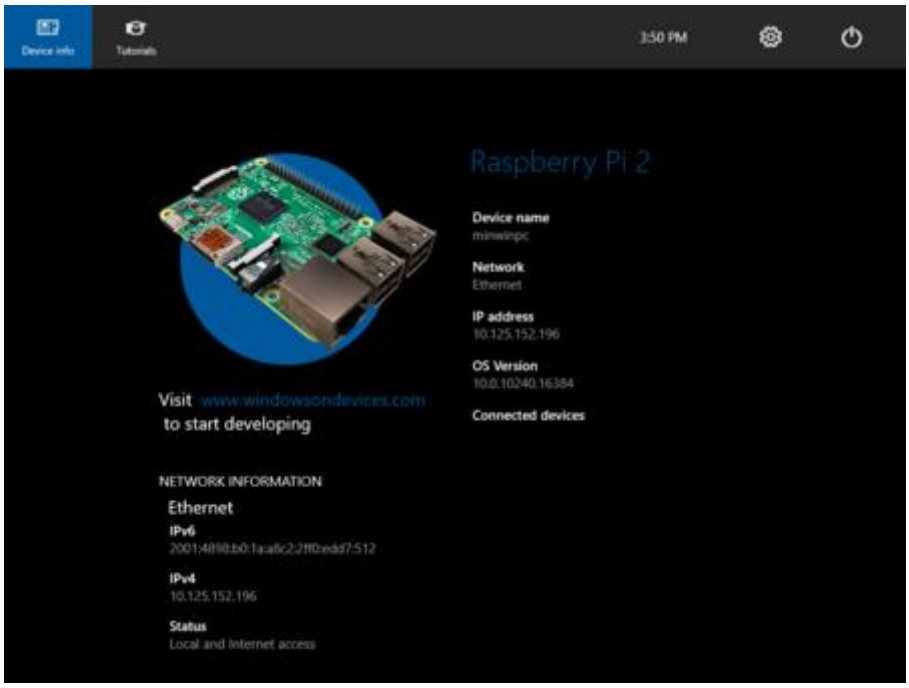

Figura 9 - Inicio de Windows IOT en una Raspberry Pi 2

A demás el módulo LoPy, programado en MicroPython, es el encargado de enviar los mensajes recibidos por LoRa a la Raspberry Pi. Podemos ver en las siguientes imágenes a los dos módulos utilizados para realizar esta función:

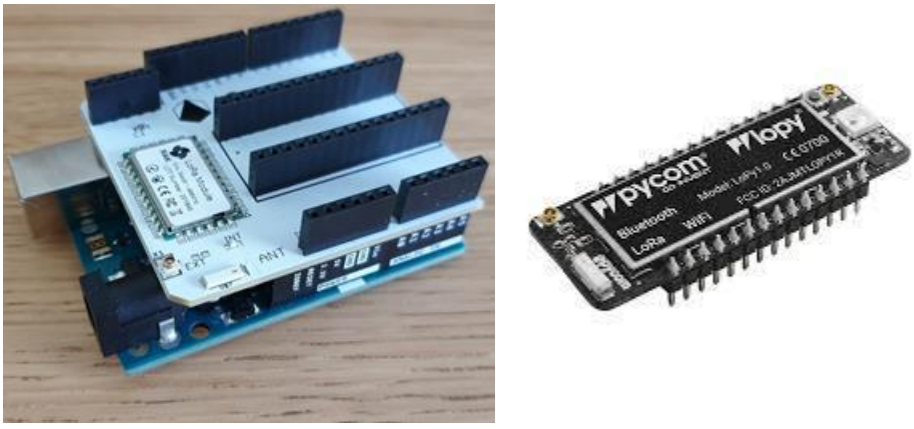

Figura 10 – (a la derecha) Módulo LoPy, encargado de recibir señales LoRa y reenviarlas via Wifi a la Raspberry; (a la izquierda) otro posible módulo del fabricante Pi-Supply con un adaptador RF LoRa™ y Arduino One

Figura 11 - Raspberry Pi. Recoge los datos enviados por LoPy como si fuesen de sensores

