UNIVERSIDAD PANAMERICANA®
Aguascalientes

FACULTAD DE INGENIERÍA

SISTEMA DOMOTICO DE CONTROL DE PERSIANAS MEDIANTE DISPOSITIVO MÓVIL CON COMUNICACIÓN ZIGBEE

TESIS

QUE PRESENTA

Carlos Alfonso Acero Juárez

PARA OBTENER EL GRADO DE

MAESTRÍA EN INGENIERÍA

CON RECONOCIMIENTO DE VALIDEZ OFICIAL DE ESTUDIOS DE LA SECRETARÍA DE EDUCACIÓN PÚBLICA, DE ACUERDO CON EL Nº 2007574 DE FECHA 29 DE JUNIO DE 2007

DIRECTORES DE TESIS

Dr. José Sebastián Gutiérrez Calderón

Dr. Pedro Manuel Rodrigo Cruz

Aguascalientes, Ags. Enero de 2017

“A mi Querida y Amada Esposa”

Agradecimientos

En este punto de mi carrera, son muy gratificantes los buenos resultados y lo logros obtenido pero, sin tu apoyo y constante motivación, sin duda el resultado de este trabajo no habría sido el mismo. Eres mi fuente de inspiración y gracias a ti, muchas cosas las he logrado. Junto a ti, seguiré creciendo personal y profesionalmente; Gracias Querida y Amada Esposa.

Agradezco también el apoyo que mis padres me han brindado y el buen ejemplo que ellos siempre predican hacia el trabajar arduamente para lograr lo que uno se propone.

A mis tutores los Doctores José Sebastián Gutiérrez Calderón y Pedro Manuel Rodrigo Cruz por su muy puntual apoyo, sus recomendaciones y aportaciones a este proyecto.

Finalmente agradezco a Dios por brindarme la oportunidad de cerrar un ciclo más en mi carrera profesional con este proyecto y por permitirme continuar en el camino de esta vida.

Resumen

El siguiente proyecto de Tesis es realizado en la ciudad de Aguascalientes, considerando que el nivel socioeconómico de las personas es medio, y partiendo del interés que existe por el uso de tecnología que facilite las actividades de la vida cotidiana, que maximice los tiempos de ocio, o incluso la inversión de tiempo en traslados de un lugar a otro, puesto que Aguascalientes crece de manera considerable.

Esta investigación comprende el diseño y desarrollo de un prototipo para un sistema domótico propio cuyo desarrollo será mediante la implementación de software libre, con la finalidad de reducir los costos de desarrollo y ofrecer un sistema capaz de mejorar la interacción de los usuarios con los dispositivos en el hogar como la automatización de persianas (en su primera etapa), aumentando el nivel de confort y satisfacción en los hogares de la ciudad de Aguascalientes.

Palabras Claves: **sistema domótico, software libre, android, arduino, ZigBee, actuador eléctrico, persianas.**

Abstract

The following project is developed in the Aguascalientes city, considering that the socioeconomic level of the people is average and base on the interest of use of technology that facilitates the activities of the daily life, to improve the leisure times, or even the investment of movement from one place to another, since Aguascalientes grows considerably.

The main intention of this investigation is to design and develop a prototype of a domotic system implementing open source software and low cost hardware to reduce the development cost and provide a functional environment which can interact with user through mobile technology having the capability to operate home devices such as roller shades (first stage), providing high comfort level and user satisfaction in Aguascalientes city homes.

Keywords: **domotic system, open software, android, arduino, ZigBee, electrical end device, rolling shutter, blind.**

Índice

Terminología y Acrónimos

AP – Access Point (Punto de Acceso)

ADSL – Asymmetric Digital Subscriber Line

API – Application Programming Interface

Bluetooth LE – Bluetooth Low Energy

CA – Corriente Alterna

DAO – Data Access Object

DNS – Domain Name System

DTO – Data Transfer Object

FFD – Full Function Device

Gateway (Puerta de Enlace)

GPS – Global Position System

HCI – Human Computer Interaction

HD – High Definition

IDE – Integrated Development Enviroment (Ambiente Integrado de Desarrollo)

IEEE – Institute of Electrical & Electronic Engineers

IoT – Internet of Things

IP – Internet Protocol

Java VM – Java Virtual Machine

Jave ME – Java Mobile Edition

KWh – KiloWatt-Hora

LAN – Local Area Network

LED – Light Emiting Diode

LiPS – Linux Phone Standards Forum

MAC – Media Access Control

OMA – Open Mobile Alliance

PAN – Personal Area Network

PCB – Printed Circuit Board

PCBA – Printed Circuit Board Assembly

PLC – Power Line Carrier (Sistema por Corriente de Portadoras)

PLC – Programable Logic Controller (Controlador Lógico Programable)

RF – Radio Frecuency

RFD – Reduced Function Device

RFID – Radio Frecuency ID

SDK – Software Development Kit

SIG – Bluetooth Special Interest Group

SMT – Surface Mountind Technology (Tecnología de Montaje Superficial)

UI – User Interface

UX – User Experience

WECA – Wireless Ethernet Compatibility Alliance

WiFi – Wireless Fidelity

WLAN – Wireless Local Arean Network

[Índice]

Lista de Figuras

Lista de Tablas

Capítulo 1.
Planteamiento del problema

1.1 Antecedentes

El hombre es creativo por naturaleza y es un ser curioso que desea conocer el entorno, derivado de ello siempre busca interactuar con él, además de tener una vida más llevadera, facilitando la supervivencia, creando viviendas, que han evolucionado en las diferentes épocas adaptándose al entorno.

Pero también adaptándose a las revoluciones industriales y tecnológicas, ya no es suficiente con tener un hogar, sino que se busca optimizarlo con herramientas y tecnologías, implementando primeramente el control remoto. Es a partir de aquí que el mundo de las comunicaciones y la electrónica comienzan un gran desarrollo ofreciendo calidad de vida, seguridad, comodidad y eficacia.

En los últimos años se desarrolla la optimización de hogares introduciendo el concepto de Domótica o automatización en el hogar, si bien es verdad que en la actualidad ha sido poco explotado, se espera un gran crecimiento y que se incremente el número de usuarios.

Por ser una tecnología de reciente implementación, que permite el control y automatización de una serie de dispositivos dentro del hogar, se está distribuyendo a costos elevados con lo que realmente, muy pocos usuarios pueden tener acceso a ella.

Sin embargo la era digital y el avance tecnológico que se vive actualmente, permite que cada vez más los avances en domótica crezcan, que se ofrezcan más funciones a los dispositivos incrementando la funcionalidad y accesibilidad, facilitando su uso e instalación.

Derivado a lo anterior y fundamentado en un análisis y diseño de prototipo de sistema domótico de bajo costo, en el trabajo realizado por investigadores de la Universidad de Antioquía (Barrera Durango, Londoño Ospina, Calvajal, & Fonseca, 2012), en sus conclusiones menciona que es posible desarrollar sistemas domóticas con tecnología propia, para así minimizar los costos y optimizar recursos, es de ahí que surge la idea de este proyecto sobre el interés de investigar y a su vez desarrollar hardware y software libre, con la finalidad de reducir los costos de desarrollo e implementación y ofrecer un sistema capaz de mejorar la interacción de los usuarios con los dispositivos en el hogar específicamente las persianas.

1.2 Justificación

Hoy en día las tecnologías móviles y la electrónica aplicada al hogar se están convirtiendo en herramientas fundamentales y necesarias en la vida cotidiana de las personas. Puertas automáticas, cerraduras electrónicas, estufas eléctricas, son algunos ejemplos de cómo la tecnología cada vez se está integrando más en el hogar. Pero, si la tecnología avanza a pasos agigantados, ¿por qué la inclusión de la tecnología de control en el hogar ha sido tan lenta? Uno de los factores principales ha sido el costo.

En México por ejemplo, se está viviendo un crecimiento en la demanda para adquisición de vivienda del 6.2% en comparación con cifras del 2014 (Federal, s.f.).

A pesar de la gran demanda de vivienda, la mayoría de ellas cuentan con las adecuaciones y los servicios mínimos necesarios con los que un hogar necesita y nunca se contempla la implementación de dispositivos que brinden seguridad, confort y ahorro de energía.

Dada la situación anterior, se plantea en esta investigación el diseño de *hardware* y *software* de domótica para dispositivos móviles de bajo costo con tecnología de uso libre y con una interfaz a través del sistema operativo para móviles Android, para que más personas puedan disfrutar y no quedar limitada a sólo ciertos sectores de la población, así las personas podrán interactuar con los dispositivos de sus hogares, de tal forma que puedan sentirse en un hogar más agradable y haya plenitud de vida.

1.3 Objetivos

1.3.1 General

Diseñar y desarrollar un sistema domótico mediante la implementación de hardware de bajo costo y desarrollo de software para móviles para la mejora en el confort en el hogar en los sectores urbanos de la ciudad de Aguascalientes.

1.3.2 Específicos

- Analizar y seleccionar las tecnologías más adecuadas.
- Definir la estructura de un sistema de comunicación mediante tecnología *ZigBee.*
- Implementar un sistema centralizado de intercambio de información mediante tecnologías de uso libre Arduino.
- Desarrollar una aplicación para móviles en la plataforma Android.

- Integrar el sistema de comunicación, el sistema centralizado y la aplicación Android para la creación de un sistema domótico para el Control de iluminación natural mediante el uso de persianas automatizadas.

Capítulo 2.
Metodología

2.1 Tipo de Investigación

Para efectos del desarrollo de la tesis se trabajará bajo una investigación de tipo Cualitativo la cual se forjará con el análisis de hechos y la descripción de datos que conforman el sistema domótico, cuyas características y aplicaciones lo hacen ser de bajo costo, así como la descripción del hardware y software que se implementará.

2.2 Hipótesis

Mediante la realización de un prototipo propio de sistema domótico de bajo costo, se puede proporcionar acceso a la tecnología de automatización en el hogar a familias de la ciudad de Aguascalientes.

2.3 Diseño de Investigación

Este trabajo presenta un diseño de investigación de tipo No Experimental puesto que su desarrollo pretende estar supervisado. El método cualitativo de manera transversal y descriptiva, verificando así que el sistema domótico puede proporcionar costos bajos para el acceso a la tecnología en los hogares de Aguascalientes.

Capítulo 3.
Marco teórico

3.1 ¿Qué es Domótica?

Desde hace bastantes años se están desarrollando numerosas soluciones para una mayor integración entre los sistemas y equipos domésticos (Junestrand, Passaret, & Vázquez, 2005). La integración tecnológica de los sistemas electrotécnicos en el hogar se ha venido denominando en muchas ocasiones como Domótica. Sin embargo, es importante conocer que, de forma estricta, se define la vivienda domótica como: "Aquella en la que existen agrupaciones automatizadas de equipos, normalmente asociados por funciones, que disponen de la capacidad de comunicarse interactivamente entre ellas a través de un bus doméstico multimedia que las integra" (Junestrand, Passaret, & Vázquez, 2005). Domótica proviene del latín *domus* (casa) y se puede derivar como tal de *Domus Informatics* o su equivalente a "Tecnologías de la Información en el Hogar" (Carbou, Diaz, Exposito, & Roman, 2011). Domótica es otro término de Casa Digital que, además de dispositivos conectados a la red que brindan confort, suele ser parte de un sistema que proporciona seguridad al usuario (Pérez Pérez, 2010).

Los diccionarios franceses incorporaron el término *domotique* a partir de 1998. Esta palabra, traducida al castellano por domótica, es originaria de la palabra latina *domus* (de la que ha derivado de la raíz *domo* que quiere decir *casa*) y de la palabra francesa *informatique* (de la que ha derivado la palabra informática) o, según otros autores, *robotique*. La domótica se aplica a la ciencia y los elementos desarrollados por ella que proporcionan algún nivel de automatización o automatismo dentro de la casa;

pudiendo ser desde un simple temporizador para encender y apagar una luz o aparato a una hora determinada, hasta los más complejos sistemas capaces de interactuar con cualquier elemento eléctrico de la casa (Huidobro Moya & Millan Tejedor, Manual de Domótica, 2010).

La construcción de una concepción bajo el contexto para el desarrollo de este documento se deriva del análisis sobre las referencias dadas en párrafos anteriores, así pues se deduce que domótica se considera a la interacción de dispositivos electrónicos y el usuario, bajo funciones autónomas que posee el hogar sobre actividades cotidianas, mas no va más allá en términos futuristas sino de actualización constante bajo el presente.

3.2 Breve historia de la domótica

El origen de la domótica se remonta a los años 70, cuando, tras muchas investigaciones, aparecieron los primeros dispositivos de automatización de edificios basados en la aún exitosa tecnología X-10. Fue así como comenzó la búsqueda por la creación del hogar ideal con la creación de diversos dispositivos y electrodomésticos que buscaban la automatización y el confort.

Estados Unidos fue uno de los primeros países en implementar sistemas automatizados de regulación de temperatura principalmente en edificios de oficinas. Una década más tarde, a finales de los 80 e inicios de los 90, se comenzaron a construir edificios con tecnología de cableado estructural, el cual tenía como principal objetivo, la conectividad a través de todo el inmueble de diversos servicios y periféricos con cableado estandarizado para la transmisión de señales de voz y control de seguridad, dando con esto y burdamente hablando, el origen de los primeros Edificios Inteligentes**.** A mediados de los años 90, los automatismos destinados a edificios de

oficinas, se comenzaron a aplicar a viviendas particulares y otro tipo de edificios, dando así el origen a la vivienda domótica. Pero no sería hasta la siguiente década, cuando pasa a ser un concepto notablemente conocido por la sociedad (Huidobro Moya & Millan Tejedor, Manual de Domótica, 2010).

Actualmente es más frecuente escuchar el término de Hogar Digital relacionado al hogar moderno donde ya es fundamental e indispensable para los habitantes el tener siempre disponible una conexión a internet, así como la conexión de diversos dispositivos y *gadgets* conectados a la red y con acceso a la información más actual.

3.3 Principales Tecnologías para la Automatización en el Hogar

Como ya se ha explicado anteriormente, en muy breves palabras, la Domótica es la aplicación de la automatización en el hogar. Es una técnica que permite integrar diversas tareas como el control de la iluminación, la climatización o la seguridad en el núcleo de una red doméstica. El desarrollo de la domótica es inseparable de la revolución en las tecnologías de la información y comunicación experimentada en los últimos tiempos. Los hogares son, cada vez más, hogares digitales en los que coexisten dispositivos que reciben, transmiten y procesan información. Las tecnologías domóticas tienden a integrarse con estas redes formando verdaderos hogares inteligentes (Moro Vallina, 2011).

En el hogar digital la instalación eléctrica se desarrolla para convertirse en una auténtica red de control que permite actuar sobre los dispositivos en función de las señales del ambiente (luz, oscuridad, viento, agua, presencia o ausencia de personas), de una programación horaria o de las órdenes enviadas desde dentro o fuera de la vivienda a través de un SMS o Internet. De este modo, la red de control deja de ser una

simple instalación eléctrica y se convierte en una auténtica red de comunicaciones, con unas entradas, salidas y una cierta lógica de control.

La implementación física de dicha red puede llevarse a cabo con diversas tecnologías: unas emplean un dispositivo central que, mediante un programa almacenado en su memoria, toma las decisiones pertinentes; otras se basan en un control descentralizado mediante dispositivos inteligentes que se comunican mediante un bus; las hay que emplean un canal de comunicación "dedicado", mientras que otras se comunican por radiofrecuencia a través de la red eléctrica o incluso mediante la propia red Ethernet doméstica; por último, existe también gran variación en cuanto a costos, complejidad y prestaciones.

Los sistemas que se emplean en la actualidad en las instalaciones domóticas pueden clasificarse según el medio de transmisión utilizado en las siguientes categorías:

- Hay sistemas que emplean autómatas o relés programables, también denominados PLC (*Programmable Logic Controller*), dispositivos que conectan entradas (sensores) y salidas (actuadores) siguiendo un programa lógico o esquema de contactos grabado en su memoria. Se trata de un sistema centralizado en la que todos los elementos se comunican con el PLC.
- En los sistemas por corrientes portadoras o PLC (*Power Line Carrier*) se emplea el cableado eléctrico de la vivienda para transmitir las señales domóticas. Este tipo de protocolo lo emplea principalmente el sistema X10.
- Los sistemas con bus de campo, la comunicación se establece mediante un cableado específico (bus) que comunica todos los nodos de la instalación, proporcionándoles además la alimentación eléctrica, aunque hay nodos que toman dicha alimentación directamente de la red. La información se transmite

en paquetes denominados telegramas. Los dos sistemas más populares que emplean este medio de transmisión son el KNX y el LONWorks.

