

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA**



Proyecto Fin de Carrera

**Red de sensores auto-configurable mediante tecnologías ZigBee y Arduino
con monitorización por aplicación Android**



AUTOR: José Salvador Montesinos Navarro
DIRECTORES: Juan Carlos Sánchez Aarnoutse
Felipe García Sánchez
Septiembre / 2013

Autor	José Salvador Montesinos Navarro
E-mail del autor	jmontesinos@hotmail.es
Directores	Juan Carlos Sánchez Aarnoutse Felipe García Sánchez
E-mail de directores	juanc.sanchez@upct.es felipe.garcia@upct.es
Título del PFC	Red de sensores auto-configurable mediante tecnologías ZigBee y Arduino con monitorización por aplicación Android.
Descriptores	ZigBee, Arduino, Android, Bluetooth, Sensores, Inalámbrico, Monitorización, Ad-hoc, Redes Mesh
<p>Resumen</p> <p>Diseño e implementación de una red inalámbrica de sensores de temperatura y luz capaz de reconfigurarse automáticamente al variar la disposición y/o disponibilidad de los nodos que la componen. Estos nodos están formados por placas Arduino que se comunican entre sí mediante la tecnología ZigBee.</p> <p>Los datos de los sensores, junto con la ruta que siguen los paquetes dentro de la red, son recopilados por un nodo coordinador capaz de transmitir dicha información vía Bluetooth a un dispositivo Android, donde una aplicación se encarga de monitorizar la red, mostrando dicha información tanto de forma gráfica como en modo texto.</p>	
Titulación	Ingeniero Técnico de Telecomunicaciones, Esp. Telemática
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de presentación	09/2013

Índice

1.- Introducción.....	3
1.1.- Planteamiento inicial	3
1.2.- Objetivos.....	3
1.3.- Organización de la memoria.....	4
2.- Tecnologías	5
2.1.- Arduino	5
2.2.- ZigBee	9
2.3.- Bluetooth	17
2.4.- Android.....	18
3.- Red de Sensores.....	22
3.1.- Topología	22
3.2.- Hardware	23
3.3.- Configuración	24
3.4.- Protocolo de comunicación	27
4.- Aplicación de monitorización.....	29
4.1.- Comunicación Bluetooth	30
4.2.- Representación de la información	31
5.- Conclusiones.....	32
5.1.- Problemas encontrados	32
5.2.- Líneas futuras	34
Anexo A. Código y flujogramas	35
A1.- Coordinador	35
A2.- Router	40
A3.- Terminal.....	43
A4.- MainActivity.....	47
A5.- AndroidManifest.....	59
Anexo B. Bibliografía y referencias	60

1.- Introducción

Esta memoria recoge información detallada acerca del proceso de diseño, desarrollo e implementación del presente Proyecto Fin de Carrera. En este primer punto se expone cuál fue la idea original del proyecto, los objetivos que se pretenden alcanzar y cómo se estructuran los siguientes capítulos de que consta este documento.

1.1.- Planteamiento inicial

En un principio este proyecto surgió de la necesidad de implementar una red inalámbrica con fines esencialmente didácticos donde se pudiera observar de forma gráfica e intuitiva cómo se forman los enlaces entre los distintos componentes del sistema. Si estos enlaces caían, la red debía poder reconfigurarse automáticamente, buscando rutas alternativas y mostrando la nueva configuración.

Para que todos los nodos de la red dispusieran de datos independientes entre sí, se optó por implementar una red de sensores donde cada nodo recogiera información ambiental de su entorno, como luminosidad y temperatura, y la transmitiera hasta un nodo coordinador encargado de recopilar todos los datos de la red. Desde este nodo coordinador los datos serían volcados a un dispositivo, como un smartphone o una tablet, donde una aplicación se encargaría de monitorizar la red y representar gráficamente su estado en cada momento.

A lo largo del proceso de diseño y desarrollo del proyecto el planteamiento didáctico inicial ha ido evolucionando, primando el potencial de las tecnologías empleadas y la utilidad práctica que una red de estas características puede ofrecer, pero sin abandonar el objetivo original.

1.2.- Objetivos

Además del fin didáctico señalado en el apartado anterior, los principales objetivos del presente proyecto son:

- Estudiar el uso, rendimiento, versatilidad y posibles aplicaciones de componentes hardware libres de bajo coste como *Arduino*.
- Profundizar en las posibilidades que ofrece la tecnología inalámbrica *ZigBee* y su uso aplicado a la implementación de una red mallada de sensores que permita una reconfiguración automática y eficiente de la misma frente a cualquier tipo de evento (entorno hostil, caídas de nodos, ingreso de nuevos nodos a la red, etc.).
- Destacar las prestaciones y versatilidad que ofrece el entorno *Android* para desarrollar aplicaciones que interactúen con tecnologías completamente independientes y la posibilidad de convertir cualquier smartphone o tablet en una potente herramienta de monitorización o control remoto de sistemas.

1.3.- Organización de la memoria

Los siguientes apartados de esta memoria se estructuran de la manera descrita a continuación:

- **Capítulo 2: Tecnologías.** Se plantea una introducción a las principales tecnologías que han sido empleadas durante la realización del proyecto, exponiendo sus principales características y los motivos por los que fueron elegidas frente a otras posibles alternativas.
- **Capítulo 3: Red de sensores.** Estudio del proceso de diseño e implementación de la red inalámbrica de sensores, así como información detallada sobre su topología, configuración, funcionamiento y características.
- **Capítulo 4: Aplicación de monitorización.** Se pormenoriza en el desarrollo y funcionamiento de la aplicación Android encargada de monitorizar la red y en el proceso de interacción entre ésta y el nodo coordinador.
- **Capítulo 5: Conclusiones.** Principales ideas que se extraen del proyecto, además de las diversas problemáticas encontradas durante su realización y algunas propuestas de posibles aplicaciones futuras que se le pueden dar al mismo.
- **Anexo A: Código y flujogramas.** Anexo que incluye los códigos íntegros de los distintos programas cargados en las placas Arduino y de la aplicación Android, así como sus correspondientes diagramas de flujo.
- **Anexo B: Bibliografía y referencias.**

2.- Tecnologías

En este apartado se estudian las distintas tecnologías que forman parte del proyecto, tales como componentes hardware, protocolos de comunicación, estándares técnicos o entornos de programación.

2.1.- Arduino

[Arduino](#)¹ es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos.

Arduino puede tomar información del entorno de toda una gama de sensores a través de sus pines de entrada y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el [lenguaje de programación Arduino](#)² (basado en [Wiring](#)³) y el entorno de desarrollo Arduino (basado en [Processing](#)⁴). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software.

Las placas pueden ser [hechas a mano](#)⁵ o [compradas](#)⁶ montadas de fábrica; el software puede [ser descargado](#)⁷ de forma gratuita. Los ficheros de diseño de referencia (CAD) están [disponibles](#)⁸ bajo una licencia abierta, así pues [permite ser adaptado según las necesidades](#)⁹.

Existe una amplia gama de placas prefabricadas que se ajustan a las distintas necesidades de cada proyecto, ofreciendo distintos tamaños, potencias, costes, modelos de microcontrolador, necesidades de alimentación, número de entradas/salidas analógicas o digitales y otras prestaciones específicas. Algunos de los modelos oficiales más utilizados son [Uno](#)¹⁰, [Leonardo](#)¹¹, [Mega ADK](#)¹², [Pro](#)¹³, [Mini](#)¹⁴, [Nano](#)¹⁵, [Fio](#)¹⁶, y un largo etc. Además existe gran variedad de clones no oficiales.

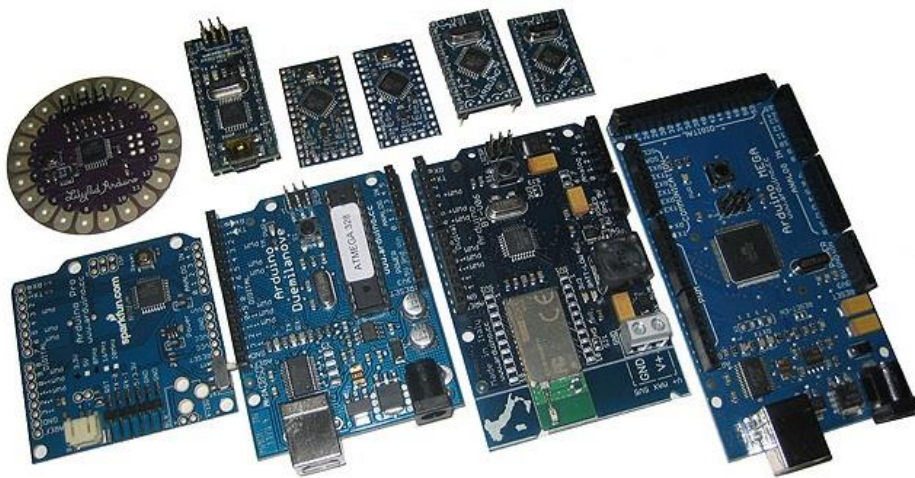


Figura 1 - Diversos modelos de placas oficiales Arduino disponibles.

El principal componente hardware de los nodos de la red es la placa *Arduino Uno*.

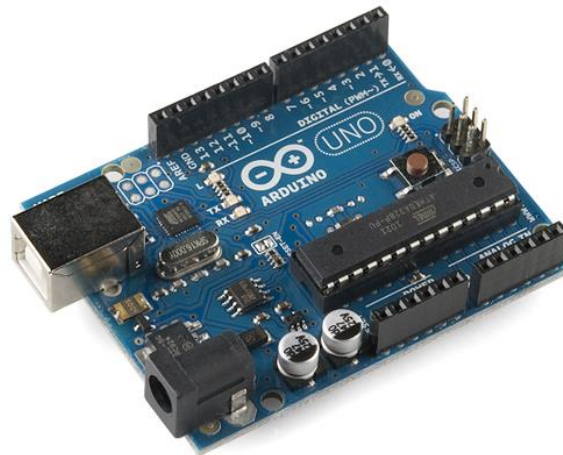


Figura 2 – Arduino Uno

Uno es el modelo de referencia de la plataforma Arduino, alcanzando un compromiso entre la sencillez de los modelos más básicos y la potencia de los más complejos. Está basado en el microcontrolador [ATmega328](#)¹⁷. Dispone de 14 pines de entrada/salida digital (de los cuales 6 pueden ser utilizados como salida PWM y 2 como puerto serie), 6 entradas analógicas, reloj cerámico de 16 MHz, puerto USB, conector de alimentación, cabezal para programación en circuito serie y un botón de reinicio, entre otros componentes.

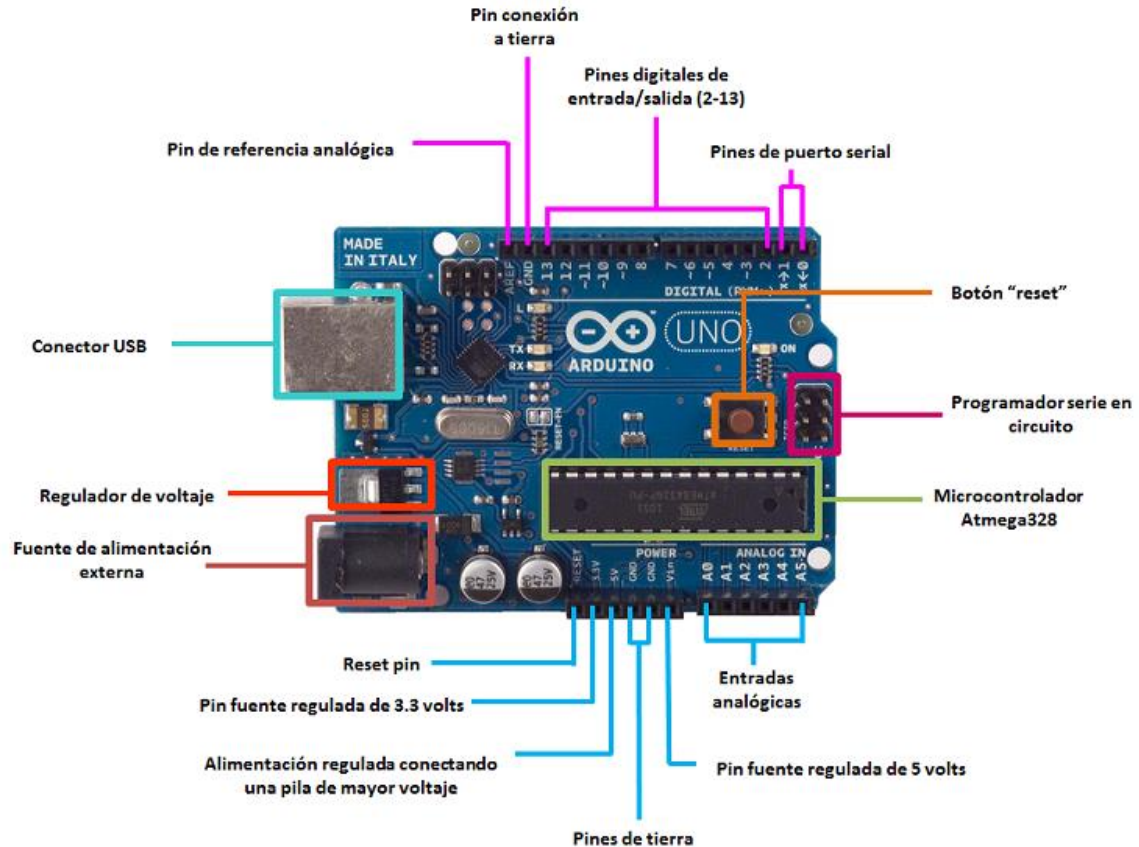


Figura 3 – Principales elementos que componen la placa Arduino Uno

Para ponerlo en funcionamiento basta con conectarlo a un PC mediante el puerto USB o alimentarlo con una batería o un adaptador AC/DC. Su voltaje de funcionamiento es de 5 voltios, admitiendo un rango de voltaje de alimentación recomendado entre 7 y 12 voltios. Proporciona dos pines de salida de alimentación de 5 y 3,3 voltios.

Posee tres unidades de almacenamiento: una memoria FLASH de 32 KB, una SRAM de 2KB y una EEPROM de 1 KB. En el siguiente [esquemático](#)¹⁸ se puede observar la disposición del circuito interno de la placa y cómo se interconectan los distintos elementos.

El modelo Uno ha experimentado varias modificaciones desde su origen, siendo la versión utilizada en este proyecto la más reciente, llamada *Revisión 3* o *R3*, que cambia la disposición en la placa de algunos pines y del botón de reset, además de proporcionar un circuito más estable de reinicio del microcontrolador.

Arduino IDE.

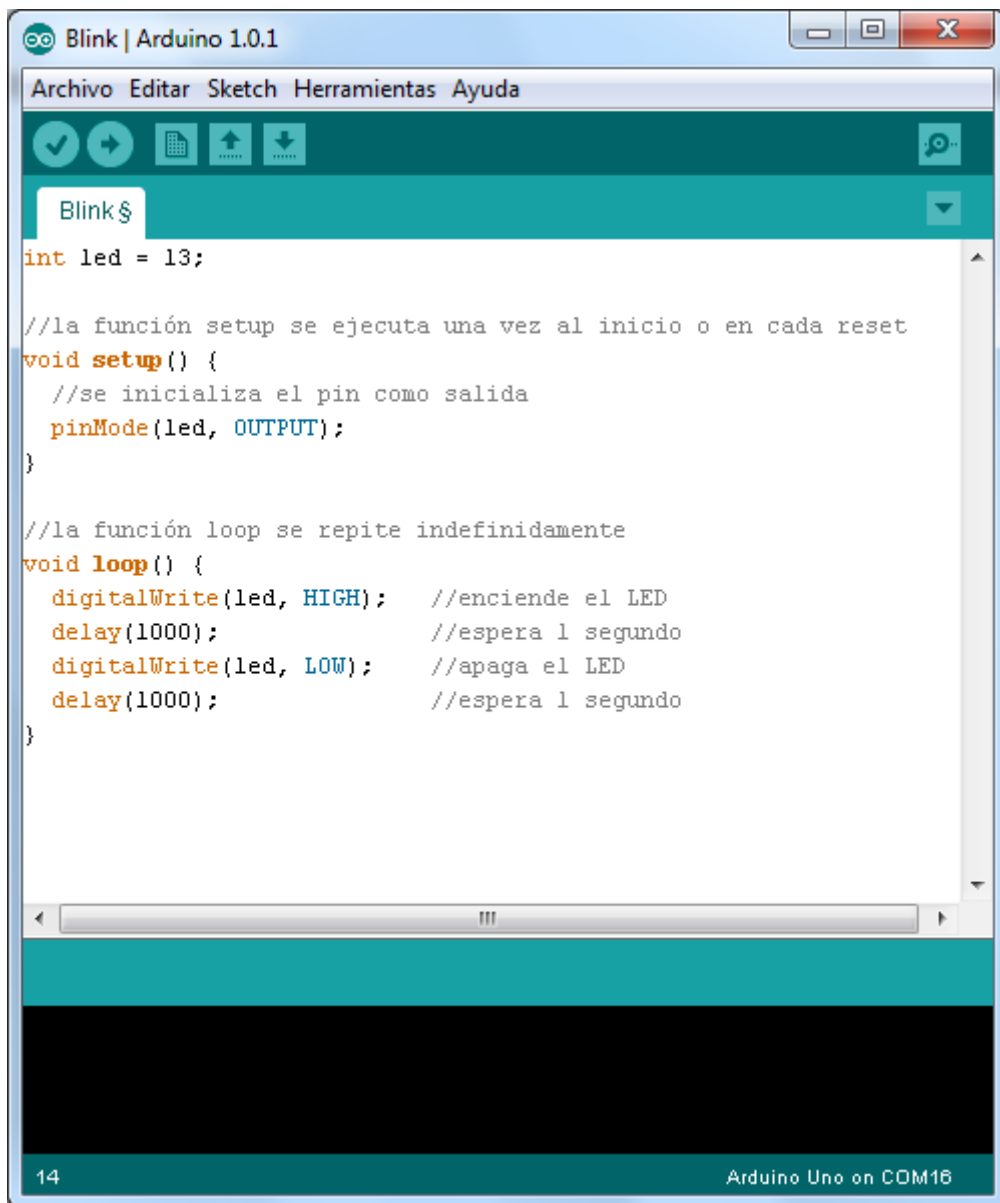
Para programar el microcontrolador se proporciona un sencillo pero eficiente entorno de desarrollo integrado o IDE que se puede descargar de forma libre desde [la web de la plataforma Arduino](#). Este software se puede utilizar en sistemas operativos Windows, Mac o Linux y proporciona herramientas como editor de código, compilador, cargador y monitor serial, además de librerías y ejemplos.

El lenguaje de programación Arduino está basado en C/C++. Los programas, denominados comúnmente *sketches*, se dividen en tres partes principales: estructura, variables y funciones. Además todo programa debe comprender dos funciones esenciales denominadas *setup()* y *loop()*.

- **setup():** esta función se carga cuando se inicia un programa o sketch. Se emplea para iniciar variables, establecer el estado de los pines, inicializar librerías, etc. Se ejecutará una única vez después de que se conecte la placa Arduino a la fuente de alimentación, o cuando se pulse el botón de reinicio de la placa.
- **loop():** esta función se ejecuta inmediatamente después de *setup()*. Se trata de un bucle infinito que se repite indefinidamente mientras la placa esté en funcionamiento y no sea reiniciada. En ella se desarrolla el código principal del programa y es la que controla el comportamiento de la placa.

En la [figura 4](#) se puede observar a modo de ejemplo un sketch básico realizado en el IDE de Arduino. Este programa hace uso del LED integrado en placa conectado al pin digital número 13 para hacerlo parpadear cada segundo. En la función *setup()* se inicializa el pin 13 como pin de salida y en *loop()* se alterna el nivel de voltaje de dicho pin entre nivel alto o bajo para encender o apagar el LED, respectivamente, con un retardo de un segundo entre cada operación.