Figura 12 – Sensores Posibles (a la derecha con Arduino One y adaptador Dragino, a la izquierda Arduino nano con adaaptador Pi-Supply)

Se compila y carga el código del módulo LoPy, que comenzará a recibir mensajes LoRa. Posteriormente, se descarga e instala el Windows IoT en una tarjeta SD, que se introducirá en la Raspberry. Para ello utilizaremos la app oficial Dashboard. Esta aplicación además de permitir la instalación del Windows sobre una tarjeta SD, nos muestra todos los dispositivos en red local que lo están ejecutando, permitiendo su administración. La obtenemos a través de la tienda de Windows, y tiene este aspecto:

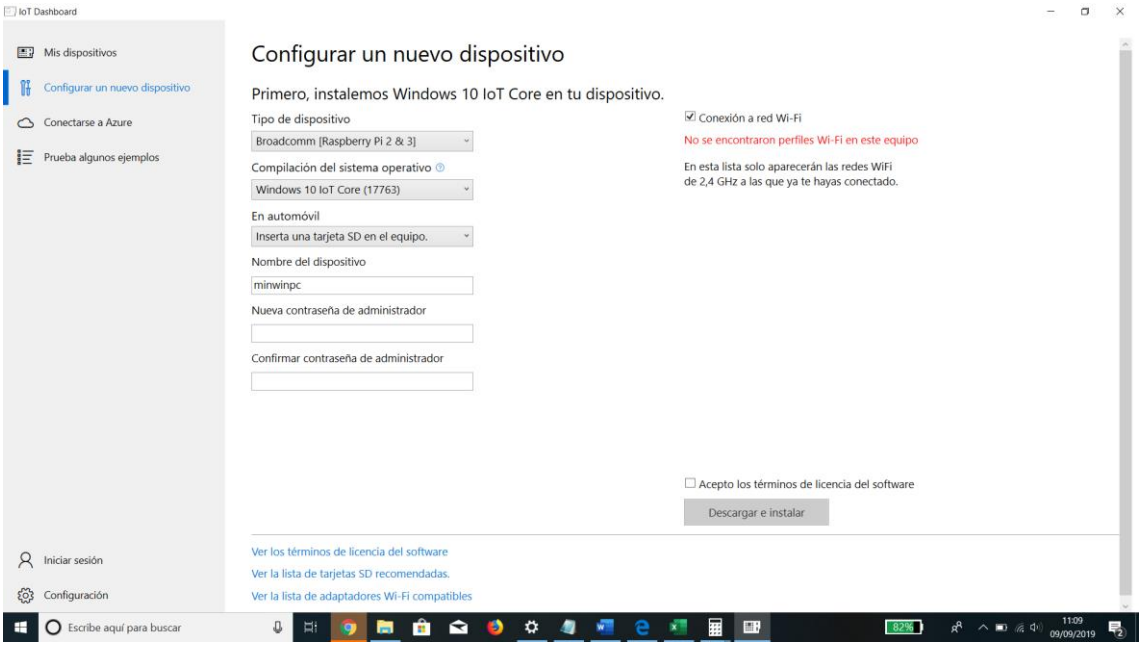

Figura 14 - Dashboard mostrando la configuración de nuevos dispositivos a través de la tarjeta SD

Es necesario conectar la Raspberry Pi a la red local para que sea detectada por dispositivos de la misma red. Se compila el programa de forma remota desde el IDE Visual Studio 2019 [33] y se ejecuta. Para ello, este entorno de desarrollo contiene un botón a ejecutar, el de equipo remoto:

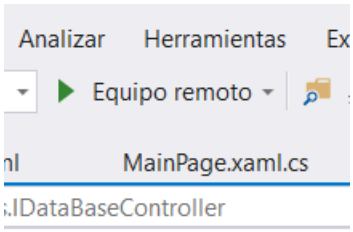

Figura 15 - Indicación del botón de compilar y ejecutar en máquina remota

A partir de ahí se puede ejecutar el cliente y probar toda la funcionalidad de este proyecto. Todas las conexiones se han realizado dentro de una misma red local, pero no habría problemas para trasladar alguna de las partes a otra red con la configuración de firewall y puertos adaptada.