- En los sistemas inalámbricos, los nodos se comunican entre sí sin necesidad de conexión por cable. Estos sistemas utilizan ondas electromagnéticas de diversas frecuencias.

En realidad, la tecnología domótica tiende cada vez más a crear sistemas híbridos en los que se integran varias de las categorías antes mencionadas, por ejemplo, muchos sistemas de bus o corrientes portadoras incluyen sensores de radiofrecuencia, existen pasarelas que comunican los PLC con sistemas de bus KNX, entre otros.

3.4 Plataforma de Uso Libre: Arduino

3.4.1 Breve descripción del Sistema Arduino

Las nuevas tecnologías y el gran avance en el software y la electrónica de uso libre, han permitido hoy en día el surgimiento de nuevas plataforma que facilitan la implementación e integración de diversas tecnologías para lograr aplicaciones fascinantes en cualquier ámbito. Una de estas plataformas ha sido Arduino.

El proyecto Arduino nació en Italia a mediados de 2005 en el Instituto de Diseño de Ivrea. Surgió por la necesidad de contar con un dispositivo que pudiera interactuar con cualquier sistema operativo, que fuera de bajo costo y que cualquier persona que quisiera realizar un proyecto desde cero lo pudiera usar (Torrente Artero, 2013).

Como tal, Arduino es una placa de hardware libre que incorpora un microcontrolador reprogramable y una seria de pines o conexiones de tipo hembra a las cuales pueden

ser conectadas diferentes tipos de sensores y actuadores para ser utilizados en diferentes aplicaciones.

De igual manera, la placa Arduino, implementa un entorno y lenguaje de programación libre que según la *Free Software Foundation,* debe ofrecer cuatro libertades básicas: usar el programa con cualquier propósito y en cualquier entorno informático, estudiar cómo funciona internamente el programa y adaptarlo a las necesidades particulares, libertad de distribuir copias y la libertad de mejorar el programa haciendo públicas las mejoras a los demás (Torrente Artero, 2013).

Existen diversas placas o modelos de la familia Arduino que, en base a las necesidades o los requerimientos de la aplicación, pueden ser utilizadas para tal propósito. En capítulos posteriores se detallará sobre los diversos tipos de placas Arduino que existen.

3.4.2 Arduino IDE y lenguaje de programación

3.4.2.1. Arduino IDE

La mayoría de los sistemas o placas que integran un microcontrolador embebido, cuentan con un software o entorno visual con el cual pueden realizar la codificación o programación de las instrucciones que le permiten realizar las tareas en base a la necesidad que se tenga.

En el caso de Arduino, se cuenta con un Entorno de Desarrollo Integrado o por sus siglas en ingles IDE (Integrated Development Enviroment) el cual es un software que integra un conjunto de herramientas que permite a los programadores poder desarrollar y codificar las funciones de código a ejecutar en la tarjeta Arduino. Este

software es totalmente gratuito (se puede obtener del sitio web de Arduino: *http://arduino.cc*), de Código Abierto y es totalmente compatible con Windows, Mac y Linux (Amariei, 2015).

La interface de esta herramienta es muy simple y cuenta con los elementos necesarios para la codificación en Arduino. La figura 1 muestra la estructura y apariencia de la interfaz principal de usuario en el sistema operativo Windows.

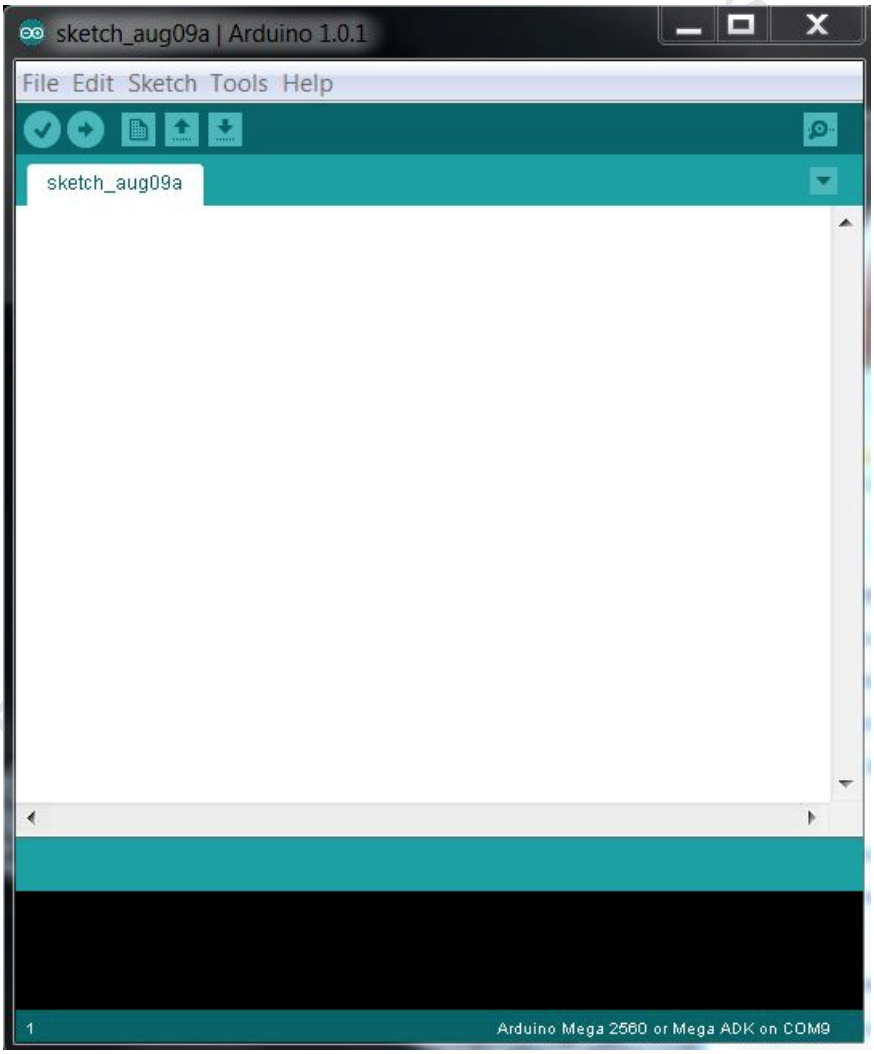


Figura 1. Interfaz de Usuario para programación de Arduino.

Como herramientas principales del IDE de Arduino podemos encontrar:

- **Menú principal**: cuenta con los elementos y las funciones básicas de una aplicación Windows como Archivo, Abrir, Cerrar, Guardar, entre otras funciones adicionales que permiten la interacción con las placas y la integración de nuevas librerías o referencias para funciones particulares en la codificación.
- **Barra de herramientas:** en este elementos contamos con diversos accesos directos a funciones específicas para el IDE. En la Tabla 1se muestran los elementos de la barra de herramientas.

Herramienta	Descripción
	La opción Verificar compila el código y valida que no contenga errores.
	En la opción Cargar nos permite compilar el código y, si no se presentan errores, cargarlo a la tarjeta Arduino.
	La opción Nuevo nos permite iniciar un nuevo sketch.
	Abrir nos permite simplemente abrir algún sketch previamente guardado.
	Como su nombre lo indica, Guardar nos permite guardar el sketch en el que se está trabajando.
	Esta opción permite abrir el Monitor de puerto Serial el cual permite la comunicación con la tarjeta Arduino.

Tabla 1. Elementos de la barra de herramientas IDE (Amariei, 2015).

- **Área de código:** es el lugar donde la codificación de las funciones a ejecutar por el Arduino son llevadas a cabo.
- **Barra de estado:** en esta área se despliegan todos los mensajes de error, notificaciones o información útil complementaria como la cantidad de memoria utilizada por el código o el estatus de descarga de un sketch en la tarjeta Arduino.

3.4.2.2 Lenguaje de programación

Al hablar de un programa en general se entiende como un conjunto de instrucciones ordenadas y agrupadas de forma adecuada y sin ambigüedades que pretende obtener un resultado determinado. Particularmente en el mundo de Arduino, los programas suelen ser llamados también "sketch" (Torrente Artero, 2013).

La plataforma Arduino se basa en el lenguaje de programación C/C++ y se debe considerar que, el programar para una plataforma embebida es un poco distinto que el codificar para plataformas para una PC (Amariei, 2015). Arduino C sigue un conjunto de normas y reglas basadas en sintaxis y estructuras que, básicamente, le dicen a la tarjeta qué es lo que debe hacer. Algunas de estas normas provienen del lenguaje C, el cual es un lenguaje estructural que se ejecuta con un orden desde el inicio o *top* hasta el final o *bottom* (Evans, 2011).

Básicamente dentro de un sketch de Arduino se pueden identificar dos funciones principales esenciales y obligatorias para el programa: la función *setup()* y la función *loop().* La primera de estas funciona para configurar los pines del Arduino, iniciar los protocolos de comunicación o ejecutar alguna función que solamente se desea ejecutar una sola vez. Cabe mencionar que esta función sólo se ejecuta una sola vez, ya sea cuando se energiza la tarjeta o cuando se reinicia (Amariei, 2015).

La función *loop()* se ejecuta continuamente mientras la tarjeta Arduino se encuentre encendida. Todo código que se encuentra dentro de esta función se ejecuta de forma secuencial desde el inicio hasta el final, una vez que la ejecución del programa llega al final, vuelve a iniciar desde el principio de manera infinita hasta que la tarjeta Arduino es des energizada o reiniciada.

3.4.3 Placas Arduino

Actualmente existe una gran variedad de placas Arduino en el mercado y, constantemente, los fabricantes están creando nuevas lo cual hace difícil hablar de todas ellas. Brevemente se mencionarán algunas de las características de las tarjetas Arduino oficiales y algunas adicionales aprobadas por Arduino.

- **Arduino UNO:** esta placa está considerada como la insignia o la principal placa Arduino ya que es la placa más usada, con la que la mayoría de las personas comienzan a adentrarse en el mundo de las placas Arduino y es la más adecuada para la mayoría de los proyectos. La versión más reciente de esta placa es la Arduino UNO R3 (lanzada en el 2011) la cual cuenta con un microcontrolador Atmel ATmega328, 15 entradas / salidas digitales y 6 analógicas. La figura 2 muestra la vista superior de una placa Arduino UNO.

Figura 2. Placa Arduino UNO

- **Arduino Leonardo:** es una de las placas más actuales. Una de sus características más importantes es que su procesador puede ser reconocido por una computadora como un dispositivo de entrada como lo puede ser un mouse o un teclado (Nussey, 2013).

- **Arduino Mega 2560 R3:** esta es una placa más grande que la UNO y está diseñada para aquellas personas que requieren más entradas, más salidas y más poder de procesamiento ya que cuenta con 54 terminales digitales y 16 analógicas.

- **Arduino Mega ADK:** en esencia, es la misma placa que la Mega 2560 con la diferencia de que ésta diseñada para interactuar con teléfonos con sistema operativo Android.

- **Arduino Nano 3.0:** es una placa Arduino miniaturizada de tan solo 0.7 X 1.7 pulgadas. Su principal cualidad es que cuenta con las mismas prestaciones que un Arduino UNO pero en una fracción de tamaño.

- **Arduino Ethernet:** su tarjeta base es un Arduino UNO pero cuanta con la capacidad para realizar conexiones a Internet adaptando una conexión Ethernet y brindando la posibilidad de actuar como un servidor Web.

- **Arduino BT:** brinda la posibilidad de permitir que el Arduino pueda comunicarse con dispositivos Bluetooth teléfonos móviles, tabletas u otros dispositivos.

3.4.4 Placas de Expansión o Shields

Además de las placas Arduino antes descritas, existen otras llamadas *Shields*. Un *shield* (del inglés "escudo") es una placa que se coloca en la parte superior de un Arduino mediante la conexión de varios pines que permiten ampliar las capacidades del Arduino y complementar su funcionalidad de una forma más compacta y estable (Torrente Artero, 2013).

Los *shields* comparten varios pines con las placas Arduino incluyendo las líneas de GND, 5V o 3V3, RESET y AREF, además suelen tomar varias terminales de entrada/salida para lograr la comunicación con la placa Arduino dejando estas bloqueadas para ser utilizadas con algún otro propósito.

Existen diversos *shields* con diversas características y funcionalidades propias pero, algunos de los *shields* oficiales que se distribuyen en el mercado son:

- **Arduino Ethernet Shield:** este *shield* le brinda a la placa Arduino la capacidad de conectarse a una red cableada TCP/IP. Esta placa cuenta con un controlador W5100 y se configura con la librería "Ethernet". También incorpora una tarjeta microSD que puede ser utilizada mediante la librería "SD". Al igual que ocurre con el chip W5100, para poder comunicarse con la tarjeta microSD la placa Arduino utiliza el protocolo SPI.

- **Arduino Wireless SD Shield:** esta placa está diseñada para permitir que una placa Arduino pueda comunicarse inalámbricamente mediante el uso de un módulo XBee o similar. A esta placa también se le puede incorporar una tarjeta microSD.

- **Arduino WiFi Shield:** mediante este *shield* se le puede añadir a la placa Arduino la funcionalidad para conectar al Arduino inalámbricamente a una red TCP/IP. Incorpora el chip HDG104 el cuan incluye una antena integrada que permite conectarse a redes Wi-Fi de tipo 802.11b y 802.11g.

- **Arduino Motor Shield:** este *shield* incorpora el chip L298P el cual está diseñado para controlar componentes que contienen inductores como relés, solenoides, motores de corriente continua o motores de paso. Gracias al chip L298P, este *shield* permite controlar la velocidad y sentido de giro de hasta dos motores de corriente directa o motores de paso de forma independiente.

- **Arduino Proto Shield:** este shield permite de forma fácil implementar circuitos diseñados personalmente. Básicamente lo que ofrece esta placa es un área de trabajo donde se pueden soldar los diferentes componentes electrónicos que se requieran para la implementación de algún proyecto en específico con cierta funcionalidad.

3.4.5 Arduino como Servidor Web

Para que una placa Arduino pueda ser conectada a una red TCP/IP, se le deberán configurar ciertos parámetros necesarios para la conexión a la red. Algunos de estos parámetros son la dirección MAC, la dirección IP, el Gateway, entre otros. Para poder ejecutar estas configuraciones en la placa Arduino se deberá colocar en el "*setup()*" del código la función *Ethernet.begin()*, la cual tiene diferentes formas de escribirse (Torrente Artero, 2013):

- **Ethernet.begin(mac):** donde “mac” se refiere a la dirección MAC de la placa la cual, en los modelos más recientes de Ethernet Shield, vendrá impresa en una etiqueta para usarla como referencia.
- **Ethernet.begin(mac, ip):** donde “ip” representa la dirección IP estática que se le asignará a la placa.
- **Ethernet.begin(mac, ip, servdns):** esta manera es muy similar a la anterior con la diferencia que se le agregar un tercer parámetro que es una dirección IP de algún servidor DNS al cual se conectará la placa.
- **Ethernet.begin(mac, ip, servdns, gateway):** a este método se le agrega la dirección IP de la puerta de enlace o *Gateway* de la red a la cual se conectará la placa.
- **Ethernet.begin(mac, ip, servdns, gateway, subnet):** a esta función se le agrega la posibilidad de configurarle una máscara de subred a la placa.

Una red “TCP/IP” suele tener una arquitectura llamada “cliente-servidor”, en donde, al establecerse comunicación entre dos equipos, uno de estos toma el rol de “cliente” mientras que el otro lo toma de “servidor”. Un cliente básicamente realiza peticiones al servidor, mientras que este último recibe estas peticiones y ofrece la mejor respuesta en base a las necesidades del cliente.

Para evitar que las peticiones de múltiples tipos de clientes lleguen a interferir entre sí cuando se conectan al servidor, existe un mecanismo donde cada servicio o recurso ofrecido por el servidor tiene definido un “número de puerto”. A través de este puerto, el servidor aceptará peticiones de un tipo de cliente de tal manera que el servidor puede tener varios puertos abiertos y cada uno de ellos escuchando posibles solicitudes de clientes sin mezclarse.

Para que una placa Arduino pueda actuar como Servidor, como se comentó en los párrafos anteriores, ésta deberá ofrecer algún recurso compartido y permitir que cualquier dispositivo que tenga la capacidad de conectarse a la placa, pueda hacer uso del recurso a través de uno o varios puertos para poder escuchar y contestar solicitudes. Para lograr esto, en el sketch del código del Arduino, en la zona de declaraciones globales, se deberá agregar una variable u objeto de tipo "EthernetServer" e inicializarlo mediante la función ***".begin()"***. Una vez inicializado el objeto, se pueden realizar diferentes eventos como enviar datos a los clientes mediante la función ***".print()"*** o ***"println()"*** o enviar datos como parámetros con la función ***".write()"***.