Adicionalmente, en el [Anexo A](#) se pueden encontrar los códigos de los sketches cargados en los microcontroladores de las placas Arduino Uno que componen los distintos nodos de la red y que determinan su comportamiento.

The image shows a screenshot of the Arduino IDE interface. The window title is "Blink | Arduino 1.0.1". The menu bar includes "Archivo", "Editar", "Sketch", "Herramientas", and "Ayuda". The toolbar contains icons for opening files, saving, and running. The current sketch is named "Blink \$". The code in the editor is as follows:

```
int led = 13;

//la función setup se ejecuta una vez al inicio o en cada reset
void setup() {
  //se inicializa el pin como salida
  pinMode(led, OUTPUT);
}

//la función loop se repite indefinidamente
void loop() {
  digitalWrite(led, HIGH); //enciende el LED
  delay(1000);             //espera 1 segundo
  digitalWrite(led, LOW);  //apaga el LED
  delay(1000);             //espera 1 segundo
}
```

The status bar at the bottom shows the line number "14" and the board configuration "Arduino Uno on COM16".

Figura 4 – Programa básico realizado en el IDE de Arduino

2.2.- ZigBee

[ZigBee](#)¹⁹ es un estándar abierto para radiocomunicaciones de baja potencia basado en la especificación [802.15.4](#)²⁰ de redes inalámbricas de área personal o [WPAN](#)²¹. Incluye un robusto y confiable protocolo de red, de bajo consumo y coste, con servicios de seguridad y capa de aplicación que garantiza la interoperabilidad entre dispositivos. ZigBee trabaja a una velocidad de datos de 250 Kbps sobre la banda libre de 2,4 Ghz, aunque también soporta las bandas de 868 y 900 MHz.

El estándar ZigBee es publicado y administrado por un grupo de empresas y fabricantes denominado [ZigBee Alliance](#)²². Por tanto, el término ZigBee es en realidad una marca registrada de este grupo y no un estándar técnico. Sin embargo, para fines no comerciales, la especificación ZigBee se encuentra disponible de forma abierta y libre al público. De esta forma la relación entre IEEE 802.15.4 y [ZigBee Alliance](#) es similar a aquella entre [IEEE 802.11](#)²³ y [Wi-Fi Alliance](#)²⁴.

Una de las características más destacables de ZigBee es su soporte a redes malladas o [mesh networking](#)²⁵. En una red mallada los nodos se interconectan entre sí de forma que existen múltiples rutas posibles entre ellos. Las conexiones entre nodos se actualizan dinámicamente y se optimizan mediante sofisticadas tablas de encaminamiento integradas. Las redes malladas son descentralizadas; cada nodo es capaz de descubrir la red por sí mismo. Además, cuando un nodo abandona la red, la topología en malla permite a los demás nodos reconfigurar las rutas en función de la nueva estructura de la red. Las posibilidades de una topología mallada y del enrutamiento ad-hoc proporcionan una mayor estabilidad y adaptación a condiciones impredecibles o caída de enlaces y nodos.

Principales características de ZigBee:

- **Bajo coste:** estándar abierto que hace innecesario el pago de patentes.
- **Bajo consumo energético:** permite prolongar ampliamente la vida de las baterías.
- **Sencillez:** cuenta con una pila de protocolos de reducido tamaño.
- **Alta confiabilidad:** redes malladas con redundancia de enlaces y canales y posibilidad de enrutamiento alternativo automático.
- **Despliegue de red sencillo:** fácil de montar y administrar gracias a las redes ad-hoc y al enrutamiento automático.
- **Alta seguridad:** proporciona integridad de datos y autenticación, haciendo uso del algoritmo de cifrado AES-128.
- **Compatibilidad:** al tratarse de un estándar abierto se garantiza la interoperabilidad entre dispositivos de diferentes fabricantes.
- **Baja latencia:** los retardos producidos en las distintas tareas de red son muy reducidos, del orden de milisegundos frente a varios segundos en otras tecnologías.
- **Gran capacidad:** una misma red puede soportar más de 65000 nodos independientes.

Estas características convierten a ZigBee en la tecnología ideal para implementar una red de sensores como la que es objeto de este proyecto. En fases iniciales del diseño se barajó la posibilidad de emplear otras tecnologías como Wi-Fi o Bluetooth pero, como se puede apreciar en la figura 5, el elevado consumo de la primera y el escaso alcance y poca escalabilidad de la segunda provocaron que fuesen descartadas en favor de ZigBee.

	ZigBee	Wi-Fi	Bluetooth
Application	Monitoring and Control	Email, Web, Video	Cable replacement
Physical/ MAC layers	IEEE 802.15.4	IEEE 802.11	IEEE 802.15.1
Data Rate	250 Kbits/s	11 & 54 Mbits/sec	1 Mbits/sec
Range	10-100 meters	50-100 meters	10 meters
Networking Topology	Mesh	Point to hub	Ad-hoc, very small networks, point to point
Operating Frequency	2.4 GHz	2.4 and 5 GHz	2.4 GHz
Complexity (Device and application impact)	Low	High	High
Security	128 bit AES and Application Layer user defined	WEP, WPA, SSID	Authentication, 64-128 bits encryption
Power Consumption	low	High	Medium
Number of devices for Network	64K	32 per access point	7
Network Latency of Sleeping slave changing to active	Devices can join an existing network in under 30ms	Device connection requires 3-5 seconds	Device connection requires up to 10 seconds
Typical Applications	Industrial control and monitoring, sensor networks, games, building automation, home control and automation, toys	Wireless LAN connectivity, broadband Internet access	Wireless connectivity between devices such as phones, PDA, laptops, headsets

Figura 5 – Comparativa entre tecnologías inalámbricas

Tipos de dispositivos ZigBee.

Los nodos que forman parte de una red ZigBee se clasifican en 3 tipos:

- **Coordinador:** el coordinador es el nodo más importante de la red. puesto que es el encargado de crearla. Por este motivo, sólo puede existir un único coordinador por red. Además, el coordinador se encarga de asignar direcciones a los nodos que se van uniendo a la red, mantiene la seguridad, administra el sistema y actúa de enlace con otras redes u otros dispositivos.
- **Router:** un router es un nodo ZigBee plenamente operativo. Puede unirse a redes existentes, transmitir, recibir y enrutar información. Actúa como intermediario entre nodos que no pueden comunicarse directamente entre sí, ampliando el alcance efectivo de la red. Además puede permitir a otros nodos el ingreso en la red. Su consumo es mayor que el de los terminales puesto que, al tener responsabilidades de enrutamiento, no puede desconectarse en ningún momento.
- **Terminal:** un terminal es un nodo provisto con la funcionalidad básica para unirse a la red y enviar y recibir información. No puede actuar como intermediario entre otros nodos. Necesita que un router o el coordinador sea su nodo padre y le permita el acceso a la red. Puede entrar de forma cíclica en un modo de bajo consumo para ahorrar energía.

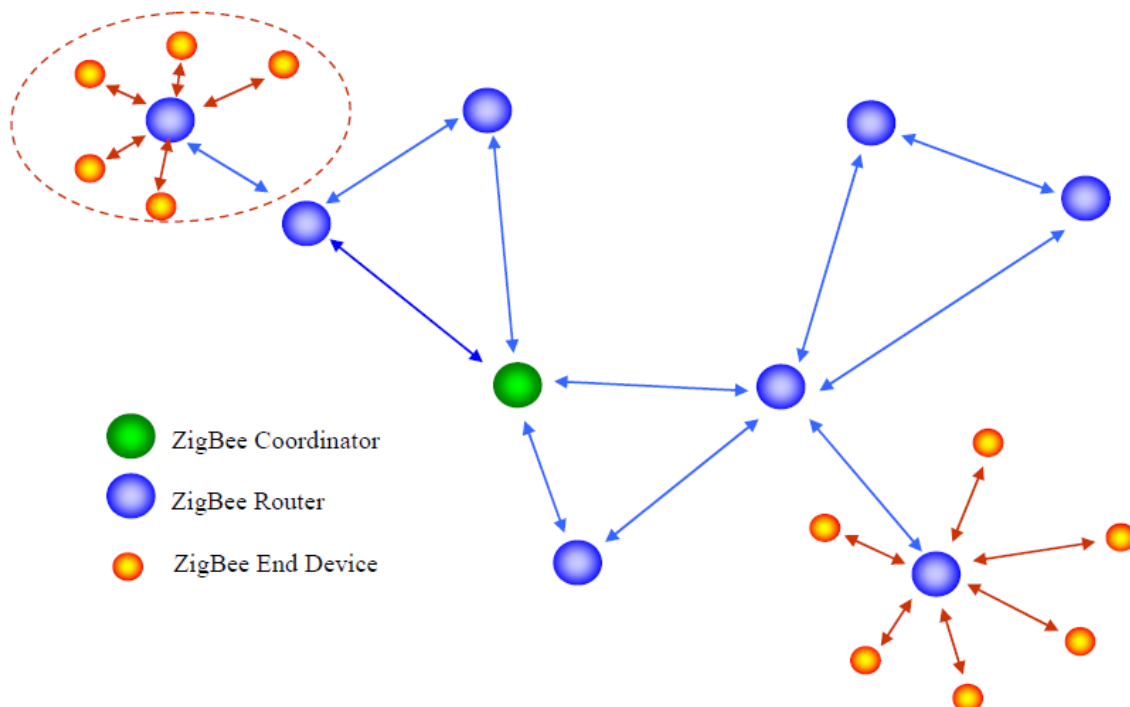


Figura 6 – Ejemplo de topología típica de una red ZigBee

Direccionamiento.

Cada dispositivo ZigBee posee una dirección única, exclusiva e inmutable de 64 bits denominada **dirección MAC** que le distingue de cualquier otro dispositivo a nivel mundial. Adicionalmente también dispone de una **dirección de red** de 16 bits que le es asignada dinámicamente por el coordinador al asociarse a la red. Dentro de una PAN cualquier dispositivo puede ser identificado unívocamente por cualquiera de estas dos direcciones. Por último, también se le puede asignar arbitrariamente un nombre identificativo formado por una cadena de caracteres, pero este nombre no garantiza su identificación inequívoca dentro de una red.

De igual modo, cada red ZigBee posee dos identificadores que permiten distinguirla de cualquier otra red de su entorno, incluso de las que operan en su mismo canal, permitiendo su coexistencia en un mismo espacio. El primero de estos identificadores se denomina **PAN ID** y es un valor de 16 bits que se establece al crearse la red. El segundo es el llamado **Extended PAN ID**, un valor de 64 bits asignado arbitrariamente por el administrador de la red y que es añadido, junto al PAN ID, en los paquetes *beacon* de señalización para permitir que los nodos que deseen asociarse puedan identificar de forma unívoca la red.

Además, dos redes ZigBee que compartan un mismo espacio se pueden diferenciar según la frecuencia del **canal** sobre el que trabajan. Cuando el nodo coordinador crea la red sondea los canales disponibles en la banda utilizada (2,4 GHz) y elige el más adecuado. Todos los nodos que se asocian a la red deben usar este mismo canal. Por defecto, los dispositivos ZigBee se encargan automáticamente de seleccionar el canal adecuado.

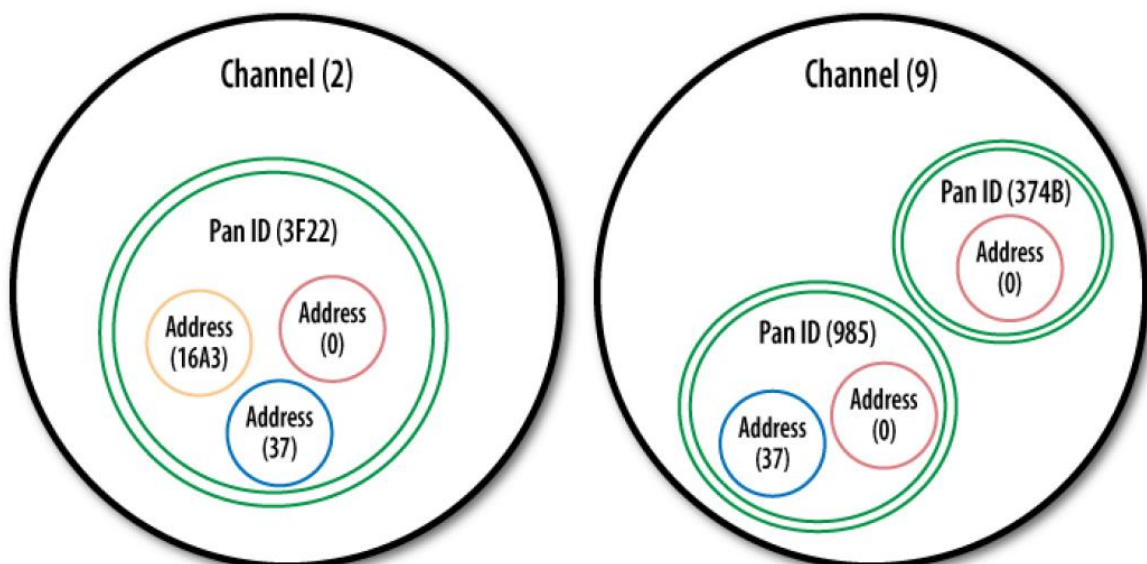


Figura 7 – Diagrama de Venn que muestra tres redes ZigBee coexistiendo en un mismo espacio gracias a que pueden ser diferenciadas unívocamente por su PAN ID y el canal de transmisión. De igual modo, los nodos que las constituyen pueden tener la misma dirección de red que los nodos de las otras redes, puesto que es un valor local a la PAN.

Seguridad.

La tecnología ZigBee garantiza la seguridad de las comunicaciones mediante el mecanismo [Advanced Encryption Standard](#)²⁶ o *AES*. Se trata de un fiable, rápido y eficiente algoritmo de cifrado en bloque que utiliza claves de 128 bits y que proporciona confidencialidad de datos, integridad en las conexiones, autenticación de los elementos del sistema y protección a la red frente a ataques malintencionados.

ZigBee soporta dos modos de manejo de claves:

- **Claves de red:** proporcionan seguridad a las comunicaciones entre nodos. Cada paquete es cifrado y enviado al siguiente nodo de la red donde es descifrado, cifrado de nuevo y retransmitido al siguiente nodo en ruta hasta su destino.
- **Claves de enlace:** proporcionan seguridad extremo a extremo. El origen del paquete lo cifra, y éste permanece cifrado mientras se transmite por la red hasta que llega a su destino donde es descifrado. Es conveniente usar este tipo de claves en situaciones donde alguno de los nodos o enlaces intermedios de la red no es completamente confiable.

XBee.

[XBee](#)²⁷ es el hardware ZigBee utilizado en el proyecto. Se trata de una gama de módulos de radio fabricados por [Digi International](#)²⁸ con una amplia variedad de modelos, componentes, firmware, potencias de transmisión y antenas. Trabajan con un voltaje de operación de 3,3 voltios y, aunque incorporan hasta 20 pines para distintas operaciones como reinicio o modo de bajo consumo, pueden funcionar haciendo uso tan sólo de sus 4 pines principales: alimentación, tierra, y entrada y salida de datos.

Cuentan con un microchip que, además de ocuparse de las tareas propias de la comunicación radio, les permite realizar operaciones con un nivel de lógica muy básico, como por ejemplo leer directamente datos de sensores y retransmitirlos sin tener que contar con procesamiento externo. Para lógica más avanzada deben conectarse a otro dispositivo que se encargue de ello, como un microcontrolador más complejo.



Figura 8 – Módulo de radio XBee Serie 2 de Digi International

Los módulos XBee se clasifican principalmente en dos tipos:

- **Serie 1:** esta serie de módulos XBee proporciona soporte para comunicaciones punto a punto de forma simple y basándose en el estándar 802.15.4. Su principal ventaja es la sencillez.
- **Serie 2:** estos módulos permiten implementar redes malladas siguiendo el protocolo ZigBee. Además poseen un mayor alcance y un menor consumo que la serie 1. Estas características convierten a los módulos de la serie 2 en los idóneos para implementar una red de sensores como la que ocupa este proyecto.

Cabe destacar que los módulos XBee de la serie 1 y la serie 2 son **incompatibles entre sí**, y no pueden interactuar entre ellos de ninguna forma. Los módulos pertenecientes a una serie sólo se pueden comunicar con otros módulos de la misma serie.

	Series 1	Series 2
Typical (indoor/urban) range	30 meters	40 meters
Best (line of sight) range	100 meters	120 meters
Transmit/Receive current	45/50 mA	40/40 mA
Firmware (typical)	802.15.4 point-to-point	ZB ZigBee mesh
Digital input/output pins	8 (plus 1 input-only)	11
Analog input pins	7	4
Analog (PWM) output pins	2	None
Low power, low bandwidth, low cost, addressable, standardized, small, popular	Yes	Yes
Interoperable mesh routing, ad hoc network creation, self-healing networks	No	Yes
Point-to-point, star topologies	Yes	Yes
Mesh, cluster tree topologies	No	Yes
Single firmware for all modes	Yes	No
Requires coordinator node	No	Yes
Point-to-point configuration	Simple	More involved
Standards-based networking	Yes	Yes
Standards-based applications	No	Yes
Underlying chipset	Freescale	Ember
Firmware available	802.15.4 (IEEE standard), DigiMesh (proprietary)	ZB (ZigBee 2007), ZNet 2.5 (obsolete)

Figura 9 – Comparativa entre la serie 1 y la serie 2 de módulos de radio XBee

Las radios XBee presentan dos modos de funcionamiento, denominados modos *AT* y *API*.

- **Modo AT:** se trata de un modo transparente donde la información recibida por el pin de entrada de datos es retransmitida por radio sin ninguna modificación. Los datos pueden ser enviados a un único destinatario (punto a punto) o bien pueden ser transmitidos a múltiples destinos (broadcast).
- **Modo API:** en este modo, los datos son encapsulados dentro de una trama que proporciona varias funcionalidades adicionales, tales como direccionamiento, confirmación de paquetes y checksum, entre otras. Existen varios tipos de tramas API y cada una ofrece distintas prestaciones: comandos de configuración, transmisión de datos, reconocimiento de mensajes, etc.

Todas las radios XBee empleadas en el proyecto funcionan bajo el modo API, que les ofrece las funcionalidades necesarias para realizar algunas de las operaciones más complejas, como la reconfiguración automática de la red, que no serían posibles en el modo AT.

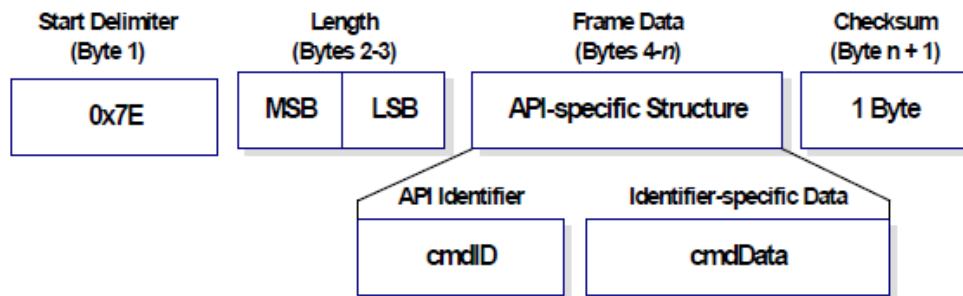


Figura 10 – Estructura de una trama API básica

Para programar los módulos XBee se debe cargar el firmware deseado y modificar el valor de los distintos parámetros que determinan su comportamiento. Para ello se utiliza un software distribuido por Digi, denominado [X-CTU](#)²⁹ y que se puede descargar de forma gratuita desde [su página web](#)³⁰. Este programa, disponible sólo para entornos Windows, permite actualizar el firmware del módulo, determinar su comportamiento como coordinador, router o terminal, leer la configuración actual, realizar pruebas de alcance o facilitar el acceso a las funcionalidades del modo API, entre otras posibilidades. Basta con conectar el módulo a un ordenador haciendo uso de un adaptador USB¹ y usar el software para programarlo con la configuración deseada.