7. Pruebas

En primer lugar, es necesario configurar el Coreógrafo al inicio de su ejecución. La ventana muestra en la ilustración inferior los campos que se han de configurar:

Figura 15 - Configuración previa al inicio de la coreografía

Los sensores van separados por comas, e indican el número de sensores que existen en la aplicación, con su ID asociado. El segundo parámetro es "Time TS" y está relacionado con el periodo del envío de datos desde el Servicio ThingsSpeak. En último lugar, el valor de "Time MF" indica cuanto tiempo en segundos debe suceder entre la comprobación de alertas y avisos que se realizan al servidor Microsoft Flow desde su servicio correspondiente.

Se ha utilizado el servicio TEST, que es el encargado de emular el sensor, que sustituye al hardware de los sensores. Este servicio envía mensajes interpretables por el proceso TCPListener.

Para probar los nuevos servicios nos centramos en las pruebas de mi desarrollo:

Para probar los servicios que he implementado he creado una mini-aplicación que se conecta a la Solución. Este cliente actúa como si fuese el cliente GUI [34] y comprueba toda la funcionalidad del proyecto.

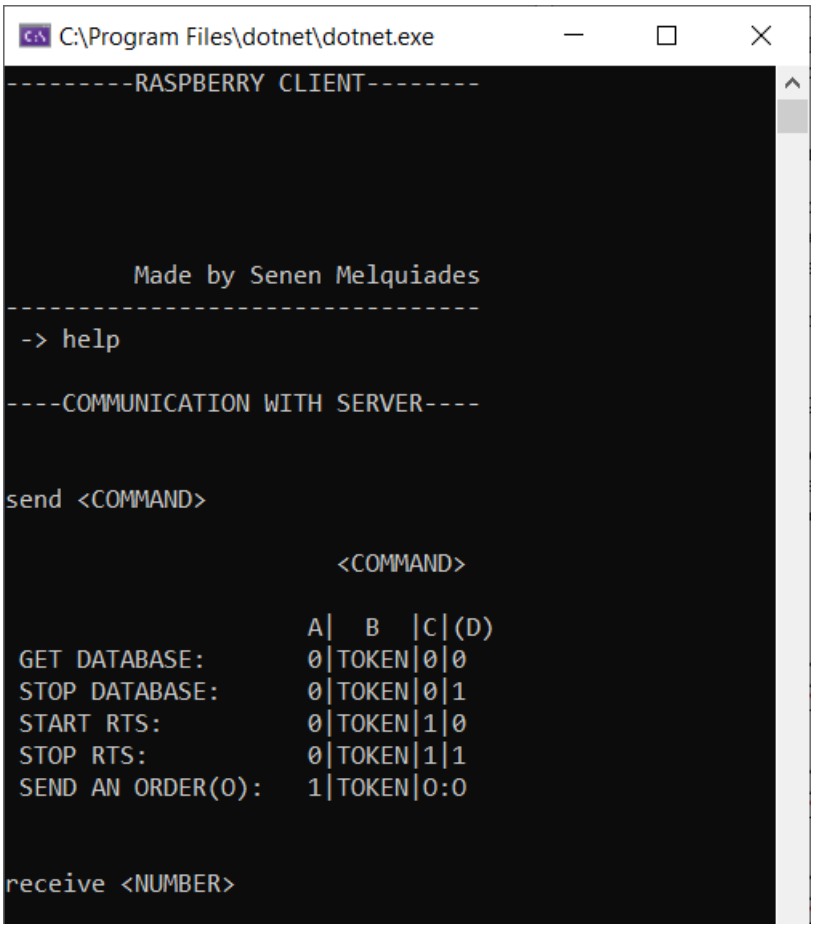

Figura 16 – Salida del comando "help"

Desde la aplicación en primer lugar podemos obtener o eliminar toda la base de datos, y el resultado se muestra por consola. Considerando un token válido como "11111", el mensaje a enviar para obtener la BD es 0|1111|0|0, mientras que para eliminarla, hay que substituir el último dígito por 1.