3.5 Tecnologías de Transmisión Inalámbrica y Comunicación

3.5.1 Generalidades: Redes Inalámbricas y de Corto Rango

Al hablar de las redes inalámbricas y sus inicios, nos debemos remontar al año 1970 al proyecto ALOHANET desarrollado en la Universidad de Hawái. Esta investigación permitió generar algunos de los pilares más importantes para las actuales tecnologías inalámbricas del siglo 21 y los fundamentos para el estándar IEEE 802.11 establecido en 1997, seguido del establecimiento del desarrollo e interoperabilidad de la *Wi-Fi Alliance* (Rackley, 2011).

No obstante, al hablar de la comunicación de Corto Alcance, en esa misma época de los 70's, el número de aplicaciones era muy limitado y estaba enfocado a tareas muy específicas, entre ellas, un transmisor para apertura de puerta de garaje por mencionar alguna. A pesar de este avance, la tecnología no estaba perfectamente desarrollada y la comunicación era susceptible de interferencias lo que ocasionaba que el garaje se

abriera momento después o de forma aleatoria del que se requería ocasionando problemas de seguridad (Bensky, 2004).

A pesar de que sistemas muy similares se encuentran en uso en la actualidad, la tecnología y los sistemas de radio comunicación han avanzado tremendamente, especialmente en el ámbito de las comunicaciones sociales, la seguridad y las tecnologías de la información.

Hoy en día existen diversas y muy sofisticadas tecnologías de comunicación de Corto alcance que permiten la implementación de más aplicaciones para la automatización y control en la industria y el hogar. Estas tecnologías se caracterizan principalmente por:

- La potencia de Radio Frecuencia (RF) puede ir desde algunos micro watts hasta 100 mili watts.
- Su rango de comunicación pude ir desde algunos centímetros hasta varios cientos de metros.
- Su principal enfoque y utilización se centra en aplicaciones en interior.
- Su transmisión se hace con sistemas de antenas omnidireccionales.
- Son dispositivo simples y de un precio muy accesible.
- Por lo general, su utilización no requiere de alguna licencia.
- Muchos de estos sistemas pueden ser operados con baterías, tanto para su transmisión como para la recepción.

En la Tabla 2, se enlistan algunas de las aplicaciones más comunes para algunos de los sistemas de comunicación de Corto rango en la actualidad:

Aplicación	Frecuencia de operación (MHz)	Características
Sistemas de Seguridad	300-500, 800, 900	Simples y de fácil instalación
Periféricos (mouse, teclados)	300-500, 800	Alta transferencia de información y bajo costo
RFID (Identificación por Radio Frecuencia)	100 KHz – 2.4 GHz	Muy corto rango con actuadores activos o pasivos
WLAN (Redes Inalámbricas de Área Local)	900 MHz, 2.4 GHz, 5 GHz	Alta y continua transferencia de tasa de datos
Micrófonos y audífonos inalámbricos	VHF, UHF	Alta fidelidad y modulación de voz y sonido
Comunicación para móviles (Bluetooth)	2.4 GHz	Transmisión de datos para móviles
Redes de sensores y actuadores (Zig Bee)	2.4 GHz	Tecnología adaptable y de bajo costo

Tabla 2. Principales aplicaciones de los sistemas de Corto Rango (Bensky, 2004).

3.5.2 Wi-Fi

En 1991, NCR/AT&T diseñaron la primer red inalámbrica para cajeros automáticos con el nombre de *WaveLAN* con velocidades de transmisión de hasta 2 Mbps, es a partir de este acontecimiento que se han generados diversos avances y logros en las aplicaciones y servicios de redes inalámbricas. El mayor hito de este desarrollo se produjo en 1999 con la publicación por parte del IEEE (*Institute of Electrical and Electronics Engineers*) del estándar 802.11b con velocidad de transmisión de hasta 11 Mbps. Desde entonces, grandes compañías como Cisco Systems o 3Com, han producido cantidades masivas de equipos para una gran variedad de aplicaciones Wi-Fi (Carballar Falcón, 2010).

Wi-Fi es una tecnología que permite que una gran variedad de equipos informáticos puedan interconectarse sin necesidad de utilizar cables. La principal aplicación del Wi-

Fi en la actualidad es la de permitir que varios ordenadores de casa u oficina puedan compartir el acceso a internet (ADSL o cable). No obstante, esta tecnología permite crear una red entre los distintos equipos para compartir todos sus recursos.

A pesar que el Wi-Fi puede tener diversas configuraciones, lo que le permite trabajar y hacer funcionar a esta tecnología es un dispositivo conocido como Punto de Acceso o *Access Point*. Este equipo suele colocarse cerca del equipo al que se conecta la red de internet que se conoce como Ruteador o *Router ADSL*. En el caso de los equipos más actuales, el *Router* y *Access Point* vienen integrados en uno mismo. Básicamente, el Punto de Acceso proporciona una conexión a internet mediante la conexión Wi-Fi desde cualquier área accesible a él y con las configuraciones y accesos requeridos para la conexión a la red.

Una de las principales ventajas del Wi-Fi es que utiliza el mismo protocolo de internet (TCP/IP). Este protocolo es también utilizado por las redes locales de cable por lo que, interconectar una red Wi-Fi con internet o con una red local cableada es realmente sencillo.

Para que un equipo informático pueda comunicarse con el Punto de Acceso, es necesario que disponga de un equipamiento conocido como Adaptador de Red, el cual es una pequeña unidad que se comunica con el Punto de Acceso vía radiofrecuencia. Esta unidad de radio es la que le ofrece al equipo la posibilidad de comunicarse de forma inalámbrica. Actualmente, la mayoría de los equipos portátiles y móviles ya incorporan de serie un dispositivo adaptador de red.

Uno de los objetivos principales de cualquier tecnología es que sea claramente diferenciable y comprensible. Esto quiere decir que sus usuarios, generalmente no expertos, puedan tener claro cómo utilizarla, qué equipos la soportan y qué se puede

esperar de ella. Para resolver esto, en el caso del Wi-Fi, los principales vendedores de soluciones inalámbricas (3Com, Aironet, Intersil, Lucent Technologies, Nokia y Symbol Technologies) crearon en 1999 una asociación conocida como WECA (*Wireless Ethernet Compatibility Alliance*) con el objetivo de diferenciar y hacer comprensible la tecnología Wi-Fi.

De esta forma, desde abril de 2000, WECA certifica la interoperabilidad de equipos según la norma IEEE 802.11b bajo la marca Wi-Fi (*Wireless Fidelity*). Esto significa que el usuario tiene la garantía de que todos los equipos que tengan el sello de Wi-Fi pueden trabajar juntos sin ningún problema, sin importar el fabricante de cada uno de ellos.

Algunas de las principales ventajas que caracteriza a las redes de Wi-Fi frente a las redes cableadas suelen ser:

- Movilidad. La libertad de movimiento es uno de los principales beneficios y más evidentes de las redes Wi-Fi. Un ordenador o cualquier otro dispositivo puede situarse en cualquier punto dentro del área de cobertura de la red sin tener que depender de si se puede hacer llegar o no un cable de red hasta el sitio.
- Desplazamiento. Con un ordenador o equipo móvil, no sólo se puede acceder a internet o a los recursos compartidos en la red local desde cualquier punto de la oficina u hogar, sino que estos pueden ser desplazados a otro lugar sin perder la comunicación.
- Flexibilidad. Las redes Wi-Fi también permiten colocar un equipo de cómputo de escritorio en cualquier lugar sin tener que realizar el más mínimo cambio de configuración en la red.

- Ahorro de Costos. Diseñar e instalar una red cableada puede llegar a tener un alto impacto, tanto económico como en tiempo y recursos, lo que el Wi-Fi puede minimizar drásticamente sobre todo en áreas algo restringidas o con limitaciones para modificación estructural del edificio.
- Escalabilidad. En una red Wi-Fi, agregar un nuevo equipo a la red es algo muy simple y no se requiere instalar ningún cable o infraestructura adicional para tener acceso a los recursos.

Pero no todos son puntos positivos y a favor para las redes Wi-Fi. También se tienen algunos puntos negativos o en contra de estas redes en comparativa con las cableadas. Quizá el principal inconveniente de la redes Wi-Fi sean las interferencias. La redes inalámbricas basadas en los estándares 802.11b, g y n funcionan utilizando el medio radioeléctrico en la banda de 2.4 GHz. Esta banda de frecuencias no requiere licencias de operación por lo que muchos equipos comerciales como teléfonos inalámbricos, microondas, entre otros, utilizan esta misma banda de frecuencias. Este hecho hace que no se tenga la garantía de que el entorno radioeléctrico esté completamente limpio para que la red funcione con un mejor rendimiento. Cuanto mayores sean las interferencias en el entorno, menor será el rendimiento de la red y esto se reflejará en una reducción de velocidad y transferencia de información. En el peor de los casos, la comunicación puede resultar imposible o nula (Carballar Falcón, 2010).

A las redes Wi-Fi también se les ha catalogado como redes menos seguras y vulnerables en comparación con las redes cableadas. Aunque esto fue así en el pasado, actualmente se le han incorporado sistemas de cifrado que permiten alcanzar un grado de seguridad similar al de una red cableada.

Desde que la Domótica ha sido parte de las nuevas tecnologías para el hogar, cada nueva tecnología o aplicación que surge en el mercado, inmediatamente busca la

forma para introducirse dentro de este mercado. No obstante, esto siempre se ha visto frenado por toda una serie de limitaciones como:

- La necesidad de nuevo cableado por todo el hogar.
- Complicaciones en la instalación.
- Alto costo de implementación.
- Baja velocidad de transmisión.
- Capacidad limitada de crecimiento.
- Compatibilidad.

Las redes Wi-Fi pueden eliminar prácticamente todas las barreras anteriores. De hecho, el Wi-Fi puede proporcionar las herramientas y la tecnología adecuada para llevar la automatización en el hogar a otro nivel y hablar de lo que actualmente se están llamando los Hogares Inteligentes. Estos hogares inteligentes, además de contar con diversas tecnologías de automatización y control, integran tres tipos de redes interconectadas:

- Red domótica o de automatización que interconecta los dispositivos eléctricos del hogar, calefacción, riego, sistemas de seguridad, etc.
- Red de entretenimiento para interconectar el Televisor, video juegos, concentrados de medios, etc.
- Red de datos para la interconexión de equipos de cómputo y dispositivos móviles como tablets o teléfonos inteligentes.

Son muchos los sistemas y dispositivos que están incorporando la tecnología Wi-Fi. En la mayoría de los casos, lo único que aporta esta tecnología es la eliminación de cables pero, no faltan los ejemplos donde esa comunicación inalámbrica es el verdadero valor del equipo.

3.5.3 Bluetooth

La tecnología de comunicación Bluetooth fue desarrollada en 1994 por la empresa sueca Ericsson con el objetivo de conseguir un sistema de comunicación de los teléfonos móviles con sus accesorios como auriculares, ordenadores, etc. (Carballar Falcón, 2010). Pero no fue hasta 4 años después cuando el SIG (*Bluetooth Special Interest Group*) integrado por Ericsson, IBM, Intel, Nokia y Toshiba iniciara el concepto para incluir la integración y conexiones entre PCs y otros dispositivos (Rackley, 2011).

El nombre de Bluetooth, que significa "diente azul" en español, procede del apoyo que tenía el rey Harald Blaatlund II, un legendario guerrero danés del siglo X. Baatlund significa *blue tooth* o diente azul. Este rey pasó a la historia por unificar las tribus noruegas, suecas y danesas (Carballar Falcón, 2010).

Las comunicaciones de Bluetooth se desarrollan mediante el modelo maestro / esclavo. Un terminar maestro pude comunicarse hasta con 256 esclavos, aunque sólo siete de estas comunicaciones pueden ser simultaneas.

Los protocolos que se utilizan en una comunicación Bluetooth son similares a los que se emplean con tecnología de infrarrojos, por lo que no hizo falta desarrollar otros nuevos pero, mientras en una comunicación por infrarrojos se requiere un enlace visual entre dispositivos, con Bluetooth no es necesario. Algunas de las especificaciones principales de la tecnología Bluetooth son las siguientes:

- Banda de frecuencia a 2.4 GHz (banda ISM).
- Soporta voz y datos de manera simultánea.
- Potencia del transmisor entre 1 y 100 mili watts, típica de 2.5 mili watts.
- Velocidad de datos de hasta 721 Kbps por pico net (versión 1.1).
- Rango de transmisión hasta de 10 metros.

- Bluetooth minimiza la interferencia potencial al emplear saltos rápidos en frecuencia de 1600 veces por segundo.
- Dado que cada enlace está codificado y protegido contra interferencia y pérdida de enlace, Bluetooth puede considerarse como una red inalámbrica de corto alcance muy segura (Huidobro Moya, Radiocomunicaciones, 2011).

Hoy en día, la tecnología Bluetooth ha tenido tanto alcance que ya se encuentra presente en el 100% de teléfonos móviles, tablets y laptops. Además, es muy utilizado en dispositivos como audífonos, altavoces, cámaras, consolas de videojuegos y periféricos como teclados, mousse, impresoras y escáneres (Gupta, 2013).

A través de los años, la tecnología Bluetooth ha ido evolucionando y se han desarrollado diferentes versiones más capaces, con mejor tasa de transferencia de datos y con menor consumo de energía. La primer versión de Bluetooth fue la 1.1 la cual contaba con una tasa de transferencia de datos de hasta 1 Mbps. Años después en el 2004 aparecería el Bluetooth 2.0 el cual tuvo una mejora en el incremento de la tasa de transferencia de datos en comparación con la versión 1.1 que fue de 1 a 2 o 3 Mbps (Rackley, 2011). La versión Bluetooth 3.0 o de alta velocidad aparecería en el 2009 permitiendo tasas de transferencia hasta de 24 Mbps.

Actualmente, Bluetooth se encuentra en la versión 4.0 (liberada en Junio de 2010) y se le ha incorporado lo que se conoce como *Bluetooth Low Energy (LE)* o de muy bajo consumo de energía. Como su nombre lo indica, esta nueva tecnología tiene como característica primordial brindar un "ultra" bajo consumo de energía en los dispositivos en los que se incorpora.

Bluetooth LE actualmente se está implementando en aplicaciones donde es muy complicado realizar recargas de baterías y es requerido una larga duración de ellas.

Esto es posible debido a que los dispositivos que utilizan esta tecnología tienen la capacidad de consumir muy poca potencia y por tal motivo, pueden operar durante meses o incluso años con una batería sin necesidad de una recarga.

Algunas de las aplicaciones en donde se está implementando la tecnología Bluetooth LE son las siguientes:

- Internet de las cosas (*IoT*).
- Dispositivos para el cuidado de la salud como Termómetros, medidores de presión sanguínea y de glucosa.
- Dispositivos para atletas y deportistas como relojes inteligentes, medidores de ritmo cardíaco, GPS.
- Automatización en el hogar.
- Controles remoto y entretenimiento en el hogar.
- Pagos con móviles.
- Sensores para aplicaciones automotrices.
- Seguridad.

3.5.4 ZigBee

ZigBee es una tecnología de baja tasa de transferencia de datos, muy bajo consumo de potencia, bajo costo y es un protocolo inalámbrico de red enfocado principalmente en el uso de aplicaciones de automatización y control a distancia. El comité de la IEEE 802.15.4 y la Alianza ZigBee trabajaron juntos y desarrollaron la tecnología comercialmente conocida como ZigBee, lanzada durante el año 2005. La expectativa de esta tecnología era reducir el costo y proveer a los dispositivos que la implementaran la capacidad de bajo consumo para comunicaciones y prolongar la vida

de la baterías de unos meses hasta varios años en aplicaciones donde no se requiera alta tasa de transferencia de datos como la tecnología Bluetooth (Garg, 2010).

ZigBee puede operar en tres diferentes anchos de banda: 2.4 GHz como estándar global, 915 MHz exclusivo para América y 868 MHz para Europa. Las tasas de transferencia de datos para estos diferentes anchos de banda se encuentran entre 250 kbps para 2.4 GHz, 40 kbps para 915 MHz y 20 kbps para los 868 MHz. Como dato adicional, cuando un nodo de una red Zig Bee es apagado, este puede encenderse y recibir un paquete de información en tan solo 15 milisegundos.

Un sistema o red ZigBee está compuesto por varios elementos. El más básico y simple puede ser un Dispositivo el cual, a su vez, puede ser un dispositivo tipo FFD (*Full Function Device*) o tipo RFD (*Reduced Function Device*). Para poder considerar una red ZigBee, se debe contar por lo menos con un FFD operando como Coordinador de la Red de Area Personal o *Personal Area Network, PAN*.