A pesar de que la única forma de actualizar el firmware de un módulo XBee es mediante el software X-CTU, para modificar otras configuraciones se puede hacer uso de cualquier otro programa terminal serie, entrar en modo comando y reconfigurar el módulo mediante el envío de [comandos AT](#)³¹.

¹ En caso de no disponer de un adaptador USB específico, se puede conectar el módulo XBee a una placa Arduino a la que previamente se ha borrado su programación o retirado el microcontrolador, y conectar dicha placa al ordenador mediante su puerto USB. De esta forma, la placa Arduino actúa como adaptador entre PC y XBee, permitiendo la configuración de éste último mediante el programa X-CTU.

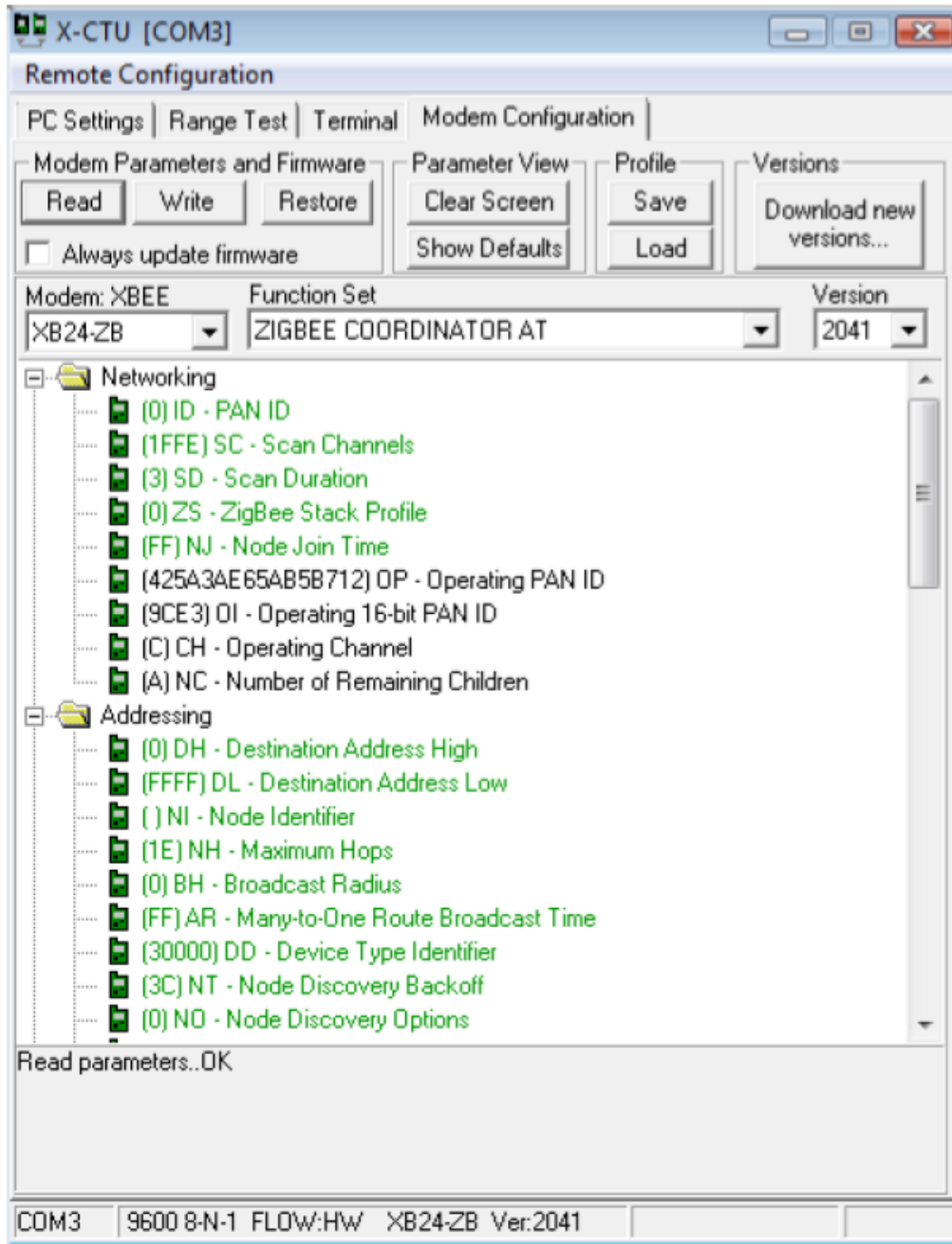


Figura 11 – Software X-CTU mostrando la configuración leída de un nodo coordinador en modo AT

2.3.- Bluetooth

[Bluetooth](#)³² es una tecnología de comunicaciones inalámbricas de corto alcance, segura, simple y ampliamente extendida que soporta canales de voz y datos. Su principal objetivo es reemplazar las conexiones cableadas entre dispositivos conservando un alto grado de seguridad. Está específicamente diseñado para que sus transeptores sean pequeños, baratos y de bajo consumo, pudiendo ser integrados en cualquier dispositivo.

Bluetooth fue especificado en el estándar [802.15.1](#)³³ que a día de hoy no se sigue manteniendo. En su lugar, un grupo de empresas llamado [Bluetooth Special Interest Group](#)³⁴, conocido también como *Bluetooth SIG*, se encarga de administrar el estándar y del desarrollo de la especificación.

La tecnología Bluetooth opera sobre la banda de frecuencias libre de 2,4 GHz utilizando 79 canales de 1 MHz. Emplea la técnica de espectro ensanchado por salto en frecuencia (*Frequency Hopping Spread Spectrum* o [FHSS](#)³⁵) que produce un cambio en la frecuencia de transmisión cada 625 microsegundos, es decir, 1600 saltos por segundo. De esta forma se minimizan las interferencias con otras comunicaciones inalámbricas que operan en la misma banda de frecuencias.

Una comunicación Bluetooth ocurre siempre entre un dispositivo maestro y uno esclavo, que previamente han negociado las características de la conexión en un proceso denominado vinculación o *pairing*. Los esclavos pueden comunicarse únicamente con el maestro y sólo cuando éste se lo indique. Sin embargo, los dispositivos Bluetooth son simétricos y pueden actuar como maestro o esclavo indistintamente. Esto permite formar redes ad-hoc que se adaptan a las necesidades de la comunicación.

Los dispositivos Bluetooth forman redes denominadas *piconets*. Cada piconet consta de un dispositivo maestro y hasta 7 esclavos. Todos los dispositivos de una piconet comparten la misma secuencia de salto en frecuencia, que suele denominarse canal FH o *Frequency Hopping*. Los maestros se encargan de establecer la red y de controlar el canal FH, determinando los instantes de salto en función de su reloj local. Los esclavos siguen la secuencia que les marca el maestro, de forma que todos comparten la misma. Cada piconet utiliza una secuencia de ensanchado distinta, permitiendo así la coexistencia de múltiples redes en un mismo espacio.

Existen tres clases de módulos Bluetooth en función de su alcance:

- **Clase 1:** hasta 100 metros. Usados principalmente en entornos industriales
- **Clase 2:** hasta 10 metros. Los más comúnmente extendidos en dispositivos móviles.
- **Clase 3:** hasta 1 metro. Su reducido alcance limita sus posibles aplicaciones.

El uso de Bluetooth está muy extendido, lo que se convierte en su principal ventaja. Su presencia en la inmensa mayoría de dispositivos móviles como smartphones, tablets e infinidad de otros dispositivos, convierte a Bluetooth en una de las tecnologías inalámbricas de mayor compatibilidad.

2.4.- Android

[Android](#)³⁶ es un sistema operativo de código abierto basado en [Linux](#)³⁷ y distribuido por [Google](#)³⁸ bajo licencia [Apache](#)³⁹. Está diseñado principalmente para dispositivos móviles de pantalla táctil, como smartphones o tablets. Su naturaleza de código abierto y lo permisivo de su licencia hace posible que el software basado en Android sea libremente creado, modificado y distribuido por cualquier desarrollador. Se facilita completo acceso al kit de desarrollo de software o [SDK](#)⁴⁰ que proporciona todas las herramientas necesarias para implementar, probar y depurar aplicaciones, junto con prestaciones adicionales como librerías API, documentación, emulador, códigos de ejemplo y tutoriales. Además, se permite a los desarrolladores la libre publicación de sus creaciones, facilitando su distribución a través de la plataforma [Google Play](#)⁴¹.

Los programas en entorno Android, denominados comúnmente aplicaciones o *apps*, se implementan en una versión personalizada del lenguaje de programación [Java](#)⁴². Para ello se puede hacer uso de la plataforma libre [Eclipse](#)⁴³ que proporciona entornos de desarrollo integrado tanto para Java como para Android. Además, recientemente ha sido publicado por Google un entorno de desarrollo específico de Android, denominado [Android Studio](#)⁴⁴, disponible de forma gratuita desde su web. Debido a que Android Studio se encuentra todavía en fases muy iniciales de su desarrollo, todo el software Android implementado en este proyecto se ha desarrollado mediante la plataforma Eclipse.

Arquitectura por capas.

El sistema operativo Android está conformado por una estructura software dividida en diversos niveles. Cada capa de esta arquitectura proporciona servicios a su capa inmediatamente superior.

- **Núcleo Linux:** todo el sistema Android se construye sobre un núcleo Linux personalizado por Google. Este nivel se encarga de interactuar con el hardware, proporcionando los controladores necesarios. Además actúa como capa de abstracción entre el hardware y el resto de capas software de la arquitectura.
- **Librerías y Android Runtime:** la siguiente capa corresponde a las librerías nativas de Android, en lenguaje C/C++. Al mismo nivel se encuentra la capa de ejecución denominada *Android Runtime*, que consta de la máquina virtual *Dalvik*, encargada de ejecutar las aplicaciones, y un conjunto de librerías Java.
- **Marco de Aplicación:** por encima de los niveles de librerías y ejecución se encuentra la capa marco de aplicación o *framework*. Esta capa gestiona funciones básicas del dispositivo y proporciona las herramientas sobre las que se construyen las aplicaciones.
- **Aplicaciones:** esta es la capa superior de la arquitectura Android. Incluye todas las aplicaciones, desde las incorporadas por defecto en Android como las desarrolladas por terceros e instaladas por el usuario.

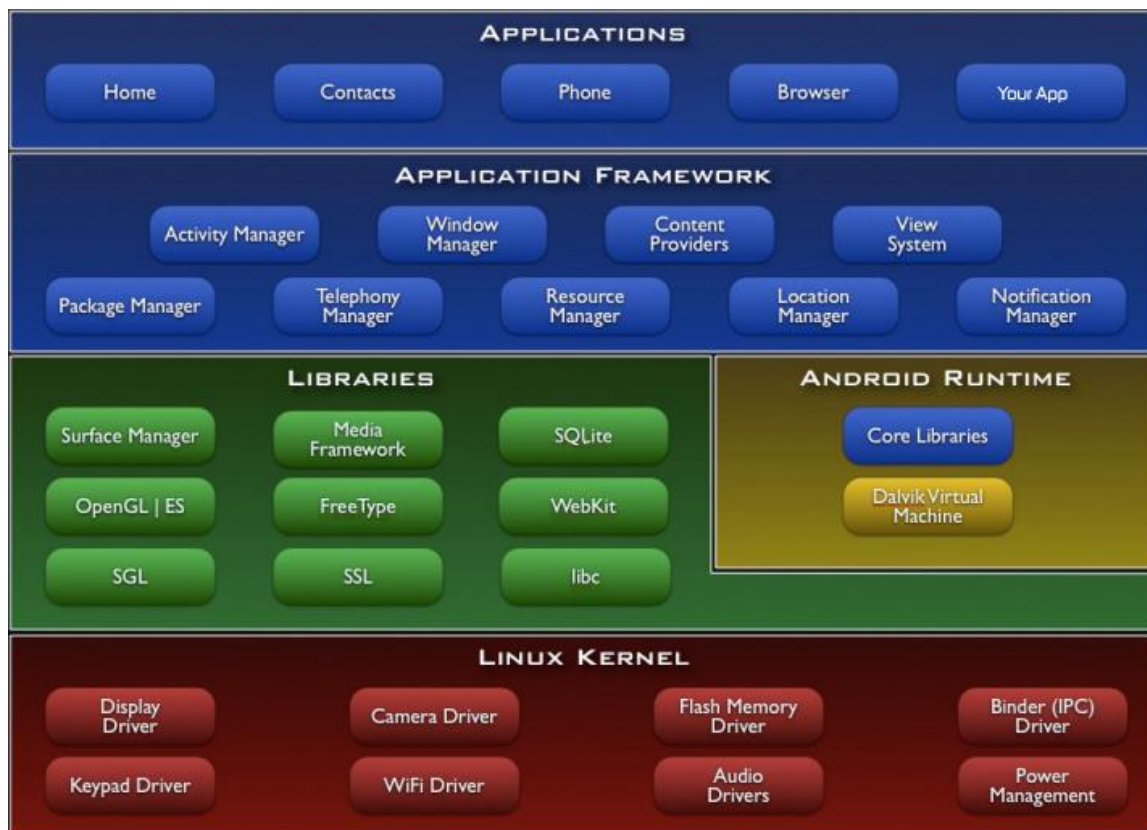


Figura 12 – Arquitectura en capas del sistema Android.

Actividades.

Una actividad es un componente de una aplicación Android que proporciona una interfaz con la que el usuario puede interactuar. Cada actividad dispone de un espacio en el que representar su interfaz. Normalmente este espacio ocupa toda la pantalla del dispositivo, aunque puede ser más pequeño o situarse por encima de otras actividades.

Una aplicación puede constar de varias actividades vinculadas entre sí. Una de estas actividades es la principal, que es presentada al usuario en el momento de lanzar la aplicación. Cada actividad puede iniciar otras actividades para realizar distintas acciones. Cuando se inicia una nueva actividad la anterior es detenida, pero preservada en una pila de actividades. Se trata de una pila *LIFO* (Last In, First Out) que permite volver a la actividad inmediatamente anterior pulsando el botón “Atrás” del dispositivo.

Durante la navegación del usuario por la aplicación, las actividades transitan entre distintos estados en lo que se denomina el [ciclo de vida de actividad](#)⁴⁵. Cuando una actividad cambia de un estado a otro, el sistema llama a determinados métodos que determinan su comportamiento al transitar a ese nuevo estado, realizando las acciones correspondientes como por ejemplo iniciar variables en caso de crearse una nueva actividad o liberar recursos en caso de detenerse. Estos métodos pueden ser modificados por los desarrolladores, personalizando su comportamiento o incluyendo acciones adicionales.

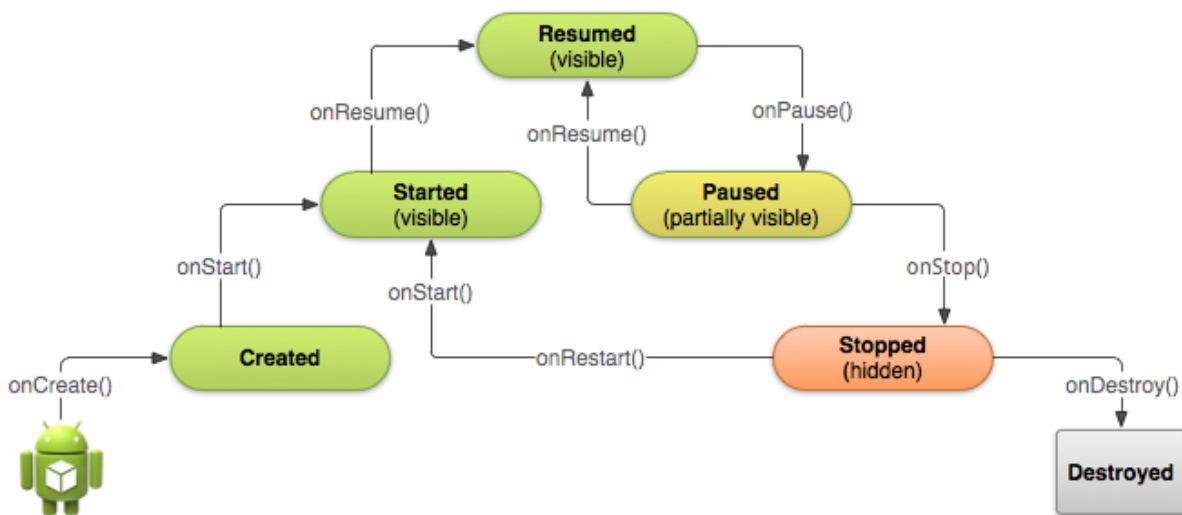


Figura 13 – Representación del ciclo de vida de una actividad Android. Se muestran los posibles estados de la actividad y los métodos que el sistema utiliza para transitar entre ellos.

Como se puede observar en la figura 13, una actividad puede encontrarse en uno de cinco estados posibles. Sin embargo sólo tres de estos estados son estáticos, es decir, la actividad puede permanecer en ellos por un periodo de tiempo prolongado. Estos estados son:

- **Ejecutándose (Resumed):** en este estado la actividad se encuentra corriendo en primer plano y el usuario puede interactuar con ella. Si la actividad pasa a segundo plano, el sistema llama al método *onPause()*.
- **Pausada (Paused):** la actividad se encuentra en segundo plano, oculta parcialmente por otra actividad que no ocupa toda la pantalla. La actividad pausada no ejecuta ningún código y no puede interactuar con el usuario. Si vuelve a primer plano se llama al método *onResume()* y se vuelve al estado de ejecución. Si por el contrario la actividad pasa por completo a segundo plano, el sistema llama a *onStop()* y transita al siguiente estado.
- **Detenida (Stopped):** la actividad está completamente oculta en segundo plano y no es visible por el usuario. Se conserva su instancia y toda la información correspondiente, pero no puede ejecutar código. El sistema llama al método *onRestart()* si la actividad vuelve a primer plano, recuperando la información guardada. Si la actividad es cerrada por completo, se ejecuta el método *onDestroy()*.

Los otros dos estados, *Creada (Created)* e *Iniciada (Started)*, son transitorios y el sistema avanza rápidamente por ellos para pasar al estado de ejecución.

AndroidManifest

Cada aplicación debe tener un archivo llamado [AndroidManifest.xml](#)⁴⁶, con ese exacto nombre, en su directorio raíz. Se trata de un archivo XML que, mediante el uso de un sistema de etiquetas y atributos, describe las características fundamentales de la aplicación y define cada uno de sus componentes. El sistema Android debe disponer de esta información antes de poder ejecutar cualquier código de la aplicación. Algunas de las funciones que lleva a cabo este archivo son:

- Dar nombre al paquete Java. Este nombre identifica unívocamente la aplicación.
- Describe los distintos componentes de la aplicación: actividades, servicios, clases, etc.
- Determina los permisos que autorizan a la aplicación para acceder a los distintos recursos del dispositivo.
- Establece la versión mínima del sistema Android que requiere la aplicación.

En el [Anexo A5](#) del presente documento se puede observar el archivo AndroidManifest de la aplicación de monitorización desarrollada en este proyecto.

Layouts

Un [layout](#)⁴⁷ define la estructura visual de la interfaz de usuario de la aplicación. Puede ser creada de dos formas: mediante la declaración de los elementos de la interfaz en un archivo XML o mediante la instanciación de los componentes del layout durante la ejecución de la aplicación. También se pueden emplear libremente estos dos métodos en combinación para conseguir los resultados deseados.