Aquí se puede ver el resultado de ambos procesos:

```
 -> send 0|1111|0|0
Socket connected to [fe80::293a:22c8:2f98:a370%7]:8084
Data:Sensor,Type,Time stamp,Values
;1,Humidity,04/09/2019 8:11:35,23,28,18
;2,Humidity,04/09/2019 8:11:45,27,32,22
;1,Humidity,04/09/2019 8:11:55,16,21,11
;2,Humidity,04/09/2019 8:12:05,20,25,15
;1,Humidity,04/09/2019 8:12:15,18,23,13
;2,Humidity,04/09/2019 8:12:25,22,27,17
;2,Humidity,04/09/2019 8:12:35,25,30,20
;1,Humidity,08/09/2019 9:57:03,26,31,21
;2,Humidity,08/09/2019 9:57:12,10,15,5
;0,Humidity,08/09/2019 9:57:22,14,19,9

 -> send 0|1111|0|1
Socket connected to [fe80::293a:22c8:2f98:a370%7]:8084
Database Deleted.
 -> send 0|1111|0|0
Socket connected to [fe80::293a:22c8:2f98:a370%7]:8084
Data:Sensor,Type,Time stamp,Values
;2,Humidity,08/09/2019 9:57:52,25,30,20
```

Figura 17 - Ejecución de prueba

Como se observa en la imagen, al solicitar la BD después de eliminarla, solo nos devuelve una fila no por que no la haya borrado, si no porque la ha insertado después del borrado debido a un proceso que genera continuamente mensajes como si fueran de sensores. Este proceso, como he mencionado anteriormente, se ejecuta desde el Servicio Test.

En segundo lugar, la opción de activar y desactivar el sistema de Tiempo Real viene determinada por el comando 0|1111|**1**|**x** . Si la X vale 0, el sistema se inicia, y si vale 1 se para.

Observamos cual es la respuesta si se inicia o se para:

```
---------RASPBERRY CLIENT--------




         Made by Senen Melquiades
---------------------------------
 -> send 0|1111|1|0
Socket connected to [fe80::293a:22c8:2f98:a370%7]:8084
RTS Status set to : 0
 -> send 0|1111|1|1
Socket connected to [fe80::293a:22c8:2f98:a370%7]:8084
RTS Status set to : 1
 ->
```

Figura 18 – Salida del Cliente

Por otro lado, también podemos enviar órdenes a los sensores con el comando 1|1111|Orden1:Orden2:Orden3… con el delimitador (:) entre cada orden. Si no ha habido problemas, la aplicación nos devolverá el siguiente mensaje indicando que la orden ha sido agregada:

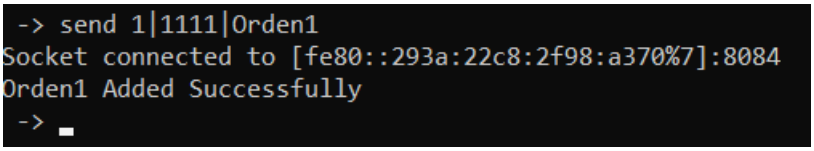

Figura 19 – Salida del Cliente

Este cliente, además, nos permite emular el dato de un sensor con el prefijo 2, Aunque no aparece en la ayuda, la segunda ilustración de este punto, se puede enviar el dato emulando ser un sensor. Con el comando 2|{DATO} podemos enviar el dato para que sea interpretado como un sensor. Todos los sensores, por tanto, deberán imponerse el prefijo "2|" en todos los mensajes que envíen a la aplicación.