Un FFD puede operar en tres modos dentro de una red: como Coordinador de la PAN, como simple Coordinador o *Router* y como Dispositivo Final o *End Device*. Un RFD es usado por lo regular en aplicaciones muy simples donde no es requerido el envío de información muy compleja. En general, un FFD pude enlazar comunicación con un RFD o con otros FFD, mientras que un RFD, sólo puede enlazar comunicación con el FFD.

La tecnología ZigBee soporta hasta tres diferentes topologías de red: tipo Estrella, Punto a Punto y tipo Árbol o *Cluster*. En la topología tipo Estrella, la comunicación es establecida entre los Dispositivos Finales y el Coordinador de la PAN. Por lo general, el Coordinador de la PAN suele estar energizado por una fuente de energía principal y siempre disponible, mientras que los Dispositivos Finales suelen usar alimentación a

baterías. Esta topología suele aplicarse principalmente en implementaciones de Automatización en el hogar, periféricos de PC, juguetes, entre otros.

En la topología Punto a Punto, en comparación con la topología tipo Estrella, cualquier dispositivo en la red puede comunicarse con cualquier otro dispositivo, siempre y cuando estos se encuentren dentro de rango. Este tipo de tecnología puede funcionar como un Ad Hoc, puede auto-organizarse y auto-configurarse. Algunas aplicaciones típicas para éste tipo de topologías pueden incluir Control y Monitoreo en la Industria, redes inalámbricas de sensores así como control de tráfico de inventarios.

La topología tipo Árbol o *Cluster* es una configuración especial de la topología Punto a Punto en la cual la mayoría de los dispositivos son de tipo FFD y donde un RFD puede conectarse a alguna rama o *Cluster* de la red siendo como una hoja al final de una rama del Árbol. Cualquier FFD puede funcionar como Coordinador y proveer servicios de sincronización a otros dispositivos Coordinadores. De cualquier manera, aunque se tienen diversos Coordinadores, sólo uno de ellos actúa como el Coordinador de la PAN (Garg, 2010). En figura 3, se pueden apreciar las diferentes topologías ZigBee.

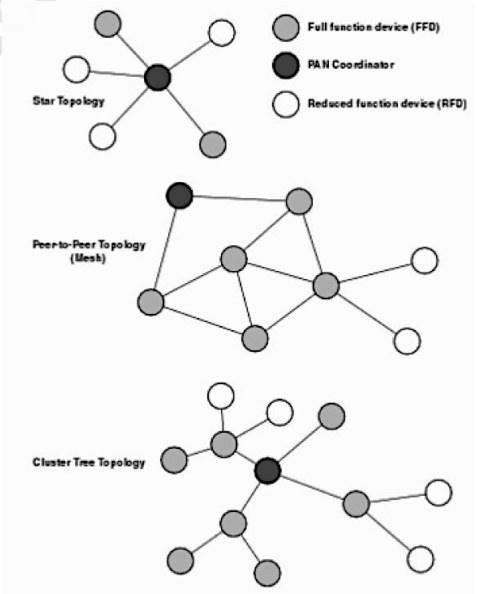


Figura 3. Topologías de Redes ZigBee (Garg, 2010).

La tecnología ZigBee ha sido implementada e utilizada en diversos tipos de aplicaciones en la actualidad incluyendo detección y localización en lugares de desastres naturales; monitoreo de sensores automotrices como sensores de presión de aire en los neumáticos; etiquetas inteligentes y sistemas de monitoreo de precisión en la agricultura como sensores de humedad para el suelo, pesticidas, herbicidas y sensores de niveles de pH. Pero, una de las más grandes aplicaciones para la IEEE 802.14.4 ha sido las redes y automatización en el hogar donde se puede mencionar control de aires acondicionados y ventiladores, sistemas de seguridad e iluminación; así como el control de dispositivos como persianas, ventanas, seguros de puertas; dispositivos para el monitoreo de la salud; así como juguetes y otras aplicaciones (Garg, 2010).

3.6 Desarrollo de Software para móviles: Android

3.6.1 ¿Qué es Android?

Hablar de Android es hablar de una nueva generación de sistema operativo para móviles diseñado para soportar aplicaciones desarrolladas para dispositivos con gran capacidad de hardware. Plataformas como Microsoft Windows Phone o IPhone de Apple, también proveen un entorno simplificado para el desarrollo de aplicaciones móviles pero, a diferencia de Android, estos priorizan la operación de las aplicaciones propietarias del sistema operativo. Es decir, ellos le dan prioridad a la ejecución de las aplicaciones propietarias del sistema operativo de las creadas por algún tercero, así como también se prioriza a las comunicaciones y a el acceso a datos del equipo (Meier, 2012).

Android brinda nuevas posibilidades para las aplicaciones móviles ofreciendo un ambiente de desarrollo abierto basado en una plataforma de Código Abierto con kernel de Linux. El acceso al hardware del dispositivo a través de las aplicaciones es posible

gracias a diversas librerías o *APIs*, así como la interacción de estos recursos a través del entorno de las aplicaciones es totalmente soportado.

En Android, todas las aplicaciones tienen la misma prioridad. Ya sean aplicaciones de terceros o propietarias del sistema operativo, todas son escritas con las mismas APIs y ejecutadas al mismo tiempo. Además, el usuario puede reemplazar la funcionalidad de alguna aplicación propietaria del sistema operativa con alguna otra de un tercero e incluso personalizar el dispositivo en base a sus criterios y necesidades.

A pesar que hoy en día el desarrollo para móviles en Android es un campo algo maduro, siguen existiendo diferentes opiniones sobre qué es exactamente Android. Para comenzar, mencionaremos lo que NO es Android:

- Android NO es una implementación de Java ME (*Mobile Edition*).
- NO es parte del LiPS (*Linux Phone Standards Forum*) o del OMA (*Open Mobile Alliance*).
- Android NO es sólo una capa de aplicaciones o librerías.
- Android NO es unos auriculares para móvil.
- NO es una respuesta de Google ante el iPhone de Apple.

Básicamente, se podría resumir que Android es una Plataforma abierta de desarrollo para móviles. Andy Rubin de Google lo describe de la siguiente manera: “La primer plataforma abierta confiable y comprensible para móviles. Esta incluye un sistema operativo, interfaces de usuario y aplicaciones – todo el software para ejecutar un teléfono móvil pero sin tener las restricciones del sistema propietario que es lo que ha obstaculizado la innovación móvil” (Meier, 2012, pág. 4).

Recientemente, Android ha expandido sus límites de sólo estar enfocado en el

desarrollo de plataformas para móviles y ha considerado un mercado que está en gran crecimiento como lo son las tablets, televisores, relojes inteligentes y muy recientemente, el de sistemas de interactivos para vehículos.

Una aplicación Android normalmente es escrita en Java como lenguaje de programación pero, es ejecutada mediante una Máquina Virtual (*VM*) llamada *Dalvik*, a diferencia de una aplicación Java que implementa una Máquina Vitual de Java (*Java VM*).

Cada aplicación Android se ejecuta en un proceso separado con la ejecución de su propia instancia de Dalvik, dejando toda la administración de los recursos como la memoria y la administración de los procesos al sistema operativo (Android), el cual, detiene o destruye procesos para administrar los recursos en base a las necesidades del sistema.

3.6.2 Por qué desarrollar en Android

Android representa un cambio, un entorno de trabajo para móviles basado en los requerimientos de los modernos dispositivos y diseñado por desarrolladores, para desarrolladores (Meier, 2012).

Con una simple, potente y de software abierto Plataforma para Desarrollo o *SDK* por sus siglas en inglés (*Software Development Kit*) la cual no necesita licencia, cuenta con excelente documentación y está respaldada por una gran comunidad de desarrolladores, Android representa una gran oportunidad de crear software para cambiar la forma en cómo y por qué la gente usa sus teléfonos móviles.

Las barreras para entrar al mundo del desarrollo para móviles de Android son mínimas:

- No se requiere certificación alguna para convertirse en un desarrollador Android.
- Google Play ofrece herramientas gratis para la comercialización y distribución de las aplicaciones.
- No existe un proceso de aprobación para la distribución de las aplicaciones.
- Los desarrolladores tienen el control total sobre sus marcas.

Desde el punto de vista comercial, más de 850,000 dispositivos móviles con Android como sistema operativo son activados diariamente y varios estudios avalan que muchos de los nuevos dispositivos que saldrán a la venta tendrán Android como sistema operativo. Otro dato importante es que a partir de Marzo de 2012, Google Play (la tienda de aplicaciones de Android) expandió el soporte de aplicaciones Android a más de 131 países, teniendo una tasa en incremento de más de 1 billón de descargas de aplicaciones al mes.

3.6.3 Historia de Android

El sistema operativo Android fue creado originalmente por Andy Rubin (co-fundador de Android Inc.) como un sistema operativo propio para dispositivos móviles a principios del 2003.

En 2005, Google adquirió Android Inc. y colocó como Director de Plataformas Móviles para Google a Andy Rubin. Muchos tienen la creencia que esta adquisición del Sistema Operativo Android de Google fue una respuesta hacia la aparición del iPhone de Apple durante ese mismo tiempo. De cualquier manera, había muchos más competidores en el mercado con sus propios sistemas operativos como el RIM de Blackberry, Symbian de Nokia o el Windows Mobile de Microsoft pero, Google consideró que la adquisición del talento y la propiedad intelectual del sistema operativo Android era un negocio

razonable e inteligente el cual permitiría a Google adentrarse en un nuevo mercado emergente, el cual era conocido como el Internet 2.0 (Wallace, 2014).

El Internet 2.0 o el Internet Móvil permitirían a los usuarios de diversos productos electrónicos, acceder a contenido en la red mediante diversos productos móviles. Esto sobre todo, incluiría tablets, teléfonos inteligentes o *smartphones*, *phablets* (un híbrido entre tablet y smartphone), consolas de videojuegos, relojes inteligentes o *smartwatches*, gafas inteligentes o *smartglasses*, robots personales y lectores de libros electrónicos.

Actualmente, plataformas de dispositivos no móviles o fuera de ámbito de las tecnologías móviles, están adoptando Android como sistema operativo. Tal es el caso de televisores inteligentes o *Smart TVs*, centros de entretenimiento, consolas y estéreos de automóviles así como otros sistemas digitales de entretenimiento.

A través del tiempo y desde su nacimiento, Android ha ido madurando y ha evolucionado hasta convertirse en un verdadero y muy confiable sistema operativo embebido de código abierto. Un sistema operativo que iniciaría con su Versión 1.0 hace unos cuantos años y, después de haber sido adquirido por Google, liberaría múltiples versiones estables hasta llegar a la más actual y estable versión 5.1 conocida como *Lollipop*. Recientemente se acaba de liberar la versión 6.0 o *Marshmallow* que está incluida en los más actuales dispositivos móviles del mercado.

En la figura 4, se puede apreciar la actual distribución de las diferentes versiones y plataformas del sistema operativo Android, así como la proporción de dispositivo en los que se encuentran instalados:

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.3%
4.1.x	Jelly Bean	16	8.1%
4.2.x		17	11.0%
4.3		18	3.2%
4.4	KitKat	19	34.3%
5.0	Lollipop	21	16.9%
5.1		22	19.2%
6.0	Marshmallow	23	2.3%

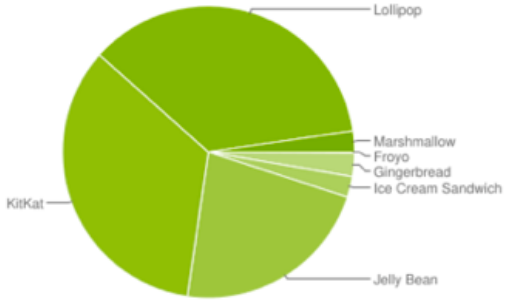


Figura 4. Distribuciones y versiones de Android (Google, 2016)

3.6.4 Aplicaciones y Ciclo de Vida en Android

3.6.4.1 Características de una Aplicación Android

Una aplicación en Android consiste básicamente en un conjunto de componentes agrupados ligados en un *manifest* de la aplicación donde se describe cada componente y cómo deben interactuar entre sí. El *manifest* es también utilizado para especificar la fuente de datos o *metadata* de la aplicación, así como los requerimientos de hardware, librerías externas y permisos requeridos (Meier, 2012).

A continuación se enlistan algunos de los elementos o componentes principales que forman parte de una aplicación en Android:

- *Activities* o Actividades. Se relaciona principalmente a la capa visual de la aplicación. Las Interfaces de Usuario o *UI* suelen estar construidas alrededor de una o más extensiones de la clase Activity. Un *Activity* utiliza *Fragments* y *Views*

para distribuir y mostrar información, así como para responder a las acciones de los usuarios. Comparado con el desarrollo de software para equipos de cómputo de escritorio, un *Activity* equivaldría a un *Form*.

- *Services* o Servicios. También llamados los trabajadores invisibles de las aplicaciones, los *Services* no requieren de una interfaz de usuario para operar y su función principal es actualizar las fuentes de datos que alimentan a un *Activity*, lanzar notificaciones y servicios de *broadcasting*, o consulta de recursos de forma paralela a la ejecución de la aplicación.
- *Content Providers*. Un *Content Provider* se encarga de administrar, compartir y almacenar durante un cierto momento, datos de la aplicación y, por lo general, es el encargado de interactuar con las bases de datos de SQL. También es usado como elemento para compartir información entre aplicaciones.
- *Intents*. Los *Intents* son una de las herramientas más utilizadas en Android. Se pueden usar para lanzar o detener *Activities* y *Services*, así como para difundir mensajes o *broadcasting* entre *Activities, Services* o los llamados *Broadcast Receivers*.
- *Broadcast Receivers*. Son los componentes que escuchan a los *Intents* en base al criterio establecido y que coincida con ellos.
- *Widgets*. Son componentes visuales de la aplicación que, por lo general, son añadidos en la pantalla de inicio del dispositivo con Android.
- *Notifications*. Son alertas lanzadas hacia el usuario para informar sobre algún estado o evento especial de alguna aplicación que el usuario no esté usando en ese momento.

Como ya se mencionó anteriormente, un elemento fundamental dentro de una aplicación Android es el archivo *manifest*. Este archivo (AndroidManifest.xml) se almacena en la carpeta raíz del proyecto y define la estructura, los componentes y los

requerimientos de la aplicación. El siguiente código muestra un pequeño ejemplo de un XML de un archivo *manifest* (Meier, 2012, pág. 56):

```
<manifest xmlns:Android="http://schemas.android.com/apk/res/android"
        package= "com.paad.myapp"
        android:versionCode="1"
        android:versionName="0.9 Beta"
        android:installLocation="preferExternal">
        [.. manifest nodes ..]
</manifest>
```

La etiqueta o nodo principal *manifest* dentro del nodo principal del XML, puede contener diversos nodos que definen los componentes, los parámetros de seguridad, las clases de prueba y los requerimientos de la aplicación. El siguiente listado describe algunos de los principales sub-nodos que pueden incluirse dentro del nodo principal *manifest* (Meier, 2012):

- uses-sdk – Este nodo permite definir la versión mínima o máxima de *SDK* que debe tener instalado el dispositivo para que la aplicación pueda funcionar de manera correcta.
- uses-configuration – En este nodo se especifica la configuración y tipos de entrada que la aplicación requiere para su utilización, es decir, si se requiere teclado QWERTY, pantalla táctil, *trackball*, entro otras.
- Uses-features – Este nodo permite definir qué tipo de hardware será requerido por la aplicación para su funcionamiento, es decir, elementos del dispositivo que permitan su correcta ejecución como audio, Bluetooth, cámara, NFC, micrófono, sensores, pantalla táctil, Wi-Fi, entre otros.
- supports-screens – Ya que Android se ha vuelto un sistema operativo para diferentes dispositivos y plataformas, la variedad de pantallas es muy diversa.

Desde las primeras pantallas de 3.2'' hasta las más actuales de tablets de 10.1'' o incluso de televisores HD de 42''. Este nodo lo que permite es definir el mejor criterio de pantalla en el que la aplicación ha sido diseñada y probada para su mejor desempeño.

- uses-permission – Este nodo, como parte de los criterios y modelo de seguridad, declara al usuario los permisos que la aplicación requiere antes de ser instalada. Estos permisos pueden incluir el uso de las APIs que podría ocasionar algún costo o algún riesgo de seguridad como realizar alguna llamada, enviar o recibir un mensaje de texto o usar el servicio de ubicación.
- application – El archivo *manifest* solo pude contener un nodo de este tipo. Aquí es donde se definen algunos de los atributos principales de la aplicación como el título, el ícono o el tema.