En el proyecto se usan principalmente dos layouts: uno para la representación gráfica de la red y otro para presentar los datos en modo texto. Ambos layouts se integran dentro de una interfaz principal mediante una estructura de vistas denominada [ViewFlipper](#)⁴⁸ que permite cambiar de una a otra con sólo pulsar un botón.

Los principales elementos de que constan los layouts desarrollados son:

- [TextView](#)⁴⁹: son campos de texto que presentan la información recibida de la red.
- [ImageView](#)⁵⁰: estos elementos permiten mostrar una imagen por pantalla.
- [Drawable](#)⁵¹: se trata de un recurso que referencia una imagen para poder ser mostrada dentro de un ImageView
- [Button](#)⁵²: botones que, al ser pulsados, permiten que el programa ejecute un código determinado.

3.- Red de Sensores

En este capítulo se estudia el proceso llevado a cabo durante las fases de diseño e implementación de la red de sensores, así como sus principales características.

3.1.- Topología

El sistema implementado en el proyecto consiste en una red mallada de nodos que se comunican entre sí mediante la tecnología ZigBee. Consta de doce nodos: un coordinador, cuatro routers y siete terminales, aunque está diseñada para ser escalable hasta sesenta nodos. Cada uno de estos nodos tiene funciones distintas:

- **Coordinador:** crea la red y acepta las uniones de los demás nodos. Es el destino final de todas las transmisiones dentro de la red, puesto que se encarga de recopilar los datos de todos los nodos. Cuenta con sus propios sensores que le permiten medir sus correspondientes datos ambientales. Además puede conectarse mediante Bluetooth a un dispositivo Android y enviarle los datos de la red para su monitorización.
- **Routers:** leen los datos de sus sensores y los envían al coordinador. Además pueden actuar como intermediarios para los terminales que se encuentran fuera del alcance del coordinador, retransmitiendo la información que éstos les envían.
- **Terminales:** además de recoger información de sus sensores, son capaces de identificar a su nodo padre, es decir, el router que actúa de intermediario entre el coordinador y ellos, e incluir dicha información en el mensaje transmitido para que pueda ser identificada la ruta que dicho mensaje ha seguido dentro de la red.

Si bien la naturaleza ad-hoc de la tecnología de radiocomunicación empleada permite un amplio grado de libertad en cuanto al despliegue de la red, para facilitar su posterior monitorización se establece una disposición fija de los nodos que hace posible la identificación inequívoca de la ruta que ha seguido cada uno de los mensajes que atraviesa la red. Esta disposición es la siguiente:

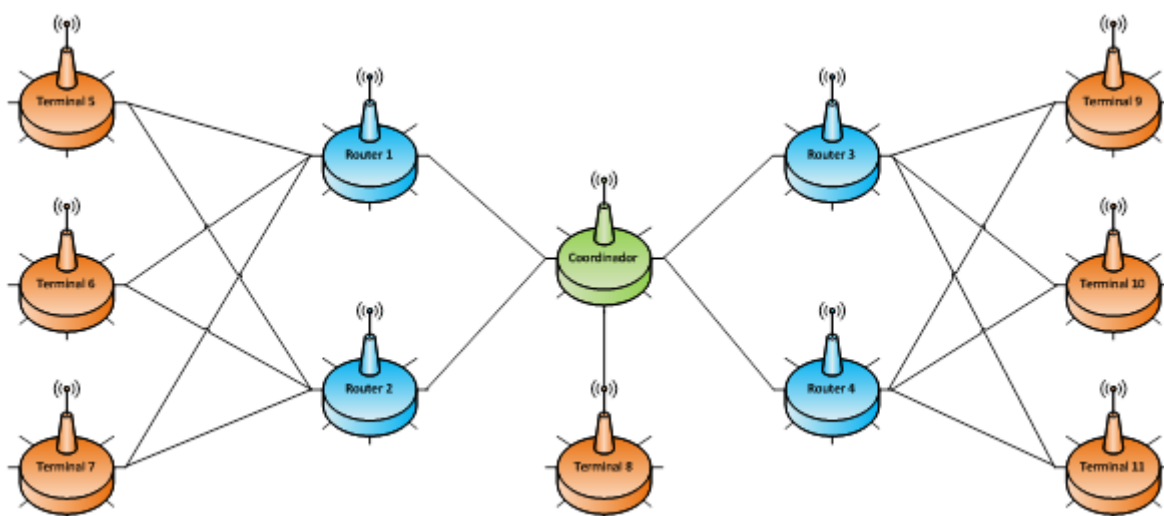


Figura 14 – Topología de la red de sensores mostrando todos los posibles enlaces entre los distintos nodos.

3.2.- Hardware

Cada nodo de la red, independientemente de su rol dentro de la misma, consta de los siguientes componentes hardware:

- **Placa Arduino Uno:** es el principal elemento del nodo, al que se conectan todos los demás componentes. Es su microcontrolador quien determina el comportamiento del nodo. Las salidas de los sensores se conectan a las entradas analógicas de la placa, permitiéndole leer sus datos. Además, mediante sus dos salidas de alimentación de 3,3 y 5 voltios, se encarga de suministrar el voltaje de operación adecuado a cada componente hardware.
- **Módulo XBee serie 2:** es el encargado de la comunicación inalámbrica con los demás nodos e implementa el protocolo ZigBee, que permite la conexión y reconfiguración automática de la red. Dependiendo de su configuración actuará como coordinador, router o terminal.
- **XBee Shield**⁵³: se trata de un adaptador especialmente diseñado para conectar una placa Arduino y un módulo XBee. Realiza todas las conexiones necesarias entre los pines de ambos elementos y dota de estabilidad al conjunto. Proporciona fácil acceso a los conectores libres de la placa, junto con un botón adicional de reinicio del microcontrolador. Dispone además de un interruptor que permite seleccionar la salida serie del módulo XBee entre radio o USB, lo que facilita su configuración.
- **Sensor de temperatura TMP36**⁵⁴: se trata de un pequeño dispositivo con tres patillas: alimentación, tierra y una salida de voltaje que varía en función de la temperatura. La placa Arduino lee este valor de voltaje por la entrada analógica A0 y lo convierte en grados Celsius mediante la fórmula:

$$\text{Temperatura } (C^{\circ}) = [(V_{out} * \text{Alimentación} * 1000) / 1023] - 500) / 10$$

- **Fotorresistencia LDR**⁵⁵: una fotorresistencia es un elemento cuya resistividad eléctrica varía en función de la luminosidad que detecta. Conectada al divisor de tensión de la figura 15 actúa como un sensor de luz. Mediante el pin analógico A1 la placa lee la salida V_{out} , que varía en función de la luz presente, y traduce este valor de voltaje a la magnitud de luminosidad Lux mediante la fórmula:

$$\text{Luminosidad } (Lux) = V_{out} * 10000 / 1023$$

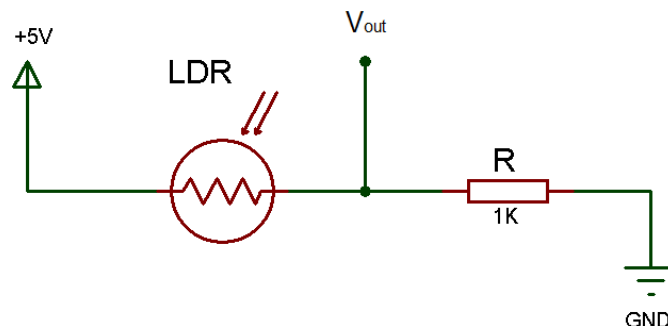


Figura 15 – Divisor de tensión que permite a la fotorresistencia LDR actuar como un sensor de luz.

- **Pila de 9 voltios:** alimenta a la placa Arduino, proporcionándole una tensión de 9 voltios que entra dentro de su rango recomendado (7-12 voltios). A su vez, la placa alimenta al resto de elementos del nodo a través de sus dos salidas de alimentación reguladas a 3,3 y 5 voltios. La pila convierte al nodo en un elemento completamente inalámbrico.
- **Módulo Bluetooth [JY-MCU](#)⁵⁶:** presente únicamente en el nodo coordinador, permite mantener una comunicación Bluetooth entre éste y el dispositivo Android encargado de la monitorización. El pin TX del módulo se conecta al pin digital número 10 de la placa, y el RX al 11. Se estudiará en detalle esta comunicación en el [capítulo 4](#).

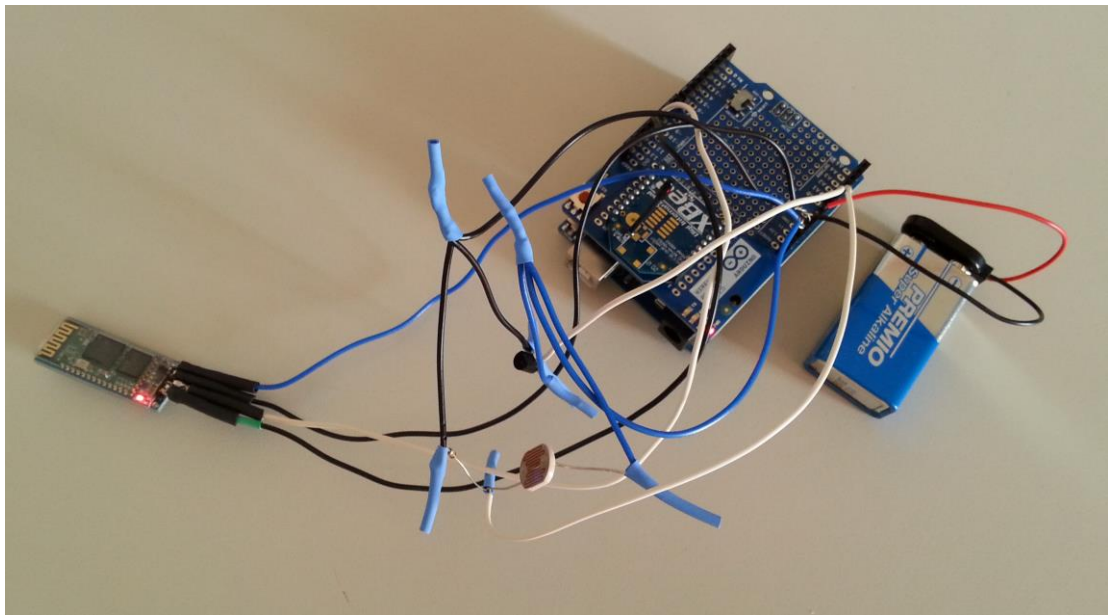


Figura 16 – Nodo coordinador de la red. De izquierda a derecha: módulo Bluetooth, sensores, módulo XBee montado sobre placa Arduino Uno, y pila de 9V.

3.3.- Configuración

Una vez interconectados correctamente los distintos componentes hardware, para que el nodo presente el comportamiento deseado es necesario configurar mediante software dos de estos elementos:

En primer lugar es necesario programar el microcontrolador de la placa mediante el [IDE Arduino](#), conectando el puerto USB de la placa a un PC y cargando el sketch correspondiente. En la realización del proyecto se han desarrollado tres sketches distintos para los tres tipos de nodo presentes en la red, que dotan a la placa del comportamiento adecuado en cada caso. En el [Anexo A](#) de esta memoria se proporciona los códigos en lenguaje Arduino de estos sketches, junto con sus correspondientes diagramas de flujo.

Por otro lado, también es necesario configurar el módulo XBee mediante la utilización del software [X-CTU](#). Para conectar el módulo al ordenador se puede hacer uso de la placa Arduino a modo de adaptador, borrando previamente el programa cargado en el microcontrolador y cambiando el interruptor del shield a la posición USB. De este modo el módulo se comunica directamente con el PC y se puede cargar en él el firmware adecuado y configurar una serie de parámetros, también denominados [comandos AT](#), que determinan su comportamiento y las características de la comunicación.

Si bien estos parámetros pueden modificarse mediante la ejecución de comandos AT desde cualquier programa terminal serie, para cargar un nuevo firmware en un módulo XBee es necesario el uso del programa X-CTU.

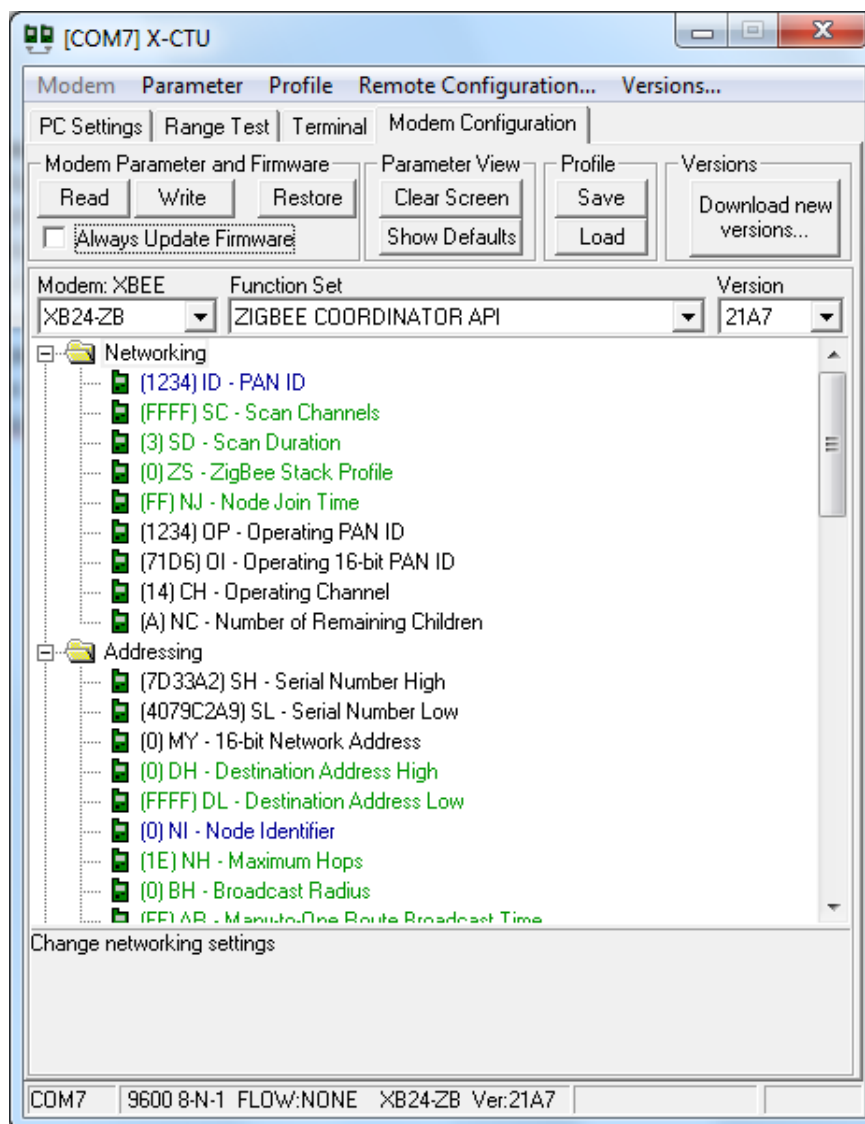


Figura 17 - Captura del software X-CTU mostrando la configuración cargada en el nodo coordinador. Los menús desplegables permiten elegir el modelo de radio XBee, su función como coordinador, router o terminal tanto en modo AT como API, y la versión del firmware. En la ventana inferior se pueden configurar los parámetros que determinan el comportamiento del módulo en lo referente a la creación de la red, el direccionamiento, la seguridad, la potencia de transmisión, etc.

Algunos de los parámetros más importantes, y que deben ser configurados en todos los módulos XBee que formen parte de la red, son:

- **ID (*Identity*):** valor de 64 bits que determina el identificador de la red (*Extended PAN ID*) a la que pertenece el nodo. La red del proyecto tiene como identificador el valor 1234.
- **NI (*Node Identity*):** cadena de caracteres usada como identificador del nodo. El microcontrolador determina dinámicamente la identidad del nodo y el origen de los mensajes a partir de este valor. Cada nodo de la red está identificado por un carácter alfanumérico: 0 para el coordinador, del 1 al 9 para los routers y una letra ASCII para los terminales.
- **DH (*Destination High*):** valor que representa los 32 bits más significativos de la dirección de 64 bits del nodo destino de los mensajes transmitidos. Siempre debe ser 0x00000000.
- **DL (*Destination Low*):** este valor son los 32 bits menos significativos de la dirección de 64 bits del nodo destino de los mensajes transmitidos. Para el coordinador este valor debe ser la dirección *broadcast* (0x0000FFFF), y para todos los demás nodos debe ser la dirección por defecto del coordinador (0x00000000).
- **EE (*Encryption Enable*):** activa (1) o desactiva (0) el algoritmo de cifrado *AES* para dotar de seguridad las transmisiones del nodo. Los nodos de la red implementada tienen activado el cifrado *AES*.
- **NK (*Netwrok Key*):** clave de red de hasta 128 bits utilizada para cifrar y descifrar los mensajes protegidos mediante *AES*. La clave de red utilizada es “*ABCD*”.
- **PL (*Power Level*):** Establece la potencia de transmisión del nodo. Este valor es un número entero entre 0 (potencia de -8 dBm) y 4 (potencia de +2dBm). Los módulos del proyecto están configurados para usar la mínima potencia para ahorrar energía y limitar en lo posible el alcance de las transmisiones.

También existen parámetros de sólo lectura que ofrecen información de diagnóstico de la red, y a los que se puede acceder mediante el software X-CTU o también ejecutando los comandos AT adecuados. Algunos de estos parámetros pueden ser:

- **MY:** dirección de red de 16 bits del nodo, asignada dinámicamente al unirse a la red.
- **MP:** dirección de red de 16 bits del nodo padre. Este valor es esencial para conocer la ruta que siguen los mensajes dentro de la red.
- **NC:** número restante de nodos hijos que pueden tomar a este nodo como padre. Si su valor es 0 ya no puede aceptar más nodos hijos.
- **DB:** potencia del último mensaje recibido por el módulo.

3.4.- Protocolo de comunicación

Para controlar la comunicación ZigBee desde el programa cargado en la placa, se hace uso de [una librería abierta XBee](#)⁵⁷ para Arduino diseñada a tal efecto por Andrew Rapp⁵⁸, que soporta las principales tramas API de los módulos XBee. En este proyecto se usan principalmente dos tramas API: la que permite ejecutar comandos AT para obtener información del módulo XBee y, sobre todo, la trama de transmisión de datos que encapsula la información que el nodo quiere transmitir.

Transmit Request				
	Frame Fields	Offset	Example	Description
API Packet	Start Delimiter	0	0x7E	
	Length	MSB 1	0x00	Number of bytes between the length and the checksum
		LSB 2	0x16	
	Frame Type	3	0x10	
	Frame ID	4	0x01	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
	64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following address is also supported: 0x000000000000FFFF - Broadcast address
		6	0x13	
		7	0xA2	
		8	0x00	
		9	0x40	
		10	0x0A	
		11	0x01	
	Reserved	LSB 12	0x27	
		13	0xFF	Set to 0xFFFE.
	Broadcast Radius	14	0xFE	
		15	0x00	Sets maximum number of hops a broadcast transmission can occur. If set to 0, the broadcast radius will be set to the maximum hops value.
	Transmit Options	16	0x00	Bitfield: bit 0: Disable ACK bit 1: Don't attempt route Discovery. All other bits must be set to 0.
	RF Data	17	0x54	Data that is sent to the destination device
		18	0x78	
		19	0x44	
20		0x61		
21		0x74		
22		0x61		
23		0x30		
24	0x41			
Checksum	25	0x13	0xFF - the 8 bit sum of bytes from offset 3 to this byte.	

Figura 18 – Estructura de una trama API de transmisión de datos.