Por último, es necesario comprobar que el servicio de tiempo real funciona, para lo que se ha habilitado el comando "receive". Este comando contiene un argumento, que indica el número de mensajes que va a querer recibir la aplicación antes de cerrar la conexión. Si ejecutamos el comando "receive 3" obtendremos la siguiente salida conforme vayan llegando los datos:

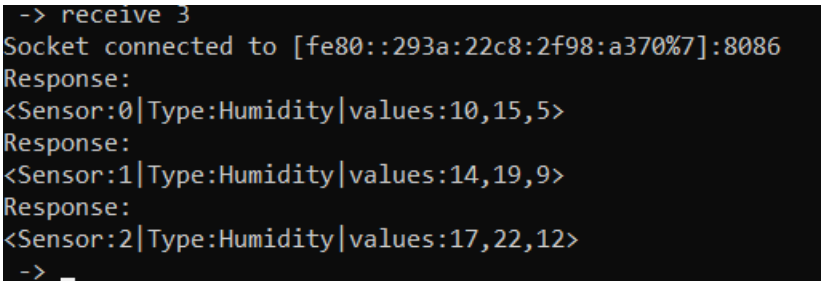

Figura 20 - Salida del Cliente

8. Conclusiones

En este proyecto he aprendido a usar la tecnología de LoRa y también a implementarla en dispositivos y aplicaciones en una red de sensores. Por otro lado, también he mejorado mucho los conocimientos de sockets que poseía, y ahora sé manejar perfectamente la herramienta de Visual Studio para .NET, por lo que podría adaptarla a más usos.

Una de las mayores complicaciones de este proyecto ha sido el no poseer los conocimientos necesarios que la solución requería, por lo que he tenido que aprender sobre varios tipos de tecnologías para redes de sensores que han sido necesarias para su implementación, aunque también creo que me ha servido para tener una mejor visión de la Ingeniería Informática. En relación a las demás complicaciones que han surgido en el desarrollo del proyecto, me ha quedado demostrado que todo tiene una solución cueste más o menos.

Por otro lado, he visto una clara relación entre los sistemas distribuidos con la solución principal, que utiliza un entorno de coreografía (que es distribuido). En segundo lugar y en el ámbito de las redes, se ha trabajado ampliamente con sockets de emisión y recepción TCP, por lo que podemos afirmar que existe una relación con la rama de Tecnologías de la Información, o con la asignatura de Redes del grado. También existe una relación con la asignatura Bases de datos, pues trabajamos con una BD en SQL. También conceptos como HTTP, REST, CSV, IoT, etc. han sido adoptados en líneas generales gracias a la carrera de Ingeniería informática.

Por último, ha sido necesario aprender una tecnología nueva, el .NET, y un lenguaje de programación, Python, debido a que no se enseñan en la carrera. También se ha estudiado una versión de Python para microcontroladores, Micro-Python, necesario para programar el componente de PyCom.

9. Trabajos futuros

Una de las características principales de esta solución es su flexibilidad, es muy adaptable a cambios, lo que permitirá que en el futuro abarque ámbitos diferentes, no solo el agronómico. Es muy fácil crear nuevos servicios sin tocar absolutamente nada del código de los servicios existentes. Por lo que se irá adaptando a nuevas necesidades o a nuevas tecnologías.

En mi caso, voy a continuar desarrollando la aplicación con el objetivo de crear un Gateway capaz de implementar una estructura de Smart-Farming.