3.6.4.2 Ciclo de vida de una aplicación Android

A diferencia de otras plataformas tradicionales de aplicaciones, las aplicaciones Android tienen un limitado control sobre su propio ciclo de vida. En lugar de esto, los componentes de las aplicaciones deben estar escuchando los cambios de estado de la aplicación misma y reaccionar de acuerdo a lo que se requiere, tomando especial atención al hecho de terminar el proceso (Meier, 2012).

Android, de una forma muy agresiva, administra sus recursos haciendo lo que sea necesario para garantizar al usuario un desempeño suave y estable del sistema. Esto llevado a la práctica puede llegar a implicar el hecho de terminar procesos de manera súbita para liberar recursos para aplicaciones que tengan mayor prioridad de ejecución.

El orden en que los procesos son terminados en Android, depende completamente de la prioridad que se tiene en las aplicaciones en proceso. La prioridad de una aplicación

es determinada por el componente que más alta prioridad tiene al momento de su ejecución. Si dos aplicaciones tienen la misma prioridad, aquella que ha tenido la prioridad durante más tiempo es la que será terminada. La interoperabilidad de los procesos también afecta el proceso de priorización de las aplicaciones ya que, si se tienen una aplicación que depende de un Servicio o *Content Provider* proporcionado por una segunda aplicación, esta última tendrá también una alta prioridad como la primera a la que provee la información.

En la figura 5 se muestran los diferentes estados que puede tener una aplicación dependiendo de la prioridad que se le va asignado:

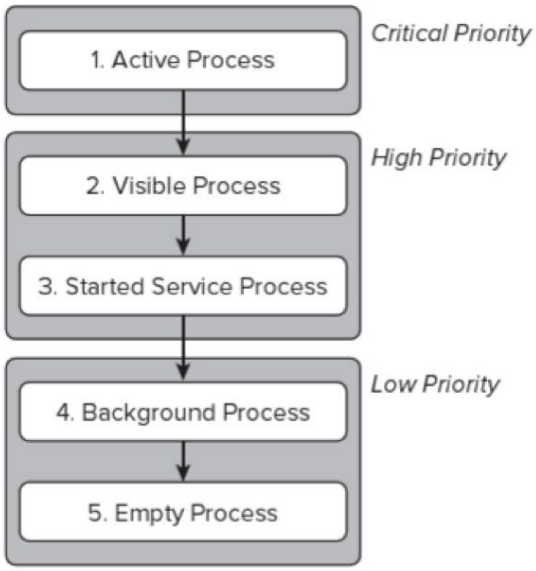


Figura 5. Criterios de prioridad de una aplicación (Meier, 2012, pág. 82).

Un buen entendimiento del Ciclo de vida de un *Activity* es vital para asegurar que la aplicación pueda brindar al usuario una buena experiencia y, a su vez, mantenga una buena relación de la utilización de recursos del sistema. El estado de cada *Activity* está determinado por la posición de ésta en el *Activity Stack* que no es más que un listado que va desde la última aplicación ejecutada hasta la más reciente. Cuando una aplicación es arrancada y se encuentra en uso, ésta se ubica en la primera posición de

la lista. Si el usuario retorna hacia atrás con el botón de Retorno o si la aplicación en primer plano es cerrada, la siguiente *Activity* en la lista es movida a la primera posición y se vuelve activa. La figura 6 muestra gráficamente este comportamiento.

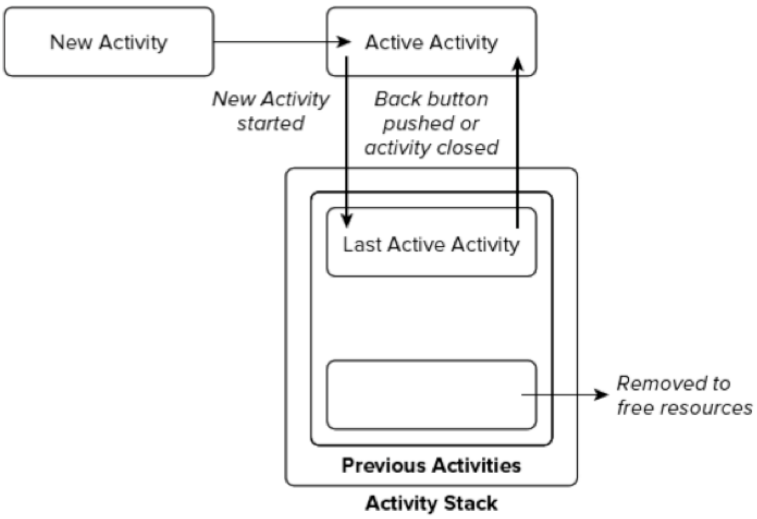


Figura 6. Comportamiento del *Activity Stack* (Meier, 2012, pág. 88).

Cuando una *Activity* es movida de posición dentro del *Activity Stack*, ésta suele cambiar de estado y puede pasar a través de las siguientes 4 transiciones:

- Activa o *Active*. Se refiere a la *Activity* que está en la primera posición del *Activity Stack* y que se encuentra en primer plano y está siendo utilizada por el usuario. Cuando otra aplicación se ponga en estado activo, esta será puesta en Pausa.
- Pausada o *Paused*. En algunos casos, la *Activity* estará visible pero no tendrá el foco, en esta condición, estará Pausada. Este estado es definido cuando una actividad transparente o que no ocupe toda la pantalla es activa enfrente de la *Activity* activa. Cuando una aplicación se vuelve totalmente opaca o no visible por alguna otra, ésta se Detiene.

- Detenida o *Stopped.* Cuando una *Activity* no es visible, se encuentra Detenida. La *Activity* se mantendrá en memoria del sistema conservando su estado e información pero, ahora se vuelve candidato para ser terminada cuando el sistema requiera recursos. Cuando una *Activity* se encuentra en este estado, es importante que se conserve la información y el estado de la *UI*, así como mantener detenida cualquier operación no crítica. Una vez que esta *Activity* se ha cerrado, se vuelve Inactiva.
- Inactiva o *Inactive.* Después de que una *Activity* ha sido terminada y antes de que ésta sea ejecutada nuevamente, se encuentra Inactiva. Al encontrarse en este estado, la *Activity* se ha removido completamente del *Activity Stack* y necesita de ser restablecida antes de ser mostrada y usada de nueva cuenta.

La transición de estos estados es algo que no se puede determinar y dependen completamente del administrador de memoria de Android. Este administrador comenzará cerrando todas aquellas aplicaciones que contienen un *Activity* en modo Inactivo seguido de las que se encuentran Detenidas. Sólo en casos muy extremos, se cerraran aquellas que estén Pausadas.

Para asegurar que las *Activities* puedan reaccionar a estos cambios de estado, Android provee una serie de métodos y eventos que son lanzados cuando este tipo de transiciones y cambios de estados son alcanzadas. La figura 7 muestra de forma esquemática el ciclo de vida y las transiciones de una *Activity* descritas anteriormente:

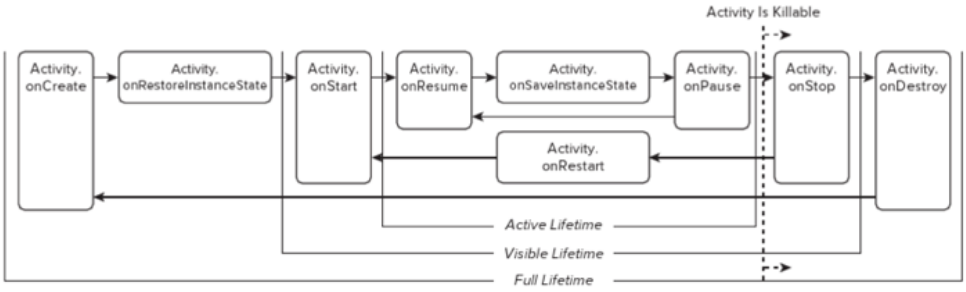


Figura 7. Esquema de ciclos de vida de una *Activity* (Meier, 2012, pág. 89).

3.6.5 Fundamentos para diseño de Interfaces de Usuario

El Diseño de Interfaces de Usuario (*User Interface, UI*), la experiencia del usuario (*User eXperience, UX*), la interacción de los usuarios (Human *Computer Interaction, HCI*) y la usabilidad son algunos de los temas más importantes que se deben considerar al momento de diseñar una Interfaz de Usuario (Meier, 2012).

En Android se manejan algunas terminologías especialmente enfocadas al diseño de interfaces de usuario que comprenden el manejo de diversos componentes como:

- *Views* o Vistas. Las Vistas son la clase base de todos los elementos y componentes gráficos para el diseño de las interfaces visuales. Todos los controles de *UI*, incluyendo las clases de esquemas o *layouts*, son derivados de la clase *View*.
- *View Group*. Son extensiones de la clase *View* y son elementos que pueden anidar o contener diferentes elementos de la clase *View*.
- *Fragments*. Éstos fueron agregados en la versión de Android 3.0 y son elementos que pueden contener o encapsular parte de la *UI* y funcionan como elementos reusables, algo similar a los *UI View Controller* en el desarrollo de aplicaciones para iPhone.
- *Activities*. Como ya se describió anteriormente, son las ventanas o marcos donde se despliegan los elementos visuales. Una *Activity* es el equivalente a una *Form* tradicional en el desarrollo para aplicaciones de escritorio en Windows. Para desplegar un *UI,* se asigna un *View* (usualmente un *layout* o *Fragment*) a una *Activity*.

Como ya se mencionaba anteriormente, todos los componentes visuales en Android descienden de la clase *View* y se puede hablar de ellos haciendo una referencia general como Vistas o *Views*.

Una *Activity* comienza con un una pantalla de inicio donde se colocan todos los elementos de *UI*. Para realizar esto, se llama al método setContentView pasándole como parámetro la *View* o el recurso de *layout* que se quiera desplegar. Se suele sobrescribir el método onCreate de una *Activity* en tiempo de ejecución para asignar un elemento de *UI* como se muestra en el código siguiente:

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
}
```

El uso de recursos de *layout* o *UI*, permite obtener una capa de presentación totalmente independiente de la capa lógica de la aplicación, permitiendo obtener flexibilidad y fácil mantenimiento sin grandes cambios de código.

Otro de los elementos importantes mencionados anteriormente son los *layouts*. Los *Layout Managers* o simplemente *layouts* son extensiones de la clase ViewGroup y son usados para anidar, distribuir y separar de una mejor manera los elementos *UI* permitiendo la creación de capas complejas en base a la necesidad requerida.

Algunos de los *layout* más usados comúnmente en Android se incluyen en el siguiente listado y suelen ser:

- *FrameLayout*
- *LinearLayout*
- *RelativeLayout*
- *GridLayout*

Estos *layouts* están diseñados para escalar y optimizar los elementos mostrados en la pantalla de los dispositivos donde se hospeda la aplicación eliminando el uso del diseño de elementos fijos con posiciones absolutas o determinadas posiciones de pixeles. Esto hace a los *layouts* particularmente usuales cuando se diseñan aplicaciones para diferentes dispositivos en Android.

Una de las formas preferidas para definir una estructura de una *layout* es mediante el uso de recursos XML externos. Estos XML deben contener un elemento raíz único y tantos elementos de *layouts* y *Views* anidados como sea necesario para construir la *UI*. En el siguiente código, se describe un pequeño ejemplo donde posicionan dos controles, un TextView y un EditText, dentro de un LinearLayout vertical.

```
<?xml version=”1.0” encoding=”utf-8”?>
<LinearLayoutxmlns:android=”http://schemas.android.com/apk/res/android”
        android:orientation=”vertical”
        android:layout_width=”match_parent”
        android:layout_height=”match_parent”
        <TextView
                android:layout_width=”match_parent”
                android:layout_height=”wrap_content”
                android:text=”Enter Text Below”
        />
        <EditText
                android:layout_width=”match_parent”
                android:layout_height=”wrap_content”
                android:text=”Text Goes Here!”
        />
</LinearLayout>
```

Para cada uno de los elementos del *layout*, se usan las constantes match_parent y wrap_content en lugar de un valor exacto de altura y ancho en pixeles. Estas contantes,

combinadas con algún *layout* que escala (como un *Linear Layout, Relative Layout* o un *Grid Layout*) ofrecen la más simple y poderosa técnica para asegurar que un *layout* sea siempre del tamaño de la pantalla independientemente de la resolución de esta.

3.6.6 Bases de Datos en Android

Android proporciona herramientas y estructuras para el manejo y la persistencia de datos mediante la implementación de bases de datos de SQLite y *Content Providers*.

Las bases de datos SQLite pueden ser usadas para almacenar los datos de una aplicación de una forma estructurada y organizada. Android ofrece una completa librería para el uso de bases de datos SQLite. Cada aplicación puede crear su propia base de datos de la cual tendrá completo control. Una vez creada la base de datos, los *Content Providers* ofrecen una forma genérica y una interfaz bien definida para el manejo de la información (Meier, 2012).

3.6.6.1 SQLite

SQLite es un Sistema para el Manejo de Bases de Datos Relacionales o *RDBMS*. Algunas de sus principales características son:

- Es de Código Abierto
- Cumple con estándares de bases de datos
- Muy ligero
- De un solo nivel

SQLite ha sido implementado como una librería compacta de C que se incluye como parte de las librerías de Android lo que permite que, las base de datos en cada

aplicación, sean parte de la misma evitando dependencias externas y simplificando el proceso de transacciones y sincronización de la información.

Existen diferentes elementos que son útiles para el manejo de las transacciones y procesos en las bases de datos SQLite. Uno de estos son los *Content Values*, los cuales son usados para insertar nuevas filas en las tablas de una base de datos. Otro elemento importante son los Cursores los cuales, son objetos que retornan una consulta o *query* a una base de datos.

Al trabajar con bases de datos, es una buena práctica encapsularla y sólo exponer los métodos públicos y contantes que se requieran para la interacción con ella, diseñando lo que se conoce como una clase *SQL Open Helper*.

La clase abstracta SQLiteOpenHelper es usada para implementar los patrones y mejores prácticas para la creación, apertura y actualización de una base de datos. Implementando un *SQLite Open Helper*, se puede ocultar la lógica usada para determinar si una base de datos debe ser creada o actualizada antes de ser abierta así como asegurar que las operaciones sean completadas eficientemente.

Es importante remarcar que las operaciones de bases de datos, especialmente la apertura y la creación de estas, pueden consumir recursos y tiempo para la aplicación. Para asegurar que esto no tenga impacto directamente sobre la experiencia del usuario, se recomienda realizar todas las operaciones a la base de datos de forma asíncrona.

Para acceder a una base de datos usando la clase *SQLite Open Helper*, se deberán mandar llamar los métodos getWritableDatabase o getRedeableDatabase con la finalidad de abrir y obtener una instancia, ya se con acceso a escritura en el caso del primer

método o con sólo acceso a lectura, para el caso del segundo, de la base de datos con la que se trabajará en la aplicación. Al mandar llamar a estos métodos, si en ese momento la base de datos aún no existiera, el *Helper* ejecutaría el método onCreate para generar la instancia a la base de datos. En el caso de que la versión de la base de datos haya cambiado, se ejecutará el método onUpgrade para obtener la versión más actualizada de esta.

Es importante mencionar que, una vez que una base de datos se haya abierto correctamente, se crea una caché de esta y, a pesar que se pudiera hacer uso de esta caché, es muy recomendable siempre volver a instanciar y usar los métodos antes mencionados siempre que se vaya a trabajar con la instancia de la base de datos.

Algunas consideraciones a tomar en cuanta al momento de diseñar una base de datos en Android son las siguientes:

- Archivos como imágenes o audio, usualmente no son almacenados dentro de las tablas de la base de datos. En lugar de esto, se deberá almacenar un string de la ruta donde se encuentra el archivo como referencia
- Es muy recomendable que todas las tablas incluyan un campo como llave auto-incrementable para cada registro ya que, para utilizar *Content Providers* un campo de ID es requerido para cada registro.

Al realizar una consulta o *query* a una base de datos, los resultados se retornan en forma de Cursores. Esto permite a Android administrar y manejar los recursos de una forma más eficiente, tomando y usando los valores en las filas y columnas en base a lo que se requiera. Para ejecutar un *query* en el objeto de la base de datos, se requiere ejecutar el método query mandándole los siguientes parámetros:

- Un booleano como opcional donde se indique si el resultado contendrá valores únicos
- El nombre de la tabla a consultar
- Un arreglo de strings como listado de las columnas que se incluirán en la consulta resultante
- Un condicional where para definir las filas a regresar como filtro de la consulta
- Un condicional group en el caso de que se requiera agrupar la filas resultantes en grupos
- Un condicional having para definir que las filas agrupadas deberán coincidir con un criterio de agrupamiento
- Un string donde se define el orden de retorno de las filas
- Un string donde se especifique el valor máximo de las filas en el resultado de la consulta.

En las líneas siguientes se muestra un pequeño ejemplo de cómo se puede realizar una consulta o *query* a una base de datos SQLite en Android.

```
String[] result_columns = new String[]{
KEY_ID, KEY_GOLD_HOARD_ACCESIBLE_COLUMN, KEY_GOLD_HOARDED_COLUMN };
String where = KEY_GOLD_HOARD_ACCESSIBLE_COLUMN + “=” + 1;
String whereArgs[] = null;
String groupBy = null;
String having = null;
String order =  null;

SQLiteDatabase db = hoardDBOpenHelper.getWritableDatabase();
Cursor cursor = db.query(HoardDBOpenHelper.DATABASE_TABLE, result_columns, where,
whereArgs, groupBy, having, order);
```

Capítulo 4.
Desarrollo

4.1 Selección de las Tecnologías

Como se ha mencionado en los párrafos anteriores, el mundo de la domótica en la actualidad está creciendo de manera exponencial y cada día son más los nuevos dispositivos y elementos que se integran a la automatización en el hogar. Debido a esta gran variedad de aplicaciones y posibilidades de implementación de tecnologías, a continuación se realizará una selección de las tecnologías más adecuadas para el prototipo de diseño domótico que se pretende construir en base a los objetivos generales y específicos planteados en un inicio.

Para el diseño del sistema domótico se han identificado principalmente cinco elementos fundamentales que integraran el prototipo y se describen en la figura 8.

Figura 8. Esquema de elementos de prototipo domótico.

Basados en el esquema anterior y con relación a la información analizada en el marco teórico, se definirán las tecnologías adecuadas para el diseño e implementación del prototipo domótico para el control en el hogar.

4.1.1 Interfaz de Usuario e Interacción

El gran auge con el que actualmente cuentan los dispositivos y tecnologías móviles ha sido tal que, hoy en día la mayoría de las personas con acceso a este tipo de aparatos cuenta al menos con un teléfono inteligente (*smartphone*), tableta electrónica, phablet (*smartphone – tableta)* o algún gadget con el que pueda estar siempre en comunicación y que cuente con una gran gama de aplicaciones y programas de mucha utilidad para su vida cotidiana.

Debido a esto, la mejor alternativa o el mejor medio de interacción que puede ser elegido para el prototipo domótico ha de ser las tecnologías móviles enfocadas a *smartphones* y tabletas electrónicas.

Teniendo claro el medio de interacción, se deberá mencionar que existen diversos sistemas operativos alojados en estos dispositivos y que son la base de la operación de ellos. Como ya se mencionó en el capítulo 3.6 (*Desarrollo para móviles: Android*) de este documento, uno de los principales sistemas operativos en la Actualidad es Android ya que, además de ser muy versátil, es de uso libre y está respaldado por un gran grupo de desarrolladores expertos que han ido mejorando esta plataforma. Por estas razones, se ha elegido Android como sistema operativo de los dispositivos para el prototipo domótico.

4.1.2 Protocolo y Medio de comunicación

Otro ámbito de gran crecimiento y que ha tenido muchos cambios y mejoras en la actualidad han sido los sistemas de comunicación, y en particular, los medios y estándares de transmisión de información. Se mencionaban y describían en el capítulo 3.5 algunas de las principales tecnologías de comunicación inalámbrica que en la mayoría de los sistemas y dispositivos de nueva generación, han sido implementadas.

Hablando principalmente de las tecnologías de comunicación aplicadas en el hogar, está de más mencionar que la más utilizada y de mayor demanda ha sido el WiFi ya que, la mayoría de los equipos que se conectan a internet en los hogares, lo hacen inalámbricamente a través de esta tecnología. Con esto, se puede definir que el tipo de tecnología a implementar para la comunicación del *smartphone o tablet* será vía WiFi. La forma en que funcionará será conectando el dispositivo móvil a la red WiFi del *Router* del hogar para establecer una conexión de red que pueda permitir la comunicación del *smartphone* con la Puerta de Enlace Residencial o Control Central del prototipo utilizando como medio de comunicación la ya mencionada tecnología.

4.1.3 Puerta de Enlace Residencial o Control Central

Dentro de los elementos del prototipo domótico, uno de los principales y de mayor importancia es el Control Central o la Pasarela Residencial. Este elemento es el encargado de procesar la información proveniente de los dispositivos de interacción y enviar las instrucciones de mando o control a los elementos receptores para que pueda ejecutarse acciones en los actuadores finales.

La peculiaridad principal de este dispositivo es la integración de dos diferentes tipos de tecnologías de transmisión / recepción de información para poder interactuar con

todos los elementos que conformaran el prototipo domótico. En primera, deberá tener la capacidad de procesar información de peticiones TCP / IP provenientes del dispositivo de interacción o *smartphone*, esto es, debe tener elementos para conexión a red LAN o WiFi. También y cómo segundo tipo de tecnología, deberá tener la capacidad de poder comunicarse utilizando los estándares ZigBee para poder interactuar con los elementos receptores y poder activar los actuadores finales.

La tecnología ZigBee es un protocolo estándar de radiocomunicación respaldado por una Alianza global de más de 400 empresas dedicadas a las Tecnologías de la Información y el Internet de las Cosas (IoT). ZigBee se ha vuelto una de las tecnologías más usadas en las implementaciones de aplicaciones domóticas ya que su implementación es muy sencilla y sigue protocolos internacionales de radiofrecuencias basadas en estándares actuales. Además de que es una tecnología muy accesible en cuanto a costos y disponibilidad en el mercado. Por estas razones, se optó por utilizar ZigBee como tecnología de comunicación para los actuadores finales.

Actualmente, existen diferentes tecnologías y sistemas embebidos que tienen la capacidad de poder integrar diferentes tipos de tecnologías de comunicación en un solo dispositivo y permitiendo cubrir el requerimiento para el prototipo del sistema domótico. Uno de estos sistemas embebidos, como ya se mencionó en el capítulo 3.4, es el Arduino.

Para lograr el propósito de poder utilizar tanto la tecnología LAN o WiFi como la ZigBee en una sola Puerta de Enlace Residencial, es necesario que a la placa Arduino se le adicione alguna placa de expansión o *shield* para poder complementar el uso de las dos tecnologías. Por tanto, se optará por utilizar la placa Arduino UNO Ethernet, la cual brinda las bondades de una placa Arduino UNO más la capacidad de tener una conexión Ethernet vía LAN. Además, a esta placa se le conectará un *XBee shield*, placa

de expansión para poder integrar un dispositivo *XBee* para la utilización de la tecnología ZigBee y poder tener comunicación con los elementos receptores y los actuadores finales.

4.1.4 Elemento Receptor

Estos dispositivos serán los intermediarios entre la Puerta de Enlace Residencial y los Actuadores finales. Deberán tener la capacidad de poder comunicarse con el Control Central para poder recibir órdenes de mando así como poder interactuar con los dispositivos finales para ejecución de instrucciones.

Para poder procesar instrucciones de la Puerta de Enlace Residencial, se deberá tener capacidad para poder recibir información mediante el estándar ZigBee. En base a esto, se deberá contar con un dispositivo XBee en configuración tipo *Dispositivo Final* el cual deberá contar con una etapa de acoplamiento hacia los actuadores. Este acoplamiento podría variar dependiendo del tipo de actuador final que le sea adaptado al elemento receptor.

4.1.5 Actuadores Finales

Como su nombre lo menciona, estos elementos serán los encargados de accionar y dar vida al prototipo domótico ya que serán los responsables de controlar la iluminación de luz natural (en el caso de las persianas), brindar una iluminación especial o de ciertas características (control de iluminación) o los que pueden brindar algún tipo de confort o ambiente a los usuarios finales.

Estos dispositivos se conectarán directamente a los elementos receptores mediante algún método de adaptación o acoplamiento en base a las necesidades del elemento a

controlar. Para este caso en particular del control de persianas, se podría optar por una etapa de acoplamiento de potencia en base al uso de pequeños relevadores ya sea de estado sólido o de enclave mecánico para activar los motores.

4.2 Diseño e Implementación de Software

Actualmente en el mundo del desarrollo de software, existen diversas tecnologías, metodologías y lenguajes de programación que permiten a los programadores crear gran variedad de utilidades y aplicaciones que facilitan la vida cotidiana.

Para desarrollar algún software o aplicación, es importante mencionar que se debe realizar todo un proceso de planeación y análisis para tomar en cuenta la funcionalidad principal de la aplicación, la estructura de datos necesaria, el diseño de las Interfaces, la usabilidad y la interacción con el usuario. Básicamente, para el diseño e implementación de la aplicación para el prototipo de sistema domótico, se estructurará el proceso en cuatro bloques de actividades:

1. Requerimientos
2. Diseño
3. Codificación
4. Pruebas de la aplicación

4.2.1 Requerimientos

Como punto de partida se deberá establecer el alcance que se quiere lograr y hasta qué punto se llegará como prototipo o primera etapa. Tomando esto como base, podemos establecer lo siguiente para el diseño de la aplicación:

- Será una aplicación nativa para Android

- La versión mínima de Sistema Operativo deberá ser la 4.0 y la versión objetivo será la 4.4 o superior
- Deberá contar con un sistema de administración de información o base de datos local para garantizar la integridad de la información.
- Deberá contar con capacidad para conexión a redes WiFi
- Debe ser una aplicación escalable y fácil de actualizar
- El diseño de interfaces será pensado para smartphones

4.2.2 Diseño

Para el diseño de la aplicación de prototipo domótico se deberán considerar tres rubros importantes:

- Diseño de la estructura de código
- Diseño de la estructura de datos (base de datos)
- Diseño de la Interface de Usuario

4.2.2.1 Diseño de la estructura de código

Para el tema del diseño y codificación, se considerará la implementación de una estructura de clases y archivos organizada en Capas para lograr una mejor organización de los todos los archivos. Añadido a lo anterior y debido a que la herramienta de desarrollo de software será Android Studio (herramienta oficial para el desarrollo de aplicaciones para Android), se mantendrá la estructura que el software aplica sobre los archivos de la aplicación, agregándole la ya mencionada estructura de Capas.

La estructura de Capas que se diseñará, contará con cuatro secciones:

- Capa de Objetos o DTOs (*Data Transfer Objects*)

- Capa Visual o de Activities
- Capa de Acceso a Datos o DAOs (*Data Access Objects*)
- Capa de clases para la base de datos

Dentro de la sección de Objetos o DTOs, se definirán las clases de cada uno de los objetos a utilizar con sus respectivas propiedades. Estos objetos serán los elementos con los que se podrá manejar la información de una forma más estructurada y organizada. Para la aplicación se crearan los siguientes dos tipos de objetos con sus respectivas propiedades

- Objeto Habitación
  - Identificador de la Habitación
  - Nombre de la Habitación
- Objeto Persiana
  - Identificador de la Persiana
  - Nombre de la Persiana
  - Estado
  - Identificador de Habitación
  - Conexión

La Capa de Activities o Visual es propiamente la carpeta que crea Android Studio donde se van alojando las clases que contienen toda la estructura de código para el funcionamiento de las interfaces de usuario en la aplicación. Estas clases se relacionan a un archivo XML el cual contiene la distribución de los componentes visuales que componen al Activity.

Los DAOs o la Capa de Acceso a Datos, son clases que sirven de intermediarios entre la Capa Visual y la base de datos. Estas clases proporcionan diversos métodos para poder

lograr consultas a la base de datos y poder administrar la información contenida en ella. Es conveniente crear un elemento DAO para cada objeto DTO, es por esto que, se incluirán los siguientes objetos como DAOs:

- DAO de Objeto Habitación
- DAO de Objeto Persiana

Dentro de cada DAO, se requerirá definir por lo menos 4 métodos para poder interactuar con la base de datos que pueden ser:

- Abrir / Cerrar conexión
- Insertar registros
- Obtener registros
- Eliminar registros

En la última capa, se define una clase que ayuda en la creación o actualización de la base de datos y donde se define la estructura que contendrá cada una de las tablas así como los tipos de datos que se podrán manejar en ellas.

4.2.2.2 Diseño de la estructura de datos (base de datos)

La estructura de información o los datos dentro de la aplicación, tendrán relación directa con los objetos y la funcionalidad de la aplicación. Como se describió en la Capa de DTOs, para la aplicación se crearan los tipos de objetos Habitación y Persiana, por tal, se deberá crear una tabla en la base de datos para cada uno de estos objetos. Adicional a esto, la tabla persiana contendrá una columna relacional para saber la ubicación de esta en una cierta habitación. De forma esquemática, se podría definir la entidad relación de la base de datos como se muestra en la figura 9.

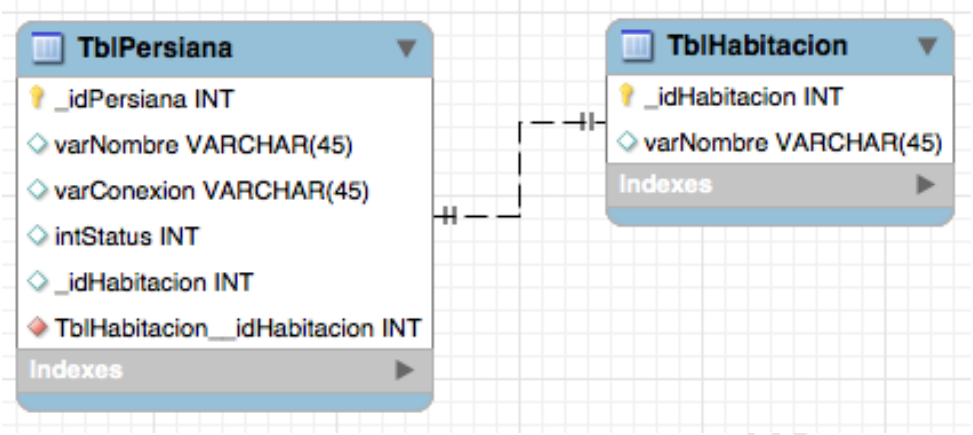


Figura 9. Esquema entidad relación de la base de datos.

4.2.2.3 Diseño de Interfaces de Usuario

Uno de los tópicos de mayor importancia al momento de la programación de un software o aplicación, es la creación de la Interfaz de Usuario. La importancia de esto radica en el hecho de que la apariencia y el aspecto visual es lo primero con lo que el usuario final entra en contacto y mucho de esto dependerá para que el usuario pueda o no continuar usando la aplicación.

Hablar de diseño de interfaces podría ser algo ambiguo ya que involucra diversos aspectos y pueden llegar a intervenir infinidad de metodología y técnicas específicas para un mejor desarrollo de los componentes visuales.

Para el diseño de las interfaz de Usuario para el prototipo de sistema domótico, se partirá de la creación de sencillos esquemas de diseño llamados "*Mockups*" considerando los diferentes componentes que pudieran integrar cada forma o *Activity* en la aplicación. Basado en lo anterior, los *mockups* de los *Activities* que se utilizaran en la aplicación serán:

- Menú principal, figura 10.

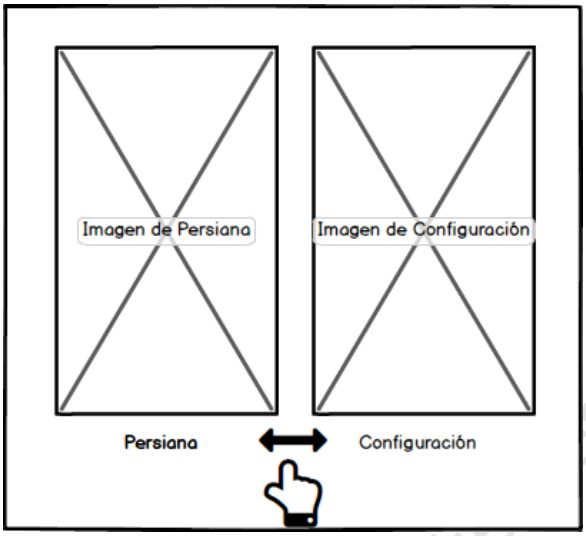


Figura 10. *Mockup* Menu principal.

- Menú de configuraciones, figura 11.

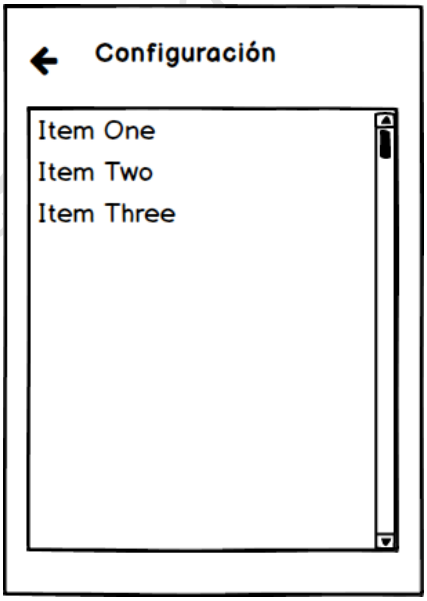


Figura 11. *Mockup* menú configuraciones.

- Listado de selección de persiana con elementos de acción, figura 12.

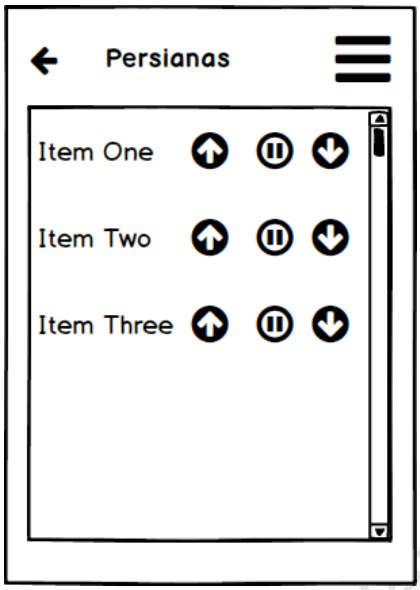


Figura 12. *Mockup* listado de selección de persiana.

- Menú para agregar persianas, figura 13.

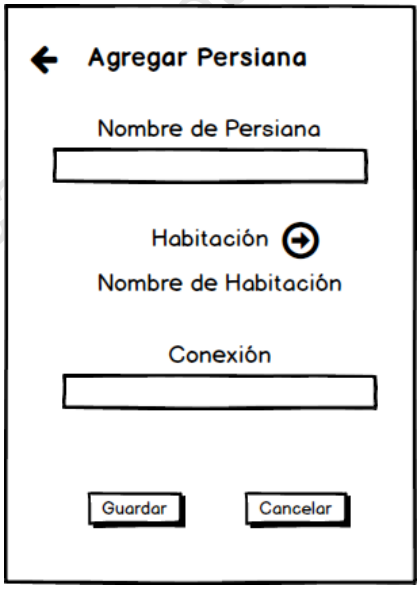


Figura 13. *Mockup* menú agregar persianas.

- Listado de selección de habitaciones, figura 14.

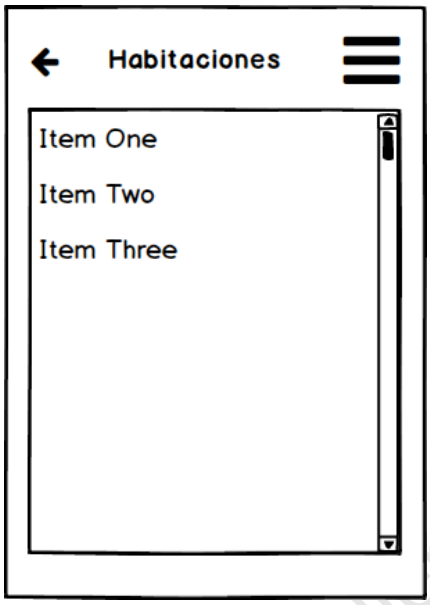


Figura 14. *Mockup* listado de selección de habitaciones.

- Menú para agregar habitaciones, figura 15.

Figura 15. *Mockup* menú agregar habitaciones.

Junto con la creación de los *mockups*, se deberá establecer un diagrama o flujo de pasos que determinaran el orden y la lógica en que la aplicación mostrará estas ventanas. En la figura 16 se muestra el flujo propuesto.

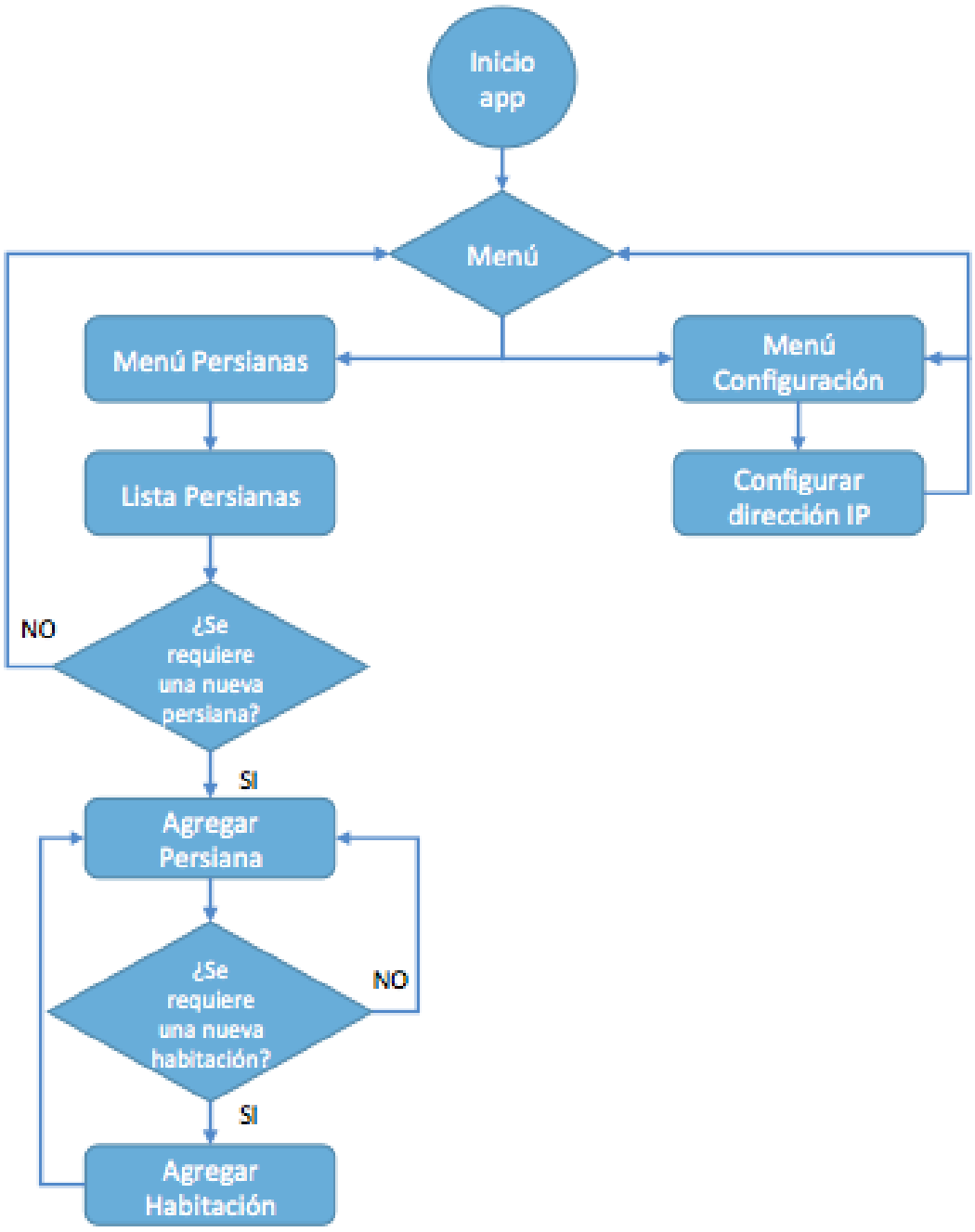


Figura 16. Flujo y lógica de despliegue de ventanas en la aplicación.

4.2.3 Codificación

Actualmente una de las herramientas principales para el desarrollo de aplicaciones móviles para Android es el IDE Android Studio, el cual integra todas las librerías y el ambiente de codificación más adecuado para la creación de *apps* de forma profesional.

Como se mencionó anteriormente, en la figura 16, el flujo de la aplicación móvil en Android comenzará su ciclo con el menú principal, es aquí donde se comenzará a realizar la codificación. Este menú constará de dos *Fragments*, el primero de ellos asociado al menú de persianas y, el segundo, servirá para poder realizar configuraciones importantes a la aplicación.

Como ya se mencionó anteriormente, la aplicación se estructurará en 4 capas principalmente. Se comenzará con la codificación de los objetos que permitirán el manejo de la información entre las diferentes capas (DTOs). Es importante remarcar que estos elementos deberán contener las propiedades principales que nos ayuden con el manejo de las columnas y variables necesarias en base al diseño de base de datos propuesto en el capítulo anterior.

El primer objeto DTO a codificar será el *DTOPersiana*. En la figura 17 se ilustra un fragmento del código donde se muestran las principales propiedades que contendrá este objeto así como su constructor con la inicialización de las variables mencionadas.

```
public class DTOPersiana {

    private int IdPersiana;
    private String NombrePersiana;
    private int StatusPersiana;
    private int IdHabitacion;
    private String Conexion;

    public DTOPersiana(){
        this.IdPersiana = 0;
        this.NombrePersiana = "";
        this.StatusPersiana = 0;
        this.IdHabitacion = 0;
        this.Conexion = "";
    }
```

Figura 17. *DTOPersiana* y sus propiedades.

El segundo objeto DTO será el *DTOHabitacion*. De igual manera, en la figura 18 se muestra un fragmento del código con las propiedades de este objeto.

```
public class DTOHabitacion {

    private int IdHabitacion;
    private String NombreHabitacion;

    public DTOHabitacion(){
        this.IdHabitacion = 0;
        this.NombreHabitacion = "";
    }
```

Figura 18. *DTOHabitacion* y sus propiedades.

Teniendo los objetos DTOs, se deberá comenzar con la programación de la capa para conexión y transacciones a base de datos. El primer elemento a crear será la clase *DBHelper* que contendrá las variables estáticas y los métodos para crear o actualizar la base de datos SQLite de la aplicación y la cual heredará características de la clase *SQLiteOpenHelper*.

En base al diseño de la entidad relación de base de datos de la figura 9, se comenzará a declarar cada una de las columnas de las tablas como variables estáticas globales dentro de la clase *DBHelper*. La figura 19 muestra la declaración de variables para las tablas de Persiana y Habitación.

```
//Tabla Persiana
public static final String TABLA_PERSIANAS = "TblPersiana";
public static final String PERSIANA_ID = "_idPersiana";
public static final String PERSIANA_NOMBRE = "varNombre";
public static final String PERSIANA_CONEXION = "varConexion";
public static final String PERSIANA_STATUS = "intStatus";
public static final String PERSIANA_HABITACION = "_idHabitacion";

//Tabla Habitacion
public static final String TABLA_HABITACION = "TblHabitacion";
public static final String HABITACION_ID = "_idHabitacion";
public static final String HABITACION_NOMBRE = "varNombre";
```

Figura 19. Variables estáticas para la creación de las tablas de base de datos.

Una vez definidas las variables correspondientes a las columnas de las tablas, se deberá crear una variable tipo *String* que contenga los *scripts* para la creación de las tablas en la base de datos de la aplicación. En la figura 20 se muestra el *script* para la creación de la tabla *TblPersiana*.

```
//Create Tabla Persisana
private static final String CREATE_TABLA_PERSIANA = "create table "
        + TABLA_PERSIANAS + "(" + PERSIANA_ID + " integer primary key autoincrement, "
        + PERSIANA_NOMBRE + " text not null, "
        + PERSIANA_CONEXION + " text not null, "
        + PERSIANA_STATUS + " integer null, "
        + PERSIANA_HABITACION + " integer," + " FOREIGN KEY (" + PERSIANA_HABITACION
        + ") REFERENCES " + TABLA_HABITACION + " (" + HABITACION_ID + "));";
```

Figura 20. *Script* para creación de tabla *TblPersiana* en base de datos.

Para poder crear la base de datos, se sobrescribirá el método *onCreate* de la clase *DBHelper* donde se deberá recibir una instancia del tipo *SQLiteDataBase*. Dentro de este método, usando la *instancia* de la base de datos que se recibe, se usará el método

execSQL pasándole como parámetro la variable anteriormente mencionada. La figura 21 muestra un fragmento de la clase mencionada y la forma en que se manda llamar el método para la creación de las tablas.

```
@Override
public void onCreate(SQLiteDatabase database) {
    database.execSQL(CREATE_TABLA_PERSIANA);
```

Figura 21. Método *onCreate* donde ser crean las tablas de la base de datos.

Teniendo la clase *DBHelper*, se procederá a la creación de las clases tipo *DAO*. Dentro de estas clases, se incluirán los métodos principales que ayudarán a la consulta de la información proveniente de la base de datos. Los métodos principales a incluir serán:

- *Open*: abrirá la instancia de la base de datos para tener acceso a ella.
- *Close*: cerrará la instancia de la base de datos.
- Insertar: se podrán agregar elementos a las tablas.
- Obtener: retornará la información contenida en una tabla.
- Eliminar: borrará un elemento existente en una tabla.

Es importante mencionar que se debe crear una clase *DAO* por cada uno de los elementos *DTO*, es decir, se tendrá una clase *DAOPersiana* y otra *DAOHabitacion*. En la figura 22 se muestra el método *insertarPersiana* de la clase *DAOPersiana*, el cual recibe algunas variables como parámetros para poder agregar un registro en la tabla *TblPersiana*.

```
public DTOPersiana insertarPersiana(String nombrePersiana, String conex,
                                    int status, int idHabita) {
    try {
        dataBase = dbHelper.getWritableDatabase();
    } catch (SQLiteException ex){
        //Log.e("No writable DB, " + ex.getMessage());
        dataBase = dbHelper.getReadableDatabase();
    }

    ContentValues values = new ContentValues();
    values.put(dbHelper.PERSIANA_NOMBRE, nombrePersiana );
    values.put(dbHelper.PERSIANA_CONEXION, conex);
    values.put(dbHelper.PERSIANA_STATUS, status);
    values.put(dbHelper.PERSIANA_HABITACION, idHabita);
    long insertId = dataBase.insert(dbHelper.TABLA_PERSIANAS, null,
            values);
    Cursor cursor = dataBase.query(dbHelper.TABLA_PERSIANAS,
            allColumns, dbHelper.PERSIANA_ID + " = " + insertId, null,
            null, null, null);
    cursor.moveToFirst();
    DTOPersiana dtoPersiana = cursorToPersiana(cursor);
    cursor.close();
    return  dtoPersiana;
}
```

Figura 22. Método *insertarPersiana* para agregar una persiana a la base de datos.

Definidos los *DTOs* y las clases *DAO*, el enfoque ahora será sobre la *Activity* principal, la cual servirá como menú principal de la aplicación. Esta *Activity* alojará un *PageAdapter* que contendrá dos *Fragments*, los cuales serán los principales elementos para poder interactuar con las persianas en la aplicación. Estos *Fragments* podrán alternarse en el *PageAdapter* mediante deslizamientos de la pantalla hacia la izquierda o derecha con una transición de animación tipo *ZoomOut*. En la figura 23 se aprecia un fragmento del código encontrado en la *Activity* principal donde se muestra el *PageAdapter* para los *Fragments*.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_my_dohme);

    ViewPager pager = (ViewPager) findViewById(R.id.viewPager);
    pager.setPageTransformer(true, new ZoomOutPageTransformer());
    pager.setAdapter(new MyPagerAdapter(getSupportFragmentManager()));
}

private class MyPagerAdapter extends FragmentPagerAdapter {

    public MyPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int pos) {
        switch(pos) {
            case 0: return FragmentCuatro.newInstance("Persianas");
            case 1: return FragmentTres.newInstance("Configuración");
            default: return FragmentCuatro.newInstance("Persianas");
        }
    }

    @Override
    public int getCount() { return 2; }
}
```

Figura 23. Principales métodos en el *Activity* principal de la aplicación.