Al iniciar su funcionamiento, cada nodo obtiene su identidad y la identidad de su nodo padre mediante comandos AT. Además, en cada iteración del bucle, se leen los datos de los sensores de temperatura y luz. Con toda esta información, el nodo crea un paquete de datos y llama a la librería XBee para que encapsule dicho paquete dentro de una trama API de transmisión de datos, que es enviada hasta su destino: el nodo coordinador. Al recibir esta trama, el coordinador llama de nuevo a la librería XBee, que extrae el paquete de datos. El nodo obtiene entonces de este paquete la información y la almacena en su base de datos local. De esta forma puede saber el origen del mensaje, la ruta que ha seguido y los datos de los sensores.

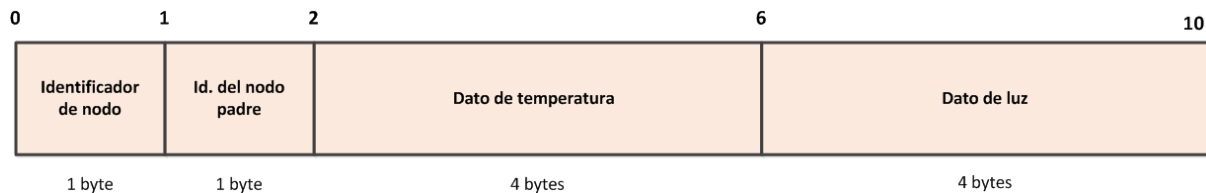


Figura 19 – Estructura del paquete de datos creado por el nodo.
Este paquete constituye el *payload* de la trama API

- **Identificador de nodo:** valor de 1 byte que identifica el nodo origen del mensaje y que coincide con el parámetro *NI*. Este valor se establece al configurar el módulo XBee y el microcontrolador puede acceder a él mediante el comando *AT NI*. El identificador del nodo coordinador es 0, los routers están numerados por un carácter numérico del 1 al 9 y los terminales por un carácter alfabético ASCII. De esta forma la red puede soportar hasta 60 nodos.
- **Identificador del nodo padre:** valor de 1 byte que determina la identidad del nodo padre. Mediante la ejecución del comando *AT MP*, el microcontrolador obtiene del módulo XBee la dirección de red del nodo padre. Una vez obtenida, se sondea a todos los posibles routers de la red con el comando *AT ND*, que solicita información como la dirección de red y el identificador *NI*. Este sondeo es necesario puesto que las direcciones de red son asignadas dinámicamente por el coordinador y no tienen un valor fijo. Entonces se compara la dirección recibida de cada nodo con la dirección del nodo padre obtenida anteriormente y, si alguna coincide, se establece su identificador *NI* como el del nodo padre.
- **Dato de temperatura:** *float* de 4 bytes que contiene el último dato leído del sensor de temperatura.
- **Dato de luz:** *float* de 4 bytes que contiene el último dato leído del sensor de luz.

Cabe destacar que, en caso de que un nodo caiga, abandone la red o una comunicación entre nodos se interrumpa por cualquier motivo, la naturaleza ad-hoc del protocolo ZigBee permite la reconfiguración automática de los enlaces entre los distintos nodos de la red. Esta reestructuración de la red se produce automáticamente cada vez que la ruta entre el coordinador y cualquier terminal se ve interrumpida y garantiza que la comunicación se lleve a cabo siempre que exista un camino posible entre nodo y coordinador,

4.- Aplicación de monitorización

Además del diseño e implementación de la red de sensores, en el proyecto se ha desarrollado una aplicación para dispositivos Android capaz de interactuar mediante Bluetooth con el nodo coordinador. La aplicación solicita al coordinador el envío de los datos de la red que éste tiene almacenados. Una vez recibidos, se procesa e interpreta la información, representando gráficamente el estado de los nodos de la red y de los enlaces existentes entre ellos.

La aplicación tiene dos vistas principales: una representación gráfica de la topología de la red y un modo texto con la información recibida del coordinador. Se puede cambiar indistintamente entre ambas interfaces en cualquier momento.

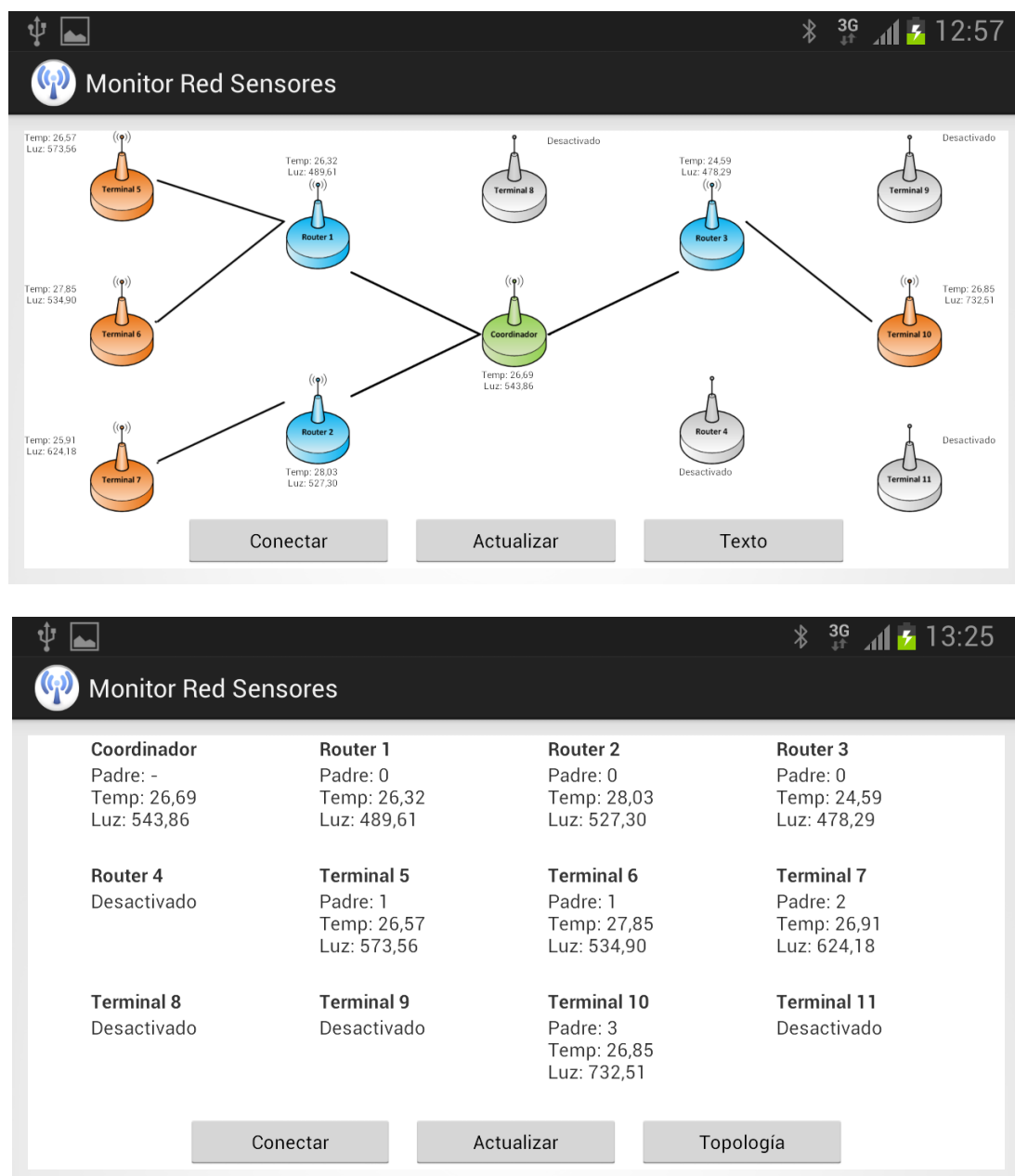


Figura 20 – Captura de pantalla de la aplicación de monitorización, mostrando un ejemplo de representación gráfica del estado de la red y su correspondiente modo texto.

4.1.- Comunicación Bluetooth

Debido a que la práctica totalidad de dispositivos Android cuenta con la posibilidad de establecer una comunicación Bluetooth, esta aplicación permite que cualquier smartphone o tablet se convierta en herramienta de monitorización de la red de sensores.

Para que la aplicación tenga acceso al módulo Bluetooth del dispositivo, es necesario otorgarle los permisos adecuados. Esta autorización se concede modificando el archivo `AndroidManifest`. Mediante su edición, se puede conceder a la aplicación autorización para acceder a los distintos recursos del dispositivo, entre otras características. En concreto, se le permite hacer uso del módulo Bluetooth al añadir la siguiente línea:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
```

Además es necesario que, previamente a la ejecución de la aplicación, el dispositivo Android y el módulo Bluetooth del nodo coordinador hayan sido vinculados mediante la interfaz estándar Bluetooth de Android. De esta forma al iniciar la aplicación ésta podrá encontrar fácilmente el nodo coordinador dentro de la lista de dispositivos Bluetooth vinculados. Sólo es necesario realizar esta vinculación una vez, puesto que queda registrada en el sistema.

Una vez vinculados ambos dispositivos y concedidos los permisos correspondientes, la aplicación ya está preparada para conectar al módulo Bluetooth del nodo coordinador y comenzar el tráfico de datos. Se presentan tres posibles interacciones entre ambos dispositivos:

- **Conectar:** es la primera acción que es necesario llevar a cabo al arrancar la aplicación. Para ello debe pulsarse el botón correspondiente que provoca que el sistema busque el módulo del nodo coordinador dentro de la lista de dispositivos Bluetooth vinculados. Una vez encontrado, se establece la conexión mediante la creación de un socket y se abren sendos flujos de entrada y salida de datos. De esta forma, la comunicación se encuentra lista para ser empleada.
- **Actualizar:** al pulsar este botón la aplicación envía al nodo coordinador una solicitud de información, consistente en el carácter ASCII 'A'. Cuando el nodo recibe esta petición envía secuencialmente al dispositivo Android los datos más recientes de todos los nodos de la red. Estos datos son transmitidos con el formato adecuado para ser directamente mostrados por pantalla en el modo texto de la aplicación. La información de cada nodo es recibida y procesada de forma independiente.
- **Desconectar:** esta acción se lleva a cabo cuando se pulsa el botón de desconexión o también cada vez que la aplicación es cerrada o pasa a segundo plano, puesto que se incluye dentro del método `onPause()`. Cuando alguno de estos eventos ocurre, la comunicación Bluetooth es cerrada, liberando los recursos empleados y cerrando los flujos de entrada y salida de datos. Para volver a establecer la conexión basta con pulsar de nuevo el botón *Conectar*.

4.2.- Representación de la información

Cuando la aplicación recibe información referente a un nodo, ésta es procesada, guardada y representada adecuadamente en la interfaz de monitorización. Esta representación dependerá de la vista que esté activa en el momento de enviar una solicitud de actualización al nodo coordinador. Se puede cambiar entre ambas vistas, que forman parte de una estructura `ViewFlipper`, pulsando el botón correspondiente.

Si la aplicación se encuentra en **modo texto** tan sólo necesita extraer el primer byte del mensaje recibido, que indica la identidad del nodo cuyos datos vienen a continuación, y mostrar por pantalla dichos datos en el campo de texto, o `TextView`, correspondiente, pues son ya transmitidos por el nodo coordinador con el formato adecuado.

Si por el contrario la aplicación está en el **modo de representación gráfica de la topología**, el procesado de los datos es más complejo. En primer lugar debe extraerse el byte que identifica al nodo origen de los restantes datos del mensaje. Si dichos datos indican que el nodo está activo, se muestra por pantalla, en el elemento `ImageView` adecuado, una imagen de dicho nodo iluminado en un color identificativo: verde para el coordinador, azul para los routers y naranja para los terminales. Sin embargo, si los datos recibidos indican que el nodo está desactivado, se mostrará una imagen grisácea para representar su inactividad.

Una vez identificado el nodo y mostrado su estado, es necesario identificar a su nodo padre. Para ello se extrae y se procesa el byte correspondiente dentro de los datos recibidos, y se representa el camino seguido dentro de la red por el paquete en su ruta hasta el coordinador. La aplicación cuenta con imágenes, denominadas **Drawables** en el contexto Android, que contemplan todas las posibles combinaciones de rutas entre los distintos nodos. Una vez se conocen el nodo origen y su nodo padre, basta con comparar dichos valores con unas tablas de estados implementadas en la aplicación y mostrar por pantalla la imagen correspondiente que representa el estado actual de los enlaces involucrados. Si por cualquier motivo un nodo terminal no tiene como padre uno de los nodos esperados, la ruta del mensaje quedará en blanco, indicando de esta forma que el despliegue de la red no es el deseado y que conviene reiniciar dicho nodo.

Por último, una vez “encendido” o “apagado” el nodo y actualizada la representación de la ruta seguida por el mensaje, tan sólo resta mostrar en el campo de texto correspondiente los datos de los sensores de temperatura y luz. De esta forma con un simple vistazo se puede tener una idea detallada del estado de la red de una forma más intuitiva y directa que en el modo texto, tal y como se puede apreciar en la [figura 20](#).

Cabe destacar que la aplicación mantiene una **base de datos local** con la información más reciente obtenida de cada nodo, por lo que cada nuevo dato recibido puede ser procesado de forma independiente. Cuando se recibe un nuevo dato de un nodo, se representará por pantalla el estado de la red en función tanto del nuevo dato de ese nodo, como de los últimos datos recibidos de los demás. De esta forma cada vez que se solicita una actualización la monitorización de la red siempre se muestra en su estado más reciente.

Para más información acerca del funcionamiento de la aplicación, se puede observar el código de la actividad que la conforma y sus correspondientes diagramas de flujo en el [Anexo A4](#).

5.- Conclusiones

El proyecto ha conseguido cumplir el objetivo de diseñar e implementar una red mallada inalámbrica de sensores capaz de reconfigurarse automáticamente para adaptarse a condiciones cambiantes tales como desconexión de nodos, modificaciones en su disposición, caídas de enlaces o ingreso de nuevos nodos a la red.

Asimismo, mediante la aplicación Android desarrollada se cuenta con la capacidad de monitorizar el estado de la red en cada momento, pudiendo observarse de forma inmediata e intuitiva la situación de los nodos y de los distintos enlaces entre ellos, así como los valores leídos por los sensores de luz y temperatura presentes en cada nodo. De esta forma, se cumple el objetivo didáctico que supuso el punto de partida del proyecto.

Otras de las principales conclusiones que se pueden extraer de la realización de este proyecto son las señaladas a continuación:

- Se ha podido evidenciar el potencial de las placas **Arduino** como elemento hardware libre de bajo coste. Su versatilidad, la potencia de su lenguaje de programación y la gran gama de modelos disponibles, junto con la posibilidad de montar placas personalizadas bajo licencia abierta, permite a las placas Arduino adaptarse a las necesidades de cualquier proyecto o prototipo que requiera del uso de un microcontrolador.
- Se han estudiado las principales características de la tecnología de comunicación inalámbrica **ZigBee**, entre las que se han destacado su sencillez y eficacia para el despliegue, configuración y administración de redes ad-hoc auto-configurables de bajo consumo, gran alcance, adaptabilidad frente a condiciones impredecibles y elevado número de nodos. Estas prestaciones convierten a ZigBee en una tecnología inalámbrica superior a otras alternativas como Wi-Fi para la realización de un proyecto de esta índole.
- Se ha destacado la gran extensión de la tecnología inalámbrica **Bluetooth** y su presencia en la práctica totalidad de dispositivos móviles de última generación, lo que la convierte en una de las tecnologías de comunicación de mayor compatibilidad. Que el nodo coordinador utilice Bluetooth para comunicarse con el dispositivo encargado de la monitorización del sistema, en lugar de mediante la tecnología ZigBee ya presente en la red, garantiza que la gama de dispositivos con los que puede interactuar sea la mayor posible.
- Se ha podido observar cómo la aplicación de monitorización desarrollada en entorno **Android** permite convertir cualquier dispositivo móvil que cuente con este sistema operativo, como puede ser un smartphone o una tablet, en una potente plataforma para monitorizar o controlar sistemas de forma remota, independientemente de las distintas tecnologías en que éstos se basen. Android ofrece de forma abierta un potente entorno de programación y todas las herramientas indispensables para programar aplicaciones que se adapten a las necesidades de cada proyecto.

5.1.- Problemas encontrados y soluciones propuestas

A lo largo del proceso de diseño y desarrollo del proyecto aparecieron diversos problemas y obstáculos que requirieron de una mayor atención y esfuerzo para poder ser solventados. Principalmente fueron consecuencia de la dificultad de llevar a la práctica algunas de las ideas planteadas en el diseño, debido a las limitaciones de las tecnologías empleadas. También hubo que descartar algunas tecnologías presentes en un diseño inicial por otras más adecuadas al proyecto.

Los principales problemas a los que hubo que enfrentarse durante la realización del proyecto fueron:

- La dificultad para determinar la ruta seguida por los mensajes dentro de la red. Debido a las limitaciones de los módulos XBee no es posible conocer la identidad del nodo padre de una radio configurada como router. Esto hace imposible determinar la ruta seguida por los mensajes enviados por un router en el caso de existir más de dos posibles caminos hasta el nodo coordinador. Este problema se solventó estableciendo una disposición fija de los nodos de la red de forma que todos los routers pudieran comunicarse directamente con el coordinador y desplegando los terminales en la periferia de la red. Los terminales sí disponen de mecanismos para conocer cuál de los routers es su nodo padre, por lo que de esta forma se puede conocer la ruta seguida por todos los mensajes que se transmiten por la red.
- En un principio se pensó en utilizar el modelo de placas [Arduino Fio](#), que ofrece la posibilidad de ser reprogramado inalámbricamente mediante una comunicación ZigBee, lo cual suponía una gran ventaja a la hora de añadir prestaciones adicionales a la red. Sin embargo dicha reprogramación inalámbrica requería necesariamente del uso de módulos XBee de la serie 1, que únicamente soportan comunicaciones punto a punto. Puesto que el objetivo del proyecto era implementar una red mallada para la que es necesario el uso de módulos XBee de la serie 2 y siendo del todo imposible cualquier tipo de interacción entre ambas series, se optó por renunciar a la reprogramación inalámbrica y descartar las placas Arduino Fio en favor de las Uno.
- La naturaleza didáctica del proyecto hacía pensar en un principio que era necesario limitar el alcance de los nodos de la red de forma que no todos ellos se pudieran comunicar directamente con el coordinador para que, de esta manera, se establecieran entre ellos los enlaces adecuados. No obstante, incluso configurando al mínimo la potencia de transmisión de los nodos, su alcance, de aproximadamente 10 metros, era demasiado grande como para que toda la red pudiera ser, en principio, desplegada en una misma habitación. Sin embargo, se descubrió que los módulos XBee funcionan de tal forma que, al conectarse a la red, eligen como nodo padre al router o coordinador que les ofrece una mejor calidad de señal. Así, todos los nodos pueden ser colocados en una misma mesa con la única precaución de que cada nodo terminal debe estar más cerca del router que debe actuar como su padre que de los demás. En caso de desconectarse dicho router, el terminal tomará como nodo padre al segundo router (o coordinador) que tenga más cerca, facilitándose así en gran medida el despliegue adecuado de la red y su administración.

5.2.- Líneas futuras

Son múltiples las posibles aplicaciones que se le pueden dar a una red de estas características. Está diseñada para ser fácilmente escalable hasta 60 nodos, o muchos más introduciendo las modificaciones adecuadas en su diseño, y basta con cambiar los tipos de sensores y actuadores de que dispone para adaptarla según las necesidades. Se puede también desarrollar una aplicación Android cuya función sea el control telemático de los elementos de la red, además de su monitorización. Por otra parte, cambiando el módulo Bluetooth del nodo coordinador por cualquier otro tipo de módulo se puede otorgar a la red la posibilidad de interactuar con otros sistemas mediante distintas tecnologías de comunicación: Wi-Fi, Ethernet, etc.

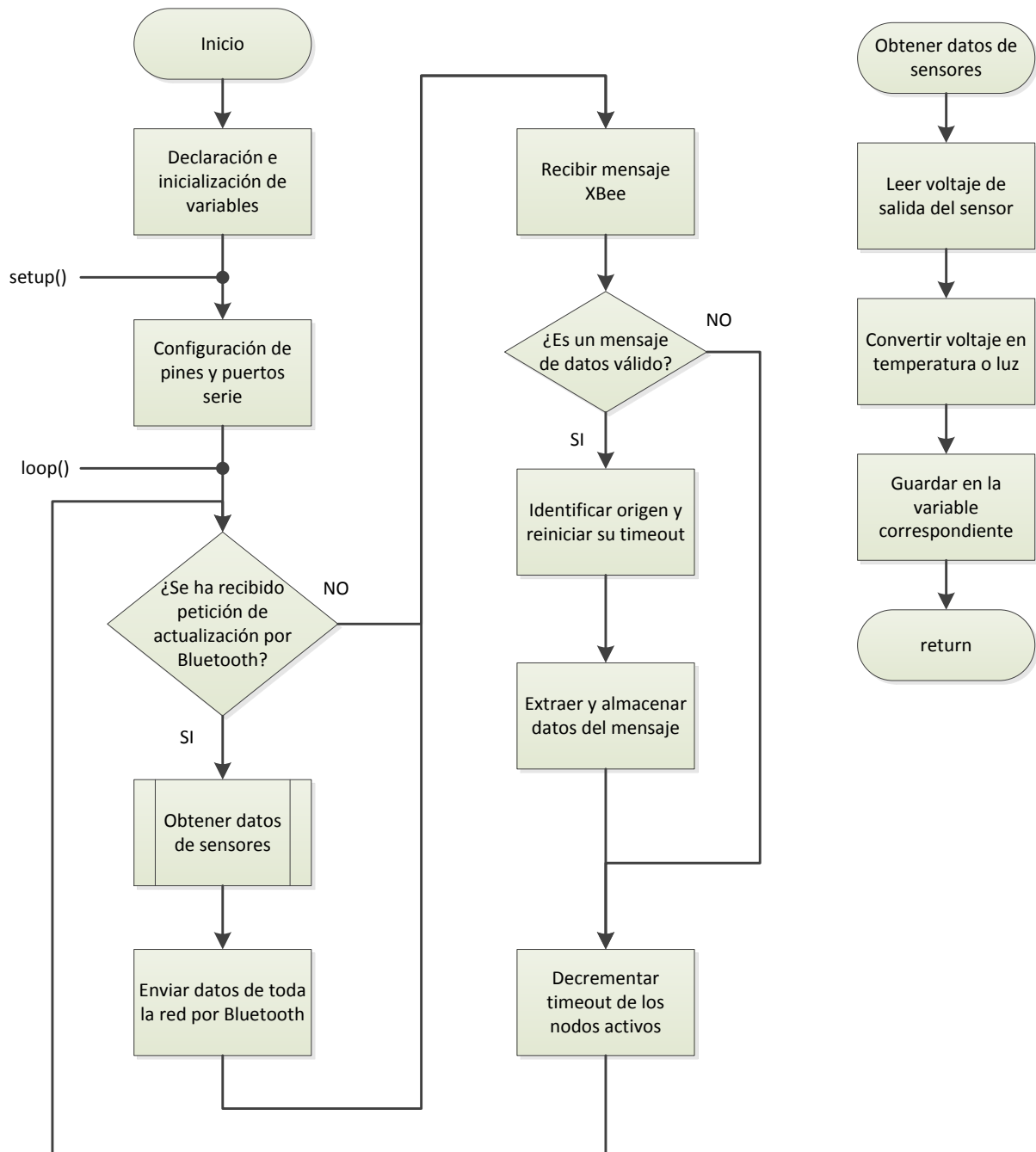
A modo de referencia se presentan a continuación algunos ámbitos en los que se emplean actualmente redes ZigBee similares a la implementada en el proyecto:

- Gestión energética inteligente.
- Domótica y automatización de edificios.
- Monitorización de sistemas.
- Servicios de vigilancia y seguridad.
- Control de procesos en plantas industriales.
- Tele-asistencia médica y control telemático de constantes vitales.
- Sistemas de calefacción y aire acondicionado.
- Y, como en este proyecto, prototipos con fines académicos.

En concreto, se está estudiando la posibilidad de aplicar una red como la diseñada en este proyecto para controlar el encendido y apagado automático de la iluminación de las aulas y salas de la universidad con el fin de optimizar el ahorro energético. Mediante el despliegue por el campus de nodos dotados de sensores de movimiento se podría determinar las estancias que están vacías en un sistema de monitorización central y proceder al apagado automático de la iluminación correspondiente. Este planteamiento puede dar origen a un futuro proyecto.

Anexo A. Código y flujogramas

A1.- Coordinador