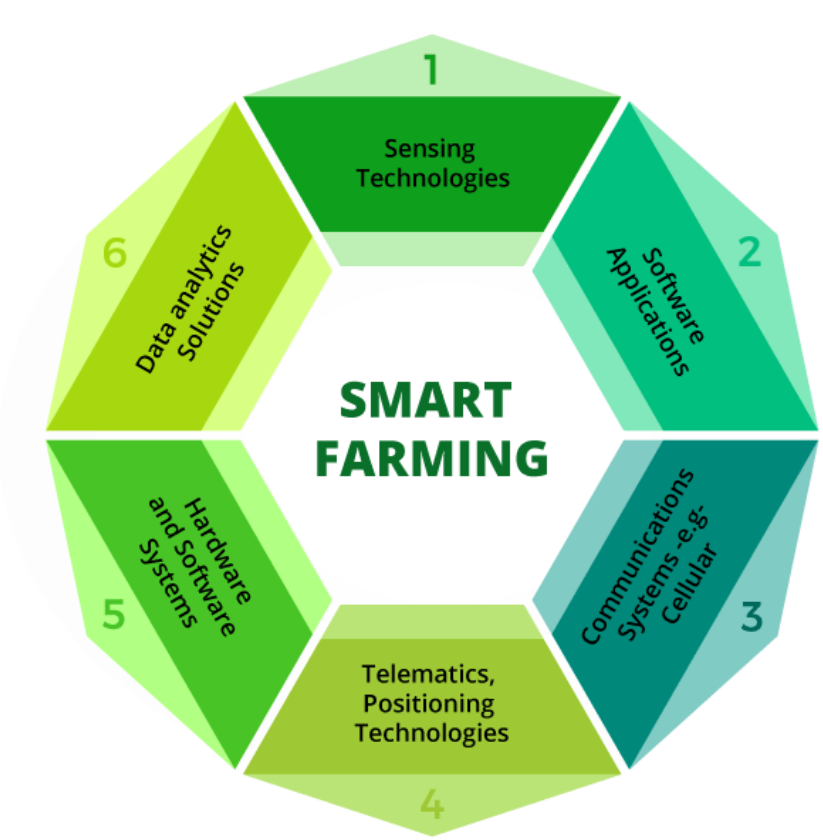

Figura 21 – Smart Farming

Este proyecto irá dedicado a varios colegios de Europa con el propósito de acercar más la informática y el mundo de la IoT a los más jóvenes. Además, debido al bajo coste del hardware y la cantidad de proyectos existentes sobre dicho hardware, es mucho más fácil desarrollar soluciones adaptadas a nuestras necesidades. Por todo esto considero que este proyecto no acaba aquí, es más, creo que nos encontramos en sus inicios.

10. Referencias

1. ITACA

Itaca.edu.gva.es

2. UPV

www.upv.es

3. SABIEN

www.sabien.upv.es

4. Raspberry Pi

https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

5. CSV

Comma separated values, valores separados por comas:

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

6. REST

Transferencia de estado representacional:

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

7. TCP

Transmission Control Protocol

https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisi%C3%B3n

8. LoRa

https://lora-alliance.org/

9. Windows IOT

https://developer.microsoft.com/es-es/windows/iot

10. ZigBee

https://zigbee.org/

11. Fog Computing

https://en.wikipedia.org/wiki/Fog_computing

12. ThingsSpeak

https://thingspeak.com/

13. HTTP

Protocolo de transferencia de hipertexto

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

14. LoRaWan

https://lorawan.es/

15. TTN

https://www.thethingsnetwork.org/

16. ETSINF

https://www.inf.upv.es/www/etsinf/es/

17. Raw LoRa

https://docs.pycom.io/tutorials/lora/lora-mac/

18. Digital Ecosystem Utilisation en IOF2020

https://www.iof2020.eu/trials/vegetables/digital-ecosystem-utilisation

19. W3C

https://www.w3c.es/

20. DAL

Capa de acceso a datos

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

21. BD

Base de datos

22. SQL

Lenguaje de consulta estructurada

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

23. Proyecto anterior de Joan Benedito Fuster, Integración de servicios IOT.

El proyecto titulado así se creó en la ETSINF. Proyecto publicado en el año 2018.

https://riunet.upv.es/handle/10251/111154

24. LoPy

https://pycom.io/product/lopy4/

25. XML

Extensible Markup Languaje

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

26. TCPClient Class

https://docs.microsoft.com/es-es/dotnet/api/system.net.sockets.tcpclient?view=netframework-4.8

27. UWP

Universal Windows Platform

https://docs.microsoft.com/es-es/windows/uwp/get-started/universal-application-platform-guide

28. ARM

https://www.arm.com/

29. C#

https://docs.microsoft.com/es-es/dotnet/csharp/

30. C y C++

C++ es una versión con mejoras de C, que permite trabajar con objetos.

https://es.wikipedia.org/wiki/C%2B%2B

31. Patrón Envoltorio

https://es.wikipedia.org/wiki/Adaptador_(patr%C3%B3n_de_dise%C3%B1o)

32. Microsoft Flow

https://flow.microsoft.com/es-es/

33. Visual Studio 2019

https://visualstudio.microsoft.com/es/vs/

34. GUI

https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario