ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROTOTIPO DE UN SISTEMA DE MONITORIZACIÓN AMBIENTAL MEDIANTE UNA RED DE SENSORES 802.15.4 PARA EL JARDÍN BOTÁNICO DE QUITO

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

RONALD FERNANDO ERAZO GODOY

DIRECTOR: ING. CARLOS ROBERTO EGAS ACOSTA, M.Sc.

Quito, julio 2018

AVAL

Certifico que el presente trabajo fue desarrollado por Ronald Fernando Erazo Godoy, bajo mi supervisión.

ING. CARLOS ROBERTO EGAS ACOSTA, M.Sc.

DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Ronald Fernando Erazo Godoy, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Ronald Fernando Erazo Godoy

DEDICATORIA

Dedico este trabajo a mi familia, en especial a mis padres Rocío y Fernando, a mis hermanos Joselyn, Elizabeth y Cristian y a mi abuelito César, quienes siempre fueron mi apoyo incondicional en este camino y me animaron a seguir adelante. Todo lo que he logrado y logré hacer será por ellos.

AGRADECIMIENTO

Agradezco a Dios, por haberme permitido llegar hasta este punto para poder cumplir con esta meta tan importante para mí.

Agradezco además a mi familia porque siempre fueron mi pilar y mis fuerzas para continuar y no rendirme. En especial a mis padres por siempre haberme brindado todo su apoyo incondicional y todos sus consejos para que yo sea una mejor persona. También agradezco a mis hermanas porque nunca dejaron de confiar en mí y siempre estuvieron conmigo en las buenas y en las malas. Además agradezco a mi abuelito César por todo su cariño porque sin el nada de esto hubiera sido posible.

Gracias a todos mis amigos Juan Pablo, Dani, Nico, Ely, Yanitza, Marce, Stephy, Cris, Zory, Domi, que de una u otra forma estuvieron ahí motivándome para culminar con este sueño.

De forma especial agradezco a los amigos que hice en la universidad, sobre todo a José, Fabián, Darío, David, que fueron también compañeros de lucha, con los que he compartido grandes momentos y una amistad verdadera.

Agradezco también a la familia Puente Chiluisa por todo el apoyo y confianza que brindaron, en especial a la señora Jenny que siempre supo decirme las palabras correctas para alentarme y a Gaby que por muchos años estuvo a mi lado siendo una de mis principales motivaciones para continuar.

Al Ing. Carlos Egas M.Sc,. quien fue una guía en el desarrollo de este trabajo de titulación, le agradezco por la confianza que depositada en mí.

Agradezco finalmente a todos los profesores de la Escuela Politécnica Nacional, por su paciencia y dedicación en esta difícil tarea de enseñar, gracias por todos los conocimientos que me transmitieron y por los consejos para nuestra vida profesional.

ÍNDICE DE CONTENIDO

AVAL I
DECLARACIÓN DE AUTORÍA II
DEDICATORIA III
AGRADECIMIENTO IV
ÍNDICE DE CONTENIDO V
ÍNDICE DE FIGURAS VII
ÍNDICE DE TABLAS X
ÍNDICE DE CÓDIGOS XI
RESUMEN XIII
ABSTRACT XIV
1. INTRODUCCIÓN 1
1.1 Objetivos 1
1.2 Alcance 2
1.3 Marco Teórico 2
1.3.1 Introducción a las redes de sensores inalámbricos 3
1.3.2 Elementos de una WSN 3
1.3.3 Arquitectura de una WSN 4
1.3.4 Topologías de una WSN 6
1.3.5 Estándar IEEE 802.15.4 9
1.3.6 Arquitectura de un nodo o mota 12
1.3.7 Herramientas de hardware 15
1.3.8 Herramientas de software 18
1.3.9 Lenguaje de programación NesC 22
1.3.10 Lenguaje de programación JAVA 26
1.3.11 Sistema operativo Android 29
1.3.12 Sistema gestor de base de datos 32
2. METODOLOGÍA 35
2.1 Requerimientos del prototipo 35
2.2 Componentes del sistema de monitorización ambiental 36
2.2.1 Aplicativo Android 37
2.2.2 Aplicación de escritorio 37

2.2.3 Nodo Gateway 37
2.2.4 Red IEEE 802.15.4 37
2.3 Diseño del prototipo 37
2.3.1 Diseño de la aplicación Android 37
2.3.2 Diseño de la aplicación de escritorio 44
2.3.3 Diseño del programa para los nodos sensores 53
2.3.4 Diseño del programa para el nodo Gateway 55
2.4 Implementación del prototipo 56
2.4.1 Implementación de la aplicación de escritorio 56
2.4.2 Implementación de la aplicación Android 66
2.4.3 Implementación de la base de datos 69
2.4.4 Implementación de software para el nodo sensor 72
2.4.5 Implementación de software para nodo Gateway 79
3. RESULTADOS Y DISCUSIÓN 85
3.1 Pruebas del prototipo 85
3.1.1 Instalación del software implementado en los nodos sensores y Gateway. 85
3.1.2 Transmisión de datos desde la red WSN 86
3.1.3 Pruebas de funcionamiento de la aplicación de escritorio 87
3.1.4 Prueba de funcionamiento de la aplicación Android 97
4. CONCLUSIONES y recomendaciones 100
4.1 Conclusiones 100
4.2 Recomendaciones 101
5. REFERENCIAS BIBLIOGRÁFICAS 102
6. ANEXOS 105

ÍNDICE DE FIGURAS

Figura 1.1 Elementos de una WSN....3
Figura 1.2 Arquitectura de una WSN....4
Figura 1.3 Topología tipo bus....6
Figura 1.4 Topología tipo árbol....7
Figura 1.5 Topología tipo estrella....7
Figura 1.6 Topología tipo anillo....8
Figura 1.7 Topología tipo malla....8
Figura 1.8 Redes tipo estrella y peer-to-peer....10
Figura 1.9 Capas del estándar IEEE 802.15.4....10
Figura 1.10 Arquitectura básica de un nodo....14
Figura 1.11 Estructura de la mota Iris XM2110....15
Figura 1.12 Diagrama de bloques de los elementos que constituyen la mota Iris X2110....17
Figura 1.13 Tarjeta interfaz MIB520CB....18
Figura 1.14 Estructura del mensaje message_t....20
Figura 1.15 Diagrama de bloques CTP....22
Figura 1.16 Red con diseminación de datos....22
Figura 1.17 Relación entre componentes de una aplicación ejemplo....23
Figura 1.18 Esquema de funcionamiento de una interfaz....24
Figura 1.19 Bloque signatura para componente módulo y configuración....25
Figura 1.20 Bloque Implementación para componentes módulo y configuración....25
Figura 1.21 Ejemplo de clases y objetos....27
Figura 1.22 Descripción de la clase Taxi....27
Figura 1.23 Compilador de Java....29
Figura 1.24 Arquitectura de Android....30
Figura 1.25 Arquitetura de un SGBD....33
Figura 1.26 Interacción entre un servidor y cliente MySQL....34
Figura 2.1 Esquema del prototipo de sistema de monitorización ambiental....36
Figura 2.2 Navegación por la aplicación Android....38
Figura 2.3 Pantalla de login para la aplicación Android....39
Figura 2.4 Pantalla de selección de ambientes....39
Figura 2.5 Diseño de la pantalla para el Ambiente 1....40

Figura 2.6 Diagrama de casos de uso de la aplicación Android 40
Figura 2.7 Diagrama de actividades de la aplicación Android 43
Figura 2.8 Diagrama de clases de la aplicación Android 44
Figura 2.9 Navegación por la aplicación de escritorio 45
Figura 2.10 Esquema de la ventana de registro 46
Figura 2.11 Esquema de la pantalla de monitorización de ambientes 46
Figura 2.12 Esquema de la ventana de gráficas estadísticas 47
Figura 2.13 Casos de uso de la aplicación de escritorio 47
Figura 2.14 Diagrama de actividades de la aplicación de escritorio 51
Figura 2.15 Diagrama de clases de la aplicación de escritorio 52
Figura 2.16 Diagrama de flujo para los sensores iris con la aplicación Nodo_Final 54
Figura 2.17 Diagrama de flujo para el nodo Gateway 55
Figura 2.18 Pantalla de Login de la aplicación 57
Figura 2.19 Pantalla AmbientesArtificiales 57
Figura 2.20 Pantalla de monitorización 58
Figura 2.21 Pantalla de gráficas estadísticas 59
Figura 2.22 Creación de la App MonitorizaciónJardínBotánico 66
Figura 2.23 Pantalla Login de la App 67
Figura 2.24 Menú para selección de ambiente 67
Figura 2.25 Pantallas para visualización de valores de los parámetros ambientales 68
Figura 2.26 Creación de la base de datos 70
Figura 2.27 Ejemplo de creación de tabla ambiente1 70
Figura 2.28 Tablas de la base de datos 71
Figura 2.29 Diagrama de conexiones entre interfaces y componentes de Nodo_Final 78
Figura 2.30 Compilación de la aplicación Nodo_Final 79
Figura 2.31 Diagrama de conexión entre interfaces y componentes de BaseStation 84
Figura 3.1 Carga de la aplicación en el nodo sensor 85
Figura 3.2 Carga de la aplicación del nodo Gateway 86
Figura 3.3 Recepción de valores en el nodo Gateway 87
Figura 3.4 Interfaz monitorizar 88
Figura 3.5 Ejemplo de mensaje de advertencia 88
Figura 3.6 Mensaje de envío de correo electrónico exitoso 89
Figura 3.7 Correo de alerta en el buzón de entrada 89
Figura 3.8 Ítems para la generación de la gráfica estadística 90

Figura 3.9 Ejemplo de gráfica estadística de temperatura 90
Figura 3.10 Ejemplo de valores tomados en el invernadero de clima frío.....91
Figura 3.11 Ejemplo de valores tomados en el invernadero de clima caliente.....93
Figura 3.12 Ejemplo de valores tomados en el orquideario de clima caliente..... 95
Figura 3.13 Ingreso de credenciales en la aplicación Android.....98
Figura 3.14 Validación de credenciales.....98
Figura 3.15 Ejemplo de valores monitorizados remotamente por la aplicación Android ...99

ÍNDICE DE TABLAS

Tabla 1.1 Propiedades fundamentales del estándar IEEE 802.15.4 9
Tabla 1.2 Características de las motas Iris X2110 16
Tabla 2.1 Caso de uso registro de usuarios en la base de datos 41
Tabla 2.2 Caso de uso ingreso de credenciales 41
Tabla 2.3 Caso de uso monitorizar ambientes artificiales 42
Tabla 2.4 Caso de uso registro de usuarios en base de datos 48
Tabla 2.5 Caso de uso ingreso de credenciales 48
Tabla 2.6 Caso de uso monitorizar ambientes artificiales 49
Tabla 2.7 Caso de uso generar gráficas estadísticas 50
Tabla 3.1 Tabla comparativa de valores de temperatura en el invernadero de clima frío 92
Tabla 3.2 Tabla comparativa de valores de humedad en el invernadero de clima frío 92
Tabla 3.3 Tabla comparativa de valores de luminosidad en el invernadero de clima frío 93
Tabla 3.4 Tabla comparativa de valores de temperatura en el invernadero de clima caliente 94
Tabla 3.5 Tabla comparativa de valores de humedad en el invernadero de clima caliente 94
Tabla 3.6 Tabla comparativa de valores de luminosidad en el invernadero de clima caliente 95
Tabla 3.7 Tabla comparativa de valores de temperatura en el orquideario de clima caliente 96
Tabla 3.8 Tabla comparativa de valores de humedad en el orquideario de clima caliente 96
Tabla 3.9 Tabla comparativa de valores de luminosidad en el orquideario de clima caliente 97

ÍNDICE DE CÓDIGOS

Código 2.1 Creación de variables para la comunicación serial 60

Código 2.2 Hilo para implementar la comunicación serial 60

Código 2.3 Recepción de datos desde el nodo Gateway 61

Código 2.4 Creación de método procesamiento_valores 61

Código 2.5 Conversión de valores de temperatura 62

Código 2.6 Conversión de valores de humedad 62

Código 2.7 Conversión de valores de luminosidad 62

Código 2.8 Presentación de valores en pantalla 63

Código 2.9 Creación de método conectado 64

Código 2.10 Métodos para uso de procedimientos almacenados 64

Código 2.11 Evento del botón Graficar 65

Código 2.12 Consulta de valores para generar gráfica estadística 65

Código 2.13 Método para conexión con la base de datos 69

Código 2.14 Consulta de valores a la base de datos 69

Código 2.15 Procedimiento almacenado sp_Nodo1 71

Código 2.16 Declaración de constantes y estructura de datos 73

Código 2.17 Interfaces utilizadas por el fichero MtsTesterP.nc 74

Código 2.18 Declaración de variables y del evento Boot 74

Código 2.19 Eventos para lectura de datos 75

Código 2.20 Empaquetado de datos 75

Código 2.21 Evento timer y transmisión de datos 76

Código 2.22 Listado de componentes utilizados por Nodo_Final 76

Código 2.23 Enlazado entre componentes e interfaces 77

Código 2.24 Fichero Makefile 77

Código 2.25 Interfaces utilizadas en BaseStationP 80

Código 2.26 Evento Boot.booted 80

Código 2.27 Evento para encendido de radio 81

Código 2.28 Evento para recepción de datos por radio 81

Código 2.29 Comunicación serial 82

Código 2.30 Enlazado de componentes en BaseStationC 83

Código 2.31 Fichero Makefile de BaseStation 83

RESUMEN

Hoy en día, las redes de sensores inalámbricos se han convertido en un campo de investigación importante gracias a los grandes beneficios que presentan debido a su bajo costo, bajo consumo energético, factor de escalabilidad, capacidad de obtener información a través de sus nodos sensores inteligentes y reenviarla de forma inalámbrica a un centro de coordinación, lo que permite desarrollar un sinfín de aplicaciones en muchas áreas como: seguridad, agricultura, medicina, medio ambiente, etc.

En este proyecto de titulación se describe el desarrollo e implementación de un prototipo de sistema de monitorización ambiental encargado de medir parámetros ambientales como son: la temperatura, humedad y luminosidad, los cuales luego serán guardados en una base de datos. El sistema usa el protocolo IEEE 802.15.4 para la transmisión de la información mediante un direccionamiento IPv4, además cuenta con una aplicación de escritorio y una aplicación Android, ambas bajo el lenguaje de programación Java, las cuales permiten visualizar al usuario los datos almacenados.

El trabajo consta de 4 capítulos en donde se describen las etapas que se siguieron para la realización del prototipo. En el primer capítulo se exponen los fundamentos teóricos sobre WSN (*Wireless Sensor Network*), así como el sistema operativo TinyOS y los diferentes lenguajes de programación usados para la elaboración del sistema de monitorización. En el segundo capítulo se describe el diseño de cada uno de los componentes que conforman el prototipo. En el tercer capítulo se presenta la implementación y las pruebas realizadas sobre todos los componentes que integran el sistema de monitorización ambiental. Finalmente el último capítulo cuenta con las conclusiones y recomendaciones obtenidas en base a los resultados logrados en el proyecto de titulación.

PALABRAS CLAVE: IEEE 802.15.4, WSN, Prototipo, Monitorización, Temperatura, Humedad, Luminosidad, Java.

ABSTRACT

Nowadays, wireless sensor networks have become an important field of research thanks to the great benefits they have due to their low cost, low energy consumption, scalability factor, ability to obtain information through their nodes and resend it wirelessly to a coordination center, which allows to develop an endless number of applications in many areas such as: security, agriculture, medicine, environment, etc.

The present final career project presents a prototype of an environmental monitoring system, which measures environmental parameters such as: the temperature, humidity and luminosity, to later be registered in a data base. The system uses the IEEE 802.15.4 protocol for the information transmission with IPv4 addressing. It also has a desktop application and an Android application, both under the Java programming language, which allow the users visualize the stored data.

The work consists of 4 chapters, where they describe the stages followed to reach the realization of the prototype. In the first chapter, the theoretical foundations of WSN (Wireless Sensor Network), as well as the operating system TinyOS and the different programming languages used for the development of the monitoring system are exposed. In the second chapter, the design of each component that be part of the prototype is described. In the third chapter, the implementation is presented. Moreover, the performance tests are carried out over all the components that be part of the environmental monitoring system. Finally, in the last chapter, the conclusions and recommendations obtained based in the results achieved in the project are exposed.

KEYWORDS: IEEE 802.15.4, WSN, Prototype, Monitoring, Temperature, Humidity, Luminosity, Java.

1. INTRODUCCIÓN

En la actualidad las WSN (*Wireless Sensor Network*) juegan un papel importante para el desarrollo de la tecnología, debido a que pueden usarse en una amplia gama de aplicaciones, además de proporcionar una integración de varios servicios; eliminando la necesidad de un medio físico para la comunicación de los diferentes dispositivos, lo que genera una mayor flexibilidad a la red.

Una de estas aplicaciones es la monitorización de entornos, en los cuales se requieren condiciones ambientales específicas. Tal es el caso del Jardín Botánico de Quito, en donde se necesitan determinadas características ambientales para el correcto desarrollo de sus trabajos de investigación, además de la preservación de la flora que éste alberga.

El principal problema que presenta este centro, es que no cuenta con un control automático ambiental, por ello, se plantea una solución basada en una red de nodos inalámbricos IEEE 802.15.4, que permita la realizar la monitorización automatizada de los diferentes ambientes artificiales con los que cuenta el centro.

1.1 Objetivos

El objetivo general de este Proyecto Integrador es: desarrollar un prototipo de sistema de monitorización ambiental con una red de sensores inalámbricos basados en IEEE 802.15.4.[1]

Los objetivos específicos de este Proyecto Integrador son:

- Recolectar y analizar información acerca de sensores IEEE 802.15.4 y del lenguaje de programación NesC[2] (*Network Embedded Systems C*) para determinar su funcionamiento.
- Desarrollar el software para los diferentes componentes que conforman el prototipo de monitorización ambiental cumpliendo con los requerimientos del usuario.
- Implementar el prototipo en base al diseño realizado.
- Analizar los resultados de las pruebas realizadas para comprobar el correcto funcionamiento del prototipo.

[1] IEEE 802.15.4: Estándar utilizado para la transmisión de datos sobre redes inalámbricos de área personal de baja velocidad [24].
[2] NesC: Lenguaje de programación orientado a componentes, que usa limitados recursos de memoria.

1.2 Alcance

El prototipo de un Sistema de Monitorización Ambiental constará de los siguientes elementos: un nodo *Gateway* (puerta de acceso), tres nodos sensores IEEE 802.15.4 y una computadora en la cual se tendrá la aplicación que permitirá capturar los datos, procesarlos y almacenarlos.

Los nodos inalámbricos tendrán conectados 3 sensores cada uno, los cuales tomarán medidas de temperatura, humedad y luminosidad, los cuales serán enviados a una aplicación de escritorio para ser analizados y almacenados en una base de datos. Si los valores obtenidos por los sensores están fuera del rango normal se generará una alerta, la cual será visualizada en la aplicación y enviada a los correos de las personas encargadas del centro. Dentro de la aplicación también se generarán gráficas estadísticas, que permitirán visualizar si han existido cambios bruscos en los parámetros ambientales, estas graficas estarán dentro de determinados intervalos de tiempo y se usarán los valores que se encuentren almacenados en la base de datos.

Se dispondrá de una aplicación Android que permita realizar una monitorización remota de los parámetros antes mencionados. Para esto, se solicitará a las personas del centro, los valores con los que se definirán los rangos permitidos para la generación de alertas dentro de los aplicativos.

El prototipo finalmente será puesto a prueba dentro de las instalaciones del Jardín Botánico de Quito para comprobar el correcto funcionamiento tanto en hardware como en software.

1.3 Marco Teórico

El marco teórico presentará una descripción concisa de los recursos de hardware y software involucrados para el desarrollo del proyecto. Se iniciará abordando conceptos relacionados con las redes de sensores inalámbricos, además del estándar IEEE 802.15.4 que es la base para este tipo de tecnología. Se presentará también, el concepto de mota, que es el dispositivo fundamental para una WSN. Posteriormente se hablará sobre el sistema operativo TinyOS[3], que permitirá trabajar con los sensores seleccionados. Se finalizará mencionando los diferentes lenguajes de programación utilizados para creación de los aplicativos que interactuarán con el usuario.

[3] TinyOS: Es un sistema operativo de código abierto basado en componentes, para redes de sensores inalámbricos.

1.3.1 Introducción a las redes de sensores inalámbricos

Una red de sensores inalámbricos se puede definir como un conjunto de dispositivos que se comunican entre sí sin que exista la necesidad de un medio guiado.

Las redes de sensores inalámbricos brindan grandes beneficios debido a su bajo costo, flexibilidad de crecimiento, facilidad de instalación y la existencia de nodos sensores inteligentes. No solo pueden emplearse en áreas de interés engorrosas y peligrosas para monitorear o controlar la región, sino que también se permiten automatizar tareas que requerirían de un individuo, para realizarlas de forma manual.

Los sensores inteligentes actuales poseen una gran capacidad de procesamiento y permiten realizar tareas de detección, sensado de parámetros específicos, reenvío de datos, automatización de procesos, entre otras, y todos alimentados a través de una batería que elimina la necesidad del cableado.

Existen numerosas aplicaciones que aprovechan las WSN como soluciones de bajo costo para observar el hábitat y el medioambiente, desde vigilancia militar y civil haciendo uso de aplicaciones de detección y seguimiento de objetivos, agricultura de precisión, monitorización de pacientes en atención médica hasta aplicativos residenciales para gestión energética y eficiencia en redes vehiculares [1].

1.3.2 Elementos de una WSN

De manera general, se puede decir que una red de sensores inalámbricos está constituida por: sensores, nodo sensor, nodo *Gateway* y estación base. La Figura 1.1 presenta los elementos principales de una WSN [2]. A continuación se revisa cada uno de ellos.

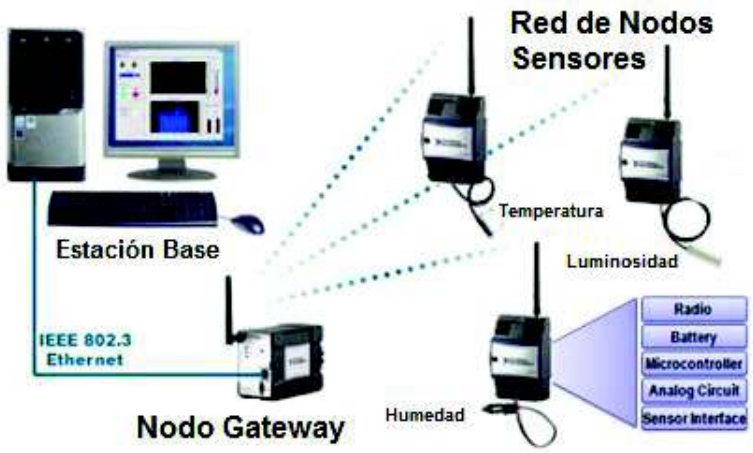

Figura 1.1 Elementos de una WSN [2]

- **Sensores:** Son dispositivos que van conectados a cada nodo a través de una conexión física. En este caso los sensores que se utilizará permiten medir temperatura, humedad y luminosidad.
- **Nodos:** Es aquel elemento que se encarga de recolectar los datos de los sensores y transmitirlos a un equipo donde se almacena la información. Los nodos pueden ser utilizados para hacer simplemente monitoreo o también para realizar tareas de control. En el presente proyecto se utilizará el lenguaje NesC para programar la funcionalidad de los nodos.
- ***Gateway*:** Su función es interconectar la WSN con una red exterior o con otra red de sensores. Además se encarga de recolectar toda la información de una red WSN y enviarla a la estación base.
- **Estación Base:** Encargada de almacenar y presentar toda la información que fue recolectada de la red al usuario, a través de una interfaz gráfica.

1.3.3 Arquitectura de una WSN

La arquitectura WSN más común sigue el modelo de arquitectura OSI[4] (*Open System Interconnection*). La arquitectura de la WSN incluye cinco capas y tres capas niveles como se muestra en la Figura 1.2. Las cinco capas definidas son: Aplicación, Transporte, Red, Enlace de Datos y la Capa Física. Mientras que los tres niveles son: administración de energía, administración de movilidad y administración de tareas. Estas capas de la WSN se utilizan para que los sensores trabajen juntos y también para aumentar la eficiencia completa de la red [3].

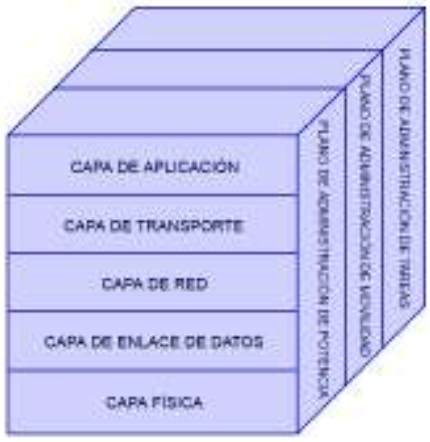

Figura 1.2 Arquitectura de una WSN

[4] El modelo OSI es usado en la comunicación de redes, y describe un conjunto de guías generales de operación para que dispositivos puedan comunicarse entre sí[25].

a. Capa Aplicación

Responsable de la gestión del tráfico y de proporcionar el software para las diferentes aplicaciones que traducen los datos de forma comprensible o de enviar consultas para obtener cierta información. Las redes de sensores pueden ser implementadas en diversas aplicaciones en diferentes campos, por ejemplo; militares, salud, ambientales, agrícolas.

b. Capa Transporte

La función de la capa transporte es proporcionar confiabilidad y evitar congestión para lo cual hace uso de protocolos destinados a realizar esta función. Estos protocolos usan mecanismos diferentes para reconocimiento de pérdida y recuperación de pérdida. La capa de transporte es exactamente necesaria cuando se planifica un sistema para contactar otras redes.

Proporcionar una conexión salto a salto es más eficiente, en términos de consumo de energía, que una conexión extremo a extremo y esa es una de las razones principales por las que TCP[5] (*Transmission Control Protocol*) no es apto para WSN. En general, los protocolos de la capa transporte se pueden separar en: controlados por paquetes, controlados por eventos. Hay algunos protocolos populares en la capa de transporte, a saber, STCP[6] (*Sensor Transmission Control Protocol*), PORT[7] (*Price-Oriented Reliable Trasport Protocol*) y PSFQ[8] (*Pump Slowly, Fetch Quickly*) [3].

c. Capa de Red

La función principal de la capa de red es el enrutamiento, tiene muchas tareas dependiendo de la aplicación, pero en realidad, su objetivo primordial es la conservación de energía; la memoria y buffers limitados y el hecho de que el sensor no tiene un identificador universal, hacen que estos deban ser auto organizados, lo que genera un impacto en la energía requerida.

La idea básica del protocolo de enrutamiento es definir una ruta confiable y rutas redundantes de acuerdo con una determinada escala llamada métrica, que difiere de uno a otro. Hay muchos protocolos disponibles para esta capa y se pueden dividir en: el

[5] TCP: El protocolo de control de transmisión, es uno de los principales protocolos de la capa de transporte del modelo TCP/IP[26].
[6] STCP: El protocolo de control de transmisión de sensores, es una ligera variación del protocolo TCP y ofrece funciones de control de congestión y confiabilidad en la transmisión.
[7] PORT: Es un protocolo confiable orientado a precios, que se asegura que el receptor reciba suficiente información de los fenómenos físicos.
[8] PSFQ: Es el primer protocolo desarrollado para proporcionar confiabilidad de datos en redes de sensores. Es un protocolo de transporte simple, escalable y robusto.

enrutamiento plano (por ejemplo difusión directa) y el enrutamiento jerárquico (por ejemplo LEACH, *The Low-Energy Adaptive Clustering Hierarchy*) o se puede dividir en: controlados por tiempo, consultas y eventos [4] .

d. Capa de Enlace de Datos

La capa de enlace es responsable de la multiplexación de flujos de datos, detección de tramas de datos, MAC y control de errores, asegura la confiabilidad de punto-punto o punto-multipunto [5].

e. Capa Física

La capa física proporciona una interfaz para transmitir un flujo de bits en un medio físico, es responsable de la selección de frecuencia, generación de frecuencia portadora, detección de señal, modulación y encriptación de datos [5].

1.3.4 Topologías de una WSN

El desarrollo y la implementación de WSN han llevado las topologías de red tradicionales en nuevas direcciones. Diferentes topologías de red de sensores inalámbricos son Bus, Árbol, Estrella, Anillo, y Malla.

a. Bus

En esta topología, un nodo que desea transmitir a otro en la red, lo hace enviando un mensaje de difusión que todos los demás nodos ven, pero solo el destinatario previsto realmente lo acepta y procesa, Figura 1.3. La topología de bus es fácil de instalar, pero tiene desventajas como la congestión del tráfico y la comunicación a través de una única ruta. Sin embargo, las redes de bus pueden funcionar bien con un número limitado de nodos [6].

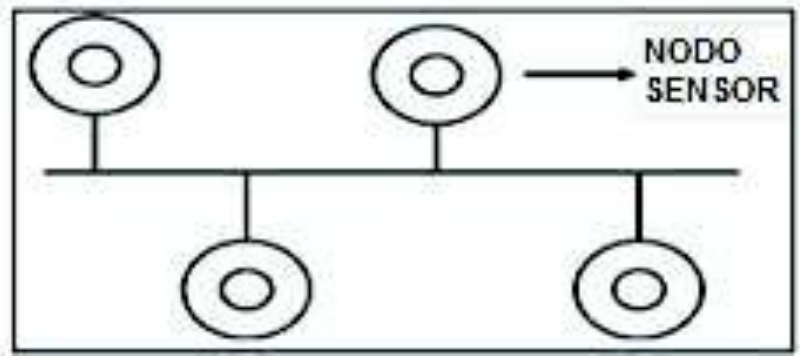

Figura 1.3 Topología tipo bus [6]

b. Árbol

La red usa un concentrador central llamado nodo raíz como el enrutador de comunicación principal. En la jerarquía, el concentrador central se encuentra un nivel más abajo del nodo raíz. Este nivel inferior forma una red tipo estrella. La red de árbol se puede considerar un híbrido de las topologías de red estrella y punto-punto, como se observa en la Figura 1.4 [6].

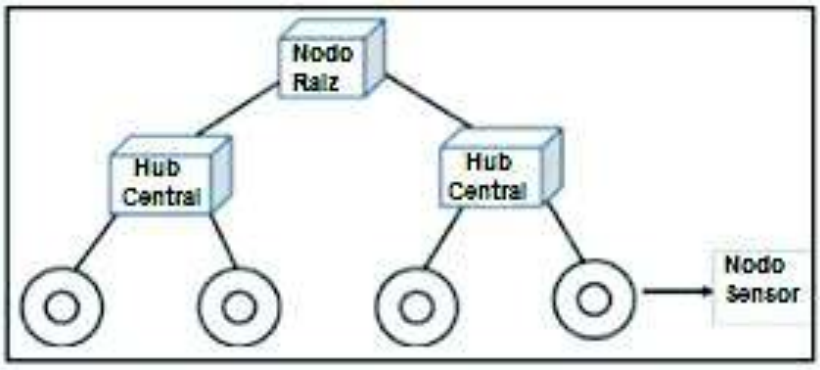

Figura 1.4 Topología tipo árbol [6]

c. Estrella

Las redes en estrella están conectadas a un concentrador de comunicación centralizado (receptor) y los nodos no se pueden comunicar directamente entre sí.

Toda la comunicación debe enrutarse a través del concentrador centralizado. Cada nodo es entonces un "cliente" mientras que el concentrador central es el "servidor o receptor", que se presenta en la Figura 1.5 [6].

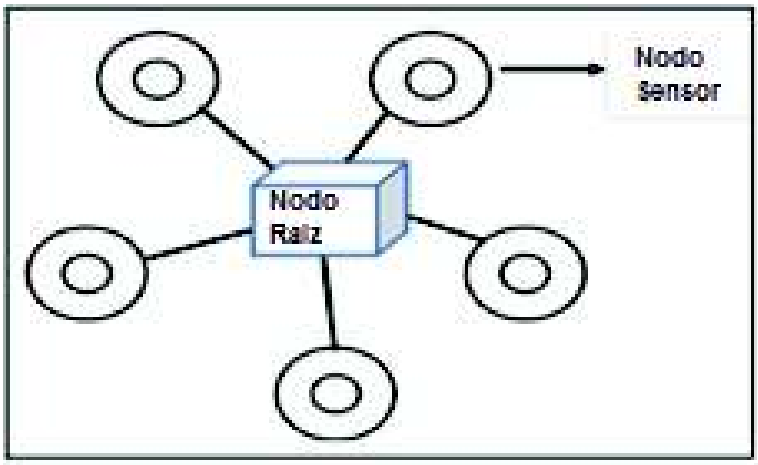

Figura 1.5 Topología tipo estrella [6]

d. Anillo

En una red en anillo, cada nodo tiene exactamente dos vecinos para fines de comunicación como se aprecia en la Figura 1.6. Todos los mensajes viajan a través de un anillo en la misma dirección ("en sentido horario" o "antihorario"). Una falla en el nodo rompe el ciclo y puede acabar con toda la red [6].

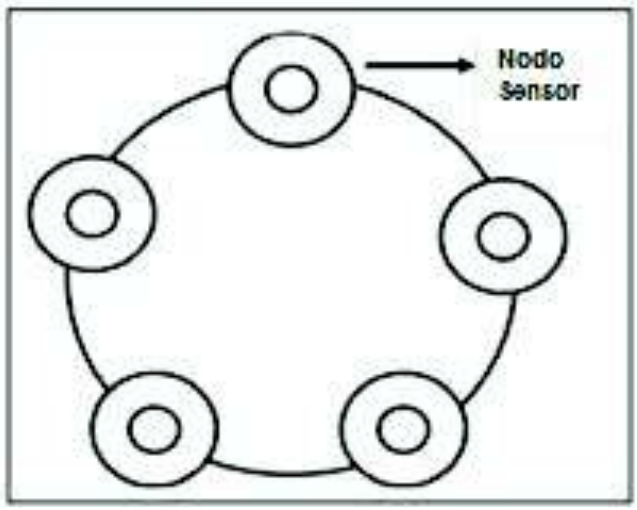

Figura 1.6 Topología tipo anillo [6]

e. Malla

Estas topologías implican que el mensaje puede tomar cualquiera de varias rutas desde el origen hasta el destino, lo cual se muestra en la Figura 1.7. Una red de malla en la que cada nodo se conecta entre sí se llama completa y en la que algunos dispositivos (nodos) se conectan solo indirectamente a otros, toma el nombre de parcial [6].

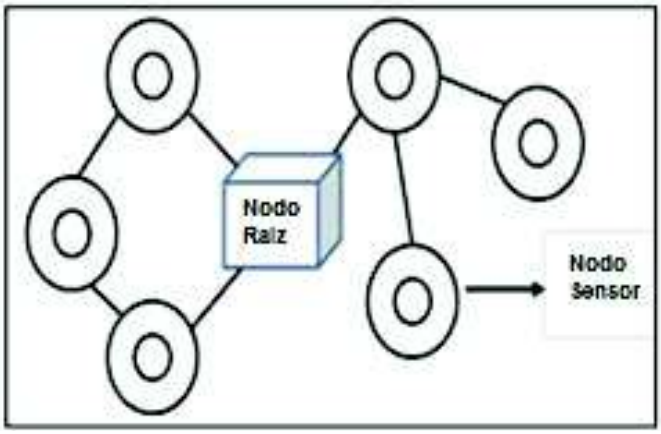

Figura 1.7 Topología tipo malla [6]

1.3.5 Estándar IEEE 802.15.4

Las características principales del estándar son: mínima complejidad, muy bajo consumo de energía, bajo costo, capacidades de procesamiento reducidas, flexibilidad de red. Es un estándar que se puede utilizar para varias aplicaciones que requieren una tasa baja en la transmisión de datos [6].

En la Tabla 1.1 se pueden apreciar las propiedades fundamentales del estándar IEEE 802.15.4.

Tabla 1.1 Propiedades fundamentales del estándar IEEE 802.15.4

PROPIEDAD	RANGO	
Tasa de transmisión de datos	868 - 868,6 MHz	20 kbps
	902 - 928 MHz	40 kbps
	2,4 – 2,4835 GHz	250 kbps
Alcance	20 – 30 m	
Latencia	Por debajo de 15 ms	
Canales	868 / 915 MHz	11 canales
	2,4 GHz	16 canales
Bandas de frecuencia	Tres PHY: 868/915 MHz y 2,4 GHz	
Direccionamiento	Cortos de 8 bits a 64 bits IEEE	
Técnica de acceso al medio	CSMA-CA y CSMA-CA ranurado	
Temperatura	-40° a 85° C	

a. Capa de Red

En las redes tradicionales, la capa de red es responsable de la construcción de la topología y mantenimiento de la red, así como de las tareas necesarias de direccionamiento y seguridad. Estos mismos servicios existen para las redes inalámbricas; sin embargo, representan un reto mayor por la necesidad de ahorro de energía que exigen estas redes.

Las redes que se construyan dentro de esta capa del estándar IEEE 802.15.4 deben auto organizarse y auto mantenerse para que de esta forma se reduzcan los costes totales para facilitar su uso.

El estándar IEEE 802.15.4 soporta la topología tipo estrella y la topología peer-to-peer, presentadas en la Figura 1.8. La topología a escoger es una elección de diseño y va a estar dado por la aplicación a la que se desee orientar. Algunas aplicaciones requieren de

conexiones de baja potencia de tipo estrella, mientras que otras requieren de una mayor área de cobertura por lo que es necesario implementar una red *peer-to-peer* [6].

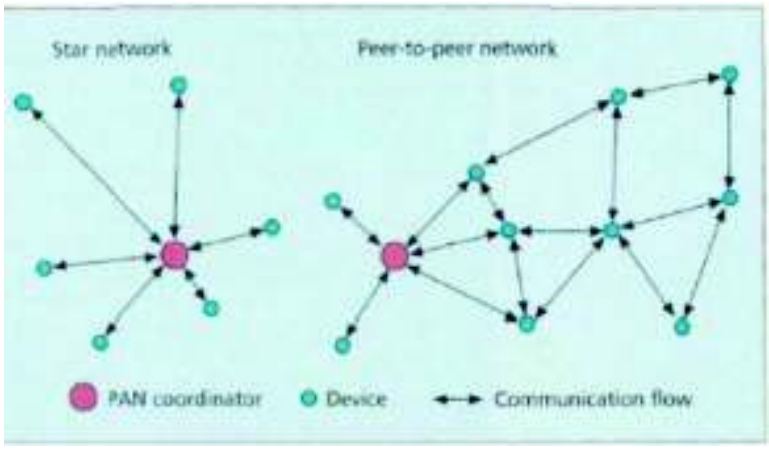

Figura 1.8 Redes tipo estrella y peer-to-peer [6]

b. Capa de Enlace de Datos

El proyecto IEEE 802 divide al DLL (*Data Link Layer*) en dos subcapas: la subcapa MAC (*Medium Access Control*) y la subcapa LLC (*Logical link control*). La LLC es común a todos los estándares 802, mientras que la subcapa MAC depende del hardware y varía respecto a la implementación física de esta capa [6].

La Figura 1.9 ilustra la forma en que están estructuradas las capas del estándar IEEE 802.15.4.

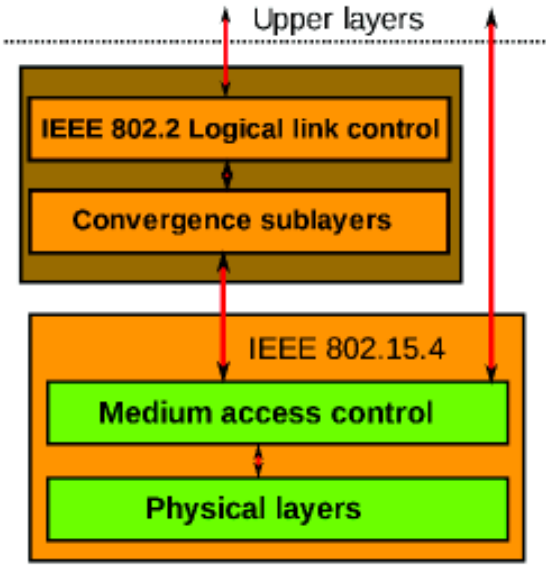

Figura 1.9 Capas del estándar IEEE 802.15.4 [7]

b.1 Subcapa MAC

Las principales características de la subcapa MAC son: la asociación y la disociación, reconocimientos de entrega de trama, mecanismos de acceso al canal, validación de trama, garantía del manejo de las ranuras de tiempo y manejo de guías. Contiene 26 primitivas[9], que permiten la transferencia de datos, tanto entrantes como salientes, así como la gestión por entidades de nivel superior.

La subcapa MAC proporciona acceso a las capas superiores a través de dos puntos de acceso de servicio (SAP). Los datos se gestionan a través del MAC-SAP, mientras que las funciones de control y monitoreo se acceden a través de la interfaz de manejo de identidades de la capa MAC, llamada MLME-SAP. Además, es responsable de generar *beacons* que permiten que los dispositivos encuentren una red existente o, en el caso de redes TDMA[10] (*Time Division Multiple Access*), que proporcionen una indicación de tiempo para que los dispositivos de los clientes accedan al canal durante períodos basados en contención y sin contención [6].

En otras palabras la capa MAC maneja la capacidad de un dispositivo para encontrar una red, asociarse a esa red y desasociarse según sea necesario. Tras el encendido, una entidad de capa superior ordena al transceptor que comience una exploración en cada canal disponible en una búsqueda de una red existente.

c. Capa PHY o Física

Esta capa realiza las siguientes funciones: transmisión y recepción de datos, evaluación y selección del canal, acceso al medio mediante CSMA/CA[11] *(Carrier Sense Multiple Access with Collision Avoidance*) y el encendido y apagado del transceptor.

PHY contiene primitivas específicas que administran el canal de radio y controlan el flujo de datos. El PHY utiliza CSMA/CA para acceder al canal de radio. Esto significa que un dispositivo con datos para transmitir escuchará primero el canal y, si el canal está libre, transmitirá su paquete. Sin embargo, si el canal está ocupado, ya sea debido a la transmisión de otra estación 802.15.4, o debido a la interferencia de una estación que no

[9] Primitivas: Son llamadas entrantes o salientes en cada una de las capas que sirven para solicitar servicios, devolver resultados, confirmar las peticiones, etc.
[10] TDMA: Es una técnica de multiplexación para transmitir señales digitales en ranuras alternas de tiempo.
[11] CSMA/CA: Es un protocolo de acceso múltiple a redes de bajo nivel con detección de portadora y prevención de colisiones, que permite a varias estaciones utilizar el mismo medio.

es 802.15.4 (horno de microondas, punto de acceso Wi-Fi, etc.), la radio esperará por un período de tiempo aleatorio antes de volver a verificar el canal para la ocupación.

Esta capa se divide en dos bandas de frecuencia para operar [6]:

- La PHY que trabaja en la banda de 2.4 GHz, que es específicamente para operaciones industriales, médicas y científicas, conocidas como ISM (Industrial, Scientific and Medical).
- La PHY que está en operación en la banda de 868/915 MHz, en donde la banda de 868 MHz es utilizada en Europa y la banda de 915 MHz está disponible en Estados Unidos para ISM.

d. Dispositivos de una WPAN según 802.15.4

El estándar puede ser configurado para manejar dos tipos de dispositivos:

- **Dispositivo de función completa (*Full Function Device*, FFD):** Cuenta con todas las características especificadas en el estándar IEEE 802.15.4. Puede funcionar como un enrutador o un dispositivo final o tener ambas funciones. Pasa la mayor parte del tiempo en estado activo atendiendo los enlaces de los demás integrantes de la red. En este tipo de dispositivos se considera que no pueden llegar a presentar problemas en su alimentación ya que suelen estar conectados a fuentes de corriente directa.
- **Dispositivos de funciones reducidas (*Reduced Function Device*, RFD):** Tiene tareas limitadas, muy específicas, como la recolección de datos o la monitorización de eventos externos. Suele contar con una fuente de alimentación limitada. Otra característica es que pasa la mayor parte del tiempo en estado inactivo; de tal manera que sólo se activa cuando se le realiza una consulta.

1.3.6 Arquitectura de un nodo o mota

El concepto que se tiene de nodo, o mota, en una WSN surge del paradigma del “cómputo ubicuo”; que propone un punto de vista diferente sobre las computadoras respecto a cómo se lo manejaba en las últimas décadas.

En este nuevo contexto, que constituye la tercera era del cómputo moderno, se define a las computadoras como pequeños dispositivos portátiles integrados con el entorno y comunicándose unos con otros entre sí con la finalidad de cumplir determinadas tareas de manera individual. Como resultado se plantean la interacción con un gran número de

dispositivos, para poder realizar tareas cotidianas en muchas áreas, y en la mayoría de los casos de manera imperceptible [8].

Debido a una demanda de herramientas cada vez menos intrusivas; se plantea retos muy importantes, ya que la capacidad del dispositivo no es el único desafío, sino también la implementación de la red al momento de diseñar la arquitectura y determinar las demás herramientas de software que serán usadas para la resolución del problema que se proponga resolver.

Aspectos como la topología que tendrá la red, la colocación de los nodos, los posibles obstáculos de comunicación, etc. son parámetros que deben ser contemplados, pues todo esto determinará como se accederá a los datos que sean recolectados por la WSN y también definirá la capacidad de adaptación que tenga el sistema al momento de dar una determinada solución [9].

Varios tipos de arquitecturas de nodos han sido planteadas hasta el momento por distintas compañías; y aunque la arquitectura de una mota depende de la funcionalidad precisa que tendrá su despliegue, del tipo de datos para el cual fue diseñado y de la manera en que será integrado a la red, actualmente el diseño de un nodo busca cubrir tres puntos principales:

- El manejo eficiente de la energía.
- Que sea de tamaño reducido.
- Que su costo de fabricación sea lo más bajo posible.

Siendo el manejo eficiente de la energía el aspecto de mayor prioridad, y por el cual se han realizado varios estudios con la finalidad de generar protocolos que permitan su optimización de este parámetro ya que es el recurso del cual depende el tiempo de funcionamiento de la mota [10], donde la mayoría de los escenarios suponen fuentes de alimentación limitada.

Actualmente en el mercado ya existen motas que permite la integración de diversas funciones como la recolección de datos, así como el procesamiento y comunicación con otros dispositivos; todo ello en una pequeña área de circuito o incluso en sólo un chip. Sin embargo, el hardware básico de un nodo, se puede visualizar en la Figura 1.10, el cual incluye los siguientes componentes:

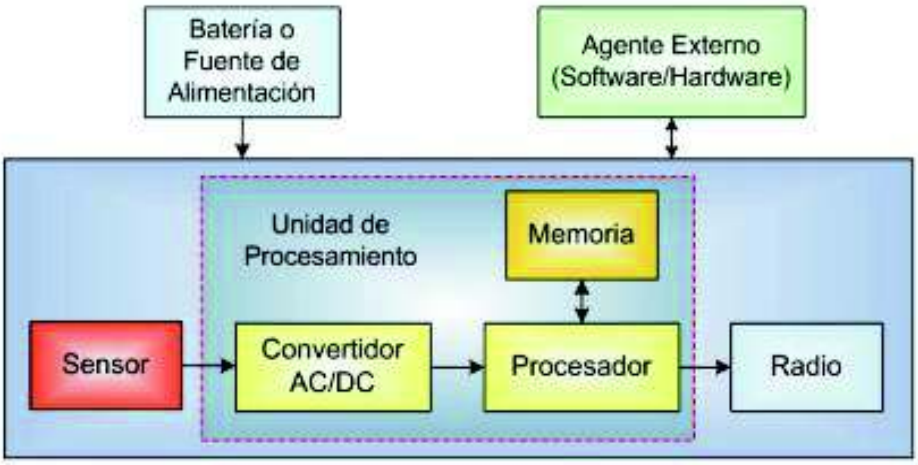

Figura 1.10 Arquitectura básica de un nodo

- **Sensor:** El sensor es un transductor que recopila la información del entorno que será monitorizado para su posterior procesamiento por parte de la unidad de procesos. Este elemento suele ser modular para dar la posibilidad de caracterizar múltiples variables.
- **Unidad de procesamiento**: Se encarga de procesar tanto la información detectada por el sensor, como las tareas proporcionadas por los protocolos de comunicación. Suele ser un microcontrolador que cuenta con un convertidor analógico digital y una memoria de programación. Esta última se considera un recurso muy escaso.
- **Módulo de radio**: Es el módulo encargado de proporcionar la comunicación inalámbrica del dispositivo para transmitir datos a otros elementos de la red. Debido a que el suministro de energía es limitado, los módulos de radio y sus antenas suelen ser de baja potencia.
- **Fuente de alimentación o de energía:** Uno de las características principales de los nodos es la autonomía que éstos poseen, es por ello que los nodos están equipados con una fuente de energía compuesta por una o más baterías. Se han realizado grandes esfuerzos por reducir el consumo de energía haciendo uso de componentes de hardware de baja potencia y de algoritmos de gestión de transmisión de datos y energía.
- **Agente**: Por lo general un agente es un programa de software, el cual es considerado como un elemento externo autónomo capaz de detectar cualquier evento o suceso que tiene lugar en el entorno en el que se encuentra, con la finalidad de cumplir un objetivo.

- **Interfaces de comunicación**: Los microcontroladores modernos cuentan con un cierto número de dispositivos periféricos, que pueden estar integrados en un mismo chip, junto con la memoria y el CPU (*Central Processing Unit*). Estos dispositivos, como el módulo de radio y los sensores, están conectados a través de una o más interfaces periféricas.

Con la finalidad de administrar de mejor manera la energía, los nodos pueden trabajar en tres estados diferentes:

- ***Sleep* (dormido)*:*** Es aquel, en el que el nodo se encuentra en un estado de hibernación, y mientras mayor tiempo pase en este estado su consumo de energía será menor.
- ***Wakeup* (despierto)*:*** Se genera a través de un estímulo o interrupción programada y hará que el nodo pase del estado de sleep al estado activo.
- ***Active* (activo)*:*** Es el estado en el que se consume mayor energía, debido a que en este estado se realiza la transmisión y recepción de la información.

1.3.7 Herramientas de hardware

Las herramientas principales de hardware para el prototipo son: el nodo sensor IRIS XM2110 y la tarjeta MIB520CB, las cuales se detallarán a continuación.

a. Mota Iris XM2110

Las motas IRIS XM2110 son la última generación de la marca MEMSIC y fueron elegidas para la realización de este proyecto. En la Figura 1.10, se puede observar el diseño que tiene esta mota [11].

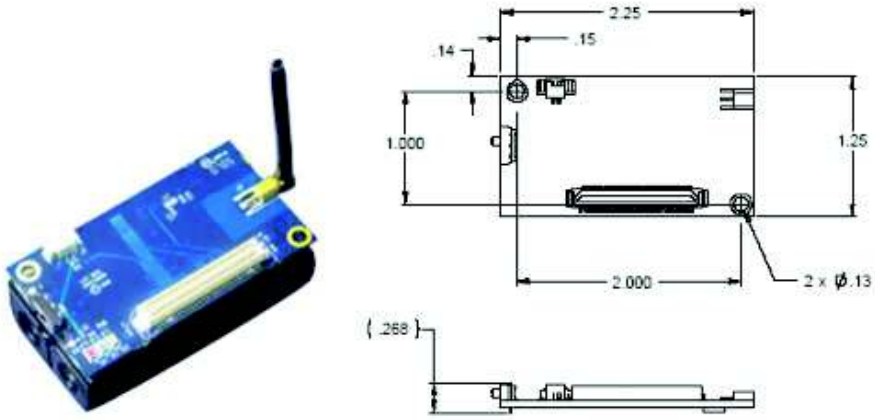

Figura 1.11 Estructura de la mota Iris XM2110 [11]

La Tabla 1.2 muestra las principales características de las motas Iris X2110.

Tabla 1.2 Características de las motas Iris X2110

Mota Iris X2110		
	Características	Observaciones
Procesador		
Memoria flash	128 KBytes	
RAM	512 KBytes	
Comunicación serial	UART	0-3 V Niveles de transición
Consumo de corriente	8 mA (Active Mode)	8 µA (Sleep Mode)
Banda de frecuencia	2045 MHz a 2480 MHz	Banda ISM
Transceptor RF		
Tasa de transmisión de datos	250 kbps	
Potencia RF	3 dBm	
Alcance en ambientes abiertos	> 300m	Antena dipolo ¼ de onda
Alcance en ambientes cerrados	> 50m	Antena dipolo ¼ de onda
Consumo de Corriente	16 mA	Modo recepción
	10 mA	Tx, -17 dBm
	13 mA	Tx, -3 dBm
	17 mA	Tx, 3 dBm
Electromecánica		
Batería	Baterías AA x2	
Fuente externa	2,7 V – 3,3 V	
Interface de usuario	3 LEDs	Amarillo, rojo y verde
Tamaño (pulg)	2,25 x 1,25 x 0,25	
Peso (gramos)	18	
Conector de Expansión	51 pines	

Trabajan en la banda de 2.4 GHz a 2.4835 GHz, tienen incorporado un microcontrolador Atmega1281, además un transceptor que es compatible con el microcontrolador Atmel

RF230, es un nodo que trabaja en bajo el estándar IEEE 802.15.4 y ZigBee[12]. Estas características ayudan triplicar el alcance de la señal y duplican la memoria de programa que tenía la generación anterior, denominadas MICAz. En la Figura 1.12 se muestra el diagrama de bloques con los elementos que constituyen la mota Iris X2110.

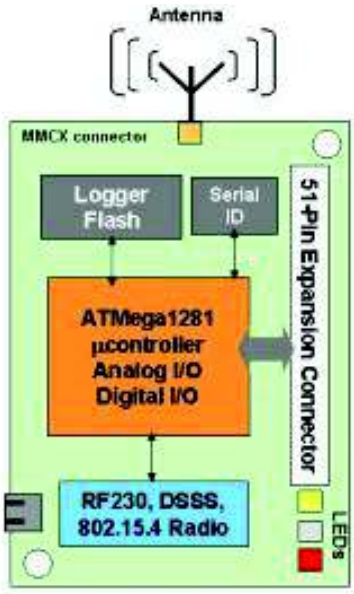

Figura 1.12 Diagrama de bloques de los elementos que constituyen la mota Iris X2110 [11]

Algunas características de la mota tipo MICAz han prevalecido como son el conector de entrada y salida I/O de 51 pines y la memoria flash serie; así como, el software de aplicación y las tarjetas sensoriales.

b. Tarjeta Interfaz MIB520CB

La tarjeta MIB520CB se caracteriza por tener un conector USB macho, mientras que la placa MIB520CA tiene un conector USB hembra. La principal función es proporcionar conectividad USB y es compatible con la familia de motas IRIS y MICA, asimismo suministra energía a los dispositivos, permite la comunicación serial y la posibilidad de instalar el *software* diseñado en los nodos sensores. En la Figura 1.13 se puede observar la tarjeta MIB520CB [10].

[12] Zigbee: es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo, basada en el estándar IEEE 802.15.4 [27].

Figura 1.13 Tarjeta interfaz MIB520CB

1.3.8 Herramientas de software

A continuación, se detalla el sistema operativo TinyOS, que es el *software* que utilizan los nodos sensores IRIS XM2110, con su lenguaje de programación llamado NesC; mientras que para desarrollar las aplicaciones de escritorio y para el dispositivo móvil Android se utilizó el lenguaje de programación Java.

a. Sistema operativo TinyOS

TinyOS tiene un modelo de ejecución basado en eventos y un modelo de programación basado en componentes, codificado por el lenguaje NesC, que es básicamente un dialecto C.

TinyOS no es un sistema operativo en el sentido tradicional; en realidad se trata de un marco de programación para sistemas embebidos y un conjunto de componentes que permiten la construcción de un "sistema operativo específico" para cada aplicación.

Un programa de TinyOS es un grafo[13] de componentes, cada uno de los cuales es una entidad independiente con capacidad computacional, que expone una o más interfaces. Estas interfaces son bidireccionales y contienen tres abstracciones computacionales: comandos, eventos y tareas. Un comando es una función que esta implementada en el componente proveedor de una interfaz, mientras que un evento es una función que se

[13] Grafo: En el ámbito de las ciencias de la computación es una estructura de datos, en concreto un tipo abstracto de datos, que consiste en un conjunto de nodos y un conjunto de arcos que establecen relaciones entre los nodos.

implementa en el componente que usa dicho comando. Las tareas se utilizan para expresar concurrencia entre componentes [12].

Está diseñado para escalar con las tendencias tecnológicas de la actualidad y soporta operaciones intensivas de concurrencia y garantiza atomicidad (es posible construir bloques de sentencias que se ejecutan por completo o no se ejecuta nada, y por lo tanto ante un fallo del sistema no puede quedar a medias). Emplea un paradigma de comunicación basado en Active Message y no maneja memoria dinámica:

- Las zonas de memoria se localizan en memoria estática.
- No existe memoria dinámica. No hay punteros.
- Una única pila asignada la tarea que se ejecuta en un momento dado.
- No existe protección de memoria.

El modelo de ejecución se basa en eventos:

- Soporta altos niveles de concurrencia en poca cantidad de espacio.
- Un único contexto de ejecución se comparte entre tareas de procesamiento no relacionadas (se evita la sobrecarga de los cambios de contexto).
- Concurrencia, gracias a tareas o procesos de larga ejecución que ejecutan hasta completarse en background sin interferir con otros eventos del sistema. Pueden ser interrumpidos por eventos del sistema de bajo nivel (concepto de concurrencia).
- Proporciona mecanismos para crear exclusión mutua en secciones de código (concepto de atomicidad).

El modelo de programación se basa en componentes (módulos):

- Cada módulo es diseñado para operar continuamente respondiendo a eventos de entrada (alarmas, *timer* del reloj, radio, etc.).
- Cuando un evento llega, trae con él el contexto de ejecución requerido.
- Las aplicaciones deben declarar implícitamente cuando han dejado de usar la CPU.

A continuación se muestran las funciones que puede ejecutar TinyOs [6]:

- Enrutamiento.
- Conversor analógico-digital.
- Identificación de un nodo.
- Reserva de memoria.
- Conversión serie-paralelo.
- Pila de comunicación.
- Radio.

- *Active Messages*.
- *Logs* del sistema.
- Reloj del sistema.
- Energía del sistema.
- Resetear el sistema.
- Generador de números aleatorios.
- Leds del sistema.
- Datos (Int a Leds, a Radio y viceversa).
- Gestión de errores CRC[14].

a.1 Estructura de Mensajes TinyOS

Los datos que intercambian los nodos sensores y el *Gateway* serán llamados mensajes y poseen una determinada estructura. Estos mensajes en TinyOS vienen identificados por variable `message_t` y consta de 4 elementos internos como se puede observar en la Figura 1.14 y son descritos posteriormente [13].

```
typedef nx_struct message_t {
  nx_uint8_t header[sizeof(message_header_t)];
  nx_uint8_t data[TOSH_DATA_LENGTH];
  nx_uint8_t footer[sizeof(message_footer_t)];
  nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
```

Figura 1.14 Estructura del mensaje message_t [13]

- ***Header* (cabecera):** Contiene la información de la interfaz de envío que se está utilizando, además permite identificar al nodo *Gateway* y sus respectivos emisores.
- ***Data* (Datos):** Contiene la información útil o carga útil, la cual es almacenada en la variable TOSH_DATA_LENGTH.
- ***Footer* (Pie del mensaje):** Permite ajustar los paquetes de datos para que puedan cumplir con el MTU requerido.
- ***Metadata* (metadatos):** Es información adicional como potencia de transmisión y códigos CRC y depende mucho del tipo de plataforma en la que se trabaje.

[14] CRC: código de detección de errores por redundancia cíclica, usado frecuentemente en redes digitales y en dispositivos de almacenamiento.

a.2 Interfaces de comunicación en TinyOS

Los nodos sensores hacen uso de ciertas interfaces para poder transmitir información a su nodo *Gateway*. Estas interfaces son [13]:

- ***Packet* (paquete)**: Proporciona un acceso básico a la estructura `message_t`, para poder realizar ciertas modificaciones a los datos.
- ***Send* (envío):** Esta interfaz permite enviar mensajes desde un nodo sensor a la red WSN, además posee un indicador para poder determinar si los mensajes llegaron exitosamente.
- ***Receive* (recepción):** Permite desempaquetar los datos del mensaje.

En TinyOS se puede realizar transmisiones simultáneas de mensajes gracias a las interfaces de tipo AM (*Active Message*). Cada mensaje tiene un propio identificador de tipo numérico, similar al uso de puertos.

a.3 Protocolos de Red en TinyOS

TinyOS posee dos protocolos de red que permiten transmitir los datos entre los nodos sensores y el nodo *Gateway* [14].

- **CTP (*Collection Tree Protocol*):** Es un protocolo de recolección de datos que utiliza una topología árbol. Éste posee un rendimiento aceptable; sin embargo, esto implica un consumo considerable de memoria de código.

CTP consta de tres elementos principales que permiten su funcionamiento, como se presenta en la Figura 1.15.

- Estimador de enlace: Monitorea la calidad del enlace enviando periódicamente mensajes de control, denominados beacons, hacia sus nodos vecinos.
- Enrutador: Analiza las respuestas de los beacons, para así seleccionar la mejor ruta hacia su *Gateway*
- Envío / Recepción: Este bloque interactúa con los dos anteriores y prepara a los paquetes para su envío y recepción. Además ayuda a evitar la duplicidad de los mensajes.

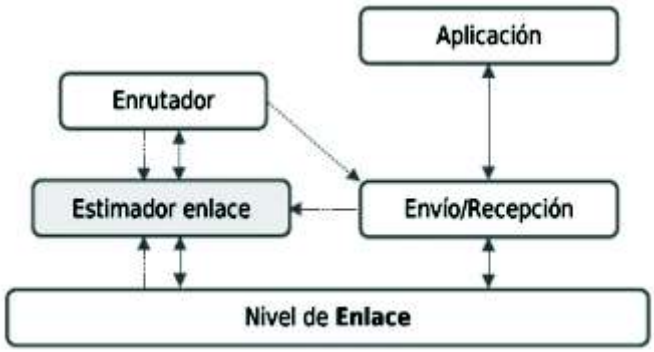

Figura 1.15 Diagrama de bloques CTP

- **Drip:** Este protocolo realiza una función inversa a la de CTP; es decir, envía mensajes desde su nodo raíz hacia los nodos finales, a través de variables de control que tienen los mensajes.

Drip utiliza dos interfaces para su funcionamiento: *DisseminationValue* y *DisseminationUpdate*. Realiza una estimación del enlace en base a la interfaz *DisseminationUpdate*, la cual trabaja con paquetes de *beacons* cada 64 metros.

La Figura 1.16 indica una configuración en donde el nodo raíz o *Gateway* se conecta con una PC, la cual le envía órdenes, para que a su vez éste retransmita a los nodos respectivos haciendo uso de los IDs. Muchas de las veces para poder alcanzar el nodo destino se deberá atravesar por nodos intermedios, como se muestra en el ejemplo.

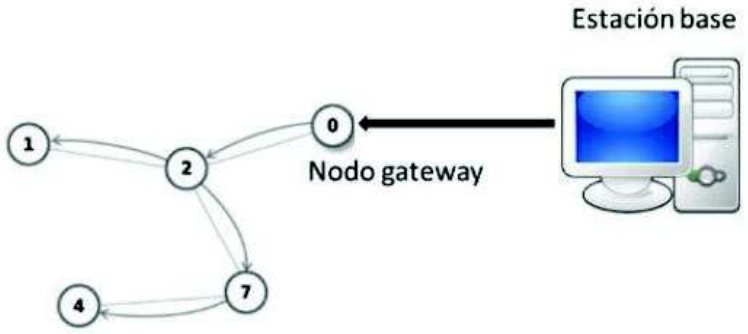

Figura 1.16 Red con diseminación de datos

1.3.9 Lenguaje de programación NesC

NesC es una variante del lenguaje de programación C, optimizado para las limitaciones de memoria de las redes de sensores. Existen varias herramientas que complementan y facilitan su uso, escritas en su mayoría en Java y en Bash [14]. Otras herramientas y librerías asociadas están escritas principalmente en C.

NesC es un lenguaje orientado a componentes y está especialmente diseñado para programar aplicaciones sobre redes de sensores, en particular en el sistema operativo TinyOS. Un programa en NesC está estructurado mediante componentes, el usuario puede crear un componente basado en otros ya creados. Dos componentes podrán comunicarse entre sí, mediante una interfaz, la cual definirá una serie de métodos (*commands* y *events*), los cuales deberán ser implementados en cada componente. Así, un método podrá solicitar la ejecución de un *command* de otro componente; por otro lado, para enviar una notificación se utilizarán un *event*, un ejemplo de esto se puede visualizar en la Figura 1.17.

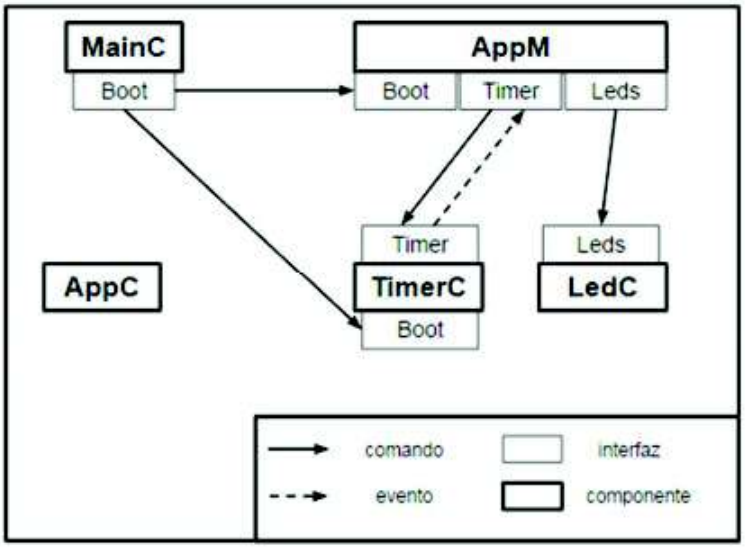

Figura 1.17 Relación entre componentes de una aplicación ejemplo

Todo componente está dividido lógicamente en tres partes: Configuración, Interfaces y Módulos. Como se ha dicho anteriormente, una aplicación escrita para TinyOS se compone de implementación de módulos y de configuraciones. En algunos casos, pueden hacerse aplicaciones solo con archivos de configuraciones. Un archivo de configuración consiste básicamente en un archivo formado por los módulos (o componentes) a utilizar en la aplicación y la relación que hay entre dichos módulos y las interfaces. Las interfaces se pueden definir como la parte "visible" de los módulos. No se puede acceder a ninguna función de un módulo si antes no ha sido definida en una interfaz.

Una interfaz puede ser provista por varios módulos distintos. Por ejemplo, se puede tener una interfaz de comunicaciones que abstraiga los procedimientos de envío y recepción de mensajes y varios módulos que hagan uso de esa interfaz, uno para cada tipo de hardware subyacente (por ejemplo no se maneja igual el hardware de comunicaciones de una mica2 que el de una telos). Los métodos serán los mismos y se llamarán igual (la interfaz es la misma), pero la implementación interna puede ser totalmente distinta.

Los módulos implementan bloques funcionales, generalmente de un nivel de abstracción más bajo que el bloque que lo referencia. En los módulos se deberá programar las acciones que llevará acabo un componente. Un mismo módulo puede proveer varias interfaces y por lo tanto puede implementar distintas funcionalidades (Por ejemplo el módulo *AMSender* provee las interfaces *Packet, AMPacket, AMSend*, etc.). Lo lógico es que todos los conjuntos de funciones (interfaces) que implementa un módulo guarden algún tipo de relación.

Los módulos se dividen en tres partes:

- *Provides*: Son las interfaces que ofrece el componente.
- *Uses*: Las interfaces que usa un componente.
- *Implementation*: Donde verdaderamente se realizan las acciones que deberá realizar el programa.

a. Interfaces

Las interfaces son usadas para la conexión entre dos componentes, uno perteneciente al usuario y otro al proveedor. Un componente usuario utiliza comandos para ejecutar funciones que le brinda su proveedor, mientras que el proveedor responde a dichas peticiones a través de eventos, todo esto lo pueden realizar gracias a la interfaz que éstos compartan. En la Figura 1.18, se observa cómo funciona una interfaz en TinyOS.

Figura 1.18 Esquema de funcionamiento de una interfaz [15]

b. Componentes

Los componentes están formados por código o algoritmo encargados de realizar funciones o tareas determinadas, es decir la funcionalidad de un programa. Las aplicaciones creadas en NesC son el producto del funcionamiento organizado de los componentes que son accedidos a través de las interfaces. Existen dos tipos de componentes principales:

módulos (*modules*), configuraciones (*Configurations*). Estos componentes se dividen en dos bloques que son, signatura e implementación.

- **Bloque Signatura:** se encuentra al inicio de un archivo de programación y contiene las interfaces que van a ser usadas por un componente, mediante el uso de las palabras reservadas *provides* y *uses.* En la Figura 1.19, se muestra un ejemplo para el componente módulo y configuración.

Componente Módulo

```
module SenseC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli>;
  uses interface Read<uint16_t>;
}
```

Componente Configuración

```
configuration LedsC {
  provides interface Leds;
}
```

Figura 1.19 Bloque signatura para componente módulo y configuración [13]

- **Bloque Implementación:** es la segunda parte del componente, tiene un identificativo mediante la palabra *implementation*. La implementación en el componente configuración consistirá en realizar un listado de componentes y enlazarlos mediante interfaces, lo cual se conoce como *wiring*. Por otro lado, para un componente módulo, la implementación contiene la lógica del programa; que consiste en la ejecución de eventos por medio de llamadas, denominadas *call*. La Figura 1.19, muestra un ejemplo de implementación de los dos tipos de componentes.

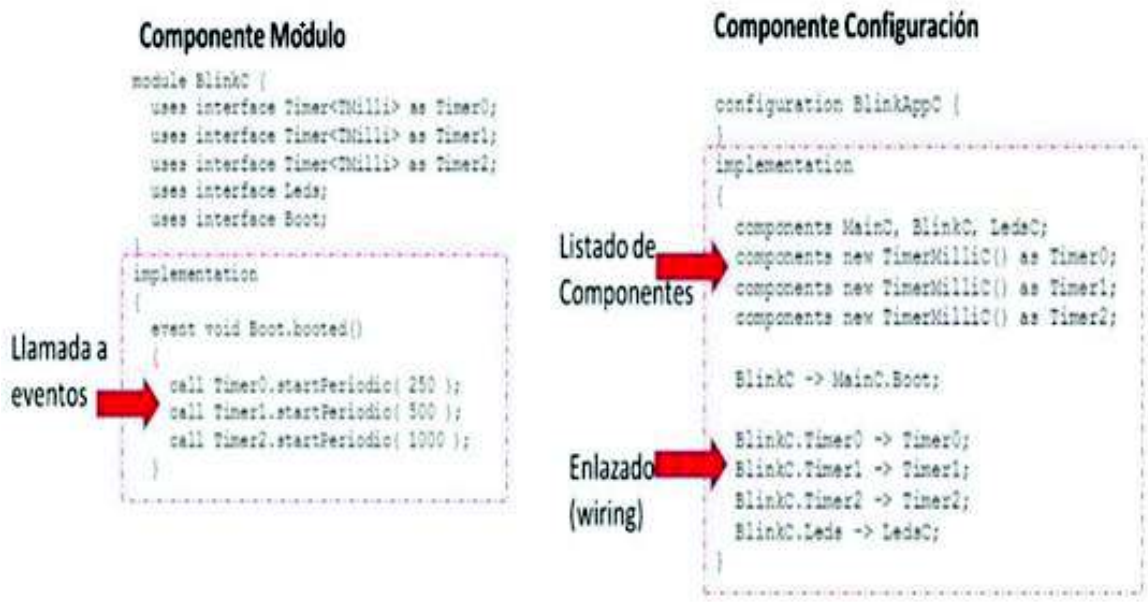

Figura 1.20 Bloque Implementación para componentes módulo y configuración [13]

c. Enlazado (*Wiring*)

El enlazado o *wiring* se da mediante 3 operadores: ←, →, e =. Las flechas indican que componente es usuario de una determinada interfaz y a su vez cual provee dicha interfaz. La flecha irá de usuario a proveedor, y utilizará el siguiente formato:

CompUsuario.InterfazUsuario **->** Comp.Proveedor.InterfazProveedor;

Con la utilización del operador **=** se podrá renombrar interfaces de otros componentes.

d. Comandos Básicos para NesC

Los comandos básicos más usados en NesC son los siguientes:

Para otorgar permisos de lectura, escritura y ejecución que permitan la instalación de programas, se usa el siguiente comando:

```
$ sudo chmod 666 ,  /dev/ttyUSBX
```

Con el siguiente comando se podrá compilar el código, pero primero hay que ubicarse en el directorio del programa, además este comando indicará si existe algún error y en qué línea del código se encuentra.

```
$ make iris
```

Para la instalación del programa en los equipos se utiliza el comando mostrado a continuación, se debe indicar el identificador, la mota en la que se va a instalar el programa y el número de puerto en que se encuentra.

```
$ make iris install, node_id mib520 , port_number
```

1.3.10 Lenguaje de programación JAVA

Java es un lenguaje de programación orientado a objetos (POO) y fue desarrollado por Sun Microsystems. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, evitando problemas como la manipulación directa de punteros o memoria [16].

Principalmente una característica que lo hace único es que es independiente de la Plataforma, del Sistema Operativo y del Hardware, lo que lo hace compatible con cualquier Computador, y portable, cumpliendo el axioma de Java “Programe una vez, y Ejecute donde sea” – “*Write Once, Run Everywhere*’’.

Las clases son lo más simple de Java. Todo en Java forma parte de una clase, es una clase o describe como funciona una clase. Todas las acciones de los programas Java se colocan dentro del bloque de una clase o un objeto. Para comprender mejor se debe definir estos dos conceptos [16]:

- **Objeto**: Entidad existente en la memoria del ordenador que tiene propiedades (atributos o datos sobre sí mismo almacenados por el objeto) y operaciones disponibles específicas (métodos).
- **Clase**: Abstracción que define un tipo de objeto especificando qué propiedades (atributos) y operaciones disponibles va a tener.

En la Figura 1.21 se tiene un diagrama que explica mediante un ejemplo el concepto de clases y objetos.

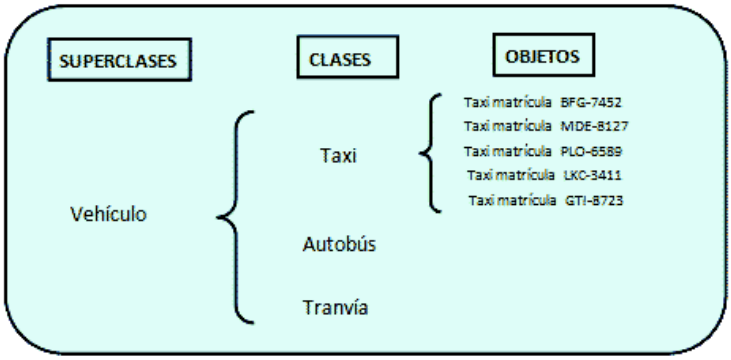

Figura 1.21 Ejemplo de clases y objetos [17]

Este ejemplo, consta de tres tipos de vehículo: taxi, autobús y tranvía, los cuales serán denominamos clases. Si se deseara definir una clase, se debe indicar sus propiedades y operaciones (métodos) disponibles, por ejemplo:

```
/* Ejemplo Clase Taxi - aprenderaprogramar.com */
Clase Taxi {
Propiedades:
            Matrícula identificativa
            Distrito en el que opera
            Tipo de motor diesel o gasolina
            Coordenadas en las que se ubica
 Operaciones disponibles:
            Asignar una matrícula
            Asignar un distrito
            Asignar un tipo de motor
            Ubicar en unas coordenadas
}
```

Figura 1.22 Descripción de la clase Taxi [17]

El haber definido así a “taxi”, significará que todo objeto de tipo Taxi que sea creado, tendrá una matrícula identificativa, un distrito en el que opera, un tipo de motor y unas coordenadas en las que se ubica.

a. Conceptos de POO en JAVA

Hay cuatro conceptos principales de la programación orientada a objetos en Java. Estos son [18]:

- **Abstracción.** Abstracción significa usar cosas simples para representar cosas complejas. En Java, la abstracción significa que cosas simples como objetos, clases y variables representan códigos y datos subyacentes más complejos. Esto es importante porque evita repetir el mismo trabajo varias veces.
- **Encapsulación.** Es la práctica de mantener los campos dentro de una clase privada, y luego proporcionar acceso a ellos a través de métodos públicos. Es una barrera protectora que mantiene seguros los datos y el código dentro de la clase. De esta forma, podemos reutilizar objetos como componentes de código o variables sin permitir el acceso abierto a los datos en todo el sistema.
- **Herencia.** Esta es una característica especial de la Programación Orientada a Objetos en Java. Permite a los programadores crear nuevas clases que comparten algunos de los atributos de las clases existentes. Esto permite construir sobre el trabajo previo sin tener mayor problema.
- **Polimorfismo.** Este concepto permite a los programadores usar la misma palabra para significar cosas diferentes en contextos diferentes. Una forma de polimorfismo en Java es la sobrecarga de métodos, es cuando el código en sí mismo implica diferentes significados. La otra forma es anulación de método, es cuando los diferentes significados están implicados por los valores de las variables suministradas.

b. Entorno de desarrollo

En el mercado existen distintos programas que permiten desarrollar en lenguaje Java, por ejemplo, Eclipse, NetBeans, Android Studio, etc. Oracle distribuye sin costo el JDK (*Java Development Kit*) formado por programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java. También existe una versión reducida del JDK, que se llama JRE (*Java Runtime Enviroment*), el JRE se encarga de solo ejecutar el código de Java mas no compilarlo [19].

Los entornos de desarrollo integrados o IDEs (*Integrated Development Enviroment*), permiten en un solo programa escribir algoritmo Java, compilarlo y ejecutarlo. Estas características permiten desarrollar aplicaciones más rápido, con la posibilidad de utilizar librerías con componentes ya desarrollados en la creación de la aplicación o programa.

c. El compilador de Java

Es una herramienta incluida en el JDK (*Java Development kit*) llamada javac, se encarga de realizar un análisis de la sintaxis del algoritmo en los archivos que tienen extensión ".java". En el caso de que no exista errores genera un archivo compilado o *bytecodes* con la extensión ".class", como se puede observar en la Figura 1.23, y en el caso de que exista errores muestra las líneas en donde se produjo este error [19].

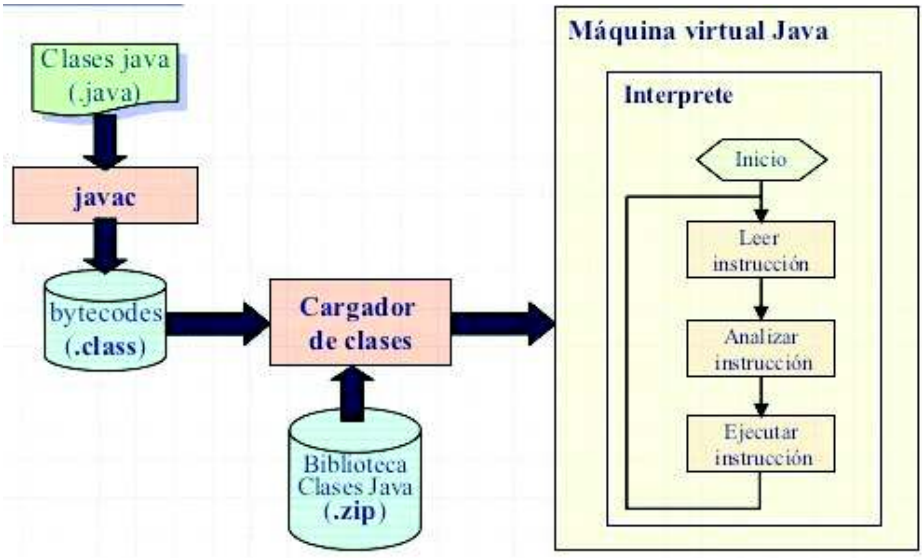

Figura 1.23 Compilador de Java [19]

1.3.11 Sistema operativo Android

Android es un Sistema Operativo de código abierto basado en Linux[15], por lo que es posible tener acceso al código fuente. Fue desarrollado para dispositivos móviles con pantallas táctiles. En sus inicios fue desarrollado por Android Inc. y que posteriormente fue comprado por Google.

[15] Linux: Es un sistema operativo libre tipo Unix; multiplataforma, multiusuario y multitarea.

Este sistema operativo se ha convertido en un modelo a seguir tanto por desarrolladores de aplicaciones como por personas conocedoras de negocios de alto impacto.

a. Arquitectura de Android

Este sistema está compuesto por varios componentes, los cuales se muestran en la Figura 1.24 y están descritos posteriormente [20].

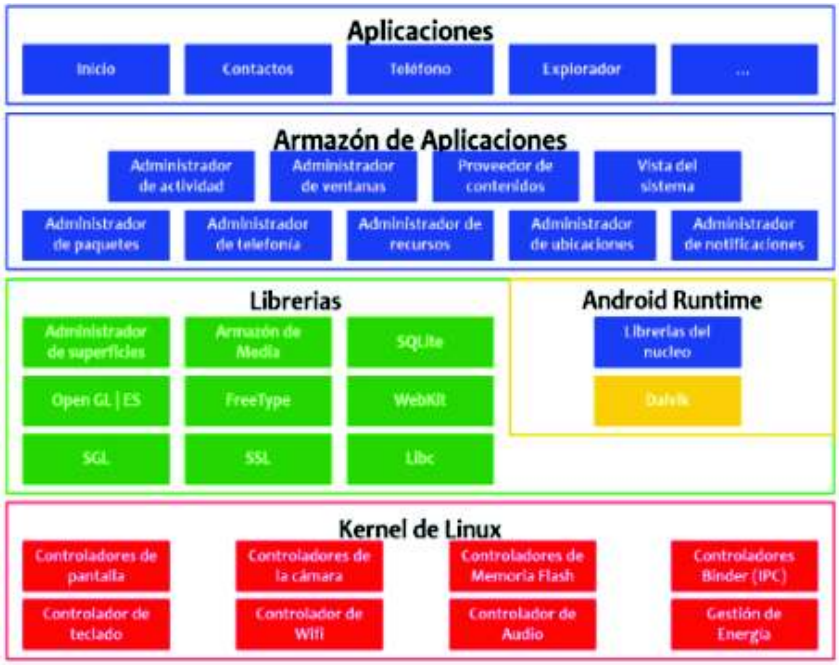

Figura 1.24 Arquitectura de Android [20]

- **Aplicaciones:** El bloque de aplicaciones está formado por un cliente de correo electrónico, navegador, mapas, contactos, calendario, programas de SMS (*Short Message Service*), entre otros.

- **Marco de trabajo de aplicaciones:** En este bloque los desarrolladores tienen acceso completo a los mismos APIs[16] (*Application Programming Interface*) de las aplicaciones base. Una de las ventajas de esta arquitectura es que fue diseñada para facilitar la reutilización de componentes, por este motivo es posible hacer uso de ciertas capacidades para más de un diseño. Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

- **Bibliotecas:** El Sistema Operativo de Android está compuesto por millones de líneas de lenguaje C/C++, que forman un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se presentan a los

[16] API: Es la interfaz de programación para el desarrollo de aplicaciones.

desarrolladores a través del marco de trabajo de aplicaciones de Android; algunas son: System C library (implementación de la biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.

- ***Runtime* de Android:** Android incluye un conjunto de bibliotecas base las cuales aportan la mayor parte de las funciones en las bibliotecas base de Java. Un componente principal de este entorno es la máquina virtual llamada Dalvik la cual fue la máquina virtual que Android usaba originalmente, y permite que cada aplicación corra su propio proceso.

b. Componentes de una Aplicación Android

Al desarrollar una aplicación en Android se tienen los siguientes componentes principales [21]:

- ***Activity*:** Es un conjunto de acciones que indican la interrelación entre el usuario y la aplicación de forma directa y está constituida por dos partes, la primera la parte gráfica consta de un archivo XML[17] (*eXtensible Markup Language*), en el cual se encuentran los elementos visibles que se muestran en la pantalla, esta parte es bastante similar a las etiquetas usadas en HTML[18] (*HyperText Markup Language*) y la parte lógica que consiste en un archivo .java donde se desarrolla todo el código propio de la actividad.

- ***Broadcast Intent Receiver*:** Gracias a este componente es posible correr una aplicación mientras se tiene otra en ejecución con el objetivo de responder eventos, estos eventos se los puede alertar a través del API *Notification Manager*, posteriormente el mismo usuario lo puede observar en la barra de estado.

- ***Service*:** *Service* es una aplicación que se ejecuta en segundo plano de forma constante, la ejecución del *service* debe tener un tiempo de vida muy alto.

- ***Content Provider*:** Este componente administra los datos de la aplicación, y el acceso a la misma, permitiendo a las aplicaciones puedan compartirlos de una manera segura.

[17] XML: Lenguaje de Marcas Extensible. Se trata de un metalenguaje, un lenguaje que se utiliza para decir algo acerca de otro.
[18]HTML: Lenguaje de marcas de hipertexto, hace referencia al lenguaje de marcado para la elaboración de páginas web.

- ***View*:** Las vistas son aquellos elementos gráficos y crean una interfaz entre la aplicación y el usuario, estas vistas pueden ser botones.
- ***Layout*:** Es un conjunto de vistas, las cuales son organizadas según necesidades del desarrollador en tipos como por ejemplo; lineales, en cuadrícula o en posición absoluta.
- ***Intent*:** Como su nombre lo indica este componente representa la voluntad de realizar una acción y es usada al lanzar una actividad, un servicio, envíos tipo broadcast, etc.
- ***Fragment*:** Los fragmentos están formados por la combinación de varias vistas para formar bloques funcionales de la interfaz de usuario, esto es necesario debido a la diferencia en el tamaño de las pantallas de los dispositivos móviles (celulares y tabletas).
- ***Android Telephony* API:** Los dispositivos móviles que funcionan con Android como sistema operativo, tienen una capa llamada RIL (*Radio Network Interface Layer*), encargada de manejar la interfaz entre el módem del dispositivo y su sistema operativo. RIL permite iniciar los servicios de telefonía haciendo uso de los protocolos de comunicaciones.

1.3.12 Sistema gestor de base de datos

Tradicionalmente, para manipular grandes cantidades de datos aparecieron inicialmente las bases de datos o "bancos de datos". Cumpliendo con la función de almacenar ordenadamente datos en un juego de ficheros, y, mediante unas aplicaciones informáticas y un sistema de índices, gestionarlas adecuadamente [22].

Al aumentar la complejidad de estos bancos de datos (elevado número de ficheros y usuarios, acceso múltiple y simultáneo a los ficheros, aumento del número de registros, etc.) se producen problemas cada vez más graves para asegurar la consistencia, la integridad y la accesibilidad de los datos, produciendo problemas de eficiencia en el tratamiento de los mismos [22].

Para resolver estos problemas aparecen en el mercado los Sistemas de Gestión de Bases de Datos (SGBD), que convierten el acceso a los datos y su gestión en una aplicación cerrada, asumiendo el papel de intermediario entre los usuarios y los ficheros, y haciéndose cargo de todos los problemas de explotación, mantenimiento y comprobación de los datos como se muestra en la Figura 1.25.

De esta manera el usuario pierde de vista todos los detalles relativos al almacenamiento físico de los de los datos tratando con ellos sólo a través de un lenguaje conceptual sencillo.

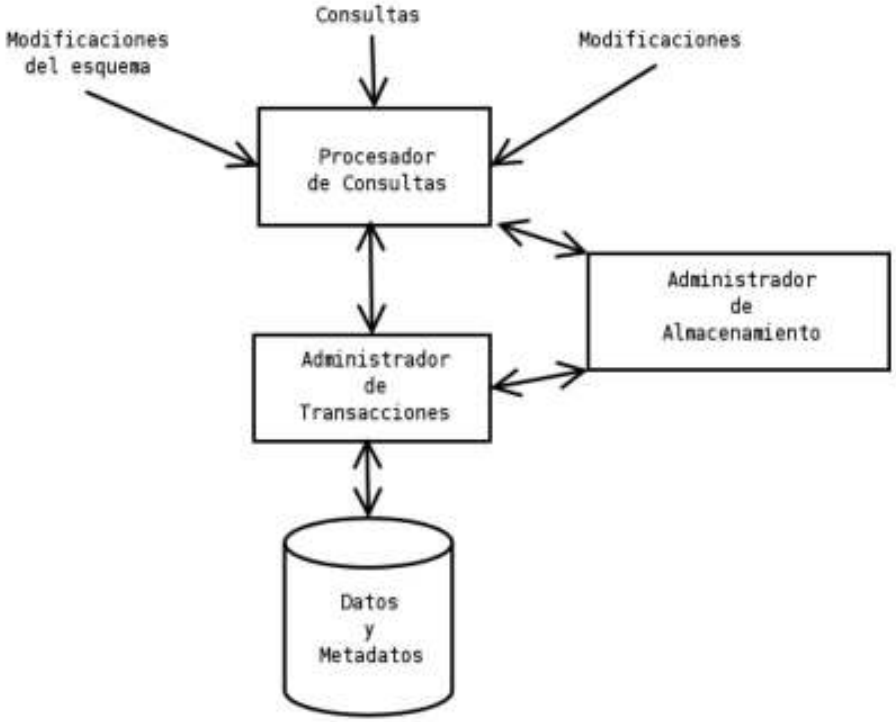

Figura 1.25 Arquitetura de un SGBD

a. Gestor de Base de Datos MySQL

MySQL es un Sistema de Gestión de Base de Datos Relacionales (RDBMS), de código abierto, basado en un Lenguaje de Consulta Estructurado (SQL). MySQL ofrece compatibilidad con PHP, Pearl, C y HTML. Al mismo tiempo, MySQL se ejecuta en prácticamente todas las plataformas, incluyendo Linux, UNIX y Windows, además de que ciertas sentencias pueden ser embebidas en código PHP y HTML para diseñar aplicativos web.

MySQL es conocida por desarrollar alta velocidad en la búsqueda de datos e información, a diferencia de sistemas anteriores [23].

En la Figura 1.26 se puede apreciar el funcionamiento desde que el cliente hace una petición, hasta que el servidor la procesa.

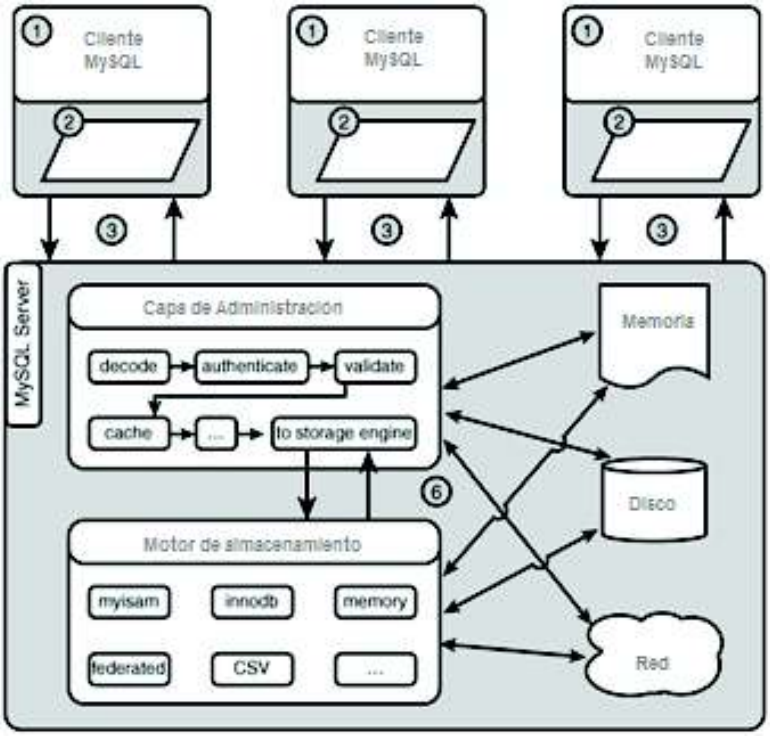

Figura 1.26 Interacción entre un servidor y cliente MySQL [23]

2. METODOLOGÍA

En el capítulo anterior, se mencionaron conceptos relacionados con la tecnología de las redes de sensores inalámbricos (WSN), detallando algunas de sus características principales. Además se describieron algunas de las herramientas de hardware y software que serán usadas en el desarrollo de este proyecto de titulación, el cual, ofrece una solución para la monitorización ambiental requerida en el Jardín Botánico de Quito.

En este apartado, en cambio, se determinarán los requerimientos del sistema, además se detallará el diseño del mismo, especificando cada uno de los componentes que lo conforman.

2.1 Requerimientos del prototipo

Hoy en día, las redes de sensores inalámbricos proveen un sinfín de aplicaciones, siendo una de éstas la monitorización de parámetros ambientales. El Jardín Botánico de Quito cuenta con ambientes artificiales los cuales necesitan ser monitoreados; y es aquí donde surge la necesidad de tener un sistema que permita la supervisión de determinados parámetros que son indispensables para la preservación de la flora albergada en este centro.

Es por ello que en este proyecto se plantea el desarrollo de un prototipo de sistema de monitorización ambiental que permita recopilar valores de temperatura, humedad y luminosidad, de forma automática; optimizando así el proceso de control de los ambientes artificiales.

Teniendo en cuenta lo anteriormente mencionado, los requerimientos que debe cumplir el prototipo son los siguientes:

- El prototipo tendrá como funciones: Recolectar valores de temperatura, luminosidad y humedad de los tres principales ambientes artificiales que posee el centro.
- Cada nodo debe ser capaz de obtener los valores de los parámetros antes mencionados.
- Los valores obtenidos deberán ser enviados al nodo *Gateway*, para que estos posteriormente sean retransmitidos a un servidor, con el correspondiente identificativo de cada nodo sensor.
- Se tendrán 2 aplicativos: Una de las aplicaciones será de escritorio, mientras que la otra aplicación será para dispositivos Android.

- La aplicación de escritorio dispondrá de una pantalla de *Login* para poder dar acceso solo a los usuarios que se encuentren registrados en la base de datos. Posteriormente se desplegará una ventana en la cual se tendrá dos opciones. La primera nos permitirá ingresar a una nueva ventana, en la que se visualizará los valores obtenidos por los sensores de los tres ambientes artificiales. La otra opción permitirá, en cambio, obtener gráficas estadísticas elaboradas con los valores tomados desde la base de datos, en períodos de tiempo determinados. Además se generará una alarma en el caso de que los valores tomados estén fuera de los rangos permitidos y se emitirá un e-mail de advertencia a los encargados del centro.
- Por otra parte, la aplicación Android tendrá como finalidad poder realizar una monitorización remota de los ambientes artificiales, además receptará las alertas que puedan ser generadas, cuando los parámetros ambientales salgan de los límites permitidos.
- El sistema contará también con una base de datos para almacenar todos los valores que sean tomados por los nodos sensores y que además servirá como fuente de datos para poder realizar gráficas estadísticas.

2.2 Componentes del sistema de monitorización ambiental

El prototipo tendrá los componentes mostrados en la Figura 2.1, constará de una aplicación Android, un servidor que tendrá una aplicación de escritorio y una base de datos, un nodo *Gateway*, una red de nodos IEEE 802.15.4 y sensores de luminosidad, humedad y temperatura.

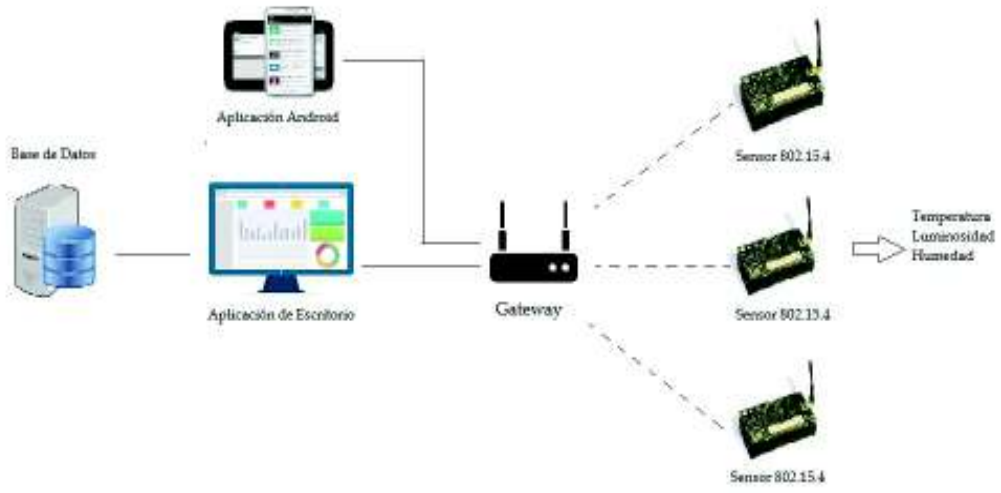

Figura 2.1 Esquema del prototipo de sistema de monitorización ambiental

2.2.1 Aplicativo Android

El aplicativo Android será un programa diseñado exclusivamente para dispositivos basados en sistemas Android. Esta aplicación será desarrollada en Android Studio, con el lenguaje de programación JAVA y permitirá realizar peticiones, con la finalidad de supervisar los valores de los parámetros ambientales.

2.2.2 Aplicación de escritorio

Se refiere a una computadora con una aplicación encargada de realizar la comunicación con la red IEEE 802.15.4, además de receptar los valores obtenidos por los sensores, para poder ser procesados. La comunicación entre la red IEEE 802.15.4 y el servidor se la realiza mediante un nodo *Gateway* conectado al computador por USB.

2.2.3 Nodo Gateway

El nodo *Gateway* se encarga de recolectar la información de la red IEEE 802.15.4 y transmitir al servidor y además enviar datos desde el servidor hacia los nodos, cumpliendo la función de un puente que conecta a la red IEEE 802.15.4 con el servidor o la aplicación. El nodo *Gateway* tendrá instalado el programa “BaseStation” desarrollado en el lenguaje de programación NesC

2.2.4 Red IEEE 802.15.4

La red de sensores IEEE 802.15.4 está formada por tres motas o nodos sensores, que en el caso particular de este proyecto de titulación serán de la marca Memsic, específicamente las motas IRIS XM2110.

Cada mota recolecta información de los sensores de temperatura, humedad y luminosidad de los principales ambientes artificiales del centro y esta información es enviada por medio del *Gateway* hacia el servidor.

2.3 Diseño del prototipo

Una vez definidos los requerimientos y componentes del Sistema de Monitorización, se procede a realizar el diseño del prototipo, así como a la implementación del mismo.

2.3.1 Diseño de la aplicación Android

La aplicación Android tiene como objetivo permitir la monitorización remota de los ambientes artificiales, por parte del usuario y también será capaz de generar alertas que puedan producirse en el caso de que los valores medidos salgan de los límites

establecidos. Debido a las funcionalidades reducidas de la aplicación, el diseño y desarrollo de ésta, no será de gran complejidad.

a. Navegación por la aplicación Android

La navegación de la aplicación es relativamente sencilla debido a baja complejidad, dicha interacción se muestra en la Figura 2.2.

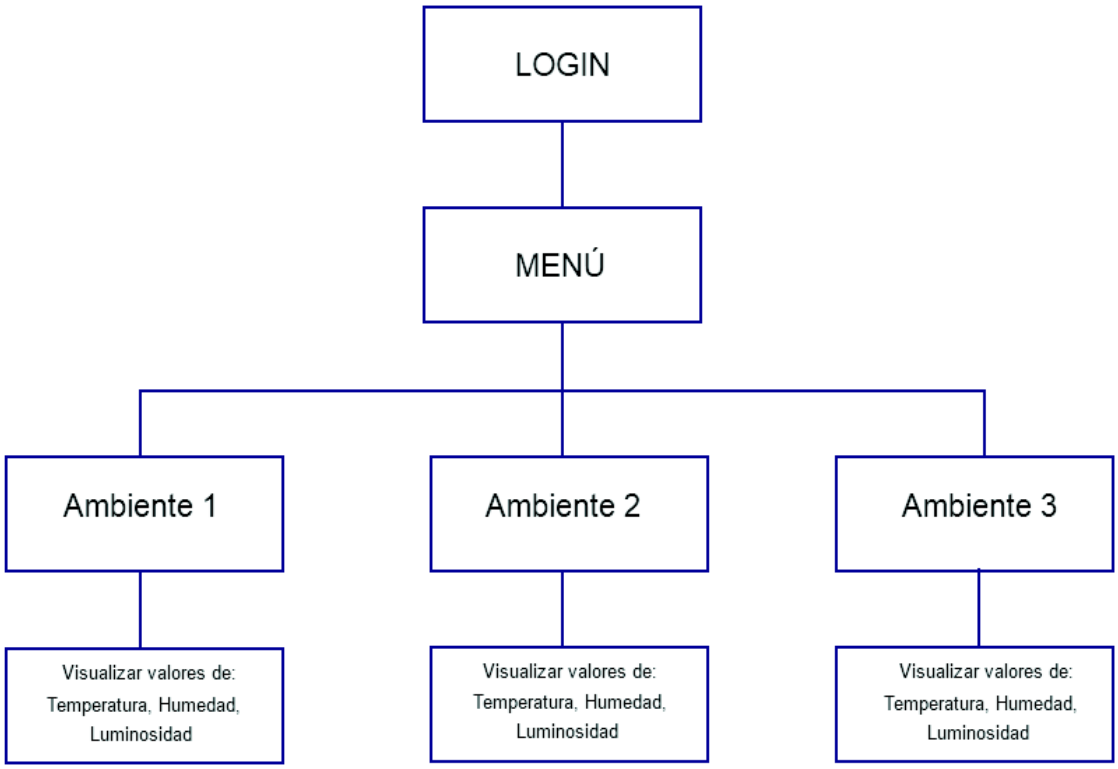

Figura 2.2 Navegación por la aplicación Android

b. Interfaz de la aplicación Android

La aplicación Android tendrá una ventana inicial para que los encargados del centro se registren ingresando un usuario y su contraseña.

Como se muestra en la Figura 2.3, la ventana tendrá dos botones, uno que permitirá el ingreso a la aplicación y otro que servirá para salir. Se incluirán mensajes de advertencia en caso de que los datos se ingresen de forma incorrecta.

Figura 2.3 Pantalla de login para la aplicación Android

Luego de desplegará una ventana, como se indica en la Figura 2.4, en la cual existirán tres botones, que permitirán ingresar a cada uno de los ambientes y poder supervisar los parámetro ambientales.

Figura 2.4 Pantalla de selección de ambientes

Luego de escoger uno de los tres botones del menú, se abrirá una ventana donde se podrán observar los valores de temperatura, humedad y luminosidad actual en el ambiente. Además se instalarán controles para visualizar en una escala los valores obtenidos en los parámetros ambientales y éstos no podrán ser manipulados por el usuario. Como se ve en

la Figura 2.5, también existirá un botón que permita regresar al menú para seleccionar otro ambiente.

Figura 2.5 Diseño de la pantalla para el Ambiente 1

c. Diagrama de casos de uso

El diagrama de casos de uso representará la forma en cómo el usuario interactúa con la aplicación Android, el cual estará basado en las especificaciones entregadas por el cliente. En la Figura 2.6 se muestra el diagrama de casos de uso para la aplicación Android.

Figura 2.6 Diagrama de casos de uso de la aplicación Android

Para un mejor entendimiento de los casos de uso que presenta la aplicación, se utilizará fichas en las que se describa cada una de las funcionalidades, incluyendo los requisitos que cada una de éstas necesita. A continuación se presentan las fichas en las que se describe el diagrama de casos de uso presentado anteriormente.

En la Tabla 2.1 se muestra de forma detallada los pasos que el administrador seguirá para registrar a los usuarios, que tendrán permitido el acceso al sistema, dentro de una base de datos.

Tabla 2.1 Caso de uso registro de usuarios en la base de datos

Caso de Uso	Registro de usuarios en base de datos
Autor	Ronald Erazo
Fecha	18/11/2017
Propósito	Tener un control de los usuarios que tendrán permitido el ingreso a la aplicación.
Actores	Administrador, Base de Datos.
Precondición	Instalar MySQL para el desarrollo de la base de datos Usuarios.
Postcondición	Los datos ayudarán a la validación de los usuarios que intenten ingresar a la aplicación
Resumen	El registro de usuarios en una base de datos facilitará la validación de los mismos en el momento en que traten de ingresar a la aplicación.

La Tabla 2.2 muestra el proceso que el usuario sigue para ingresar las credenciales y acceder a la aplicación Android.

Tabla 2.2 Caso de uso ingreso de credenciales

Caso de Uso	Ingreso de credenciales
Autor	Ronald Erazo
Fecha	18/11/2017
Propósito	Realizar una validación de usuario para ingreso al sistema.
Actores	Usuario, Aplicación Android, Base de Datos.
Precondición	Realizar una conexión con la Base de Datos.
Postcondición	Si los datos son válidos el usuario podrá ingresar al sistema.
Resumen	El usuario debe suministrar sus datos para que éstos sean validados en la base de datos y de ser correctos se permitirá el acceso a la aplicación; caso contrario, se le negará el acceso.

Finalmente en la Tabla 2.3 se describe el proceso que sigue el usuario que quiere monitorear remotamente los ambientes artificiales del Jardín Botánico.

Tabla 2.3 Caso de uso monitorizar ambientes artificiales

Caso de Uso	Monitorizar ambientes artificiales
Autor	Ronald Erazo
Fecha	18/11/2017
Propósito	Visualizar los valores obtenidos por los Nodos Sensores.
Actores	Usuario, Aplicación Android, Base de Datos.
Precondición	Realizar una conexión con la base de datos.
Postcondición	Se visualizará los valores de temperatura, luminosidad y humedad, los cuales se obtendrán mediante una consulta a la base de datos.
Resumen	Realizada la conexión con la base de datos, el programa realizará una consulta para obtener los últimos valores, de los parámetros ambientales, registrados en la base de datos.

d. Diagramas de actividades

En la Figura 2.7 se presenta el diagrama de actividades de la aplicación Android, en la cual se describe el flujo de trabajo que se tendrá entre los usuarios y la aplicación Android, desde que se inicia hasta que se cierra la aplicación.

Primero, el usuario deberá iniciar la aplicación e ingresar las credenciales. La aplicación realizará una validación de estos datos, para permitir el ingreso del usuario, caso contrario desplegará un mensaje indicando que los datos ingresados son incorrectos.

Una vez que el usuario haya logrado ingresar, deberá seleccionar el ambiente que desee monitorear, en este momento, la aplicación realizará una consulta a la base de datos y presentará los valores en la pantalla para que el usuario pueda visualizarlos. En caso de que el usuario desee monitorear otro ambiente, solo deberá regresar a la pantalla anterior y seleccionarlo.

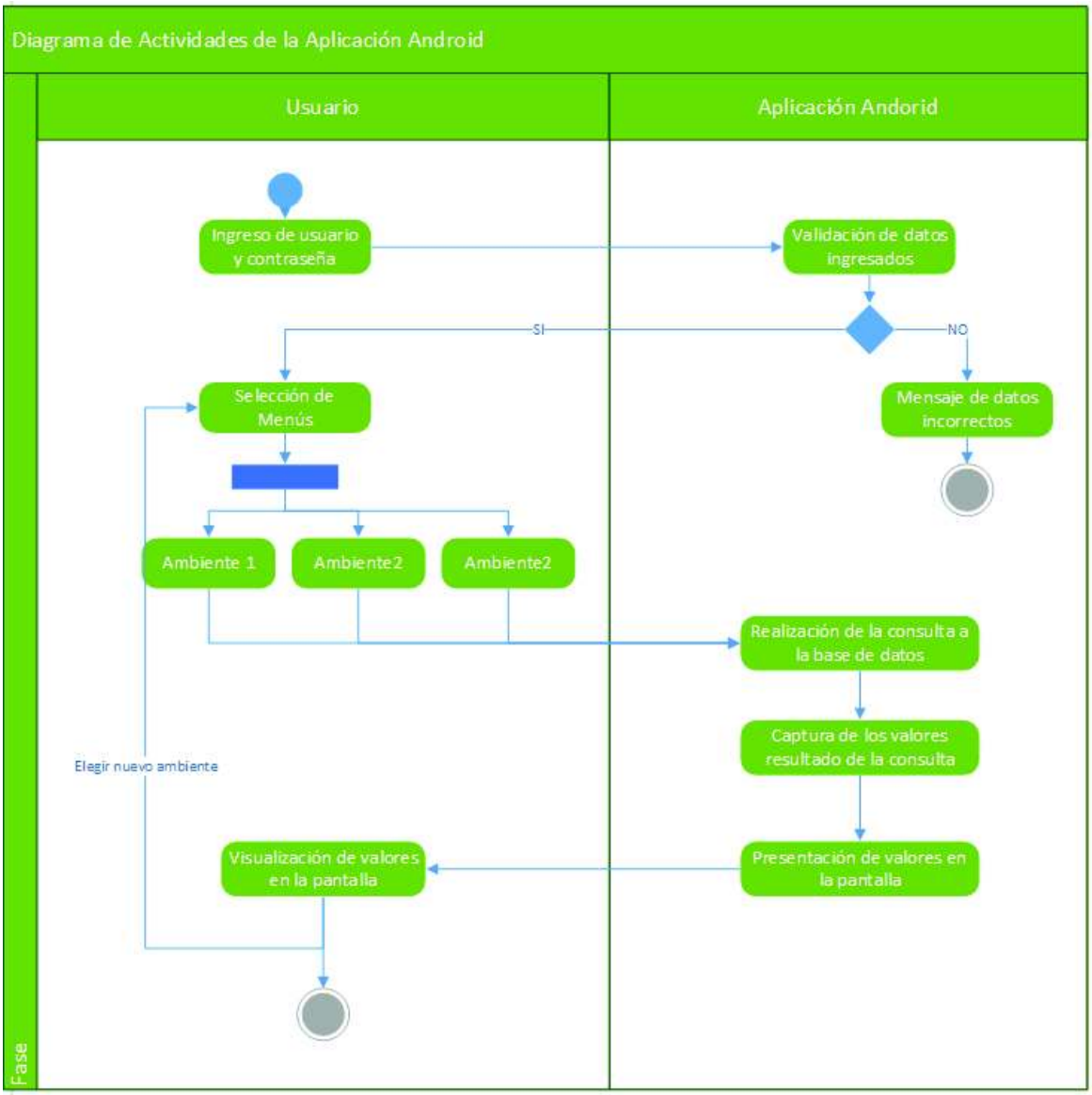

Figura 2.7 Diagrama de actividades de la aplicación Android

e. Diagrama de clases

La aplicación “MonitorizaciónJardínBotánico” estará compuesta de las siguientes clases e interfaces, como se muestra en la Figura 2.8:

- **Valores:** Tendrá métodos encargados de tomar las variables de la temperatura, humedad y luminosidad.
- **Registro (interfaz):** En esta interfaz se realizará la validación de los usuarios, por medio de la base de datos, para el ingreso a la app.

- **Ambiente 1, Ambiente 2, Ambiente 3 (interfaces):** Serán pantallas donde se desplegarán los valores de los parámetros ambientales, obtenidos de la base de datos.

- **UsuarioArea (interfaz):** Será la interfaz donde se presente el menú de los tres ambientes artificiales para que el usuario pueda seleccionarlos.

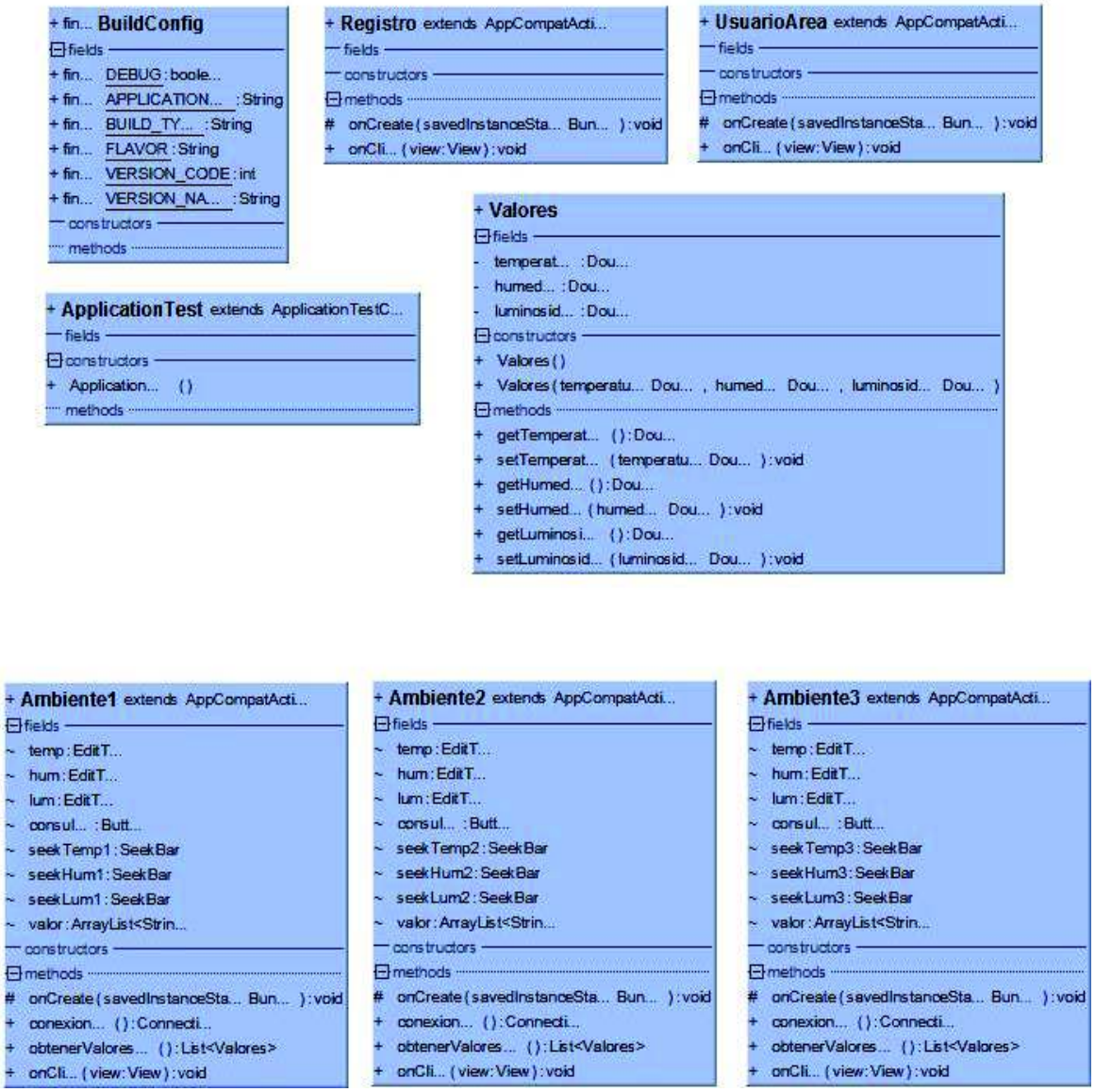

Figura 2.8 Diagrama de clases de la aplicación Android

2.3.2 Diseño de la aplicación de escritorio

La aplicación de escritorio será quien reciba los valores tomados por los nodos sensores y los mostrará en una ventana para que el usuario pueda supervisar los ambientes artificiales. Además la aplicación gestionará los datos para almacenarlos en una base de

datos. Los valores almacenados en la base de datos servirán luego para elaborar gráficas estadísticas, que podrá ayudar a los encargados del centro en tareas de investigación.

a. Navegación por la aplicación de escritorio

La aplicación de escritorio tendrá dos ventanas principales, una en donde se visualizará los valores obtenidos por los nodos sensores y otra en la que se mostrará gráficas estadísticas definidas en un intervalo de tiempo. La navegación dentro de la aplicación se puede observar en la Figura 2.9.

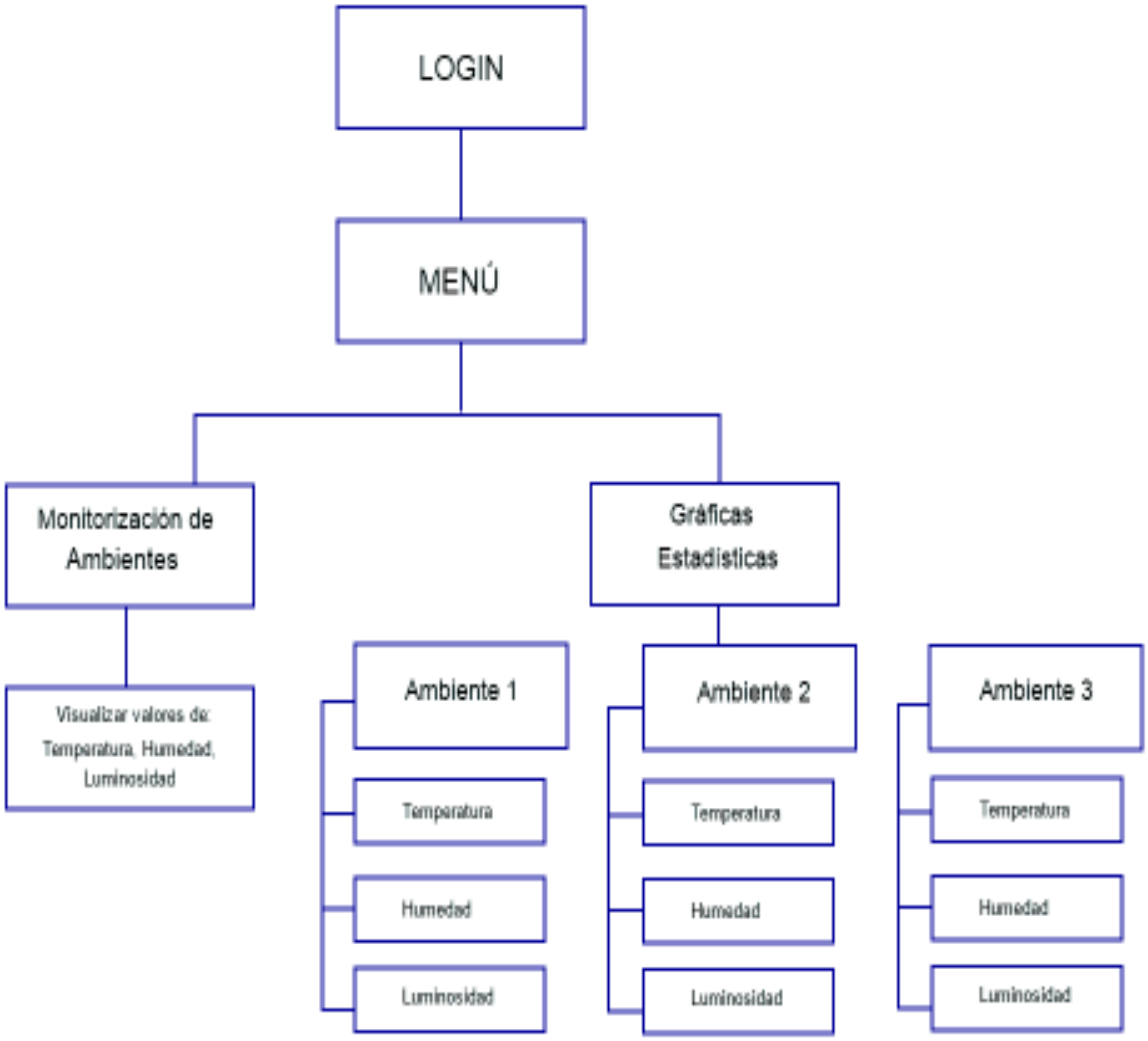

Figura 2.9 Navegación por la aplicación de escritorio

b. Interfaz de la aplicación de escritorio

La aplicación de escritorio tendrá una pantalla inicial de registro, en donde se deberá ingresar usuario y contraseña como se puede ver en la Figura 2.10.

Figura 2.10 Esquema de la ventana de registro

Posteriormente se abrirá una ventana en donde se tendrá un menú con dos opciones. La primera opción, llevará a una ventana en donde se podrá monitorear los tres principales ambientes artificiales. Aquí se visualizarán los valores de temperatura, humedad y luminosidad de los ambientes, como se puede apreciar en la Figura 2.11.

Figura 2.11 Esquema de la pantalla de monitorización de ambientes

La segunda opción, presentará a una ventana en la que se podrá realizar gráficas estadísticas de los parámetros ambientales, que se presenta en la Figura 2.12. Para esto,

se seleccionará el parámetro ambiental y el ambiente del que se quiere graficar, todo definido dentro de un rango de tiempo determinado, el cual también deberá ser elegido.

Figura 2.12 Esquema de la ventana de gráficas estadísticas

c. Diagrama de casos de uso

Para el diseño del diagrama de casos de uso se consideró los requerimientos del usuario y la funcionalidad que debe tener el sistema. Esto permitirá entender de mejor forma cual es la relación entre el usuario y el sistema a desarrollarse, y que es lo que el usuario puede hacer con el mismo.

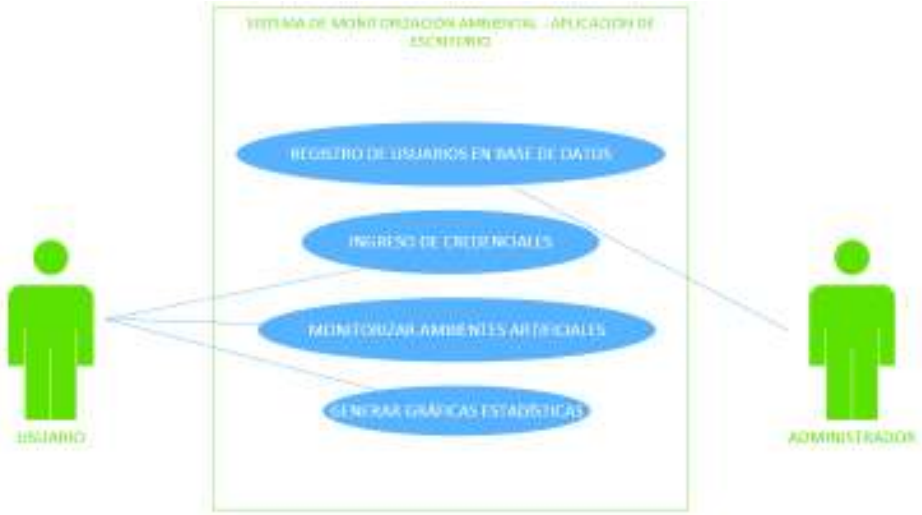

Figura 2.13 Casos de uso de la aplicación de escritorio

Para la descripción de los casos de uso se realizarán fichas en las que se definen y se concreta lo que se espera de cada una de las funcionalidades del sistema.

Básicamente, se deben definir los actores que participan, una precondición y una postcondición; es decir, deberá responder al propósito del caso de uso y dar un breve resumen la acción. A continuación, se presentan las fichas correspondientes al diagrama de casos de uso de la aplicación.

La Tabla 2.4 presenta el proceso que el administrador seguirá para el registro de usuarios en la base de datos.

Tabla 2.4 Caso de uso registro de usuarios en base de datos

Caso de Uso	Registro de usuarios en base de datos
Autor	Ronald Erazo
Fecha	18/11/2017
Propósito	Tener un control de los usuarios que tendrán permitido el ingreso a la aplicación.
Actores	Administrador, Base de Datos.
Precondición	Instalar MySQL para el desarrollo de la base de datos Usuarios.
Postcondición	Los datos ayudarán a la validación de los usuarios que intenten ingresar a la aplicación
Resumen	El registro de usuarios en una base de datos facilitará la validación de los mismos en el momento en que traten de ingresar a la aplicación.

Posteriormente, en la Tabla 2.5 se muestran los pasos que se seguirán para validar las credenciales ingresadas por el usuario al intentar acceder al sistema de monitorización.

Tabla 2.5 Caso de uso ingreso de credenciales

Caso de Uso	Ingreso de credenciales
Autor	Ronald Erazo
Fecha	17/11/2017
Propósito	Realizar una validación de usuario para ingreso al sistema.
Actores	Usuario, Aplicación de Escritorio, Base de Datos.

Precondición	Realizar una conexión con la base de datos.
Postcondición	Si los datos son válidos el usuario podrá ingresar al sistema.
Resumen	El usuario debe suministrar sus datos para que éstos sean validados en la base de datos y de ser correctos se permitirá el acceso al sistema; caso contrario, se le negará el acceso.

Una de las opciones que tiene el usuario al ingresar al sistema es realizar la monitorización de los ambientes artificiales del Jardín Botánico, la cual se detalla en la Tabla 2.6.

Tabla 2.6 Caso de uso monitorizar ambientes artificiales

Caso de Uso	**Monitorizar ambientes artificiales**
Autor	Ronald Erazo
Fecha	17/11/2017
Propósito	Visualizar los valores obtenidos por los Nodos Sensores.
Actores	Usuario, Aplicación de Escritorio, Nodo *Gateway*, Base de Datos.
Precondición	Realizar una comunicación serial con el Nodo *Gateway*.
Postcondición	Se visualizará los valores de temperatura, luminosidad y humedad, los cuales quedarán almacenados en la base de datos.
Resumen	Realizada la comunicación serial, el programa se verá capaz de receptar las tramas con los parámetros ambientales, desde el nodo *Gateway*, y éstos serán visualizados por el usuario a través de la interfaz gráfica y almacenados en la base de datos.

Finalmente, con los datos recolectados por los sensores y procesados por la aplicación de escritorio, se podrá generar gráficas estadísticas, que podrán ser visualizadas por el usuario para poder observar el comportamiento de los parámetros ambientales durante el día; para esto, se realizará consultas a la base de datos con el fin de obtener los valores registrados, dependiendo de la gráfica que se desee obtener. El detalle de esta función se visualiza en la Tabla 2.7.

Tabla 2.7 Caso de uso generar gráficas estadísticas

Caso de Uso	Generar gráficas estadísticas
Autor	Ronald Erazo
Fecha	17/11/2017
Propósito	Generar gráficas estadísticas utilizando los valores almacenados en la base de datos.
Actores	Usuario, Aplicación de Escritorio, Base de Datos
Precondición	Realizar una conexión con la Base de Datos. Seleccionar el parámetro, ambiente y fecha para generar la gráfica.
Postcondición	Visualización de gráfica estadística de las variables seleccionadas.
Resumen	Realizada la comunicación serial, el programa se verá capaz de receptar las tramas con los parámetros ambientales, desde el nodo *Gateway*, y éstos serán visualizados por el usuario a través de la interfaz gráfica y almacenados en la base de datos.

d. Diagramas de actividades

En la Figura 2.14 se presenta el diagrama de actividades correspondiente a la aplicación de escritorio, en el cual se puede visualizar el flujo de trabajo, desde el punto de inicio hasta el punto final, entre la aplicación de escritorio y el usuario que ingrese a ésta.

Inicialmente el usuario deberá ingresar a la aplicación, a través de datos que deberán ser validados. En caso de que éstos sean incorrectos, se generará un mensaje de advertencia. Una vez ingresado al sistema, se tendrá que seleccionar en el menú, si se desea monitorizar los ambientes o realizar gráficas estadísticas. En el caso de seleccionar la opción de monitorización, se desplegará una pantalla, con lo que se iniciará un hilo, en el cual se empezará a recibir los datos enviados por el *Gateway* de la WSN; estos datos serán presentados en la pantalla dentro de las cajas de texto correspondientes a cada ambiente y serán también enviados a la base de datos para ser almacenados. Si por otro lado, se elige la generación de gráficas estadísticas; el usuario deberá seleccionar los datos como el ambiente, el parámetro ambiental y la fecha, para poder realizar la consulta. La aplicación capturará los valores obtenidos de la consulta a la base de datos y mediante la librería JFreeChart los interpretará y presentará una gráfica estadística con ellos.

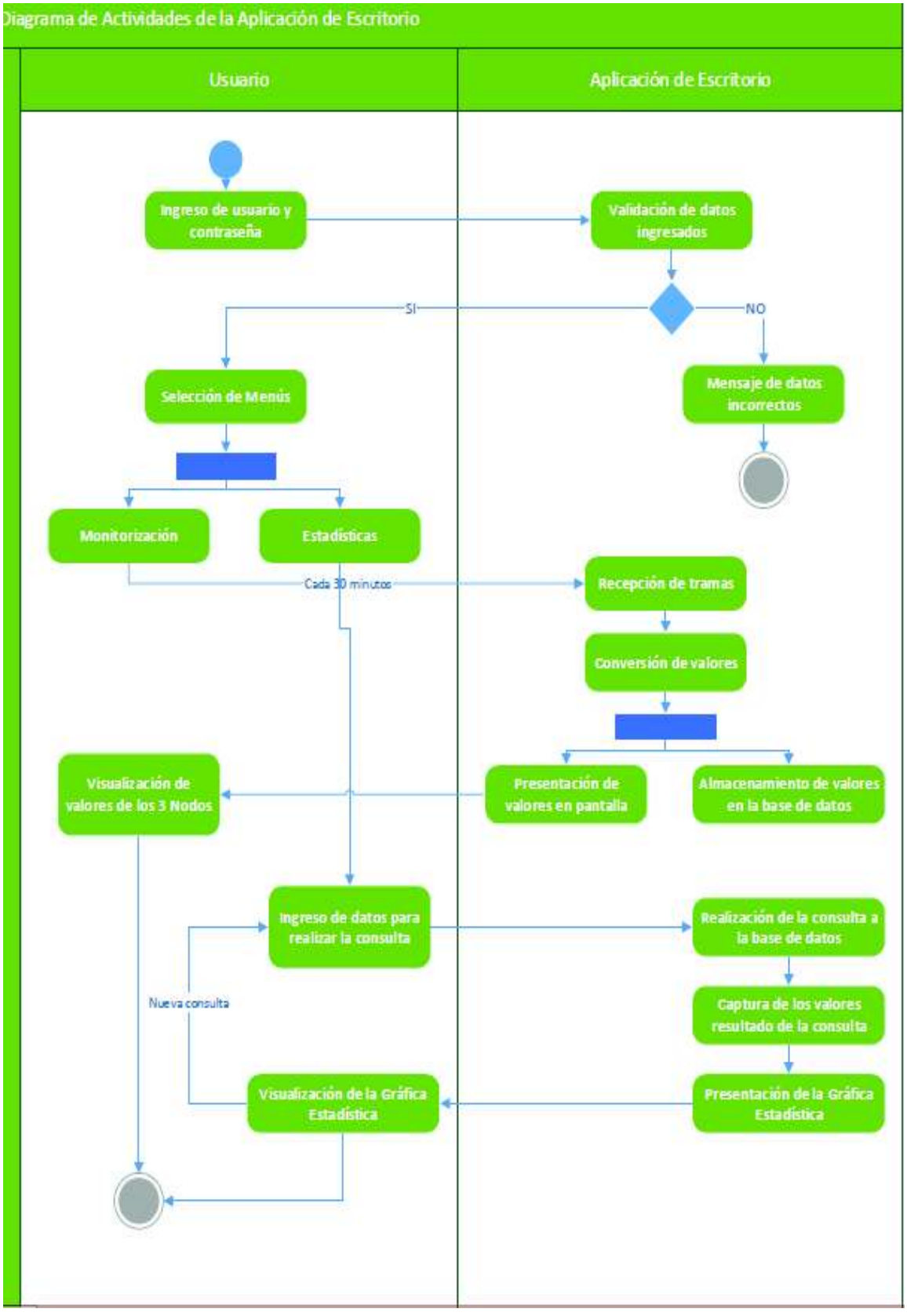

Figura 2.14 Diagrama de actividades de la aplicación de escritorio

e. Diagrama de clases

La Figura 2.15 presenta el diagrama de clases perteneciente a la aplicación "SistemaMonitorizaciónAmbiental".

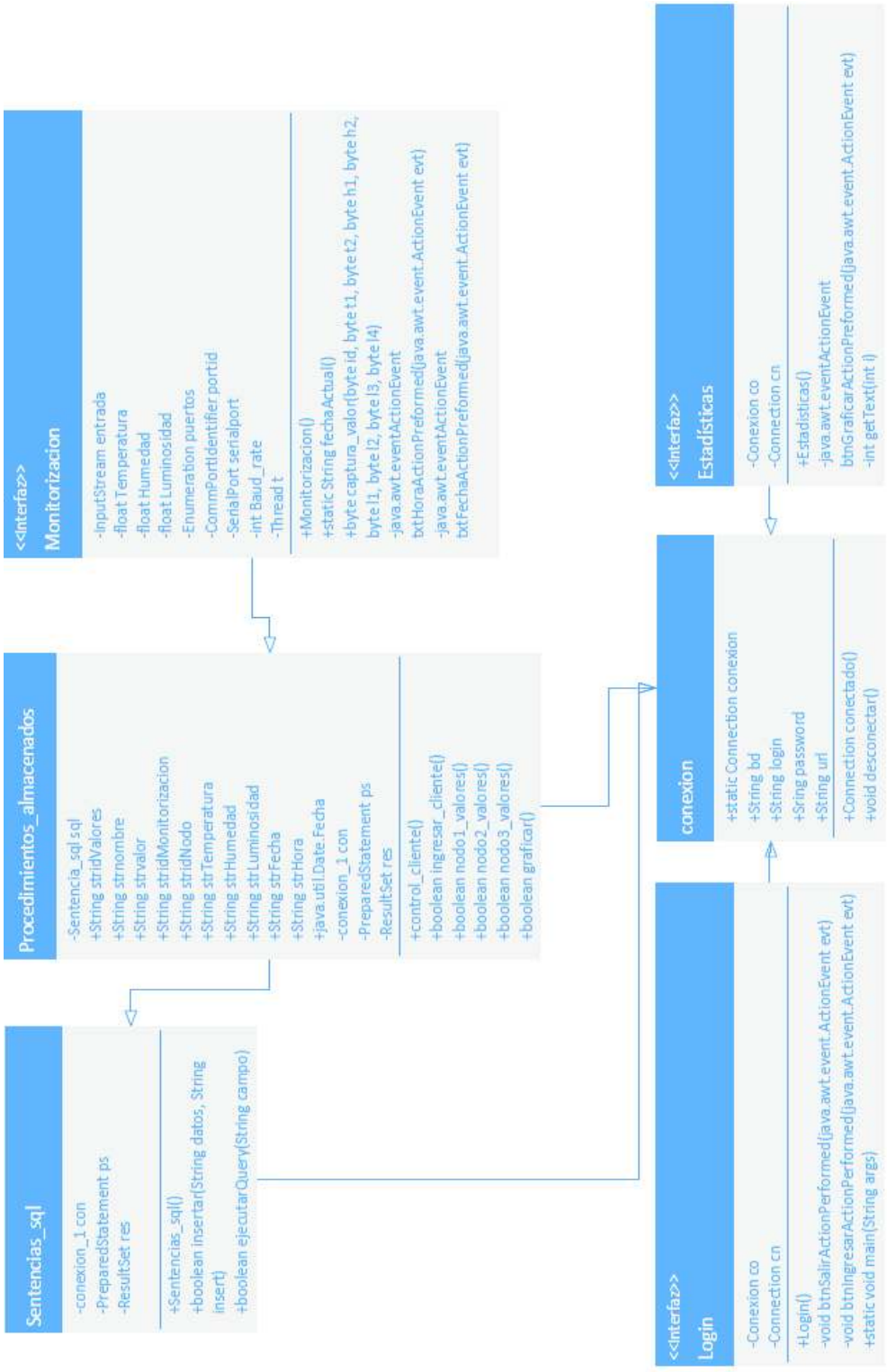

Figura 2.15 Diagrama de clases de la aplicación de escritorio

La aplicación “SistemaMonitorizaciónAmbiental” estará compuesta de las siguientes clases principales:

- **Conexión:** Tendrá métodos encargados de realizar la conexión con la base de datos.
- **Sentencias_sql:** Será la clase encargada de ejecutar las consultas a la base de datos.
- **Procedimientos_Almacenados:** Será una clase que contenga métodos que ejecutarán procedimientos almacenados, en la base de datos, a través del método ejecutarQuery de la clase Sentencias_sql.
- **Login (interfaz):** En esta interfaz se realizará la validación de los usuarios, por medio de la base de datos, para ingreso a la aplicación. Para lograrlo hará uso del método conectado de la clase Conexión.
- **Monitorización (interfaz):** Monitorización será una interfaz en la cual se realizará la comunicación con el nodo *Gateway* y se presentarán los valores, obtenidos por los sensores, a los usuarios.
- **Estadísticas (interfaz):** Por medio de esta interfaz se generará gráficas estadísticas, que permitan observar el comportamiento de los parámetros ambientales durante el día.

2.3.3 Diseño del programa para los nodos sensores

Los nodos sensores iris XM2110CB deben realizar lecturas de temperatura, humedad y luminosidad para luego enviar estos valores a través de su interfaz de radio al nodo *Gateway*, que será el encargado de retransmitirlos a la aplicación de escritorio a través de su interfaz serial.

El desarrollo del software para cumplir con las tareas antes mencionadas requiere el dominio del lenguaje de programación NesC, el cual permite una programación modular a través del enlazado (*wiring*) de interfaces con componentes.

Mediante el diagrama de flujo de la Figura 2.16, se presenta la secuencia que tendrá cada nodo sensor iris en la aplicación “Nodo_Final”.

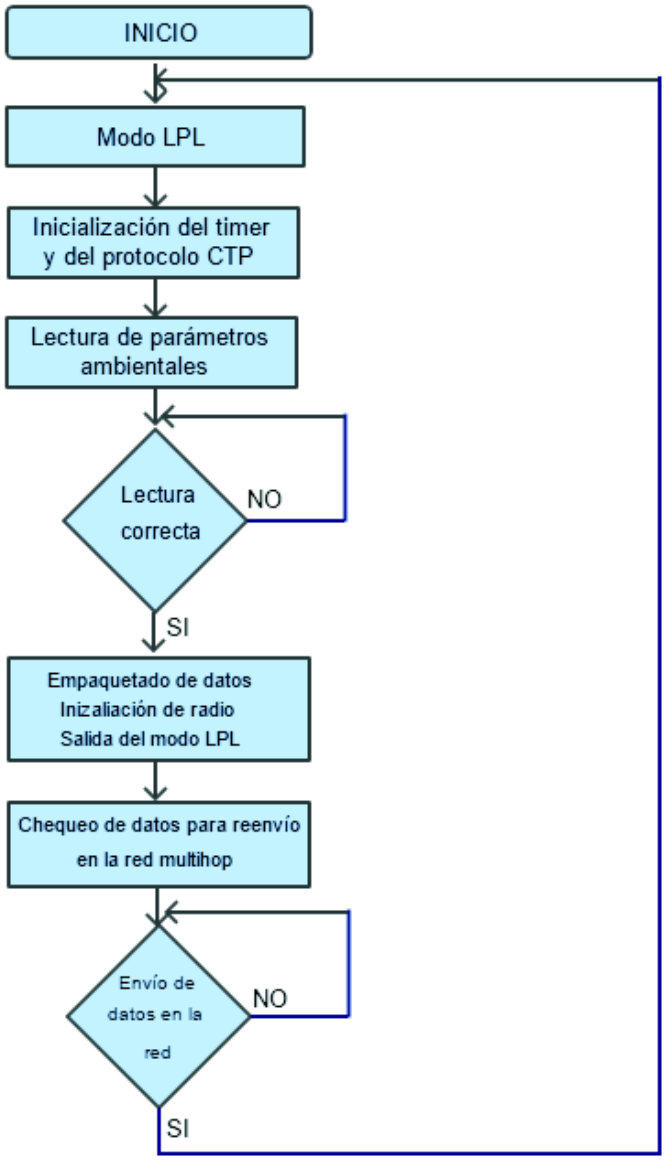

Figura 2.16 Diagrama de flujo para los sensores iris con la aplicación Nodo_Final

A continuación, se describe la secuencia de ejecución que tendrá cada nodo en la red WSN:

- La primera etapa será el encendido, donde cada nodo debe inicializar las variables, protocolos de enrutamiento y estructura de datos correspondiente.
- Cada nodo entrará en modo de bajo consumo LPL (*Low Power Listening*, escucha en bajo consumo). Además se inicializa el temporizador (*timer*), mediante el cual se controla los tiempos de transmisión de datos a la red.
- Las lecturas de luminosidad, humedad y temperatura se realizará en función de la configuración del *timer* y con lo cual se pasará al siguiente módulo.

- Luego se saldrá del modo LPL y se realizará el empaquetado de datos, para ser enviados por radio al *Gateway*, ya que el modo LPL permite recepción de paquetes vecinos de manera paralela a la ejecución del programa. Una vez finalizado el envío de datos se realiza un nuevo llamado a la interfaz LPL, generando un bucle que estará latente mientras un nodo se encuentre encendido.

2.3.4 Diseño del programa para el nodo Gateway

El programa que tendrá cargado el nodo *Gateway* debe permitirle a éste posicionarse como *root* o raíz, con la finalidad de que pueda recibir información para luego procesarla y enviarla a una interfaz serial en un computador.

Mediante un diagrama de flujo que se puede observar en la Figura 2.17, se detalla la secuencia que tendrá el nodo *Gateway*.

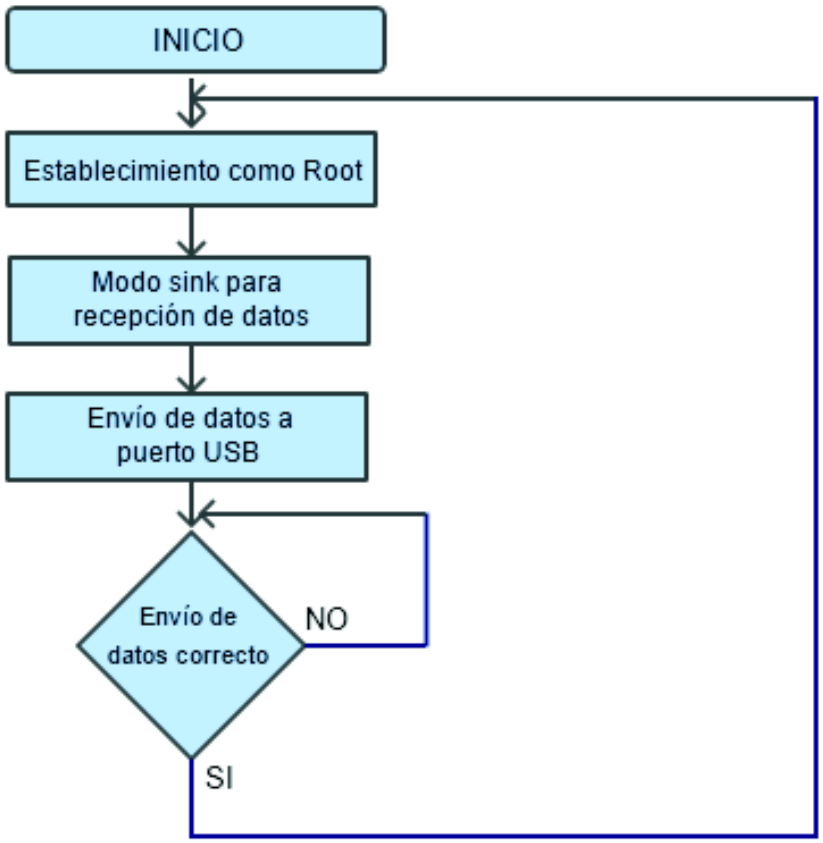

Figura 2.17 Diagrama de flujo para el nodo Gateway

Por lo tanto, para el diseño de este programa se debe cumplir con los siguientes parámetros:

- Establecer al nodo *Gateway* MIB520CB como nodo *root* de la red WSN para recibir toda la información de la misma.
- Procesar la información recibida por el nodo *Gateway* para luego transmitirla vía comunicación serial a un computador.
- Comprobar el correcto envío de datos, como muestra fiable de la transmisión de los mismos hacia el computador que contiene la aplicación colectora, evitando así la pérdida de información.
- Este nodo siempre estará en modo escucha, pues no necesita entrar en modo de bajo consumo, ya que recibe alimentación de la interfaz USB (Universal Serial Bus).

2.4 Implementación del prototipo

En esta sección, se detallará el proceso de implementación de los componentes que forman farte del prototipo.

2.4.1 Implementación de la aplicación de escritorio

Para la implementación de la aplicación de escritorio se utilizó el entorno de desarrollo NetBeans 8.1, con el lenguaje de programación Java. Se empezó creando un nuevo proyecto *Java Application* llamado SistemaMonitorizaciónAmbiental, que será el encargado de recibir y procesar los datos de los sensores.

Se inició por la elaboración de toda la interfaz gráfica del programa, para luego continuar con el desarrollo del código que permita a la aplicación realizar todas las funciones especificadas anteriormente, en los requerimientos del prototipo.

a. Interfaz gráfica de la aplicación de escritorio

La interfaz gráfica fue implementada de acuerdo a lo especificado en la fase de diseño.

- Se inició con el desarrollo de la pantalla de *Login,* cuyo resultado se muestra en la Figura 2.18, la cual cuenta con:

- **3 JLabels (Etiqueta de texto):** lblUsuario, lblPass, Logo
- **2 JTextField (Campo de texto):** txtUsuario, txtPass
- **2 JButtons (Botón):** btnIngresar, btnSalir

Figura 2.18 Pantalla de Login de la aplicación

- Posteriormente, se creó un nuevo *Frame* llamado AmbientesArtificiales que tiene dos opciones para seleccionar, en donde se realizan las tareas de monitorización de ambientes y la presentación de gráficas estadísticas. La Figura 2.19 presenta el diseño del Frame.

Figura 2.19 Pantalla AmbientesArtificiales

El Frame contiene un JDesktopPane llamado desktop y un JMenuBar llamado Menu. El JDesktopPane permitirá mostrar diferentes JInternalFrames, los cuales contendrán las diferentes funcionalidades del programa, que serán elegidas mediante las opciones del JMenuBar.

➢ En la Figura 2.20 se puede observar el JInternalFrame llamado Monitorizacion, que como indica su nombre, permitirá la publicación de los valores de temperatura, humedad, luminosidad obtenidos por los nodos sensores desde los ambientes artificiales. Para la implementación de la interfaz se hizo uso de los siguientes controles:

- **7 JPanels (Panel):** Que servirán para poder realizar una distribución del espacio dentro del Frame.
- **24 JLabels (Etiqueta de texto):** Estos fueron usados para escribir los diferentes títulos, textos en general y para poder insertar las imágenes de los parámetros ambientales que son: temperatura, humedad y luminosidad, ya que en el Lenguaje Java no se puede hacer el uso de un control específico para el manejo de éstas.
- **11 JTextField (Campo de texto):** Los cuales se utilizaron para poder imprimir los datos que son tomados desde la trama enviada por el *Gateway*. Además se utilizarán también para imprimir la fecha y hora actual dentro del programa.
- **9 JSlider (Deslizador):** Estos permitirán tomar los valores de los JTextField y graficarlos dentro de una escala, logrando una visualización animada de los valores obtenidos desde los sensores.

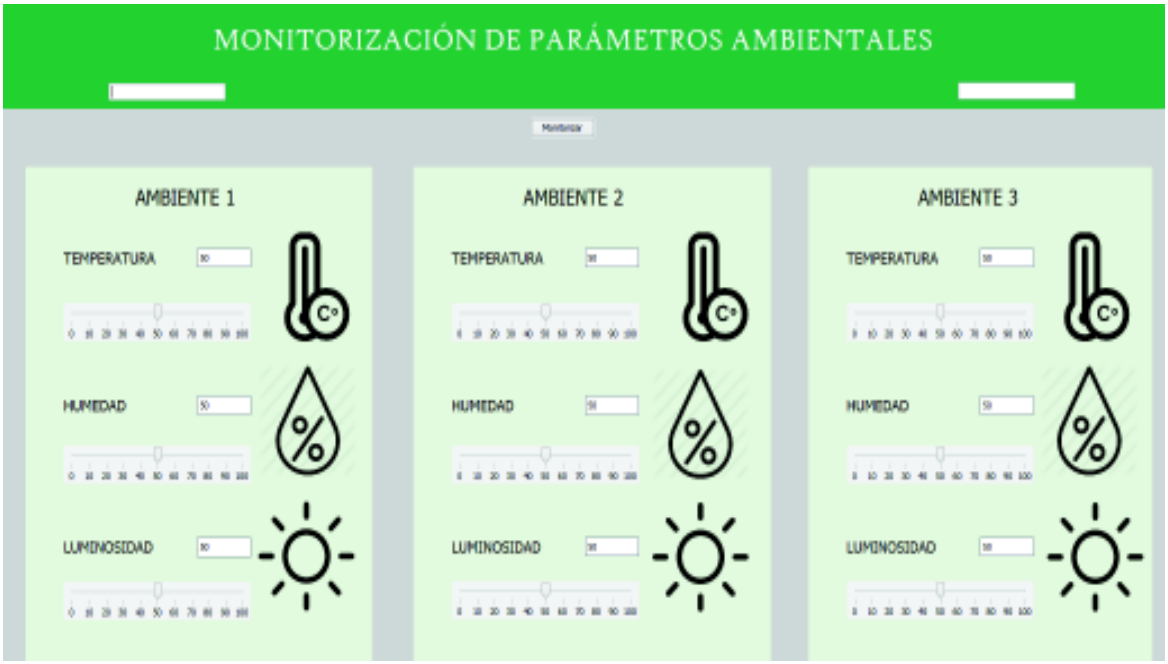

Figura 2.20 Pantalla de monitorización

- Como siguiente paso, se creó otro JInternalFrame llamado Estadísticas que se muestra en la Figura 2.21, cuya función es presentar gráficas con los valores de temperatura, humedad o luminosidad, en una determinada fecha, de los diferentes ambientes artificiales.

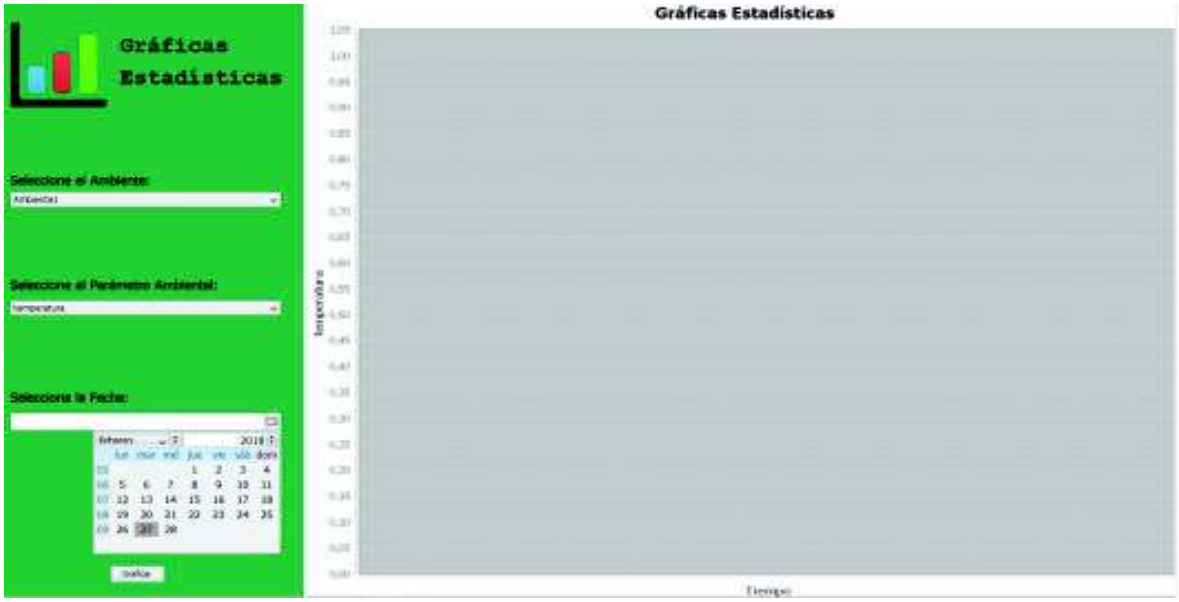

Figura 2.21 Pantalla de gráficas estadísticas

Para la implementación de la interfaz de este JInternalFrame se utilizaron los siguientes controles:

- **2 JPanels (Panel):** Para facilitar la distribución del espacio dentro del Frame, al momento de colocar los controles.
- **7 JLabels (Etiqueta de texto):** Usados para imprimir títulos y textos en general, que servirán para guiar al usuario en el manejo de la interfaz.
- **2 JComboBox (Caja de opciones):** Utilizados para poder seleccionar el ambiente y el parámetro, del cual se desea obtener una gráfica estadística.
- **1 JDateChooser (Selector de fecha):** Para poder capturar la fecha, para poder hacer la consulta a la base de datos, la cual retornará los valores para realizar la gráfica.
- **1 JButton (Botón):** Con el que se enviará a ejecutar la acción de realizar la consulta a la base de datos, para posteriormente imprimir la gráfica en uno de los JPanels.

b. Código de la Aplicación de Escritorio

En esta sección, se explicarán los puntos esenciales del código que permiten realizar las principales funciones de la Aplicación, desde que se realiza la captura de valores obtenidos por los sensores, hasta el almacenamiento de éstos en la base de datos. El código fuente de la aplicación de escritorio se encuentra en el Anexo C.

➢ Comunicación Serial

Para poder realizar una comunicación serial se deben definir los parámetros de comunicación, como son: el puerto de comunicación, la tasa de bits, bits de parada, la paridad, lo cual se puede observar en el Código 2.1.

```
38          float Temperatura;
39          float Humedad;
40          float Luminosidad;
41          CommPortIdentifier portId;
42              //Busca puertos activos y los alamcena en un objeto
43          Enumeration puertos;
44              //Clase encargada de abrir un puerto
45          SerialPort serialport;
46              //Clase para almacenar datos seriales
47          static InputStream almacenar = null;
48              //Velocidad de lectura de un puerto: Nodo iris
            private static int Baud_rate = 57600;
50              //Inicializacion de hilo para lectura serial
51          Thread t;
```

Código 2.1 Creación de variables para la comunicación serial

Para la implementación de la comunicación se creó un hilo, en el cual se realiza primero la identificación de los puertos que están activos y a continuación se establece los parámetros de comunicación, que se pasarán a la clase serialport, la cual se encargará de abrir un puerto para la comunicación. Los datos obtenidos a través del puerto serial serán almacenados en la variable almacenar, que es de tipo InputSream, como lo muestra el Código 2.2.

```
59      puertos=CommPortIdentifier.getPortIdentifiers();
60          //Instanciamos hilo para lectura serial
61          t=new Thread(new Hilo());
62          //Bucle para buscar puertos USB activos
63          while(puertos.hasMoreElements()){
64          //Almacenamos datos en variables portId
65              portId=(CommPortIdentifier) puertos.nextElement();
66              System.out.println(portId.getName());
67          //El nodo Gateway siempre se establece en el puerto USB1
68                  if (portId.getName().equalsIgnoreCase("COM5")){
69                      try{
70          //Parametros que permiten la lectura serial
71                          serialport=(SerialPort)portId.open("Lectura serial", 500);
72                          serialport.setSerialPortParams(Baud_rate, SerialPort.DATABITS_8,
73                          SerialPort.STOPBITS_1,SerialPort.PARITY_NONE );
74                          almacenar =serialport.getInputStream();
75          //Iniciamos hilo para lectura serial
                            t.start();
```

Código 2.2 Hilo para implementar la comunicación serial

- **Procesamiento de datos obtenidos desde el Gateway**

Luego de realizada la comunicación serial se debe procesar los datos que llegan desde el *Gateway*. Estos datos serán almacenados en un arreglo de bytes de tamaño 17, que se encuentra en la línea 214 del Código 2.3, del cual se obtendrán los valores de: Id de la mota (`aux[5]`), temperatura (`aux[9], aux[10]`), humedad (`aux[11], aux[12]`) y luminosidad(`aux[13], aux[14], aux[15], aux[16]`) en formato hexadecimal, que posteriormente deberán ser transformados a formato decimal. Para realizar la validación de la trama se utilizó el byte 7 (`aux[6]`), cuyo valor siempre deberá ser 8 en caso de ser una trama válida, como se observa en la línea 219 del Código.

```
207     private class Hilo implements Runnable{
208
209     @Override
        public void run(){
211         while(true){
212         //Comprobamos el booleana para iniciar timer
213             try {
214                byte []aux = new byte[17];
215                byte valor;
216                valor = (byte) almacenar.read(aux);
                   Thread.sleep(50);
218                if (valor>0){
219                if(aux[6]==8){
220         //Se captura los valores que contiene los datos dentro de la
221                procesamiento_valores(aux[5],aux[9],aux[10],aux[11],
222                          aux[12],aux[13],aux[14],aux[15],aux[16]);
223                       }
224              }
```

Código 2.3 Recepción de datos desde el nodo Gateway

Para el procesamiento de estos valores se creó un método llamado `procesamiento_valores`, que se encuentra en la línea 109 del Código 2.4. Este método tomará de argumentos los valores capturados en el arreglo mencionado anteriormente y procederá a realizar la conversión de un formato a otro de cada uno de los parámetros ambientales.

```
109     public  void procesamiento_valores(byte id ,byte t1,byte t2,byte h1,byte h2, byte l1, byte l2, byte l3, byte l4){
110         /*-------Conversion de datos para voltaje y Temperatura-----------*/
111
112             control_cliente control = new control_cliente();
113
114             int i=(int) ((id));
115             String estado="estado On";
```

Código 2.4 Creación de método procesamiento_valores

En el Código 2.5 se muestra el algoritmo usado en conversión de la temperatura, para pasar de hexadecimal a decimal.

```
117             //Conversión de Temperatura
                    float Temperatura;
119                 int T=(int) ((t1*256)+t2);
120                 Temperatura=(float) (-38.4+(0.0084*(float)T));
121                 System.out.println(id);
122                 System.out.println(Temperatura);
```

Código 2.5 Conversión de valores de temperatura

El Código 2.6 se utilizó para convertir la humedad de hexadecimal a un valor decimal.

```
124         //Conversión de humedad
125             int H= (int) ((h1*256)+h2);
126             int Hum = H;
                float Humedad= (float) ((-2.0468+0.0367*(float)Hum-0.0000015955*Math.pow((float)Hum,
128                             (float )2))+(Temperatura-25)*(0.01+0.00008*(float)Hum));
129             System.out.println(Humedad);
```

Código 2.6 Conversión de valores de humedad

Y finalmente se realiza la conversión de la luminosidad para lo que se usa 4 bytes, como se visualiza en el Código 2.7.

```
130             //Conversión de Luminosidad
131                 int VisibleLight= (int) ((l1*256)+l2);
132                 int InfraredLight = (int) ((l3*256)+l4);
133                 float CNT1, R, R1, Luminosidad1, Luminosidad2;
134
135                 int chordVal, stepVal;
136                 final int CHORD_VAL[]={0,16,49,115,247,511,1039,2095};
137                 final int STEP_VAL[]={1,2,4,8,16,32,64,128};
138                 chordVal=(VisibleLight>>4) & 7;
139                 stepVal=VisibleLight & 15;
140                 Luminosidad1=CHORD_VAL [chordVal]+stepVal*STEP_VAL[chordVal];
141
142                 chordVal=(InfraredLight>>4)&7;
143                 stepVal=VisibleLight & 15;
144                 Luminosidad2=CHORD_VAL[chordVal]+stepVal*STEP_VAL[chordVal];
145
146                 R=((float)Luminosidad2)/((float)Luminosidad1);
147                 R1=(float) 3.13*R;
148                 float R2= (float)Math.pow((float) 2.41, R1);
149                 float Lux=(float)(Luminosidad1*0.46/ R2);
150                 System.out.println(Lux);
```

Código 2.7 Conversión de valores de luminosidad

Estos valores finales se imprimirán en los JTextFields del JInternalForm llamado Monitorización, el cual fue mencionado anteriormente.

Debido a que se recibe información de los tres nodos, se usará la instrucción `Switch Case`, tomando como argumento el ID del nodo para diferenciar la información que corresponde a cada uno.

A continuación se presenta el Código 2.8, que cumple con esta funcionalidad.

```
157                 switch(i){
158
159                     case 2 :
160                         txtN1Temp.setText(valorTemp);
161                         txtN1Hum.setText(valorHum);
162                         txtN1Lum.setText(valorLum);
163
164                         control.strIdMonitorizacion = "0";
165                         control.strIdNodo= "2";
166                         control.strTemperatura= txtN1Temp.getText();
167                         control.strHumedad= txtN1Hum.getText();
168                         control.strLuminosidad=txtN1Lum.getText();
169                         control.strFecha= txtFecha.getText();
170                         control.strHora=txtHora.getText();
171                         control.nodo1_valores();
172                     break;
173
174
175                     case 3 :
176                         txtN2Temp.setText(valorTemp);
177                         txtN2Hum.setText(valorHum);
178                         txtN2Lum.setText(valorLum);
179
180                         control.strIdMonitorizacion = "0";
181                         control.strIdNodo="3";
182                         control.strTemperatura= txtN2Temp.getText();
183                         control.strHumedad= txtN2Hum.getText();
184                         control.strLuminosidad=txtN2Lum.getText();
185                         control.strFecha= txtFecha.getText();
186                         control.strHora=txtHora.getText();
187                         control.nodo2_valores();
188                     break;
```

Código 2.8 Presentación de valores en pantalla

- **Conexión con la base de datos**

Para la conexión con base de datos se creó una clase llamada Conexión, la cual hace uso de una librería MySQL JDBC Driver, que permite realizar la conexión con una base de datos MySQL. Básicamente se definen los parámetros necesarios para realizar la conexión como: dirección donde se encuentra la base de datos, nombre de la base, usuario y contraseña.

El Código 2.9 muestra la definición de los parámetros y la conexión con la base de datos llamada “tesis”.

```
16      public static Connection  conexion= null;
17      public String bd = "jardinbotanico_db";
18      public String login = "root";
19      public String password = "casia";
20      public String url = "jdbc:mysql://127.0.0.1:3306/"+bd;
21
22
23       public Connection conectado(){
24          try{
25
26            Class.forName("com.mysql.jdbc.Driver");
27
28            conexion =  (com.mysql.jdbc.Connection) DriverManager.getConnection(url,login,password);
29            Statement stm = conexion.createStatement();
30          }
31            catch(SQLException e)
32            {
33
34                JOptionPane.showMessageDialog(null, "ERROR DE CONEXIÓN"+e);
35
```

Código 2.9 Creación de método conectado

- **Manejo de la base de datos**

Una de las principales funcionalidades del código consiste en el manejo de la base de datos, es decir, se debe poder insertar valores que fueron obtenidos desde los sensores, dentro de la base de datos.

Para poder realizar esta operación se creó una clase llamada Procedimientos_Almacenados, la cual tiene tres métodos que ejecutan diferentes sentencias SQL para poder insertar los valores dentro de su respectiva tabla, mediante procedimientos almacenados que están configurados en la base de datos.

En el Código 2.10, se puede observar la implementación de uno de los métodos que permitirán insertar los valores del Nodo1 en la base de datos.

```
56      public boolean nodo1_valores()
57      {
58          String strSQL="";
59          strSQL=strSQL +"call sp_Nodo1 ('"+this.strIdMonitorizacion+"'"
60                  + ",'"+strIdNodo+"','"+strTemperatura+"',"
61                  + "'"+strHumedad+"','"+strLuminosidad+"','"
62                  +strFecha+"','"+strHora+"');";
63          System.out.println(strSQL);
64          sql.ejecutarQuery(strSQL);
65          System.out.println(strSQL);
66          return true;
67      }
```

Código 2.10 Métodos para uso de procedimientos almacenados

- **Gráficas estadísticas**

Otra de las funcionalidades de la aplicación es poder imprimir gráficas estadísticas del comportamiento de los diferentes parámetros ambientales. Para poder generar la gráfica se debe realizar una consulta a la base de datos para que retorne los valores necesarios para poder graficarla. El evento se implementó en el botón `btnGraficar`, el cual se puede observar en el Código 2.11.

De la línea 201 a la línea 206, se obtiene los valores para realizar la consulta. El ambiente y parámetro ambiental, a los que se desea analizar, se los selecciona de dos combobox; mientras que la fecha será seleccionada de un JCalendar, estos valores se los utilizará en la consulta que está definida en las líneas 209 y 210 del Código.

```
198     private void btnGraficarActionPerformed(java.awt.event.ActionEvent evt) {
199         // TODO add your handling code here:
200
201         String ambiente = cmbAmbiente.getSelectedItem().toString();
202         String parametro = cmbParametro.getSelectedItem().toString();
203         String dia = Integer.toString(jdtCalendario.getCalendar().get(Calendar.DAY_OF_MONTH));
204         String mes = Integer.toString(jdtCalendario.getCalendar().get(Calendar.MONTH)+1);
205         String year = Integer.toString(jdtCalendario.getCalendar().get(Calendar.YEAR));
206         String fecha = (year + "/" + mes + "/" + dia);
207         try{
208
209             String query = "select hora," + parametro +" from "
210                     + ambiente + " where fecha ='"  + fecha + "'" ;
```

Código 2.11 Evento del botón Graficar

Los valores que retornen de la consulta serán almacenados en la variable `dataset`, como se observa en la línea 211; que serán utilizados como argumentos para generar la gráfica en las líneas 212 y 213.

```
211             JDBCCategoryDataset dataset=new JDBCCategoryDataset(cn,query);
212             JFreeChart chart= ChartFactory.createLineChart("Gráficas Estadísticas","Tiempo",
213                     parametro,dataset,PlotOrientation.VERTICAL,false,true,true);
214             BarRenderer renderer = null;
215             CategoryPlot plot = null;
216             renderer= new BarRenderer();
217             ChartFrame frame = new ChartFrame("Gráfica Estadística", chart);
218
219             ChartPanel panel = new ChartPanel (chart);
220             panelGrafica.setLayout(new java.awt.BorderLayout());
221             panelGrafica.add(panel);
222             panelGrafica.validate();
223           } catch (Exception ex) {
224         }
```

Código 2.12 Consulta de valores para generar gráfica estadística

2.4.2 Implementación de la aplicación Android

Para la implementación de la Aplicación Android se utilizó el entorno de desarrollo Android Studio. De igual forma, se inició creando una nueva App llamada

"MonitorizacionJardinBotanico", como se muestra en la Figura 2.22, cuya función principal es monitorizar remotamente los ambientes artificiales a través de la base de datos.

Inicialmente se elaboró la interfaz gráfica de la aplicación, para luego continuar con el desarrollo del código que permita a la aplicación realizar las tareas designadas.

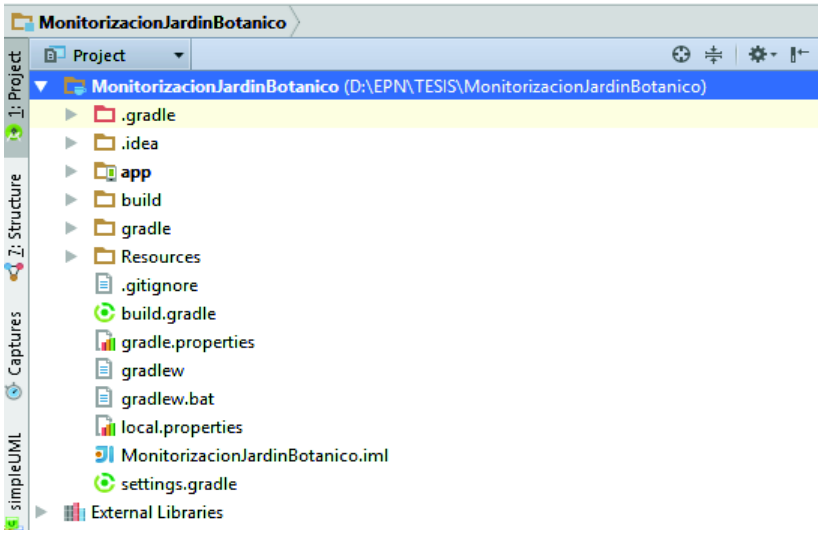

Figura 2.22 Creación de la App MonitorizaciónJardínBotánico

a. Interfaz gráfica de la aplicación Android.

La interfaz gráfica fue implementada de acuerdo a lo especificado en la fase de diseño.

- Se inició con el desarrollo de la página de *Login* como se muestra en la Figura 2.23, la cual cuenta con los siguientes controles:

- **1 TextField (Campo de texto):** Usado para el ingreso del usuario.
- **1 PasswordField (Campo de contraseñas):** En el que se ingresará la contraseña.
- **1 Button (Botón):** En donde se programará la validación del usuario que desee ingresar a la aplicación.

Figura 2.23 Pantalla Login de la App

- Una vez realizada la validación se abrirá la pantalla de la Figura 2.24, en donde se tiene un menú para poder ingresar a los diferentes ambientes. Esta pantalla contará con siguientes controles:

- **1 TextView (Etiqueta de texto):** Que tendrá un mensaje para indicar al usuario que debe seleccionar un ambiente.
- **3 Buttons (Botón):** Los cuales permitirán al usuario ingresar a cada uno de los diferentes ambientes, para poder consultar los valores de los parámetros ambientales.

Figura 2.24 Menú para selección de ambiente

- Finalmente se tendrán 3 pantallas, presentadas en la Figura 2.25, que representan a cada uno de los ambientes artificiales y éstas permitirán visualizar los últimos valores registrados en la base de datos. Para el diseño de esta pantalla, se usó los siguientes controles:

- **4 TextViews (Etiqueta de texto):** En donde se especifica el ambiente y el nombre de los parámetros medidos.
- **3 Textfields (Campo de texto):** Que será donde se muestre los valores recuperados de la base de datos.
- **1 Button (Botón):** Que ejecutará el evento para realizar la consulta a la base de datos.
- **3 SeekBar (Barra):** Donde se mostrará de manera gráfica los valores medidos dentro de una escala.

Figura 2.25 Pantallas para visualización de valores de los parámetros ambientales

b. Código de la aplicación Android

En este punto, se explicará el código de las principales funcionalidades de la aplicación Android. El código fuente completo de la aplicación Android se encuentra en el Anexo D.

- **Conexión con la base de datos**

Para realizar la conexión con Base de Datos se creó un método llamado `conexionBD`, que utiliza la librería MySQL JDBC Driver. En el Código 2.13, se puede observar la definición de los parámetros necesarios para realizar la conexión como: dirección y puerto para acceder a la base de datos, el nombre de la base, usuario y contraseña.

```
public Connection conexionBD(){
    Connection conexion=null;

    try{
        StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
        StrictMode.setThreadPolicy(policy);
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        conexion= (Connection) DriverManager.getConnection
                ("jdbc:mysql://192.168.1.10:3306/jardinbotanico_db,root,tesis");


    }catch (Exception e){
        Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_SHORT).show();

    }
    return conexion;
```

Código 2.13 Método para conexión con la base de datos

- **Consulta a la base de datos**

La aplicación Android tiene como principal tarea permitir una monitorización remota de los ambientes artificiales; para conseguirlo se realizará una consulta a la base de datos, obteniendo los últimos valores ingresados. En el código 2.14, se muestra el método `obtenerValoresBD`, que recuperará los últimos valores de temperatura, humedad y luminosidad del ambiente artificial correspondiente.

```
public List<Valores> obtenerValoresBD(){
    List<Valores> valor=new ArrayList<>();

    try{
        Statement st = conexionBD().createStatement();
        ResultSet rs = st.executeQuery
        ("SELECT temperatura, humedad, luminosidad from ambiente1 order by idmonitorizacion desc limit 1");
        valor.add(new Valores(rs.getDouble("temperatura"), rs.getDouble("humedad"), rs.getDouble("luminosidad")));
        temp.setText((CharSequence) valor.get(0));
        hum.setText((CharSequence) valor.get(1));
        lum.setText((CharSequence) valor.get(2));

    }catch (SQLException e){

        Toast.makeText(getApplicationContext(),e.getMessage(),Toast.LENGTH_SHORT).show();
    }

    return valor;
}
```

Código 2.14 Consulta de valores a la base de datos

2.4.3 Implementación de la base de datos

En la implementación de la base de datos se usó MySQL Workbrench 6.3, que es una herramienta visual de diseño de base datos, que evita tener que trabajar mediante una línea de comandos. Se inició creando la base de datos, cuyo nombre es

“JardinBotanico_db”, con nombre de usuario “root”, a la cual se accede a través del puerto 3306; como se puede observar en la Figura 2.26.

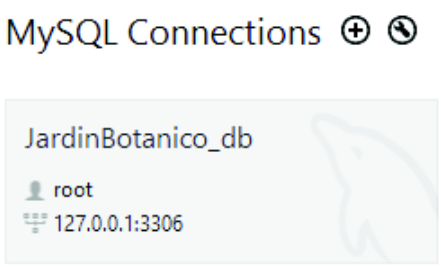

Figura 2.26 Creación de la base de datos

Como se mencionó anteriormente, MySQL Workbrench proporciona una interfaz para el diseño de la base. En la Figura 2.27, podemos ver la creación de la tabla ambiente1, en la que se tendrá los siguientes campos:

- **Idmonitorización:** Que es un campo auto incrementable, y servirá como una clave primaria.
- **Idnodo:** Será el identificador del nodo del que se obtenga los valores.
- **Temperatura, humedad y luminosidad:** Servirán para guardar los valores obtenidos por los nodos sensores.
- **Fecha y hora:** Son campos que almacenarán el instante en el que fueron registrados los datos de los parámetros ambientales.

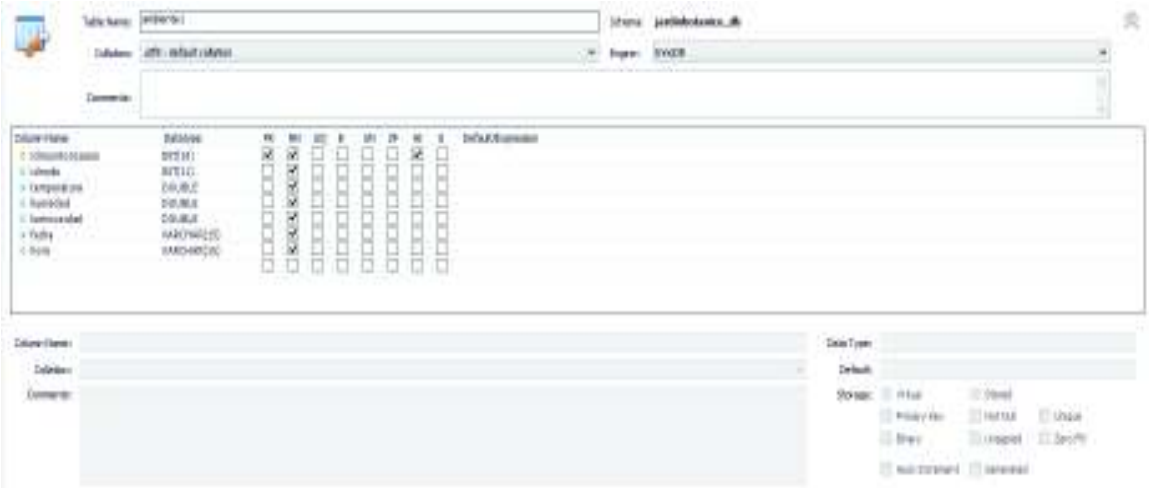

Figura 2.27 Ejemplo de creación de tabla ambiente1

Un similar proceso se utilizó para la creación de las demás tablas de la base de datos, que se presentan en la Figura 2.28 y se detallan a continuación.

- **Usuarios:** Esta tabla tendrá registrados a los usuarios que tiene permitido el acceso al sistema de monitorización y servirá para realizar la validación de los mismos.
- **Ambiente1, Ambiente2 y Ambiente3:** Son las tablas que registrarán los valores que hayan sido tomados por los nodos sensores, después de haber sido procesados por la aplicación de escritorio.

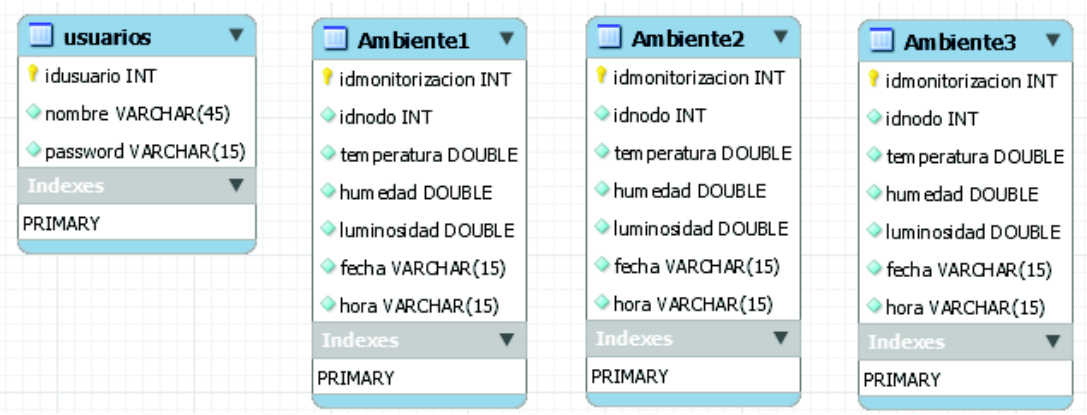

Figura 2.28 Tablas de la base de datos

Para insertar los valores enviados desde la aplicación de escritorio, en la base de datos, se hizo uso de procedimientos almacenados. En el Código 2.15, se observa un ejemplo de uno de ellos llamado “sp_Nodo1”, para el registro de los valores que se hayan tomado en el ambiente artificial 1, perteneciente al invernadero de clima frío.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_Nodo1`(in idmonitorizacion int, in idnodo int, in temperatura double,
in humedad double , in luminosidad double , in fecha varchar(15), in hora varchar(15))
BEGIN

    INSERT INTO `ambiente1`
    (
    `idmonitorizacion`,
    `idnodo`,
    `temperatura`,
    `humedad`,
    `luminosidad`,
    `fecha`,
    `hora`

    )

    VALUES
    (
    idmonitorizacion,
    idnodo,
    temperatura,
    humedad,
    luminosidad,
    fecha,
    hora

    );
END
```

Código 2.15 Procedimiento almacenado sp_Nodo1

2.4.4 Implementación de software para el nodo sensor

Para la implementación de la red WSN fue necesario utilizar las siguientes herramientas, las cuales fueron explicadas en el Capítulo 1:

- Kit de desarrollo MEMSIC que contiene los tres nodos sensores iris XM2110CB y el nodo *Gateway* MIB520CB.
- Sistema Operativo Linux Ubuntu 14.04 LTS de 32 bits.
- Paquete TinyOS 2.1.2 para Linux.
- Software Java-Eclipse 4.2.1
- Librería Yeti-2 para Eclipse.

a. Código de la aplicación "Nodo_Final"

La aplicación para los nodos sensores en la red WSN es "Nodo_Final" (Anexo B), desarrollada en java eclipse y que utiliza las herramientas de programación NesC.

Esta aplicación permite enviar los valores capturados por los nodos sensores al *Gateway*, el consumo eficiente de energía y la realización de lecturas de voltaje y temperatura.

Para la implementación de la aplicación "Nodo_Final" fue necesario desarrollar cuatro ficheros:

- DataMsg.h: Se encarga de definir la estructura de los paquetes de datos a transmitir, a través de una estructura `enum` dentro del mismo en el que se enumera a las constantes que tiene la aplicación.
- Mts400TesterP.nc: Contiene toda la lógica de la aplicación.
- Mts400TesterC.nc: Define los componentes e interfaces utilizados por Mts400TesterC para luego relacionarlos entre sí, a través de un enlazado (*wiring*).
- Makefile: Fichero encargado de contener las reglas de compilación, así como también las rutas a las librerías utilizadas dentro de esta aplicación.

Al momento de compilar la aplicación se genera un único archivo binario que más tarde será cargado en los nodos sensores iris.

- **Fichero DataMsg.h**

La aplicación necesita de tres constantes indispensables: una para el manejo de la transmisión de datos, otra para el funcionamiento del `timer` y finalmente una constante para el modo de escucha LPL. Dentro de este fichero fue definida la estructura `typedef datamsg`, que contiene datos de tipo entero sin signo que son: Id del nodo, temperatura, humedad y luminosidad. Tanto la definición de constante como la estructura de datos se observan en el siguiente código:

```
//definición de constantes
enum{
//constante para la transmisión de datos
AM_DATAMSG=1,
};
//estructura de los datos del mensaje
typedef nx_struct datamsg{
        nx_uint16_t Temp_data;
        nx_uint16_t Hum_data;
        nx_uint16_t VisLight_data;
        nx_uint16_t InfLight_data;
}datamsg_t;
```

Código 2.16 Declaración de constantes y estructura de datos

La constante `AM_DATAMSG` del Código 2.16, está definida de forma hexadecimal, ya que trabaja directamente con el transceptor del nodo sensor que necesita de esta notación; mientras que, las dos constantes restantes trabajan en base a lo que requieran sus interfaces, en este caso ambas constantes definen el intervalo de tiempo en milisegundos.

- **Fichero Mts400TesterP.nc**

Dentro de este fichero se debe incluir a `DataMsg.h` para poder utilizar la estructura antes definida, mediante `#include DataMsg.h`. Además se debe enlistar las interfaces usadas para cumplir con las funcionalidades del sistema.

Al momento de escribir una interfaz dentro de este fichero, obliga a ejecutar su evento; así mismo, para poder crear la lógica del programa es necesario llamar a cada evento con la palabra reservada `call`.

En el Código 2.17, se observa la lista de interfaces utilizadas, así como sus propósitos dentro del programa.

```
#include "DataMsg.h"
module Mts400TesterP {
//interfaz de booteo
uses interface Boot;
//interfaz para uso de leds
uses interface Leds;
//interfaz para control de radio
uses interface SplitControl;
//interfaces para lectura de Temperatura, Humedad y Luminosidad
uses interface Read<uint16_t> as Temperature;
uses interface Read<uint16_t> as Humidity;
uses interface Read<uint8_t> as VisibleLight;
uses interface Read<uint8_t> as InfraredLight;
//interfaz para control de timer
uses interface Timer<TMilli> as TimerTx;
//interfaz para envio de datos
uses interface AMSend;
}
```

Código 2.17 Interfaces utilizadas por el fichero MtsTesterP.nc

La aplicación Mts400TesterP es un fichero tipo módulo, por lo que debe tener dentro de su implementación la declaración de variables, luego se da paso a la ejecución del único evento que no necesita de un llamado `Boot`. Dentro de este evento se procede al llamado del evento `SplitControl.start()`.

```
implementation {
//Variables para datos
uint16_t Temp_data,Hum_data,VisLight_data;
//Estructura del mensaje
message_t message;
//Evento Boot
event void Boot.booted() {
call SplitControl.start();
}
```

Código 2.18 Declaración de variables y del evento Boot

Si el encendido del evento que controla la radio falla se vuelve a intentar hasta que este evento se inicie con éxito. Dentro del evento `SplitControl` llama a la lectura de temperatura, para que posteriormente sean llamados los demás eventos para lectura de humedad y luminosidad, que se observa en el Código 2.19.

```
//Condición en la que si falla la toma de datos, se inicie de nuevo el proceso
event void SplitControl.startDone(error_t err){
if(err==SUCCESS){
                call Temperature.read();
                call Leds.led0On();
                }else{
                call SplitControl.start();
                }
        }
//llamado de método para la lectura de temperatura
        event void Temperature.readDone(error_t err, uint16_t data){
                Temp_data=data;
                call Humidity.read();
        }
//llamado de método para la lectura de humedad
        event void Humidity.readDone(error_t err, uint16_t data){
                Hum_data=data;
                call VisibleLight.read();
        }
//llamado de método para la lectura de luminosidad
        event void VisibleLight.readDone(error_t err, uint8_t data){
                VisLight_data=data;
                call InfraredLight.read();
        }
        event void InfraredLight.readDone(error_t err, uint8_t data){
        InfLight_data=data;
```

Código 2.19 Eventos para lectura de datos

Como se observa en el código 2.20, se procede a hacer un *casting* de datos y se establece una estructura de los datos, y se procede a llamar al evento AMSend para realizar el envío de datos.

```
//Casteo de datos para introducir dentro de la estructura

datamsg_t* packet = (datamsg_t*)(call AMSend.getPayload(&message, sizeof(datamsg_t)));
        packet-> Temp_data =Temp_data;
        packet-> Hum_data = Hum_data;
        packet-> VisLight_data = VisLight_data;
        packet-> InfLight_data = data;
//llamado del evento AMSend para envío de datos
call AMSend.send(AM_BROADCAST_ADDR, &message, sizeof(datamsg_t));
}
```

Código 2.20 Empaquetado de datos

Para que se ejecute periódicamente el código se hace un llamado al evento *timer* el cual se lo ha configurado en 15000 milisegundos. Es decir, se realizará el envío de datos cada 15 segundos, como se puede ver en el Código 2.21.

```
//llamado del evento timer para enviar datos periódicamente
        event void AMSend.sendDone(message_t* bufPtr, error_t error){
        call TimerTx.startPeriodic(15000);
        call Leds.led1Toggle();
        }
        event void SplitControl.stopDone(error_t err){}
        event void TimerTx.fired(){
// TODO Auto-generated method stub
        call Temperature.read();
        }
}
```

Código 2.21 Evento timer y transmisión de datos

- **Fichero Mts400TesterC.nc**

Encargado de relacionar a los componentes con las interfaces utilizadas por Mts400TesterC, dicha operación se la realiza a través del *wiring* (enlazado). Tal como se mencionó en el Capítulo 1, el enlace se realiza de usuario a proveedor, pero al ser una aplicación sencilla sus componentes son provistos. Como primer paso, se enlista los componentes a utilizar en la parte de la implementación, ya que el bloque signatura estará vacío al no proveer su componente a otra aplicación. En el Código 2.22, se observa el listado de componentes junto con su propósito dentro del software utilizado.

```
#include "DataMsg.h"
configuration Mts400TesterC {
}
implementation {
//Componente para inicialización del programa
        components MainC;
//Componente para control de LEDs
        components LedsC;
//Componente para control de Temperatura
        components new SensirionSht11C();
//Componente para contro de Luminosidad
        components new Taos2550C();
//Componente para control de Radio
        components ActiveMessageC;
//Componente para encío de datos
        components new AMSenderC(AM_DATAMSG);
//Componente para control de sensores
        components Mts400TesterP as App;
//Componente para control del Timer
        components new TimerMilliC();
```

Código 2.22 Listado de componentes utilizados por Nodo_Final

Una vez enlistados los componentes, se procede a realizan el enlazado de los mismo a través del operador "->". El fichero Mts400TesterC.nc, es un componente que utiliza las funcionalidades de los demás componentes que se encuentran en la librería TinyOS. Para facilidad se parametriza al fichero como App. En el Código 2.23 se observa el enlazado respectivo.

```
        App.Boot -> MainC;
//Led de Envío
        App.Leds -> LedsC;
//Envío de Datos
        App.AMSend -> AMSenderC;
        App.SplitControl -> ActiveMessageC;
//Temperatura
        App.Temperature -> SensirionSht11C.Temperature;
//Humedad
        App.Humidity -> SensirionSht11C.Humidity;
//Luminosidad
        App.VisibleLight -> Taos2550C.VisibleLight;
        App.InfraredLight -> Taos2550C.InfraredLight;
// Temporizador de Envío
        App.TimerTx->TimerMilliC;
```

Código 2.23 Enlazado entre componentes e interfaces

- **Fichero Makefile**

Se requiere de este fichero cuando se compila la aplicación. En el Código 2.24 se observa el componente que enlaza los recursos de la aplicación Mts400TesterP y el tipo de sensor utilizado (MTS400).

```
COMPONENT=Mts400TesterC

SENSORBOARD=mts400

include $(MAKERULES)
```

Código 2.24 Fichero Makefile

En la Figura 2.29 se observa el diagrama de conexiones que existen entre interfaces y componentes dentro de la aplicación Nodo_Final. El nombre de la interfaz en el diagrama se representa con flechas, mientras que el componente que proveerá servicio a Nodo_Final se encuentra en los recuadros. Cabe recalcar que los componentes que se muestran entrecortados llevan una interfaz parametrizada.

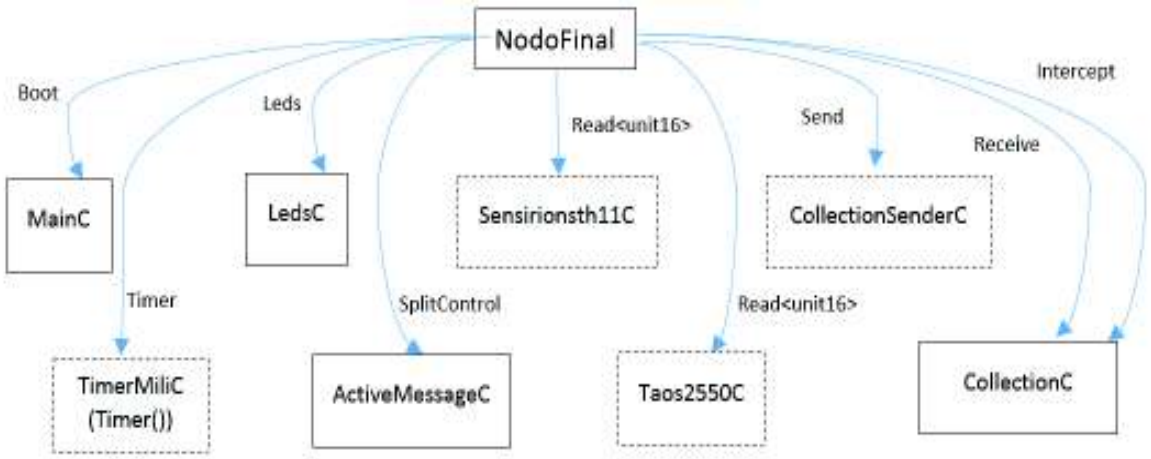

Figura 2.29 Diagrama de conexiones entre interfaces y componentes de Nodo_Final

A continuación, se detalla cada una de las funciones de los componentes mostrados en el diagrama de la Figura 2.29:

- MainC: Se trata de un componente que otorga la capacidad de inicializar las funciones de TinyOS en la aplicación creada.
- TimerMillic: Componente encargado del control de la temporización de las lecturas hechas por la tarjeta MTS400.
- LedsC: Controla el encendido de leds en la placa del nodo sensor iris, esta placa contiene los leds RGB.
- ActiveMessageC: Se encarga del control de envío de datos a la red WSN a través del transceptor de radio.
- Sensirionsth11C: Este componente permite realizar la lectura de la temperatura y humedad requerida en la aplicación Nodo_Final.
- Taos2550C: Componente encargado de realizar la toma del valor de luminosidad requerida en la aplicación Nodo_Final.

Una vez que se elabora los ficheros, se abre una sesión en una terminal de Linux para luego ir a la carpeta que contiene a la aplicación “workspace/Nodo_Final/src” y a continuación compilar la aplicación con un `make`, seguido del nombre de la plataforma, en este caso iris, tal como se puede observar en la Figura 2.30. Dentro de la compilación de la aplicación se muestran datos como el del espacio que ocupará la aplicación NodoFinal en la memoria ROM y RAM, así como mensajes de advertencia.

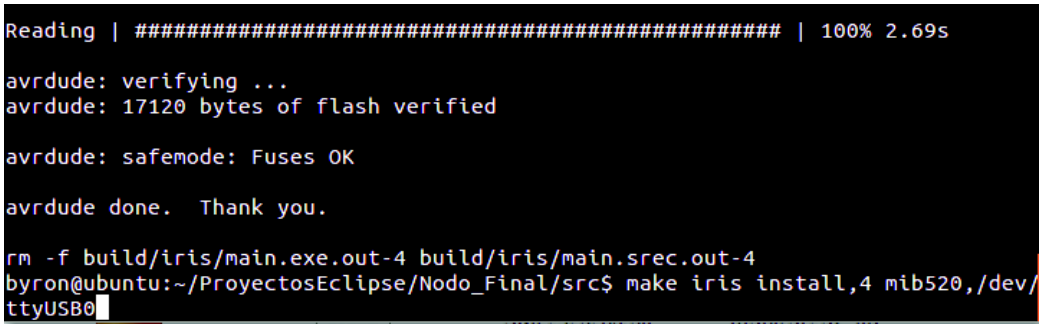

Figura 2.30 Compilación de la aplicación Nodo_Final

Por otra parte, para asignar un identificador a cada nodo sensor, se debe conectar mediante puerto serial el nodo MIB520CB acoplado al nodo al que se desee cargar la aplicación y ejecutar el segmento de código dentro de la carpeta "/Workspace/Nodo_Final/src" en una terminal Linux:

```
Make iris install, N mib520, /dev/ttyUSBx
```

Donde N, representa el identificador que se da al nodo sensor, mientras que MIB520 es la placa programadora que permite transmitir el código seguido del puerto serial x, que se conecta a dicha placa.

2.4.5 Implementación de software para nodo Gateway

Las herramientas usadas para la implementación del software son las mismas que se utilizaron en los nodos sensores iris; es decir, la aplicación eclipse en Ubuntu junto con la librería Yeti-2.

a. Código de la aplicación "Base_root"

Para la implementación de la aplicación "Base_root" (Anexo B), fue necesario implementar tres ficheros:

- **BaseStationP**

Es el fichero encargado de contener la lógica de programación del nodo *Gateway*. Es un fichero privado, debido a la seguridad que debe poseer al tratarse del nodo raíz.

Para que esta aplicación pueda utilizar la estructura serial es necesario incluir la librería mediante un `#include serial.h`, además utiliza interfaces para recepción de radio y envío serial; pues su único objetivo es recolectar los datos de la red TinyOS, procesarlos y enviarlos a una computadora. En el código 2.25, se observan las interfaces utilizadas que sirvieron para cumplir con dicha funcionalidad.

```
//INTERFAZ DE INICIO
interface Boot;
//INTERFACES PARA CONTROL SERIAL Y RADIO
interface SplitControl as SerialControl;
interface SplitControl as RadioControl;
//INTERFACES PARA CTP
interface StdControl as RoutingControl;
interface RootControl;
//INTERFACES PARA COMUNICACION SERIAL
interface AMSend as UartSend[am_id_t id];
interface Receive as UartReceive[am_id_t id];
interface Packet as UartPacket;
interface AMPacket as UartAMPacket;
//INTERFAZ PARA RECEPCION DE DATOS
interface Receive as RadioReceive;
//INTERFAZ PARA CONTROL DE LEDS
interface Leds;
```

Código 2.25 Interfaces utilizadas en BaseStationP

Luego de listar las interfaces, se procede a ejecutar el evento `Boot.booter`, que es el primer evento que se ejecuta en una aplicación TinyOS. Dentro de este se crean dos bucles FOR, los cuales sirven para encerar los buffers tanto para recepción de radio como envío serial; también se valida si dichas interfaces están listas para recibir y transmitir, como se observa el código 2.26.

```
event void Boot.booted() {
//Cola para envió de datos seriales
for (i = 0; i < UART_QUEUE_LEN; i++)
uartQueue[i] = &uartQueueBufs[i];
uartIn = uartOut = 0;
uartBusy = FALSE;
uartFull = TRUE;
//Cola para recepción de datos de radio
for (i = 0; i < RADIO_QUEUE_LEN; i++)
radioQueue[i] = &radioQueueBufs[i];
radioIn = radioOut = 0;
radioBusy = FALSE;
radioFull = TRUE;
//Validacion de interfaces de Radio y serial
if (call RadioControl.start() == EALREADY)
radioFull = FALSE;
if (call SerialControl.start() == EALREADY)
uartFull = FALSE;
}
```

Código 2.26 Evento Boot.booted

Dentro del evento de radio, del Código 2.27 se ejecuta un condicional `if` que sirve para garantizar el encendido de dicha interfaz, para luego permitir que el nodo se posesione como nodo raíz, que en los nodos sensores iris siempre debe tener un ID=1.

```
event void RadioControl.startDone(error_t error) {
if (error == SUCCESS)
radioFull = FALSE;
// El encendido de la radio tiene que ser exitoso para la recepción de datos
call RoutingControl.start();
if(TOS_NODE_ID ==1)
//Establecemos a este nodo como Root
call RootControl.setRoot();
}
```

Código 2.27 Evento para encendido de radio

Una vez encendida la interfaz de la radio en el *Gateway*, este se encuentra listo para la recepción de datos, la cual se realiza en el evento `RadioReceive` que se ejecuta inmediatamente después de que el nodo se posesiona como raíz. Dentro de este evento, los datos son tratados de acuerdo con el orden de llegada FIFO (*First in First out*) para que esto suceda se utiliza booleanos, encargados de garantizar la disponibilidad. Lo anteriormente expuesto se observa en el código 2.28, que utiliza la lógica para traspaso de la información.

```
message_t* receive(message_t *msg, void *payload, uint8_t len) {
message_t *ret = msg;
//este evento tiene que garantizar atomicidad para el correcto envió de datos
atomic {
//Se procesa un dato si la interfaz está desocupada
if (!uartFull)
{
ret = uartQueue[uartIn];
uartQueue[uartIn] = msg;
// Se crea una cola de espera para los datos
uartIn = (uartIn + 1) % UART_QUEUE_LEN;
//Se comprueba si los datos ingresados son los mismos que los de salida
if (uartIn == uartOut)
uartFull = TRUE;
if (!uartBusy)
{
//Se llama a la tarea uartSend que envía los datos por comunicación serial a la PC
post uartSendTask();
uartBusy = TRUE;
}
```

Código 2.28 Evento para recepción de datos por radio

Una tarea en TinyOS es equivalente a una función en C, que ejecuta un bloque de códigos; así mismo, para llamar a una tarea se ejecuta la palabra reservada post.

La tarea que se encarga de realizar la comunicación serial es `uartSentTask( )`,que se encuentra en el Código 2.29, dentro de la misma se arma el formato serial a enviar, seguido de una atomicidad para garantizar el correcto envío de datos. Además, se realizan varios condicionales IF que controlan la transmisión eficiente del mensaje.

```
am_id_t id;
am_addr_t addr, src;
message_t* msg;
am_group_t grp;
atomic //atomicidad para garantizar la transmisión de datos seriales
if (uartIn == uartOut && !uartFull)
{
//Cambio de estado de booleano para nuevo envío
uartBusy = FALSE;
return;
}
msg = uartQueue[uartOut];//Se pone a mensaje en lista de salida
//Se define la estructura del mensaje
call UartPacket.clear(msg);
call UartAMPacket.setSource(msg, src);
call UartAMPacket.setGroup(msg, grp);
//Comprobacion de envió de datos
if (call UartSend.send[id](addr, uartQueue[uartOut], len) == SUCCESS)
call Leds.led1Toggle();
else
{
failBlink();
post uartSendTask();//Llamada de nuevo al evento
}
```

Código 2.29 Comunicación serial

Los datos no son almacenados en ningún momento en BaseStationP; por el contrario, una vez transmitidos son borrados, pues su único objetivo es transmitirlos a una PC.

- **BaseStationC:**

Es un fichero encargado de ensamblar a las interfaces con los diferentes componentes requeridos para la implementación.

Este fichero se encarga de relacionar a los componentes con sus interfaces, para extraer las propiedades requeridas en el nodo *Gateway* de las librerías de TinyOS.

Como se observa en el código 2.30, se hacen enlaces para controlar la comunicación por radio y serial, además de establecer al nodo como raíz y recepción de paquetes de datos de la red formada por el protocolo CTP.

```
//CONTRO; DE RADIO Y SERIAL
BaseStationP.RadioControl -> Radio;
BaseStationP.SerialControl -> Serial;
//WIRING  PARA COMUNICACION  SERIAL
BaseStationP.UartSend -> Serial;
BaseStationP.UartReceive -> Serial.Receive;
BaseStationP.UartPacket -> Serial;
BaseStationP.UartAMPacket  -> Serial;
//RECEPCION DE PAQUETES DE RADIO
BaseStationP.RadioReceive ->Colector.Receive[0x93];
//ESTABLECIMIENTO  DE NODO COMO ROOT
BaseStationP.RootControl->Colector;
BaseStationP.RoutingControl->Colector;
//CONTROL DE LEDS
BaseStationP.Leds -> LedsC;
```

Código 2.30 Enlazado de componentes en BaseStationC

- **Makefile**

Es un fichero indispensable que contiene las reglas de compilación, así como rutas a librerías usadas y limitaciones del programa. No fue definida una estructura de datos, ya que el nodo *Gateway* encapsula la información recibida de la red a través de la interfaz serial de estructura definida.

Al igual que el anterior fichero, Makefile contiene las rutas y reglas de compilación, con la diferencia de que en este caso al tratarse de una red conformada por tres nodos sensores IRIS, se excluye el uso de ACK para que la red no se ralentice y los nodos puedan estar más tiempo en modo ahorro de energía.

```
COMPONENT=BaseStationC
CFLAGS += -DCC2420_NO_ACKNOWLEDGEMENTS
CFLAGS += -DCC2420_NO_ADDRESS_RECOGNITION
CFLAGS += -DTASKLET_IS_TASK
CFLAGS += -I$(TOSDIR)/lib/net\
-I$(TOSDIR)/lib/net/le\
-I$(TOSDIR)/lib/net/ctp
include $(MAKERULES)
```

Código 2.31 Fichero Makefile de BaseStation

En el código 2.31, el componente llamado a compilar es BaseSationC, que es donde se realiza el enlazado. La Figura 2.31 muestra el diagrama de conexión que existe entre interfaces y componentes en la aplicación BaseStation del nodo *Gateway*.

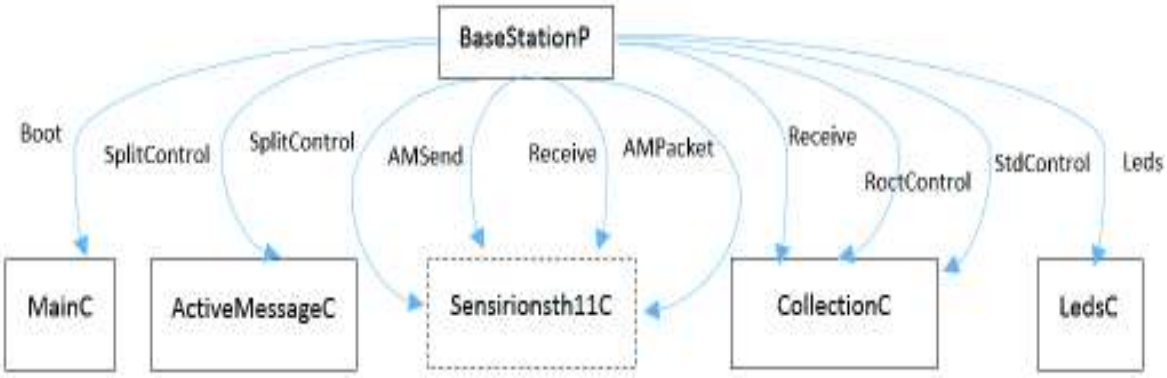

Figura 2.31 Diagrama de conexión entre interfaces y componentes de BaseStation

La función que cumple cada componente se detalla a continuación:

- **MainC:** Función principal de la aplicación que sirve para inicializar los servicios de TinyOS, dentro de ésta se ejecuta el evento `boot.booted` que no necesita ningún llamado y que es punto de partida para la aplicación.
- **ActiveMessageC:** Controla el envío de datos de cada sensor iris a través del transceptor.
- **SerialActiveMessageC:** Componente utilizado para la transmisión de datos vía comunicación serial. A pesar de que, es necesario invocar a los dos eventos (recepción y envió) para respetar las reglas de TinyOS, en el presente trabajo se utilizó únicamente el evento que sirve para enviar datos.
- **CollectionC:** Establece al nodo *Gateway* como *root* (raíz) para que éste entre en modo *Sink* (colector) y recepte los datos de la red WSN.
- **LedsC:** Componente encargado de controlar los leds que permiten informar sobre la recepción de datos.

3. RESULTADOS Y DISCUSIÓN

En este capítulo, se describen las pruebas realizadas al prototipo de monitorización ambiental para el jardín botánico con la finalidad de comprobar que se cumpla con todos los parámetros establecidos en el alcance del proyecto y además los resultados obtenidos en las mismas.

Entre las pruebas realizadas se encuentran: pruebas de comunicación entre los diferentes componentes del prototipo, verificación de los valores obtenidos por los sensores de temperatura, humedad y luminosidad, constatación del correcto funcionamiento de las diferentes funcionalidades del sistema.

3.1 Pruebas del prototipo

Se realizaron pruebas de cada una de las etapas que tiene el prototipo con el fin de comprobar que se cumplió con los requerimientos planteados en el capítulo anterior.

3.1.1 Instalación del software implementado en los nodos sensores y Gateway.

Un aspecto importante es realizar la correcta instalación del software en los nodos, ya que de no ser así pueden producirse errores al momento de transmitir datos. Si existieran errores al momento de cargar los algoritmos de deberá mostrar un mensaje de advertencia.

Como primer paso se carga la aplicación “Nodo_Final”, la cual fue detallada en el capítulo anterior, en cada uno de los nodos sensores Iris XM2110CB. Para poder realizar la carga se utilizó la placa MIB520CB que se debe conectar al computador, donde se encuentra la aplicación, a través del puerto serial. En la Figura 3.1, se puede observar la carga de la aplicación en uno de los nodos sensores.

Figura 3.1 Carga de la aplicación en el nodo sensor

Un aspecto importante que se debe considerar antes de realizar la carga es otorgar los permisos de lectura, escritura y ejecución, a través de la consola de Linux, con el siguiente comando:

```
$sudo chmod 777 /dev/ttyUSB*
```

A continuación se debe cargar la aplicación en el nodo *Gateway*, que será el encargado de recibir los datos de los nodos sensores y transmitirlos a aplicación en computador, dicha aplicación se llama “Base_root” y se la detalló en el capítulo anterior. En la siguiente Figura 3.2 se puede visualizar como se carga la aplicación al nodo *Gateway* a través del puerto serial. De igual forma, que el caso anterior, se deben otorgar los permisos respectivos para hacerlo.

Figura 3.2 Carga de la aplicación del nodo Gateway

3.1.2 Transmisión de datos desde la red WSN

La red de sensores inalámbricos se comunica con la aplicación de escritorio a través del nodo *Gateway*, por medio del puerto serial COM.

El nodo *Gateway* transmite tramas las cuales contienen información de los valores recolectados por los sensores; además del identificador del nodo y ciertas banderas. Estos valores están en un formato hexadecimal y deberán ser interpretadas por la aplicación de escritorio. Un ejemplo de esto se puede observar en la Figura 3.3

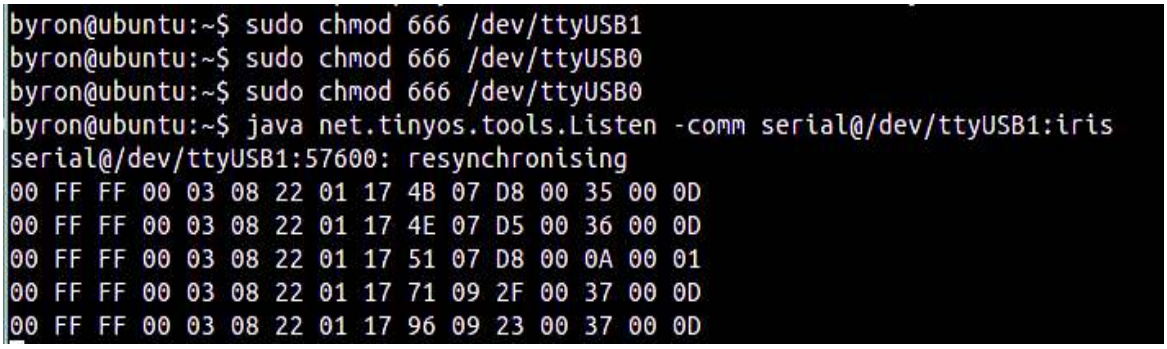

Figura 3.3 Recepción de valores en el nodo Gateway

Los principales campos que dentro de esta trama son:

Byte [5]: Corresponde al identificador del nodo.

Bytes [9] [10]: Conforman el dato de temperatura.

Bytes [11] [12]: Forman el valor de humedad.

Bytes [13] [14] [15] [16]: Construyen el valor de luminosidad.

3.1.3 Pruebas de funcionamiento de la aplicación de escritorio

En la aplicación de escritorio se comprobará la recepción de los datos desde el nodo *Gateway*, el envío de los mismos hacia la base de datos, la obtención de las gráficas estadísticas y generación de notificaciones en caso de que los valores superen los límites recomendables.

a. Recepción de datos

Después de que los datos hayan sido recolectados por el nodo *Gateway*, éste los transmitirá a la aplicación de escritorio, la cual hace el uso de la librería `RxTxcomm`, que utiliza un conjunto de clases para facilitar la comunicación serial entre el computador y el nodo *Gateway*.

Para la recepción de datos se ejecuta un hilo que se encarga de la lectura de tramas y la respectiva extracción de los valores que se encuentra en la carga útil de la trama. Es decir, se extraerán los valores de temperatura, humedad y luminosidad, los cuales serán transformados de bytes a strings, como se presenta en el Código 2.2, para a su vez ser presentados en los `JtextFields` de la pantalla (Monitorizacion.java), que se observa en la Figura 3.4.

Figura 3.4 Interfaz monitorizar

b. Mensajes de advertencia

Los mensajes de advertencia permiten alertar al usuario que se encuentre monitoreando los ambientes, que determinado parámetro sobrepasó los límites establecidos. Un ejemplo se visualiza en la Figura 3.5, en donde la temperatura del invernadero de clima frío ha excedido los límites establecidos

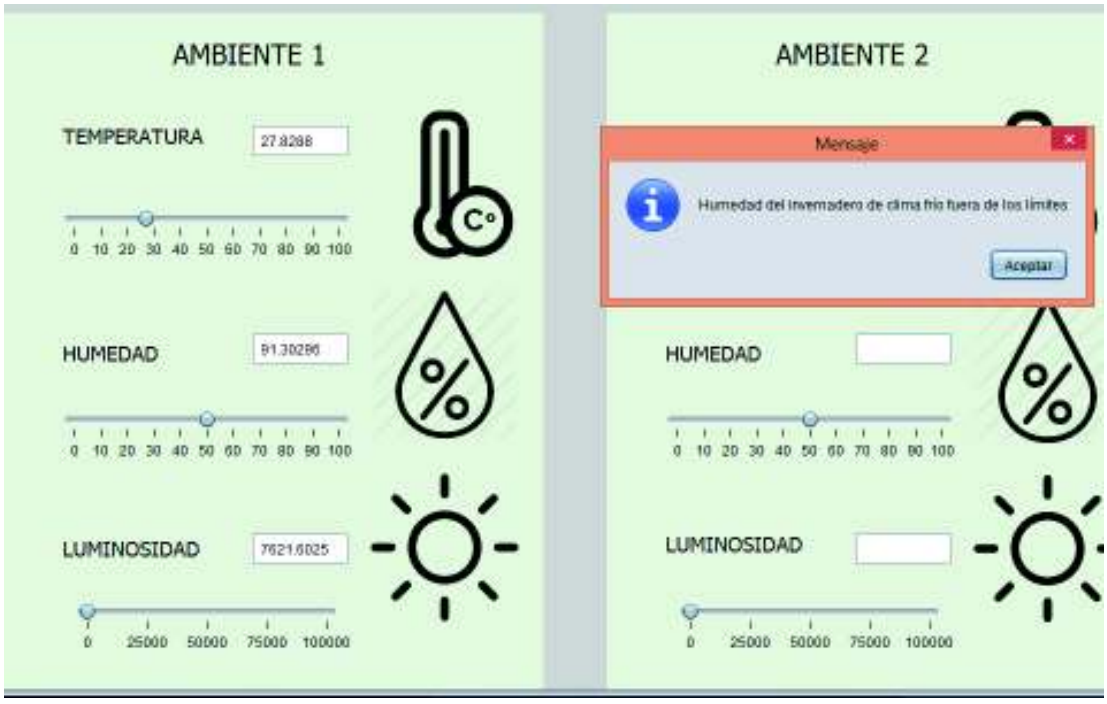

Figura 3.5 Ejemplo de mensaje de advertencia

Posteriormente, la aplicación creará correo electrónico en el que se informará del evento ocurrido y presentará un mensaje para indicar que el mail fue enviado correctamente, como se observa en la Figura 3.6.

Figura 3.6 Mensaje de envío de correo electrónico exitoso

En la Figura 3.7 se aprecia que el correo llegó exitosamente al destinatario, informando de la alerta generada.

Figura 3.7 Correo de alerta en el buzón de entrada

c. Generación de gráficas

Otra funcionalidad de la aplicación es la generación de gráficas estadísticas, para lo cual se debe especificar el ambiente artificial, el parámetro ambiental y el día del cual se desea obtener la gráfica. En la Figura 3.8 se puede apreciar un ejemplo de los ítems que se deben configurar.

La aplicación tomará estos valores para realizar una consulta a la base de datos y a través de la librería JFreeChart, presentar la gráfica estadística.

Figura 3.8 Ítems para la generación de la gráfica estadística

En la Figura 3.9, se puede apreciar un ejemplo en el que se requirió saber el comportamiento de la temperatura, el 17 de abril del 2018, en el ambiente artificial 1. En el eje Y, se tendrá los valores de temperatura, mientras en el eje X estará el tiempo en que se tomaron los datos.

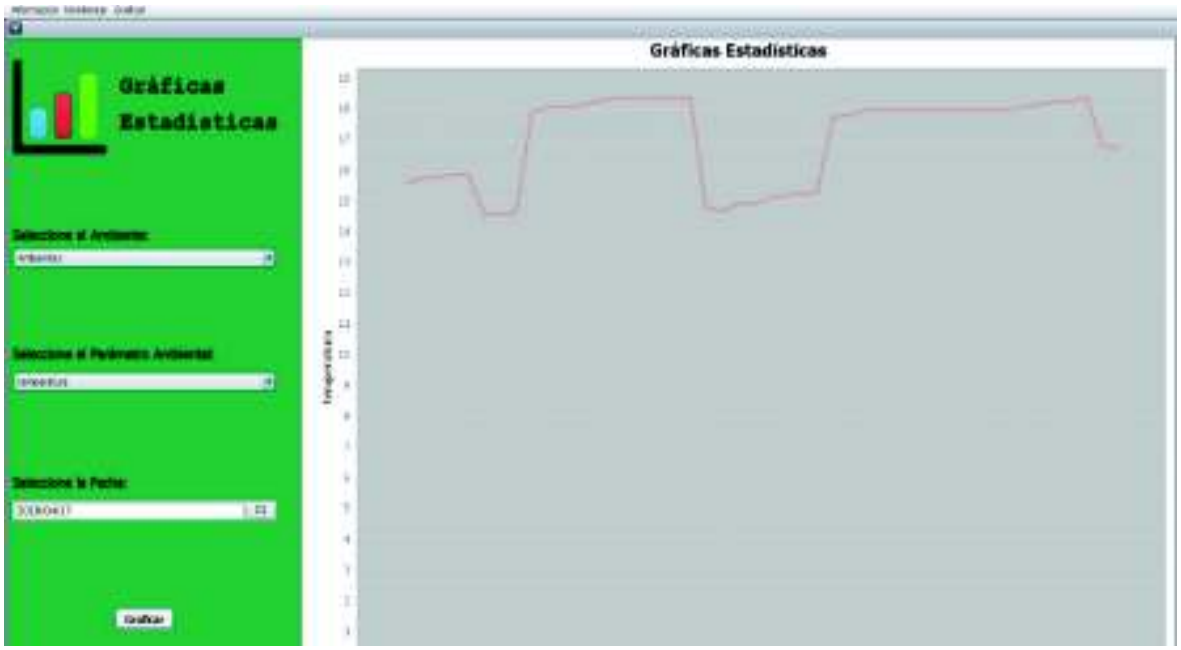

Figura 3.9 Ejemplo de gráfica estadística de temperatura

d. Pruebas de valores receptados

En la realización del proyecto, un gran desafío fue la calibración de los sensores para que los valores capturados coincidan con los datos de los dispositivos de medida del jardín botánico, denominados “valores reales”. Luego de obtener valores estables por parte de los sensores se procedió a tomar 10 medidas en cada ambiente, a diferentes horas, para determinar el margen de error que tiene el prototipo.

Se realizaron las pruebas dentro de 3 ambientes artificiales que forman parte del Jardín Botánico de Quito, los cuales son:

- **Invernadero de clima frío**

En la Figura 3.10 se muestra un ejemplo de los valores recolectados mediante el prototipo versus los valores tomados a través de los dispositivos de medida con los que cuenta el invernadero de clima frío y una aplicación para dispositivos móviles que permite realizar la medición de la luminosidad.

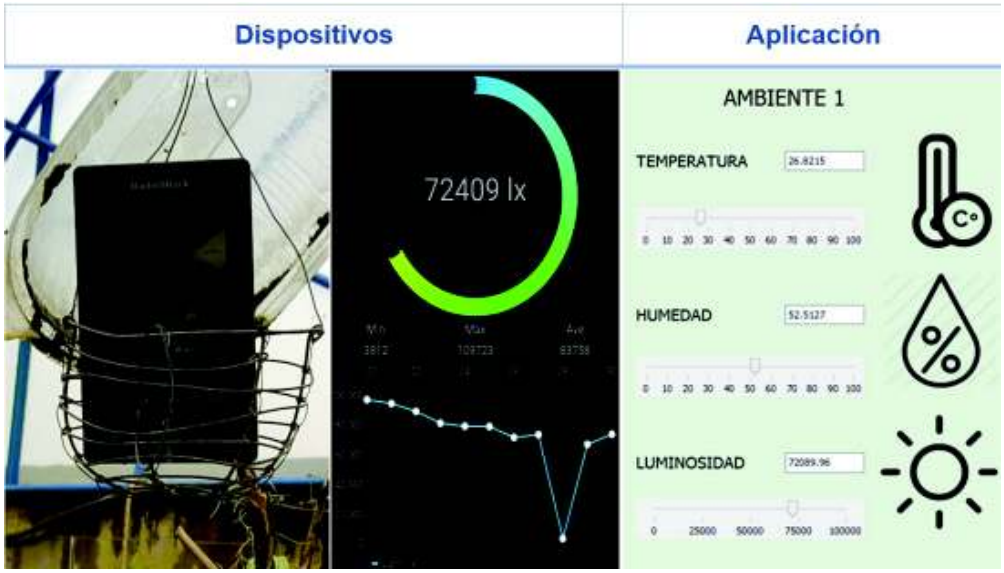

Figura 3.10 Ejemplo de valores tomados en el invernadero de clima frío

A continuación se presentan tablas comparativas con los valores obtenidos y el cálculo del porcentaje de error alcanzado en el nodo 1. Los valores de temperatura se presentan en la Tabla 3.1, los de humedad en la Tabla 3.2 y los de luminosidad en la Tabla 3.3.

Tabla 3.1 Tabla comparativa de valores de temperatura en el invernadero de clima frío

HORA	TEMPERATURA PROTOTIPO (°C)	TEMPERATURA REAL (°C)	% ERROR (%)
07:15	18.6234	17.2	8.14
08:19	18.8211	17.9	5.03
10:07	19.125	18.4	3.80
11:30	21.5928	21.9	1.36
12:32	24.1251	23.1	4.33
13:42	24.8314	26.5	6.42
14:50	25.1211	27.2	7.72
16:00	23.1252	23.8	2.94
17:01	21.3216	19.9	7.04
17:50	19.9213	19.1	4.19
ERROR PROMEDIO NODO 1			5.1%

Tabla 3.2 Tabla comparativa de valores de humedad en el invernadero de clima frío

HORA	HUMEDAD PROTOTIPO (%)	HUMEDAD REAL (%)	% ERROR (%)
07:15	53,1242	51	4.12
08:19	49,9112	51	2.15
10:07	48.5536	50	2.80
11:30	48.9037	50	2.20
12:32	48.0144	49	2.02
13:42	48.1941	49	1.84
14:50	49.1773	49	0.37
16:00	49.7641	48	3.54
17:01	50.5593	49	3.27
17:50	50.9206	49	3.88
ERROR PROMEDIO DE HUMEDAD-NODO 1			2.62%

Tabla 3.3 Tabla comparativa de valores de luminosidad en el invernadero de clima frío

HORA	LUMINOSIDAD PROTOTIPO (%)	LUMINOSIDAD REAL (%)	% ERROR
07:15	41319.31	40355	2.38
08:19	60734.17	60125	1.01
10:07	72823.96	68126	6.89
11:30	74377.12	76952	3.35
12:32	79844.93	80612	0.95
13:42	82276.42	85727	4.02
14:50	80004.99	84693	5.54
16:00	79138.74	80991	2.28
17:01	66933.19	68129	1.76
17:50	63613.42	62415	1.92
ERROR PROMEDIO DE LUMINOSIDAD-NODO 1			3.01 %

Los errores promedios para cada uno de los parámetros ambientales presentados en el nodo sensor 1 son: temperatura 5.1%, humedad 2.62% y luminosidad 3.01%.

- **Invernadero de clima caliente**

En la Figura 3.11 se observa un ejemplo de los valores tomados con el prototipo versus los valores tomados a través de los dispositivos de medición dentro del invernadero de clima caliente.

Figura 3.11 Ejemplo de valores tomados en el invernadero de clima caliente

A continuación se presentan los valores obtenidos y el cálculo del porcentaje de error alcanzado en el nodo 2, los cuales se muestran en las Tablas 3.4, 3.5, 3.6 para los parámetros de temperatura, humedad y luminosidad respectivamente.

Tabla 3.4 Tabla comparativa de valores de temperatura en el invernadero de clima caliente

HORA	TEMPERATURA PROTOTIPO (°C)	TEMPERATURA REAL (°C)	% ERROR
07:15	26.5122	25.1	5.58
08:19	26.8211	25.8	3.88
10:07	27.3257	27.4	0.38
11:30	29.5824	28.9	2.42
12:32	30.5197	29.9	2.01
13:42	31.2963	30.8	1.62
14:50	31.5181	30.7	2.60
16:00	29.1252	28.8	1.04
17:01	27.3216	26.9	1.49
17:50	25.9213	26.2	1.15
ERROR PROMEDIO NODO 2			2.21 %

Tabla 3.5 Tabla comparativa de valores de humedad en el invernadero de clima caliente

HORA	HUMEDAD PROTOTIPO (%)	HUMEDAD REAL (%)	% ERROR
07:15	51.4213	53	3.02
08:19	52.3927	53	1.13
10:07	54.5716	54	1.30
11:30	54.9037	55	0.18
12:32	54.9144	55	0.18
13:42	55.9901	55	1.82
14:50	55.7831	55	1.45
16:00	54.7419	55	0.55
17:01	53.1932	54	1.48
17:50	53.9206	54	0.19
ERROR PROMEDIO DE HUMEDAD-NODO 2			1.13 %

Tabla 3.6 Tabla comparativa de valores de luminosidad en el invernadero de clima caliente

HORA	LUMINOSIDAD PROTOTIPO (%)	LUMINOSIDAD REAL (%)	% ERROR
07:15	53441.25	54918	2.69
08:19	65734.17	63126	4.13
10:07	75323.96	78129	3.59
11:30	81950.22	80237	2.13
12:32	86463.23	85159	1.63
13:42	85765.92	83913	2.21
14:50	83892.51	81252	3.25
16:00	68724.84	76021	9.60
17:01	56312.59	61382	8.26
17:50	53713.42	51038	5.24
ERROR PROMEDIO DE LUMINOSIDAD-NODO 2			4.27%

Los errores promedios para cada uno de los parámetros ambientales presentados en el nodo sensor 2 son: temperatura 2.2%, humedad 1.13% y luminosidad 4.27%.

- **Orquideario de clima caliente**

La Figura 3.12 muestra un ejemplo de los valores tomados con los dispositivos de medición versus el prototipo en el orquideario de clima caliente.

Figura 3.12 Ejemplo de valores tomados en el orquideario de clima caliente

Finalmente en las Tablas 3.7, 3.8 y 3.9 se muestran los valores de temperatura, humedad y luminosidad obtenidos por el nodo 3, y el respectivo cálculo de error.

Tabla 3.7 Tabla comparativa de valores de temperatura en el orquideario de clima caliente

HORA	TEMPERATURA PROTOTIPO (°C)	TEMPERATURA REAL (°C)	% ERROR
07:15	28.5102	28.1	1.42
08:19	28.9171	28.4	1.76
10:07	29.2573	29.7	1.35
11:30	29.5824	28.9	2.42
12:32	30.5197	29.5	3.39
13:42	31.4218	31.8	1.26
14:50	31.7580	30.7	3.26
16:00	30.9817	30.3	1.98
17:01	29.1215	28.9	0.69
17:50	28.9243	28.2	2.48
ERROR PROMEDIO NODO 3			2.01%

Tabla 3.8 Tabla comparativa de valores de humedad en el orquideario de clima caliente

HORA	HUMEDAD PROTOTIPO (%)	HUMEDAD REAL (%)	% ERROR
07:15	54.7193	55	0.54
08:19	54.3712	53	2.45
10:07	52.0116	53	1.69
11:30	50.5137	51	0.98
12:32	48.9475	48	1.87
13:42	44.1201	44	0.22
14:50	41.7831	42	0.71
16:00	42.7419	42	1.67
17:01	43.9732	43	2.09
17:50	44.5207	45	1.11
ERROR PROMEDIO DE HUMEDAD-NODO 3			1.33%

Tabla 3.9 Tabla comparativa de valores de luminosidad en el orquideario de clima caliente

HORA	LUMINOSIDAD PROTOTIPO (%)	LUMINOSIDAD REAL (%)	% ERROR
07:15	59252.91	60047	1.32
08:19	62771.22	65503	4.17
10:07	79223.96	76125	4.07
11:30	84519.52	85982	1.70
12:32	93063.23	92601	0.50
13:42	94136.35	96607	2.56
14:50	86123.87	83592	3.03
16:00	73081.39	75791	3.58
17:01	66012.59	65707	0.46
17:50	61424.61	62532	1.77
ERROR PROMEDIO DE LUMINOSIDAD-NODO 3			1.96%

Los errores promedios para cada uno de los parámetros ambientales presentados en el nodo sensor 3 son: temperatura 2.01%, humedad 1.33% y luminosidad 1.96%.

Se puede observar, que la variación de los valores obtenidos entre los dispositivos que posee el Jardín Botánico y los obtenidos con el prototipo no superan el 10%, para cada caso analizado; mientras que en promedio, el índice de error no supera el 5%; lo cual es un error aceptable y se debe a la interferencia que se puede generar por el material del que son construidos los ambientes artificiales y a la diferente sensibilidad que poseen los sensores del prototipo y de los dispositivos de medición del jardín botánico.

3.1.4 Prueba de funcionamiento de la aplicación Android

a. Validación de credenciales

Al momento de acceder a la aplicación Android el usuario debe ingresar sus credenciales para que éstas sean validadas a través de la base de datos, como se muestra en la Figura 3.13.

Figura 3.13 Ingreso de credenciales en la aplicación Android

En caso de que las credenciales ingresadas no se encuentren en la base de datos o hayan sido ingresadas incorrectamente, se desplegará un mensaje de advertencia que se lo presenta en la Figura 3.11.

Figura 3.14 Validación de credenciales

b. Conexión a la base de datos

Para comprobar el funcionamiento de la conexión con la base de datos, se realizó la consulta de los valores para el Ambiente 1, los cuales se presentan en la Figura 3.12.

Como se puede observar, se logró realizar la consulta con éxito, obteniendo así los últimos registros de temperatura, humedad y luminosidad, almacenados en la base de datos.

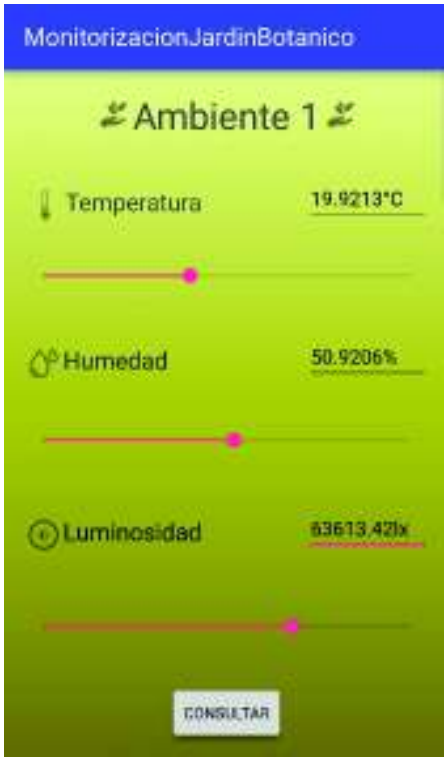

Figura 3.15 Ejemplo de valores monitorizados remotamente por la aplicación Android

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- El uso de redes de sensores inalámbricos incorpora nuevas funcionalidades que benefician a los usuarios finales, debido a la flexibilidad y la posibilidad de automatizar procesos que éstos proveen. En este caso, el prototipo de un sistema de monitorización ambiental utilizando sensores inalámbricos IEEE 802.15.4, genera una mayor eficiencia al momento de obtener valores de los ambientes artificiales y almacenarlos, evitando el uso de personal para la realización de esta tarea.

- El lenguaje de programación NesC, perteneciente a TinyOS, brinda facilidades al programador, gracias a que está orientado a componentes, lo que permitió crear nuevos componentes basados en otros existentes.

- Un complemento importante que se debe añadir al IDE Eclipse es Yeti 2, debido a que facilita la programación en el lenguaje NesC y permite la visualización de errores, además de brindar mayor facilidad para crear directorios necesarios en la implementación del código para los nodos inalámbricos.

- El correcto diseño de los diagramas de casos de uso, de actividades y de clases en la fase de diseño, fueron de vital importancia para tener claras las funciones que debían tener los programas, permitiendo facilitar las acciones a seguir en la fase de implementación.

- El uso de los IDEs Netbeans y Android Studio que proveen mejores herramientas avanzadas para programar y crear interfaces, frente a otros entornos de desarrollo. Lo que facilitó el desarrollo de las aplicaciones para escritorio y dispositivo móviles.

- Para poder calibrar los sensores del prototipo fue necesario usar los dispositivos de medición que posee el jardín botánico e ir modificando el algoritmo hasta obtener los valores más aproximados posibles.

- La infraestructura metalizada que poseen los ambientes artificiales generaba interferencias entre los nodos sensores y el *Gateway* por lo que no se podía sobrepasar los 12 metros de distancia.

- Los valores obtenidos de temperatura, humedad y luminosidad, mediante los sensores que forman parte de nodo inalámbrico iris XM2110CB, difieren con respecto a los valores que fueron tomados con los sensores que posee el Jardín

Botánico, esto se debe a la diferente calidad y sensibilidad que tienen los dos tipos de sensores. Sin embargo, el porcentaje de error promedio no supera el 5% con lo que se puede concluir que es un error aceptable y que el prototipo fue llevado a cabo con éxito.

4.2 Recomendaciones

- Si se desea usar sensores externos a través de la tarjeta MDA300CA, es necesario la revisar el manual de usuario de las placas MTS/MDA de la marca Crossbow para evitar inconvenientes con los valores obtenidos por estos sensores.

- El kit de desarrollo MEMSIC utilizado para el desarrollo del prototipo es de uso estudiantil, por lo que tiene limitaciones como el alcance y susceptibilidad a interferencias en lugares sin línea de vista. Por lo que se recomienda utilizarlos en ambientes abiertos, sin exceder los alcances establecidos en la norma.

- Para la instalación y uso de TinyOS se recomienda usar una máquina física, ya que al hacerlo a través de una máquina virtual produce problemas de procesamiento al momento de compilar los programas, que serán cargados en los nodos inalámbricos.

- Antes de realizar la programación de la aplicación Android, se debe escoger correctamente la versión del sistema operativo que se va a utilizar para que no se genere incompatibilidad con el dispositivo en el que se vaya a instalar.

- Se podría mejorar el prototipo implementando la presentación de un registro, en el que conste los valores que sobrepasen los límites establecidos. Además, se puede modificar la aplicación para que la generación de gráficas estadísticas no esté limitada a un solo día, sino que pueda ser elaborada en rangos de tiempo distintos.

- El presente Trabajo de Titulación puede ser usado para futuros proyectos en los que no se limite solo a la medición de los tres parámetros que son temperatura, humedad y luminosidad; sino que también se puede usar estos valores para automatizar el encendido de las bombas con las que se riega a las plantas.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] J. F. Quintanar Villarreal, “Sistema de monitorización ambiental con redes inalámbricas de sensores para el Insectario del CINVESTAV-IPN Zacatenco,” p. 116, 2010.

[2] “Redes De Sensores Inalámbricos: Elección de los Bloques Funcionales.” [Online]. Available: http://redes-de-sensores-inalambricos.blogspot.com/p/eleccion-de-los-bloques-funcionales.html.

[3] J. Yick, B. Mukherjee, and D. Ghosal, “Wireless sensor network survey,” Comput. Networks, vol. 52, no. 12, pp. 2292–2330, Aug. 2008.

[4] S. Halawani and A. W. Khan, “Sensors Lifetime Enhancement Techniques in Wireless Sensor Networks - A Survey,” May 2010.

[5] A. Koubâa and M. Alves, “Lower Protocol Layers for Wireless Sensor Networks: A Survey.,” Nov. 2005.

[6] J. J. García, H. Directores, A. Javier, G. Sánchez, and F. García Sánchez, “Desarrollo y Estudio de Capacidades de Redes 802.15.4 Usando Dispositivos Micaz.”

[7] “Internet of Things Connectivity Option Analysis: IEEE 802.15.4 technologies – IoT Daily.” [Online]. Available: https://iot-daily.com/2015/03/11/internet-of-things-connectivity-option-analysis-ieee-802-15-4-technologies/.

[8] M. Weiser, “The Computer for the 21st Century.”

[9] I. Stojmenovic, “HANDBOOK OF SENSOR NETWORKS ALGORITHMS AND ARCHITECTURES.”

[10] R. De La Memoria and P. S. Hernández, “Reducción del consumo de energia del protocolo ZigBee implementando un protocolo MAC de bajo consumo.”

[11] “Powerful Sensing Solutions for a Better Life Mote Processor Radio & Mote Interface Boards User Manual MPR/MIB User’s Manual.”

[12] S. Pardo, H. Directora, I. Soraluze, and L. Gardeazabal, “IMPLEMENTACIÓN DE UNA APLICACIÓN DE SEGUIMIENTO DE AVES UTILIZANDO MOTAS IRIS CON EL SISTEMA OPERATIVO TINYOS.”

[13] P. Levis, “TinyOS Programming,” 2006.

[14] J. Manuel, L. Egea, D. Fernando, and L. López, “Estudio e Implementación de un Sistema de Seguimiento de Vehículos con una Red de Sensores Inalámbrica,” 2012.

[15] J. Antonio, T. Galisteo, D. Juan, and M. E. González, “Red de sensores inalámbricos para monitorización de instalaciones eléctricas de baja tensión.”

[16] D. J. Eck, Introduction to Programming Using Java, Sexta. Geneva, NY, 2011.

[17] “Conceptos de objetos y clases en Java. Definición de instancia. Ejemplos básicos y prácticos. (CU00619B).” [Online]. Disponible en: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=411:conceptos-de-objetos-y-clases-en-java-definicion-de-instancia-ejemplos-basicos-y-practicos-cu00619b&catid=68&Itemid=188.

[18] “Basic Concepts of Java Programming.” [Online]. Disponible en: http://mrbool.com/basic-concepts-of-java-programming/30152.

[19] “Lenguaje Java y Entorno de Desarrollo.” [Online]. Disponible en: http://www.jtech.ua.es/j2ee/2006-2007/doc/sesion01-apuntes.pdf.

[20] “Arquitectura Android - Software de Comunicaciones.” [Online]. Disponible en: https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android.

[21] A. J. Gallego, Introducción al desarrollo de aplicaciones Android, Primera. Creative commons, 2017.

[22] A. U. L. Uis and Q. U. R. Ipoll, “Sistemas de gestión de Base de Datos,” p. 21, 2008.

[23] “Definición de MySQL.” [Online]. Disponible en: https://www.definicionabc.com/tecnologia/mysql.php.

[24] I. Howitt and J. A. Gutierrez, “IEEE 802.15.4 low rate - wireless personal area network coexistence issues,” in 2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003., vol. 3, pp. 1481–1486.

[25] Paul Simoneau, “The OSI Model: Understanding the Seven Layers of Computer Networks Expert Reference Series of White Papers,” p. 11.

[26] J. Postel, “Transmission Control Protocol,” p. 85, 1981.

[27] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran, “Study on ZigBee technology,”

in 2011 3rd International Conference on Electronics Computer Technology, 2011, pp. 297–301.

6. ANEXOS

ANEXO A. Instalación de las herramientas de software utilizadas (CD adjunto)

ANEXO B. Algoritmo desarrollado para el nodo sensor Iris XM2110 (CD adjunto)

ANEXO C. Algoritmo desarrollado para la aplicación de escritorio (CD adjunto)

ANEXO C. Algoritmo desarrollado para la aplicación Android (CD adjunto)