```

#include <XBee.h>
#include <SoftwareSerial.h>

#define NUM_NODOS 12 //Declaración de constantes
#define NUM_ROUTERS 4
#define BT_TX 10
#define BT_RX 11

const unsigned int PIN_TEMP = 0;
const unsigned int PIN_LUZ = 1;
const unsigned int LED = 13;
const unsigned int BAUD_RATE = 9600;
const float ALIMENTACION = 5.0;

XBee xbee = XBee(); //Declaración de variables
SoftwareSerial BluetoothSerial(BT_TX, BT_RX);
XBeeResponse response = XBeeResponse();
ZBRxResponse rx = ZBRxResponse();
ModemStatusResponse msr = ModemStatusResponse();

union u_tag1 {
    byte t[4];
    float ftemp;
} temp;

union u_tag2 {
    byte l[4];
    float fluz;
} luz;

typedef struct{
    byte padre;
    byte timeout;
    float temperatura;
    float luminosidad;
} Nodo;

Nodo nodo[NUM_NODOS];

int i = 0;
byte origen = 0;
byte padre = 0;
byte buffer[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

void setup() {
    pinMode(LED, OUTPUT);
    Serial.begin(BAUD_RATE);
    xbee.begin(BAUD_RATE);
    BluetoothSerial.begin(BAUD_RATE);

    parpadeo(LED, 3, 50);
}

void loop() {

//si se recibe por Bluetooth una petición de actualización por parte de la
aplicación de monitorización, se leen los datos de los sensores locales y
se envía por Bluetooth los datos más recientes de toda la red.

```

```

if (BluetoothSerial.available()){
  if (BluetoothSerial.read() == 'A'){
    temp.ftemp = get_temp();
    luz.fluz = get_luz();
    nodo[0].timeout = NUM_NODOS;
    nodo[0].padre = 0;
    nodo[0].temperatura = temp.ftemp;
    nodo[0].luminosidad = luz.fluz;

    enviar_datos();
  }
}

xbee.readPacket(1000); //se espera 1 seg a recibir un mensaje XBee

//si se recibe un mensaje válido, sus datos se extraen al buffer de
entrada, se identifica el origen, y los datos se almacenan en la
posición correspondiente del array de nodos.

if (xbee.getResponse().isAvailable() ) {
  if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE) {
    xbee.getResponse().getZBRxResponse(rx);
    if (rx.getOption() == ZB_PACKET_ACKNOWLEDGED) {
      parpadeo(LED, 10, 10);
    }
    else {
      parpadeo(LED, 2, 20);
    }
  }

  for (i=0; i<10; i++){
    buffer[i] = rx.getData(i);
  }

  origen = buffer[0];
  if (origen >= 'A') origen = origen - 'A' + NUM_ROUTERS + 1;
  nodo[origen].timeout = NUM_NODOS;
  nodo[origen].padre = buffer[1];

  for(i=0;i<4;i++){
    temp.t[i] = buffer[i+2];
  }

  nodo[origen].temperatura = temp.ftemp;

  for(i=0;i<4;i++){
    luz.l[i] = buffer[i+6];
  }

  nodo[origen].luminosidad = luz.fluz;
}

//si es otro tipo de mensaje (control, error, desconocido, etc) el
LED parpadea para indicarlo.

else if (xbee.getResponse().getApiId() == MODEM_STATUS_RESPONSE){
  xbee.getResponse().getModemStatusResponse(msr);

  if (msr.getStatus() == ASSOCIATED){
    parpadeo(LED, 10, 10);
  }
}

```

```

        else if (msr.getStatus() == DISASSOCIATED){
            parpadeo(LED, 10, 10);
        }

        else{
            parpadeo(LED, 5, 10);
        }
    }

    else {
        parpadeo(LED, 1, 25);
    }
}
actualizar_nodos();           //se decrementa el timeout de los nodos
}

void parpadeo(int pin, int times, int wait) {
//función que controla el parpadeo del led
    for (int i = 0; i < times; i++) {
        digitalWrite(pin, HIGH);
        delay(wait);
        digitalWrite(pin, LOW);

        if (i + 1 < times) {
            delay(wait);
        }
    }
}

void actualizar_nodos(){
//función que reduce el timeout de los nodos
    for(i=0; i<NUM_NODOS; i++){
        if (nodo[i].timeout > 0){
            nodo[i].timeout -= 1;
        }
    }
}

void enviar_datos(){
//funcion que envía por Bluetooth todos los datos almacenados en el array
de nodos

    for(i=0; i<NUM_NODOS; i++){
        BluetoothSerial.write(i);
        if(nodo[i].timeout > 0){
            BluetoothSerial.print("Padre: ");
            if(i==0) BluetoothSerial.println("-");
            else BluetoothSerial.println(nodo[i].padre);
            BluetoothSerial.print("Temp: ");
            BluetoothSerial.println(nodo[i].temperatura);
            BluetoothSerial.print("Luz: ");
            BluetoothSerial.println(nodo[i].luminosidad);
        }
        else BluetoothSerial.println("Desactivado");

        BluetoothSerial.write(0x7f);    }
}

```

```

void mostrar_datos() {

//función auxiliar que envía los datos por el monitor serial. No es llamada
durante la ejecución, pero se conserva para posible depuración futura.

boolean actividad = false;
Serial.println("Datos de los nodos activos:");

for(i=0; i<NUM_NODOS; i++){
  if(nodo[i].timeout > 0){
    if(!actividad) actividad = true;

    Serial.println("-----");

    Serial.print("Datos recibidos del sensor numero ");
    Serial.println(i);
    Serial.print("Cuyo padre es el nodo numero ");
    Serial.println(nodo[i].padre);

    Serial.print("Temperatura: ");
    Serial.print(nodo[i].temperatura);
    Serial.println(" C");

    Serial.print("Luminosidad: ");
    Serial.print(nodo[i].luminosidad);
    Serial.println(" lux");

    delay(100);
  }
}

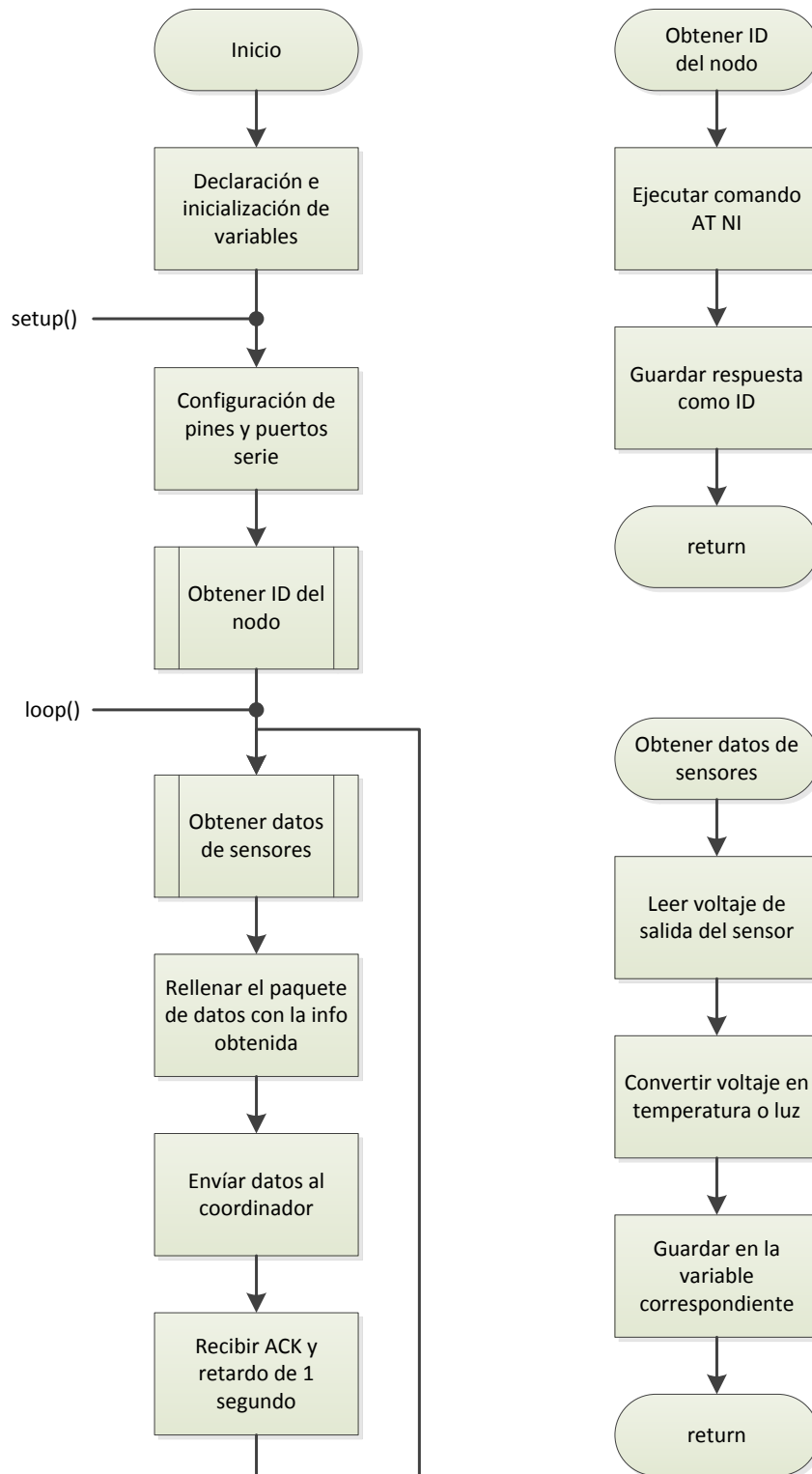
if(!actividad) Serial.println("No se ha registrado ningún nodo activo");
}

const float get_temp(){
//función que lee el sensor de temperatura
const int temp_voltaje = analogRead(PIN_TEMP);
const float t = temp_voltaje * ALIMENTACION / 1023;
return ((t * 1000 - 500) / 10);
}

const float get_luz(){
//función que lee el sensor de luz
const int luz_voltaje = analogRead(PIN_LUZ);
const float l = luz_voltaje * ALIMENTACION / 1023;
return (l * (10000 / ALIMENTACION));
}

```


A2.- Router



```

#include <XBee.h>

#define COORDINADOR 0x4079c2a9

const unsigned int PIN_TEMP = 0;           //declaración de constantes
const unsigned int PIN_LUZ = 1;
const unsigned int LED = 13;
const unsigned int BAUD_RATE = 9600;
const float ALIMENTACION = 5.0;

union u_tag1 {                             //declaración de variables
    byte t[4];
    float ftemp;
} temp;

union u_tag2 {
    byte l[4];
    float fluz;
} luz;

int i = 0;
byte id = 0xFF;
byte datos[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
XBee xbee = XBee();
XBeeAddress64 direccion = XBeeAddress64(0x0013a200, COORDINADOR);
ZBTxRequest tx = ZBTxRequest(direccion, datos, sizeof(datos));
ZBTxStatusResponse txStatus = ZBTxStatusResponse();

void setup(){
    pinMode(LED, OUTPUT);                 //se inicializan puertos y pines
    Serial.begin(BAUD_RATE);
    xbee.setSerial(Serial);
    obtener_ID();                          //se obtiene la ID del nodo
}

void loop(){
    temp.ftemp = get_temp();               //se lee el dato de temperatura
    luz.fluz = get_luz();                  //se lee el dato de luz
    datos[0] = id;                         //se rellena el paquete de datos
    datos[1] = 0;

    for(i=0;i<4;i++){
        datos[i+2] = temp.t[i];
    }

    for(i=0;i<4;i++){
        datos[i+6] = luz.l[i];
    }

    xbee.send(tx);                         //se envía la trama de datos

    parpadeo(LED, 1, 100);

    if (xbee.readPacket(500)) {           //se espera a recibir respuesta

        //si la respuesta es correcta el LED parpadea rápido; si no, lento.
        if (xbee.getResponse().getApiId() == ZB_TX_STATUS_RESPONSE) {
            xbee.getResponse().getZBTxStatusResponse(txStatus);
        }
    }
}

```

```

        if (txStatus.getDeliveryStatus() == SUCCESS) {
            parpadeo(LED, 5, 50);
        } else {
            parpadeo(LED, 3, 500);
        }
    }

    delay(1000);

}

}

const float get_temp(){
    //función que lee el sensor de temperatura y devuelve el dato en grados.
    const int temp_voltaje = analogRead(PIN_TEMP);
    const float t = temp_voltaje * ALIMENTACION / 1023;
    return ((t * 1000 - 500) / 10);
}

const float get_luz(){
    //función que lee el sensor de luz y devuelve el dato en lux.
    const int luz_voltaje = analogRead(PIN_LUZ);
    const float l = luz_voltaje * ALIMENTACION / 1023;
    return (l * (10000 / ALIMENTACION));
}

void parpadeo(int pin, int times, int wait) {
    //función que controla el parpadeo del led
    for (int i = 0; i < times; i++) {
        digitalWrite(pin, HIGH);
        delay(wait);
        digitalWrite(pin, LOW);

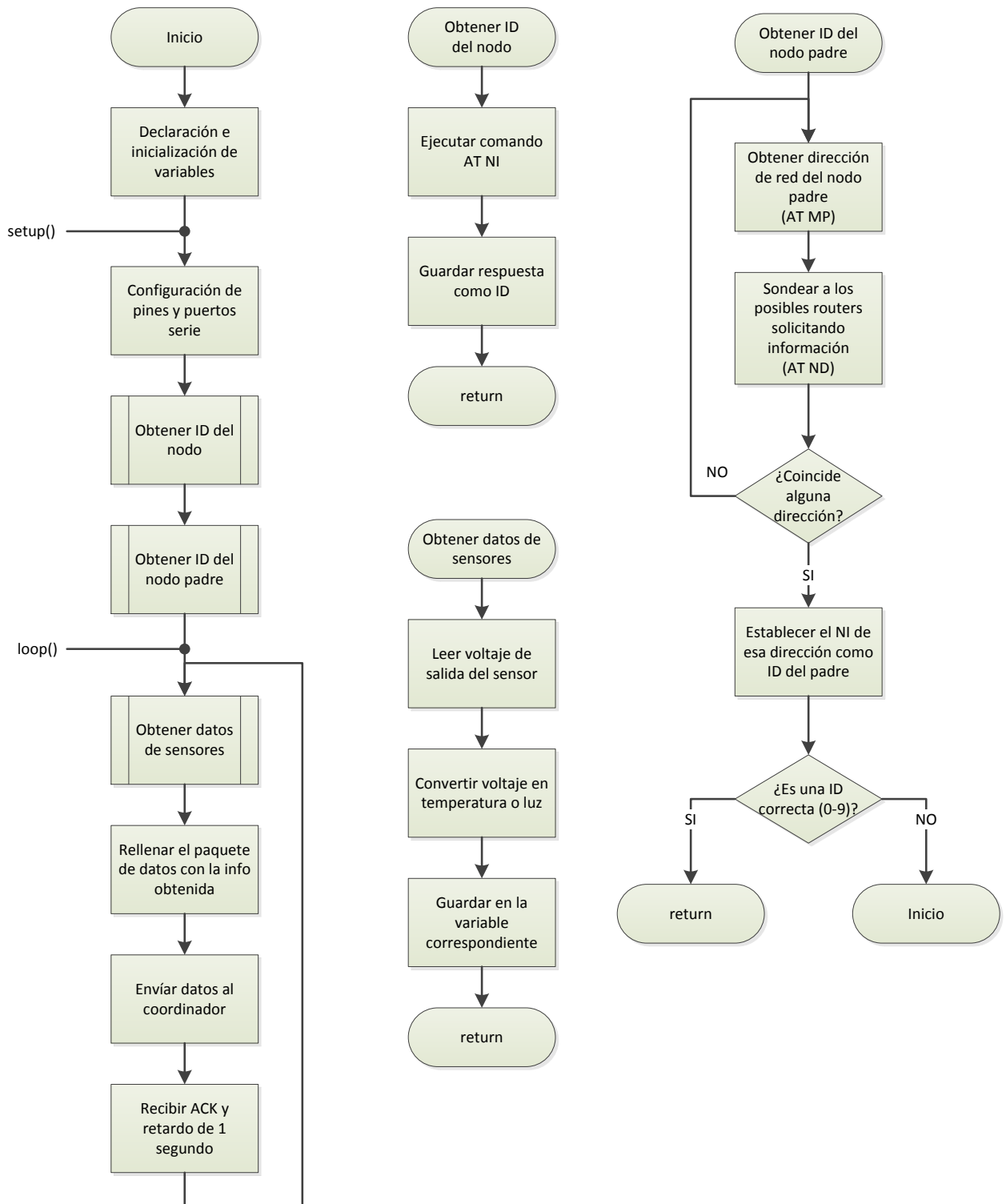
        if (i + 1 < times) {
            delay(wait);
        }
    }
}

void obtener_ID(){
    //función que obtiene el ID del nodo mediante el comando AT NI
    byte comando_NI[] = {'N', 'I'};
    AtCommandRequest atRequest = AtCommandRequest(comando_NI);
    AtCommandResponse atResponse = AtCommandResponse();

    xbee.send(atRequest);
    if (xbee.readPacket(5000)) {
        if (xbee.getResponse().getApiId() == AT_COMMAND_RESPONSE) {
            xbee.getResponse().getAtCommandResponse(atResponse);
            if (atResponse.isOk()) {
                if (atResponse.getValueLength() > 0) {
                    id = atResponse.getValue()[0] - '0';
                }
            }
        }
    }
}
}

```

A3.- Terminal



```

#include <XBee.h>
#define COORDINADOR 0x4079c2a9

const unsigned int PIN_TEMP = 0;           //declaración de constantes
const unsigned int PIN_LUZ = 1;
const unsigned int LED = 13;
const unsigned int BAUD_RATE = 9600;
const float ALIMENTACION = 5.0;

union u_tag1 {
  byte t[4];
  float ftemp;
} temp;

union u_tag2 {
  byte l[4];
  float fluz;
} luz;

int i = 0;                                 //declaración de variables
byte id = 0xFF;
byte id_padre = 0xFF;
byte id_padre_aux = 0xFF;
byte dir_padre[] = { 0xFF, 0xFE };
byte dir_aux[] = { 0xFF, 0xFE };
byte datos[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
XBee xbee = XBee();
XBeeAddress64 direccion = XBeeAddress64(0x0013a200, COORDINADOR);
ZBTxRequest tx = ZBTxRequest(direccion, datos, sizeof(datos));
ZBTxStatusResponse txStatus = ZBTxStatusResponse();

void setup(){
  pinMode(LED, OUTPUT);                   //se inicializan puertos y pines
  Serial.begin(BAUD_RATE);
  xbee.setSerial(Serial);

  delay(1000);

  obtener_ID();                           //se obtiene la ID del nodo
  buscar_padre();                          //se obtiene la ID del nodo padre
}

void(* resetFunc) (void) = 0;             //función que resetea el Arduino

void loop(){

  temp.ftemp = get_temp();                 //se lee el dato de temperatura
  luz.fluz = get_luz();                   //se lee el dato de luz
  datos[0] = id;                           //se rellena el paquete de datos
  datos[1] = id_padre;

  for(i=0;i<4;i++){
    datos[i+2] = temp.t[i];
  }

  for(i=0;i<4;i++){
    datos[i+6] = luz.l[i];
  }

  xbee.send(tx);                           //se envía la trama de datos
  parpadeo(LED, 1, 100);
}

```

```

if (xbee.readPacket(500)) { //se espera a recibir respuesta.

    //si la respuesta es correcta el LED parpadea rápido; si no, lento.
    if (xbee.getResponse().getApiId() == ZB_TX_STATUS_RESPONSE) {
        xbee.getResponse().getZBTxStatusResponse(txStatus);
        if (txStatus.getDeliveryStatus() == SUCCESS) {
            parpadeo(LED, 5, 50);
        } else {
            parpadeo(LED, 3, 500);
            resetFunc();
        }
    }
    delay(1000);
}
}

const float get_temp(){
//función que lee el sensor de temperatura y devuelve el dato en grados
    const int temp_voltaje = analogRead(PIN_TEMP);
    const float t = temp_voltaje * ALIMENTACION / 1023;
    return ((t * 1000 - 500) / 10);
}

const float get_luz(){
//función que lee el sensor de luz y devuelve el dato en lux
    const int luz_voltaje = analogRead(PIN_LUZ);
    const float l = luz_voltaje * ALIMENTACION / 1023;
    return (l * (10000 / ALIMENTACION));
}

void parpadeo(int pin, int times, int wait) {
//función que controla el parpadeo del led
    for (int i = 0; i < times; i++) {
        digitalWrite(pin, HIGH);
        delay(wait);
        digitalWrite(pin, LOW);
        if (i + 1 < times) {
            delay(wait);
        }
    }
}

void obtener_ID(){
//función que obtiene el identificador del nodo

    byte comando_NI[] = {'N','I'};
    AtCommandRequest atRequest = AtCommandRequest(comando_NI);
    AtCommandResponse atResponse = AtCommandResponse();

    while(1){
        xbee.send(atRequest);
        if (xbee.readPacket(500)) {
            if (xbee.getResponse().getApiId() == AT_COMMAND_RESPONSE) {
                xbee.getResponse().getAtCommandResponse(atResponse);
                if (atResponse.isOk()) {
                    if (atResponse.getValueLength() > 0) {
                        if(atResponse.getValue()[0]>='A' &&
atResponse.getValue()[0]<='Z'){
                            id = atResponse.getValue()[0];
                            break;
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    }
}

delay(100);
}
}

void buscar_padre(){ //función que obtiene la identidad del nodo padre
byte id_aux = '0';
id_padre = 0xFF;
dir_padre[0] = 0xFF;
dir_padre[1] = 0xFE;

while(dir_padre[0]==0xFF && dir_padre[1]==0xFE){
    sendAtCommand_MP();
    delay(100);
}

for (i=0; i<10; i++){
    //se sondea a los posibles nodos padre mediante AT ND

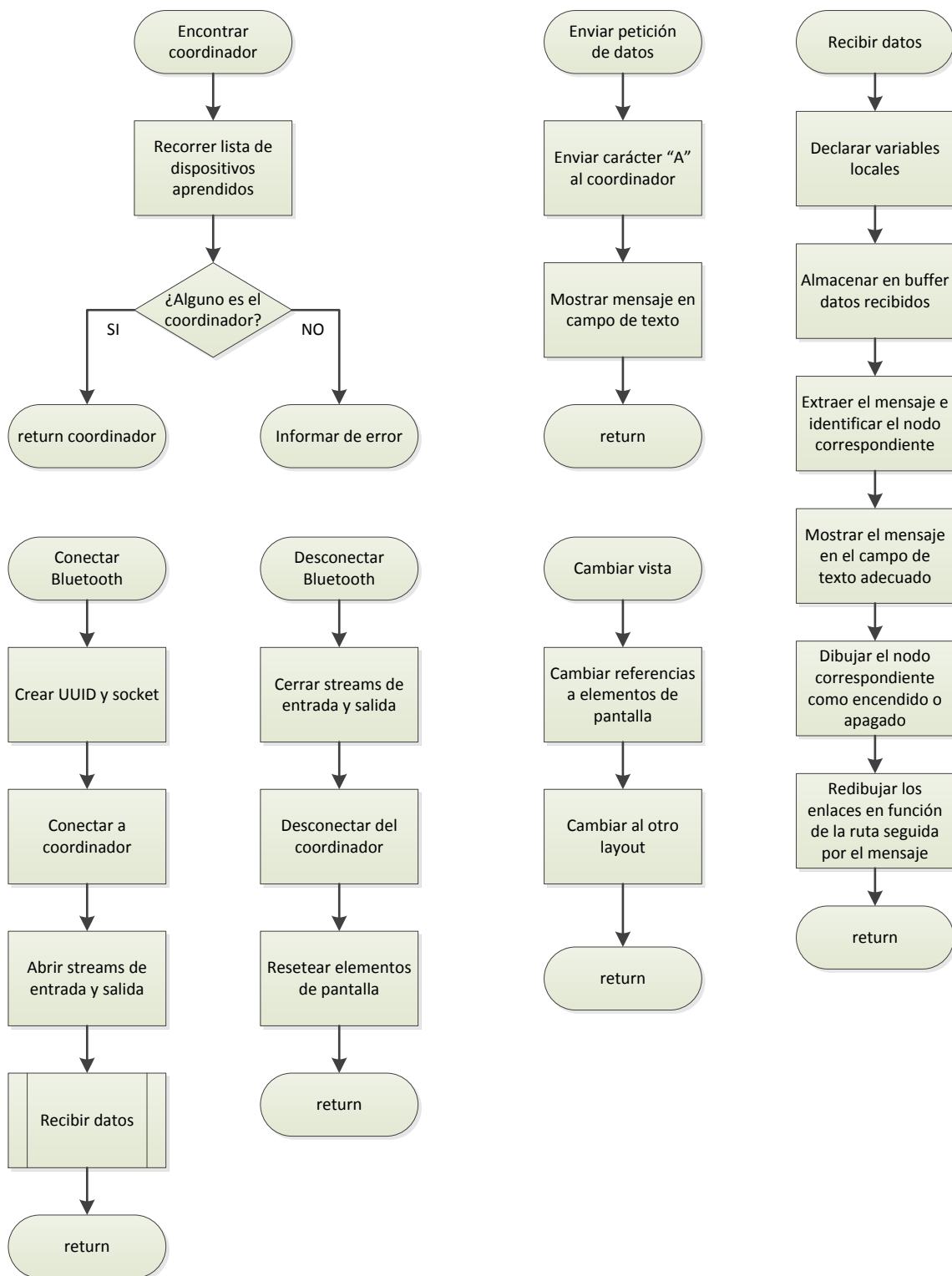
    dir_aux[0] = 0xFF;
    dir_aux[1] = 0xFE;
    delay(100);
    sendAtCommand_ND(&id_aux);
if(word(dir_aux[0], dir_aux[1]) == word(dir_padre[0],dir_padre[1])){
id_padre = id_padre_aux;
break;
}
    id_aux++;
}
delay(100);
if (id_padre < 0 || id_padre > 9) resetFunc();
}

void sendAtCommand_MP() {
//función que obtiene la dirección red del nodo padre

byte comando_MP[] = {'M','P'};
AtCommandRequest atRequest = AtCommandRequest(comando_MP);
AtCommandResponse atResponse = AtCommandResponse();

xbee.send(atRequest);
if (xbee.readPacket(500)) {
    if (xbee.getResponse().getApiId() == AT_COMMAND_RESPONSE) {
        xbee.getResponse().getAtCommandResponse(atResponse);
        if (atResponse.isOk()) {
            if (atResponse.getValueLength() > 0) {
                dir_padre[0] = atResponse.getValue()[0];
dir_padre[1] = atResponse.getValue()[1];
            }
        }
    }
}
}
}
}

```

```

package es.upct.monitorredsensores;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Set;
import java.util.UUID;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.ViewFlipper;

public class MainActivity extends Activity
{
    TextView[] myLabel = new TextView[12];
    BluetoothAdapter mBluetoothAdapter;
    BluetoothSocket mmSocket;
    BluetoothDevice mmDevice;
    OutputStream mmOutputStream;
    InputStream mmInputStream;
    Thread workerThread;
    byte[] readBuffer;
    int readBufferPosition;
    int counter;
    volatile boolean stopWorker;
    Button openButton;
    Button sendButton;
    Button changeButton;
    ImageView nodo_img[] = new ImageView[12];
    ImageView ruta_img[] = new ImageView[5];
    int nodo_draw[] = new int[24];
    int i;
    boolean[] activado = new boolean[12];
    int[] padre = {0,0,0,0,0,0,0,0,0,0,0,0};
    ViewFlipper vf;
    boolean texto = false;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setRequestedOrientation (ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        setContentView(R.layout.activity_main);
        vf = (ViewFlipper) findViewById(R.id.vf);

        //Botones
        openButton = (Button) findViewById(R.id.open);
        sendButton = (Button) findViewById(R.id.send);
        changeButton = (Button) findViewById(R.id.change);
    }
}

```

```

//Espacio reservado para las imágenes de los nodos
nodo_img[0] = (ImageView) findViewById(R.id.coordinador_img);
nodo_img[1] = (ImageView) findViewById(R.id.router1_img);
nodo_img[2] = (ImageView) findViewById(R.id.router2_img);
nodo_img[3] = (ImageView) findViewById(R.id.router3_img);
nodo_img[4] = (ImageView) findViewById(R.id.router4_img);
nodo_img[5] = (ImageView) findViewById(R.id.terminal5_img);
nodo_img[6] = (ImageView) findViewById(R.id.terminal6_img);
nodo_img[7] = (ImageView) findViewById(R.id.terminal7_img);
nodo_img[8] = (ImageView) findViewById(R.id.terminal8_img);
nodo_img[9] = (ImageView) findViewById(R.id.terminal9_img);
nodo_img[10] = (ImageView) findViewById(R.id.terminal10_img);
nodo_img[11] = (ImageView) findViewById(R.id.terminal11_img);

//Imágenes de los nodos
nodo_draw[0] = R.drawable.coordinador_off;
nodo_draw[1] = R.drawable.coordinador_on;
nodo_draw[2] = R.drawable.router1_off;
nodo_draw[3] = R.drawable.router1_on;
nodo_draw[4] = R.drawable.router2_off;
nodo_draw[5] = R.drawable.router2_on;
nodo_draw[6] = R.drawable.router3_off;
nodo_draw[7] = R.drawable.router3_on;
nodo_draw[8] = R.drawable.router4_off;
nodo_draw[9] = R.drawable.router4_on;
nodo_draw[10] = R.drawable.terminal5_off;
nodo_draw[11] = R.drawable.terminal5_on;
nodo_draw[12] = R.drawable.terminal6_off;
nodo_draw[13] = R.drawable.terminal6_on;
nodo_draw[14] = R.drawable.terminal7_off;
nodo_draw[15] = R.drawable.terminal7_on;
nodo_draw[16] = R.drawable.terminal8_off;
nodo_draw[17] = R.drawable.terminal8_on;
nodo_draw[18] = R.drawable.terminal9_off;
nodo_draw[19] = R.drawable.terminal9_on;
nodo_draw[20] = R.drawable.terminal10_off;
nodo_draw[21] = R.drawable.terminal10_on;
nodo_draw[22] = R.drawable.terminal11_off;
nodo_draw[23] = R.drawable.terminal11_on;

//Espacio reservado para representar los enlaces entre nodos
ruta_img[0] = (ImageView) findViewById(R.id.tr_img);
ruta_img[1] = (ImageView) findViewById(R.id.rc_img);
ruta_img[2] = (ImageView) findViewById(R.id.t8c_img);
ruta_img[3] = (ImageView) findViewById(R.id.cr_img);
ruta_img[4] = (ImageView) findViewById(R.id.rt_img);

//Campos de texto para mostrar los datos de los nodos
myLabel[0] = (TextView) findViewById(R.id.label0);
myLabel[1] = (TextView) findViewById(R.id.label1);
myLabel[2] = (TextView) findViewById(R.id.label2);
myLabel[3] = (TextView) findViewById(R.id.label3);
myLabel[4] = (TextView) findViewById(R.id.label4);
myLabel[5] = (TextView) findViewById(R.id.label5);
myLabel[6] = (TextView) findViewById(R.id.label6);
myLabel[7] = (TextView) findViewById(R.id.label7);
myLabel[8] = (TextView) findViewById(R.id.label8);
myLabel[9] = (TextView) findViewById(R.id.label9);
myLabel[10] = (TextView) findViewById(R.id.label10);
myLabel[11] = (TextView) findViewById(R.id.label11);

```

```

//Se activa el Bluetooth del dispositivo
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

if(mBluetoothAdapter == null)
{
    myLabel[0].setText("Ningún dispositivo Bluetooth disponible");
}

if(!mBluetoothAdapter.isEnabled())
{
    Intent enableBluetooth = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBluetooth, 0);
}

//Botón Conectar
openButton.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View v)
    {
        if(mmOutputStream==null){

            try
            {
                findBT();
                openBT();
            }
            catch (IOException ex) { }

            if(mmOutputStream!=null){
                myLabel[0].setText("Bluetooth conectado");
                openButton.setText("Desconectar");
                nodo_img[0].setImageResource(nodo_draw[1]);
            }
            else myLabel[0].setText("Bluetooth no conectado");

        }

        else{

            try
            {
                closeBT();
            }
            catch (IOException ex) { }

        }
    }
});

//Botón Actualizar
sendButton.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View v)
    {
        try
        {
            if(mmOutputStream!=null)
                sendData();
        }
    }
});

```

```

                else
                    myLabel[0].setText("No encontrado");
            }
        catch (IOException ex) { }
    }
});

//Botón Desconectar
changeButton.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View v)
    {
        cambiarVista();
    }
});
}

public void onPause() {

    //Método que cierra la comunicación Bluetooth al cerrar la aplicación
    super.onPause();
    if(mmOutputStream!=null){
        try
        {
            closeBT();
        }
        catch (IOException ex) { }
    }
}

void findBT() //Método que busca el módulo Bluetooth del nodo coordinador
{

    Set<BluetoothDevice> pairedDevices =
mBluetoothAdapter.getBondedDevices();
    if(pairedDevices.size() > 0)
    {
        for(BluetoothDevice device : pairedDevices)
        {
            if(device.getName().equals("COORDINADOR"))
            {
                mmDevice = device;
                break;
            }
        }
    }
}

void openBT() throws IOException
{
    //Método que conecta al Bluetooth del coordinador
    UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    mmSocket = mmDevice.createRfcommSocketToServiceRecord(uuid);
    mmSocket.connect();
    mmOutputStream = mmSocket.getOutputStream();
    mmInputStream = mmSocket.getInputStream();

    beginListenForData();}
}

```

```

void beginListenForData()
{
    //Método que recibe los datos de la red y los representa gráficamente
    final Handler handler = new Handler();
    final byte delimiter = 0x7f; //Caracter que marca el fin de la
transmisión

    stopWorker = false;
    readBufferPosition = 0;
    readBuffer = new byte[1024];
    workerThread = new Thread(new Runnable()
    {
        public void run()
        {
            while(!Thread.currentThread().isInterrupted() && !stopWorker)
            {
                try
                {
                    int bytesAvailable = mmInputStream.available();
                    if(bytesAvailable > 0)
                    {
                        byte[] packetBytes = new byte[bytesAvailable];
                        mmInputStream.read(packetBytes);
                        for(int i=0;i<bytesAvailable;i++)
                        {
                            byte b = packetBytes[i];
                            if(b == delimiter)
                            {

                                byte[] encodedBytes = new byte[readBufferPosition];
                                System.arraycopy(readBuffer, 0, encodedBytes, 0, encodedBytes.length);
                                final int id = (int)encodedBytes[0];
                                final String data;
                                if(encodedBytes[1]=='P' && !texto){
                                    //si son datos de un nodo y estamos en modo gráfico, se extrae lo
importante
                                    data = new String(encodedBytes, 11, encodedBytes.length-11, "US-
ASCII");
                                    padre[id] = (int)(encodedBytes[8] - '0');
                                }
                                else{
                                    //si el nodo está desactivado, se guarda el mensaje "Desactivado".
                                    data = new String(encodedBytes, 1, encodedBytes.length-1, "US-
ASCII");
                                }
                                if(encodedBytes[1]!='P') padre[id] = 0;
                            }

                            readBufferPosition = 0;

                            handler.post(new Runnable()
                            {
                                public void run()
                                {
                                    //Se actualiza el campo de texto con los datos recibidos
                                    myLabel[id].setText(data);

                                    //Se muestra el nodo encendido o apagado según corresponda
                                    if(id>0 && id<12){
                                        if(data.charAt(0) == 'T'){
                                            nodo_img[id].setImageResource(nodo_draw[(2*id)+1]);
                                            activado[id]=true;
                                        }
                                    }
                                }
                            });
                        }
                    }
                }
            }
        }
    });
}

```

```

}

else{
    nodo_img[id].setImageResource(nodo_draw[2*id]);
    activado[id]=false;
}

//Se dibujan las rutas entre routers y coordinador
if(id==1 || id==2){

    if(!activado[1] && !activado[2]){
        ruta_img[1].setImageResource(R.drawable.rcr00);}

    else if(activado[1] && !activado[2]){
        ruta_img[1].setImageResource(R.drawable.rc10);}

    else if(!activado[1] && activado[2]){
        ruta_img[1].setImageResource(R.drawable.rc01);}

    else if(activado[1] && activado[2]){
        ruta_img[1].setImageResource(R.drawable.rc11);}

}

else if(id==3 || id==4){

    if(!activado[3] && !activado[4]){
        ruta_img[3].setImageResource(R.drawable.rcr00);}

    else if(activado[3] && !activado[4]){
        ruta_img[3].setImageResource(R.drawable.cr10);}

    else if(!activado[3] && activado[4]){
        ruta_img[3].setImageResource(R.drawable.cr01);}

    else if(activado[3] && activado[4]){
        ruta_img[3].setImageResource(R.drawable.cr11);}

}

//Se dibujan las rutas entre terminales y routers
else if(id==5 || id==6 || id==7){

    if(padre[5]<1 || padre[5]>2){
        if(padre[6]<1 || padre[6]>2){
            if(padre[7]<1 || padre[7]>2)

ruta_img[0].setImageResource(R.drawable.trt000);
            else if(padre[7]==1)
                ruta_img[0].setImageResource(R.drawable.tr001);
            else if(padre[7]==2)
                ruta_img[0].setImageResource(R.drawable.tr002);
        }
        else if(padre[6]==1){
            if (padre[7]<1 || padre[7]>2)
                ruta_img[0].setImageResource(R.drawable.tr010);
            else if(padre[7]==1)
                ruta_img[0].setImageResource(R.drawable.tr011);
            else if(padre[7]==2)
                ruta_img[0].setImageResource(R.drawable.tr012);
        }
    }
}

```

```

else if(padre[6]==2){
    if (padre[7]<1 || padre[7]>2)
        ruta_img[0].setImageResource(R.drawable.tr020);
    else if(padre[7]==1)
        ruta_img[0].setImageResource(R.drawable.tr021);
    else if(padre[7]==2)
        ruta_img[0].setImageResource(R.drawable.tr022);
    }
}

else if(padre[5]==1){
    if(padre[6]<1 || padre[6]>2){
        if(padre[7]<1 || padre[7]>2)
            ruta_img[0].setImageResource(R.drawable.tr100);
        else if(padre[7]==1)
            ruta_img[0].setImageResource(R.drawable.tr101);
        else if(padre[7]==2)
            ruta_img[0].setImageResource(R.drawable.tr102);
    }
    else if(padre[6]==1){
        if (padre[7]<1 || padre[7]>2)
            ruta_img[0].setImageResource(R.drawable.tr110);
        else if(padre[7]==1)
            ruta_img[0].setImageResource(R.drawable.tr111);
        else if(padre[7]==2)
            ruta_img[0].setImageResource(R.drawable.tr112);
    }
    else if(padre[6]==2){
        if (padre[7]<1 || padre[7]>2)
            ruta_img[0].setImageResource(R.drawable.tr120);
        else if(padre[7]==1)
            ruta_img[0].setImageResource(R.drawable.tr121);
        else if(padre[7]==2)
            ruta_img[0].setImageResource(R.drawable.tr122);
    }
}

else if(padre[5]==2){
    if(padre[6]<1 || padre[6]>2){
        if(padre[7]<1 || padre[7]>2)
            ruta_img[0].setImageResource(R.drawable.tr200);
        else if(padre[7]==1)
            ruta_img[0].setImageResource(R.drawable.tr201);
        else if(padre[7]==2)
            ruta_img[0].setImageResource(R.drawable.tr202);
    }
    else if(padre[6]==1){
        if (padre[7]<1 || padre[7]>2)
            ruta_img[0].setImageResource(R.drawable.tr210);
        else if(padre[7]==1)
            ruta_img[0].setImageResource(R.drawable.tr211);
        else if(padre[7]==2)
            ruta_img[0].setImageResource(R.drawable.tr212);
    }
    else if(padre[6]==2){
        if (padre[7]<1 || padre[7]>2)
            ruta_img[0].setImageResource(R.drawable.tr220);
        else if(padre[7]==1)
            ruta_img[0].setImageResource(R.drawable.tr221);
        else if(padre[7]==2)
            ruta_img[0].setImageResource(R.drawable.tr222);
    }
}

```



```

    }
}

else if(id==8){

    if(activado[8])
        ruta_img[2].setImageResource(R.drawable.t8c1);
    else
        ruta_img[2].setImageResource(R.drawable.t8c0);

}

else if(id==9 || id==10 || id==11){

    if(padre[9]<3 || padre[9]>4){
        if(padre[10]<3 || padre[10]>4){
            if(padre[11]<3 || padre[11]>4)

ruta_img[4].setImageResource(R.drawable.trt000);
            else if(padre[11]==1)
                ruta_img[4].setImageResource(R.drawable.rt001);
            else if(padre[11]==2)
                ruta_img[4].setImageResource(R.drawable.rt002);
        }
        else if(padre[10]==3){
            if(padre[11]<3 || padre[11]>4)
                ruta_img[4].setImageResource(R.drawable.rt010);
            else if(padre[11]==1)
                ruta_img[4].setImageResource(R.drawable.rt011);
            else if(padre[11]==2)
                ruta_img[4].setImageResource(R.drawable.rt012);
        }
        else if(padre[10]==4){
            if(padre[11]<3 || padre[11]>4)
                ruta_img[4].setImageResource(R.drawable.rt020);
            else if(padre[11]==1)
                ruta_img[4].setImageResource(R.drawable.rt021);
            else if(padre[11]==2)
                ruta_img[4].setImageResource(R.drawable.rt022);
        }
    }

}

else if(padre[9]==3){
    if(padre[10]<3 || padre[10]>4){
        if(padre[11]<3 || padre[11]>4)
            ruta_img[4].setImageResource(R.drawable.rt100);
        else if(padre[11]==1)
            ruta_img[4].setImageResource(R.drawable.rt101);
        else if(padre[11]==2)
            ruta_img[4].setImageResource(R.drawable.rt102);
    }
    else if(padre[10]==3){
        if(padre[11]<3 || padre[11]>4)
            ruta_img[4].setImageResource(R.drawable.rt110);
        else if(padre[11]==1)
            ruta_img[4].setImageResource(R.drawable.rt111);
        else if(padre[11]==2)
            ruta_img[4].setImageResource(R.drawable.rt112);
    }
    else if(padre[10]==4){

```

```

        if(padre[11]<3 || padre[11]>4)
            ruta_img[4].setImageResource(R.drawable.rt120);
        else if(padre[11]==1)
            ruta_img[4].setImageResource(R.drawable.rt121);
        else if(padre[11]==2)
            ruta_img[4].setImageResource(R.drawable.rt122);
    }
}

else if(padre[9]==4){
    if(padre[10]<3 || padre[10]>4){
        if(padre[11]<3 || padre[11]>4)
            ruta_img[4].setImageResource(R.drawable.rt200);
        else if(padre[11]==1)
            ruta_img[4].setImageResource(R.drawable.rt201);
        else if(padre[11]==2)
            ruta_img[4].setImageResource(R.drawable.rt202);
    }
    else if(padre[10]==3){
        if(padre[11]<3 || padre[11]>4)
            ruta_img[4].setImageResource(R.drawable.rt210);
        else if(padre[11]==1)
            ruta_img[4].setImageResource(R.drawable.rt211);
        else if(padre[11]==2)
            ruta_img[4].setImageResource(R.drawable.rt212);
    }
    else if(padre[10]==4){
        if(padre[11]<3 || padre[11]>4)
            ruta_img[4].setImageResource(R.drawable.rt220);
        else if(padre[11]==1)
            ruta_img[4].setImageResource(R.drawable.rt221);
        else if(padre[11]==2)
            ruta_img[4].setImageResource(R.drawable.rt222);
    }
}
}
}

});

}

else
{
    readBuffer[readBufferPosition++] = b;
}
}
}
catch (IOException ex)
{
    stopWorker = true;
}
});

workerThread.start();
}

```

```

void sendData() throws IOException{

    //Método que envía por Bluetooth una petición de actualización de datos
    String msg = "A";
    msg += "\n";
    mmOutputStream.write(msg.getBytes());
    myLabel[0].setText("Petición enviada");
}

void closeBT() throws IOException{

    //Método que desconecta la comunicación Bluetooth y limpia la pantalla
    stopWorker = true;
    mmOutputStream.close();
    mmInputStream.close();
    mmSocket.close();
    mmOutputStream=null;

    openButton.setText("Conectar");
    ruta_img[0].setImageResource(R.drawable.trt000);
    ruta_img[1].setImageResource(R.drawable.rcr00);
    ruta_img[2].setImageResource(R.drawable.t8c0);
    ruta_img[3].setImageResource(R.drawable.rcr00);
    ruta_img[4].setImageResource(R.drawable.trt000);
    for(i=0; i<12; i++){
        myLabel[i].setText("");
        nodo_img[i].setImageResource(nodo_draw[2*i]);
    }
}

public void cambiarVista(){

    //Método que alterna entre las vistas Topología y Texto
    if(!texto){
        changeButton.setText("Topología");
        texto=true;

        myLabel[0] = (TextView) findViewById(R.id.txt0);
        myLabel[1] = (TextView) findViewById(R.id.txt1);
        myLabel[2] = (TextView) findViewById(R.id.txt2);
        myLabel[3] = (TextView) findViewById(R.id.txt3);
        myLabel[4] = (TextView) findViewById(R.id.txt4);
        myLabel[5] = (TextView) findViewById(R.id.txt5);
        myLabel[6] = (TextView) findViewById(R.id.txt6);
        myLabel[7] = (TextView) findViewById(R.id.txt7);
        myLabel[8] = (TextView) findViewById(R.id.txt8);
        myLabel[9] = (TextView) findViewById(R.id.txt9);
        myLabel[10] = (TextView) findViewById(R.id.txt10);
        myLabel[11] = (TextView) findViewById(R.id.txt11);

        for(i=0; i<12; i++){
            myLabel[i].setText("");
        }
    }

    else{
        changeButton.setText("Texto");
        texto=false;

        ruta_img[0].setImageResource(R.drawable.trt000);
        ruta_img[1].setImageResource(R.drawable.rcr00);

```

```

ruta_img[2].setImageResource(R.drawable.t8c0);
ruta_img[3].setImageResource(R.drawable.rcr00);
ruta_img[4].setImageResource(R.drawable.trt000);

myLabel[0] = (TextView) findViewById(R.id.label0);
myLabel[1] = (TextView) findViewById(R.id.label1);
myLabel[2] = (TextView) findViewById(R.id.label2);
myLabel[3] = (TextView) findViewById(R.id.label3);
myLabel[4] = (TextView) findViewById(R.id.label4);
myLabel[5] = (TextView) findViewById(R.id.label5);
myLabel[6] = (TextView) findViewById(R.id.label6);
myLabel[7] = (TextView) findViewById(R.id.label7);
myLabel[8] = (TextView) findViewById(R.id.label8);
myLabel[9] = (TextView) findViewById(R.id.label9);
myLabel[10] = (TextView) findViewById(R.id.label10);
myLabel[11] = (TextView) findViewById(R.id.label11);

for (i=0; i<12; i++){
    myLabel[i].setText("");
    if (i>0) nodo_img[i].setImageResource(nodo_draw[2*i]);
}
}
vf.showNext();
}}

```

A5.- AndroidManifest

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.upct.monitorredsensores"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.BLUETOOTH" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="Monitor Red Sensores"
        android:theme="@style/AppTheme" >
        <activity
            android:name="es.upct.monitorredsensores.MainActivity"
            android:label="Monitor Red Sensores" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
            />
            </intent-filter>
        </activity>
    </application>

</manifest>

```

Anexo B. Bibliografía y referencias

Bibliografía

- Michael Margolis, *Arduino Cookbook*, O'Reilly, 2011
- Maik Schmidt, *Arduino: A Quick-Start Guide*, Pragmatic Programmers, 2011
- Massimo Banzi, *Getting Started With Arduino*, O'Reilly, 2011
- Emily Gertz & Patrick Di Justo, *Environmental Monitoring With Arduino*, O'Reilly, 2011
- Robert Faludi, *Building Wireless Sensor Networks*, O'Reilly, 2011
- <http://www.arduino.cc>
- <http://developer.android.com/index.html>
- <http://stackoverflow.com>

Referencias y Enlaces.

¹ <http://www.arduino.cc/es/>

² <http://arduino.cc/es/Reference/HomePage>

³ <http://wiring.org.co/>

⁴ <http://www.processing.org/>

⁵ <http://arduino.cc/es/Main/ArduinoBoardSerialSingleSided3>

⁶ <http://arduino.cc/en/Main/Buy>

⁷ <http://arduino.cc/es/Main/Software>

⁸ <http://arduino.cc/es/Main/Hardware>

⁹ <http://arduino.cc/es/Main/Policy>

¹⁰ <http://arduino.cc/en/Main/ArduinoBoardUno>

¹¹ <http://arduino.cc/en/Main/ArduinoBoardLeonardo>

¹² <http://arduino.cc/en/Main/ArduinoBoardADK>

¹³ <http://arduino.cc/en/Main/ArduinoBoardPro>

¹⁴ <http://arduino.cc/en/Main/ArduinoBoardMini>

¹⁵ <http://arduino.cc/en/Main/ArduinoBoardNano>

¹⁶ <http://arduino.cc/en/Main/ArduinoBoardFio>

¹⁷ <http://www.atmel.com/Images/doc8161.pdf>

¹⁸ http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf

¹⁹ <http://www.digi.com/technology/rf-articles/wireless-zigbee>

²⁰ <http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf>

²¹ <http://standards.ieee.org/about/get/802/802.15.html>

²² <http://www.zigbee.org/>

²³ <http://standards.ieee.org/about/get/802/802.11.html>

24 <http://www.wi-fi.org/>

25 <http://www.meshnetworks.com/>

26 <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>

27 <http://www.digi.com/xbee/>

28 <http://www.digi.com/es/>

29 <http://www.digi.com/support/kbase/kbaseresultdet?id=2125>

30 <http://www.digi.com/support/productdetail?pid=3352&osvid=57&type=utilities>

31 http://examples.digi.com/wp-content/uploads/2012/07/XBee_ZB_ZigBee_AT_Commands.pdf

32 <http://www.bluetooth.com/Pages/Bluetooth-Home.aspx>

33 <http://www.ieee802.org/15/pub/TG1.html>

34 <https://www.bluetooth.org/en-us>

35 http://sorin-schwartz.com/white_papers/fhvsds.pdf

36 <http://www.android.com/>

37 <http://www.linux.org/>

38 <https://www.google.com/intl/es/about/>

39 <http://www.apache.org/licenses/LICENSE-2.0>

40 <http://developer.android.com/sdk/index.html>

41 <https://play.google.com/store>

42 <http://www.java.com/es/>

43 <http://www.eclipse.org/>

44 <http://developer.android.com/sdk/installing/studio.html>

45 <http://developer.android.com/training/basics/activity-lifecycle/index.html>

46 <http://developer.android.com/guide/topics/manifest/manifest-intro.html>

47 <http://developer.android.com/guide/topics/ui/declaring-layout.html>

48 <http://developer.android.com/reference/android/widget/ViewFlipper.html>

49 <http://developer.android.com/reference/android/widget/TextView.html>

50 <http://developer.android.com/reference/android/widget/ImageView.html>

51 <http://developer.android.com/reference/android/graphics/drawable/Drawable.html>

52 <http://developer.android.com/reference/android/widget/Button.html>

53 <http://arduino.cc/es/Main/ArduinoXbeeShield>

54 http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf

55 <http://www.farnell.com/datasheets/1699948.pdf>

56 <http://academic.cleardefinition.com/wp-content/uploads/2012/08/AnnotatedDiagram.pdf>

57 <http://code.google.com/p/xbee-arduino/>

58 <http://code.google.com/u/andrew.rapp@gmail.com/>

Figuras

Figura 1 - <http://www.robotshop.com/blog/en/files/arduino-microcontrollers.jpg>

Figura 2 - <http://www.arduino.dev.com/wp-content/uploads/2012/01/09950-01.jpg>

Figura 3 - <http://4.bp.blogspot.com/-nfoEZFX6OME/T3-u05afydI/AAAAAAAAABXw/gYx-NyeGM3Q/s1600/arduino2.png>

Figura 4 - Captura de pantalla del entorno de desarrollo integrado (IDE) de Arduino

Figura 5 -

[http://www.sena.com/products/industrial_zigbee/ZigBee Technology Overview for SENAWeb Page.V1.pdf#page=10](http://www.sena.com/products/industrial_zigbee/ZigBee_Technology_Overview_for_SENA_Web_Page.V1.pdf#page=10)

Figura 6 -

[http://www.sena.com/products/industrial_zigbee/ZigBee Technology Overview for SENAWeb Page.V1.pdf#page=4](http://www.sena.com/products/industrial_zigbee/ZigBee_Technology_Overview_for_SENA_Web_Page.V1.pdf#page=4)

Figura 7 – Robert Faludi, *Building Wireless Sensor Networks*, O'Reilly, 2011

Figura 8 - <http://www.matlog.com/58-816-thickbox/modules-xbee-zb-serie-2.jpg>

Figura 9 - Robert Faludi, *Building Wireless Sensor Networks*, O'Reilly, 2011

Figura 10 -

http://www.socialledge.com/sjsu/images/9/9e/CmpE146_F12_T2_XBeeAPIFrame.png

Figura 11 - Robert Faludi, *Building Wireless Sensor Networks*, O'Reilly, 2011

Figura 12 - <http://ows.edb.utexas.edu/sites/default/files/users/nqamar/system-architecture.jpg>

Figura 13 - <http://developer.android.com/training/basics/activity-lifecycle/starting.html>

Figura 14 – Topología representada mediante el programa Microsoft Visio

Figura 15 – Esquema de divisor de tensión realizado con el programa ISIS

Figura 16 – Fotografía del nodo coordinador de la red

Figura 17 – Captura del software X-CTU

Figura 18 - ftp://ftp1.digi.com/support/documentation/90000991_B.pdf#page=48

Figura 19 - Paquete de datos representado mediante el programa Microsoft Visio

Figura 20 – Captura de pantalla de la aplicación de monitorización desarrollada