Como se muestra en la figura 23, el *PageAdapter* del *Activity* principal de la aplicación, requiere de dos *Fragments* para el despliegue del menú principal. El primero es el *FragmentCuatro*, el cual contiene un *ImageView* con un ícono de una persiana el cual, al ser presionado, sirve de acceso directo al menú de la lista de persianas. En la figura 24 se muestra una sección del código del *FragmentCuatro* con el evento *onCreateView* donde se asocia el archivo xml de la vista a la clase para luego agregarle un evento de *onClick* al *ImageView* para mostrar el *Activity* con el listado de persianas existentes en la base de datos de la aplicación. En la figura 25 se muestra parte del xml con los componentes visuales que serán mostrados en este *Fragment*. En la figura 26 se muestra una vista previa de aspecto visual del *FragmentCuatro*.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragmentcuatro, container, false);

    TextView tv = (TextView) view.findViewById(R.id.tvFragCuat);
    tv.setText(getArguments().getString("msg"));

    ImageView image = (ImageView) view.findViewById(R.id.imgPer);
    image.setOnClickListener((view) → {
            Intent intent = new Intent(view.getContext(), lista_persianas.class);
            startActivity(intent);
    });

    return view;
}
```

Figura 24. Método *onCreateView* del *FragmentCuatro.*

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/background_dark" >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:src="@drawable/persiana"
        android:id="@+id/imgPer"/>
```

Figura 25. XML del diseño visual para el *FragmentCuatro.*

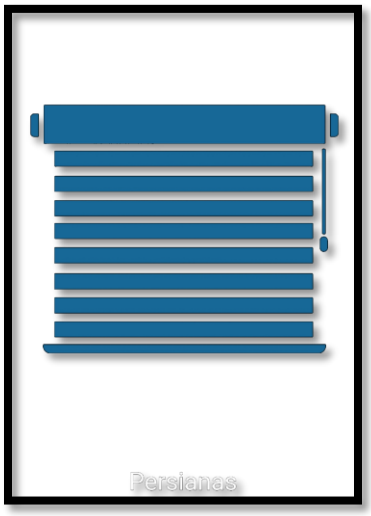


Figura 26. Vista previa del aspecto visual del *FragmentCuatro.*

4.2.4 Pruebas de la aplicación

Dentro del IDE de desarrollo de aplicaciones Android Studio, el software cuenta con la opción de poder emular en dispositivo virtual para ejecutar pruebas con la aplicación programada. Aunque esto es una ventaja, es necesario realizar diversas configuraciones previas para que el dispositivo virtual pueda ser emulado correctamente. Otra opción disponible para la ejecución de las pruebas, es la integración de un dispositivo móvil físico a la plataforma, donde solo es necesario conectar el dispositivo a través de un puerto USB disponible en la PC y elegir el modelo del dispositivo móvil, sobre el cual se realizaran las pruebas.

Para la ejecución de las pruebas de la aplicación se ha elegido la segunda opción. Las pruebas se realizarán en un dispositivo móvil con sistema operativo Android con la versión 5.0 de nombre Lollipop con número de API 21. Para comenzar la ejecución de las pruebas en el móvil, se deberá iniciar la aplicación presionando el botón de *Arrancar app* en el menú de opciones del IDE . En la figura 27, se muestra la opción el botón para iniciar la aplicación.

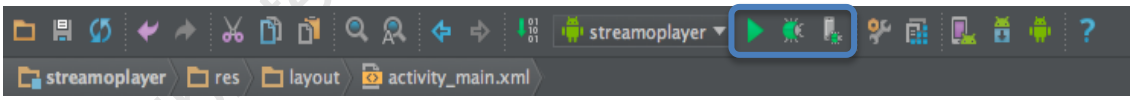


Figura 27. Opciones para ejecutar las pruebas de la aplicación.

Una vez presionado el botón de arranque de aplicación, el software nos pedirá que seleccionemos el dispositivo donde se ejecutarán las pruebas. En este caso, optaremos por seleccionar el dispositivo conectado al equipo (figura 28).

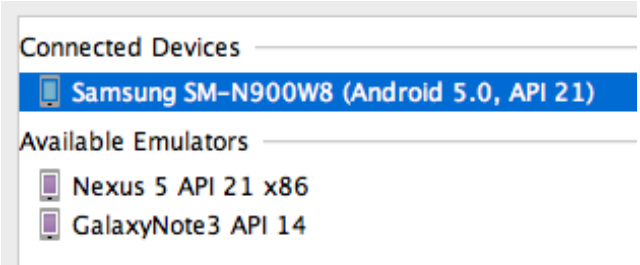


Figura 28. Lista de dispositivo para la ejecución de pruebas de la aplicación.

Seleccionado el dispositivo, el software comenzará a compilar la aplicación y la instalará en el dispositivo seleccionado para las pruebas. En caso de que el código contenga algún error, se desplegará en la ventana de mensajes del IDE mostrando una descripción del problema encontrado durante la compilación. Para nuestro caso, al momento de realizar la primera ejecución, se encontró un pequeño error de sintaxis en la clase principal de la aplicación. En la figura 29 se muestra cómo el IDE despliega los mensajes de error encontrados durante la compilación.

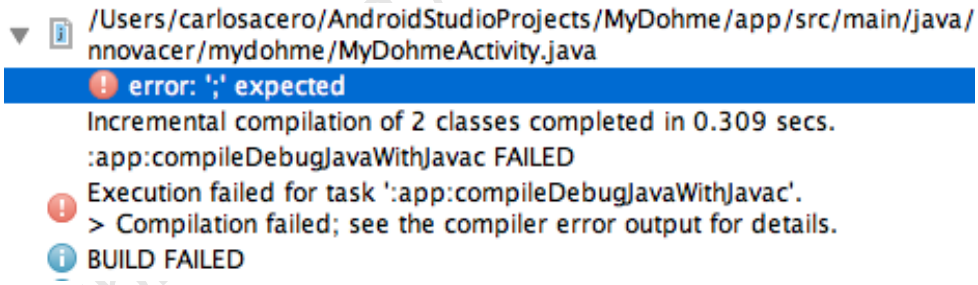


Figura 29. Mensaje de error al momento de compilar la aplicación.

Corregido el error, se procedió de nueva cuenta a ejecutar la aplicación resultando en una correcta compilación del código e instalación de la aplicación en el dispositivo. En la figura 30 se muestra la pantalla del dispositivo con el menú principal de la aplicación en la opción de Persianas, en la figura 31 se muestra la transición al momento de cambiar de opción en el menú y en la figura 32 se puede apreciar la opción de Configuración en el mismo menú principal.

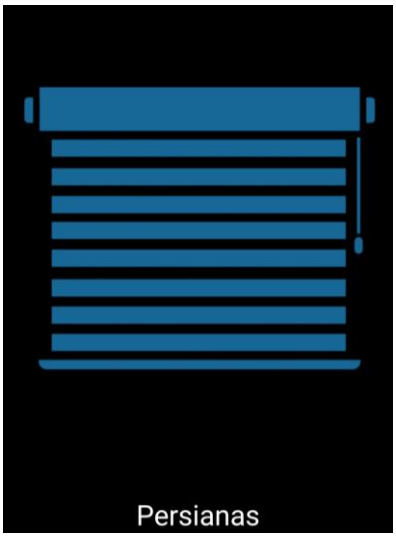


Figura 30. Menú principal en la opción de Persianas activa.

Figura 31. Transición de opciones en el menú principal.

Figura 32. Menú principal en la opción de Configuración activa.

La primera opción a probar será la del menú de Persianas. Para ingresar, bastará con tener en el menú principal la opción de Persianas activa y hacer clic sobre la imagen donde se validarán los siguiente puntos:

1. Mostrar listado de persianas
2. Agregar una persiana a la lista de persianas
3. Mostrar listado de habitaciones
4. Agregar una habitación a la lista de habitaciones
5. Al activar los botones de dirección arriba o abajo, se deberá mostrar un mensaje con la dirección del botón presionado.

Para poder validar el punto número 1, primeramente se deberán cumplir los puntos número 2, 3 y 4. Para agregar una persiana a la lista, se deberá presionar el botón de opciones ubicado en la barra superior derecha (figura 33), el cual desplegará la opción para Agregar Persiana (figura 34).

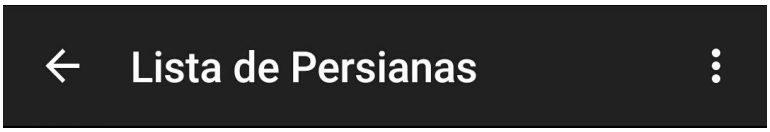


Figura 33. Botón de opciones identificado con 3 puntos.

Figura 34. Opción Agregar Persiana.

Presionada la opción para agregar persiana, se mostrará un *Activity* con un formulario donde se deberá completar la información solicitada para poder agregar la persiana a la lista como se muestra en la figura 35.

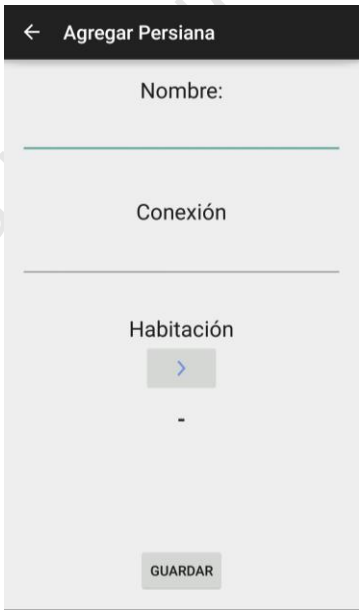


Figura 35. *Activity* para Agregar Persiana.

Los campos a completar en el formulario son Nombre (identificación que se le dará a la persiana), Conexión (este dato se obtendrá del número de serie del XBee instalado en

el actuador final el cual será el encargado de activar la persiana) y Habitación (se seleccionará de un listado que será desplegado al presionar el botón con la flecha). Completos los dos primeros campos, se deberá proceder a seleccionar la habitación donde se instalará la persiana.

La información del campo de Habitación deberá provenir de un listado de Habitaciones previamente registradas en la aplicación. Para dar de alta una habitación, basta con presionar el botón con la flecha debajo de la etiqueta de Habitación. Al realizar esto, se mostrará una *Activity* con un listado de Habitaciones disponibles para seleccionar. Para dar de alta una habitación, al igual que en el listado de persianas, se deberá presionar el botón de opciones ubicado en la barra superior derecha (los tres puntos) y seleccionar la opción de Agregar Habitación. La figura 36 muestra esta opción.

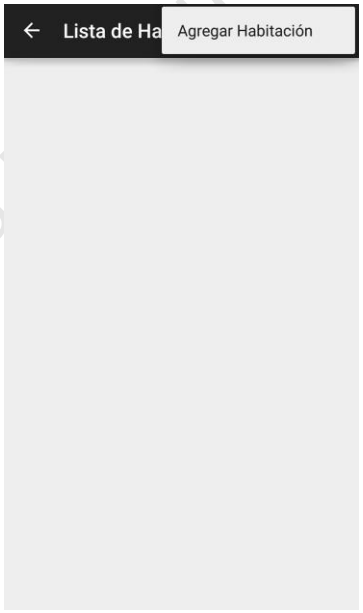


Figura 36. Opción para Agregar Habitación.

Seleccionada la opción de Agregar Habitación, se mostrará un *Activity* con el campo de Nombre donde se deberá registrar la habitación como se muestra en la figura 37.

Figura 37. *Activity* para Agregar Habitación.

Una vez que ser ingresa el nombre de la habitación, se presiona el botón Guardar para registrarla en la lista de habitaciones. La figura 38 muestra un ejemplo de una habitación agregada a la lista.

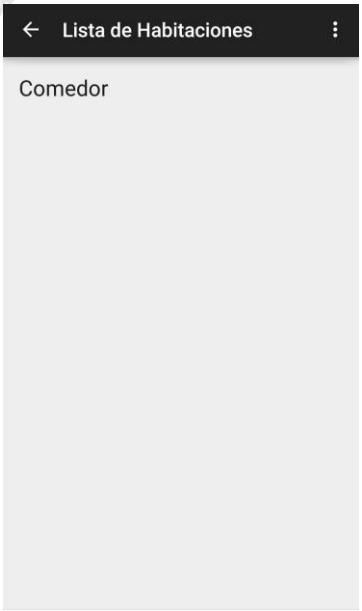


Figura 38. Lista de habitaciones.

Teniendo nuestra habitación disponible en la lista, se deberá seleccionar dejando presionado sobre ella durante un par de segundos y, automáticamente, la aplicación regresará al *Activity* para dar de alta la persiana mostrando el nombre de la habitación que se ha seleccionado. La figura 39 muestra un ejemplo de cómo se podría completar la información en ese formulario.

Figura 39. Formulario completo para agregar una persiana.

Una vez ingresada toda la información, bastará con presionar el botón Guardar para agregar la persiana a la lista. La aplicación automáticamente regresa al *Activity* donde se enlistan las persianas agregadas agregándole los botones de funciones para subir, bajar y detener la persiana. La figura 40 muestra una persiana agregada a la lista.

Figura 40. Lista de persianas agregadas a la aplicación.

La segunda prueba a realizar será en el menú de Configuración. Como se mencionó en los párrafos anteriores, esta opción se encuentra en el menú principal, sólo se deberá realizar un gesto con el dedo para deslizar la opción de Persianas hacia la izquierda y mostrar la de Configuración (como se muestra en la figura 31). Teniendo el menú disponible, sólo bastará hacer clic sobre la imagen para ingresar. La figura 41 muestra las opciones a configurar dentro de la aplicación.

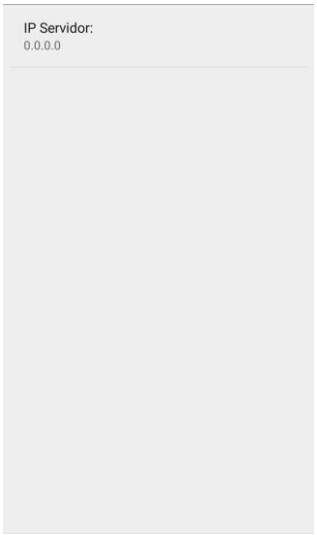


Figura 41. Lista de configuraciones para la aplicación.

Para esta versión de la aplicación, sólo se está considerando una opción de Configuración que es la dirección IP del servidor (en este caso el servidor será parte de la Puerta de Enlace Residencial) que estará recibiendo las peticiones de la aplicación. Este dato se deberá establecer hasta el momento en que se tenga completo el diseño de la Puerta de Enlace Residencial que será la interface entre la aplicación del dispositivo móvil y los elementos a controlar, en este caso, la persiana a operar.

Como se pudo apreciar, la aplicación realiza las acciones correctamente al momento de agregar habitaciones y persianas. Se puede determinar que la aplicación móvil funciona sin ningún problema y en base al diseño anteriormente planteado. Posteriormente, se realizarán pruebas donde se estará integrando el dispositivo móvil donde se instaló la aplicación a la misma red de la Puerta de Enlace Residencial para poder enviar peticiones y poder verificar que haya comunicación entre estos dos elementos.

Es importante mencionar que se debe ingresar correctamente el número de serie del XBee en el campo de “Conexión” del Formulario para agregar una persiana, esto con la finalidad de enlazar una conexión correcta entre la Puerta de Enlace Residencial con el Actuador Final. Este número de serie es único para cada dispositivo XBee y se encuentra ubicado en la parte posterior de cada dispositivo. En la figura 42 se muestra un ejemplo de número de serie de XBee y donde se ubica en el componente.

Figura 42. Número de serie de un XBee.

4.3 Diseño e Implementación de Hardware

Parte importante de la estructura y construcción del prototipo de sistema domótico es el diseño e implementación de los elementos electrónicos que conformarán o harán parte del control, la comunicación y la actuación con los elementos finales o, por resumirlo todo en un solo termino, estamos hablando del hardware.

Es importante remarcar que el software es la esencia fundamental y el elemento base que se encargará internamente de cubrir las necesidades que el sistema demanda, pero sin el hardware, no existiría la posibilidad de poder interactuar con los elementos físicos y que son los que, al final, hacen posible que un sistema cobre vida.
Como se mencionaba en el apartado 4.1 de este mismo capítulo, el prototipo de sistema domótico constará principalmente de 5 elementos fundamentales (como se puede apreciar en la figura 8 de ese mismo apartado) pero, dentro de esta sección, sólo se estará haciendo énfasis principalmente en la Puerta de Enlace Residencial y los Actuadores Finales.

La Puerta de Enlace Residencial es el elemento central de control del prototipo domótico ya que será el encargado de recibir las peticiones desde el Smartphone y direccionarlas a los actuadores finales en base a los requerimientos del usuario. Constará básicamente de un Arduino UNO que actuará como Servidor Web a través del uso de un Ethernet Shield que será conectado al dispositivo proveedor del servicio de red inalámbrica en el hogar (*Router*) así como de un Xbee Shield que, mediante el dispositivo XBee, será el encargado de enviar los mensajes y los mandos de control al actuador final el cual, tendrá acoplada la persiana motorizada a controlar instalada en la residencia.

De esta forma, el Actuador Final será el dispositivo encargado de interactuar directamente con la persiana motorizada. Será el que estará recibiendo las instrucciones de control que vendrán de la Puerta de Enlace Residencial mediante la utilización de comunicación vía ZigBee. Este dispositivo, adicional al elemento XBee, utilizará algunos componentes electrónicos para poder realizar el acoplamiento hacia el motor de la persiana y poder controlar su accionamiento.

En la figura 43, se muestra un esquema gráfico de los elementos principales de hardware que integraran el prototipo domótico.

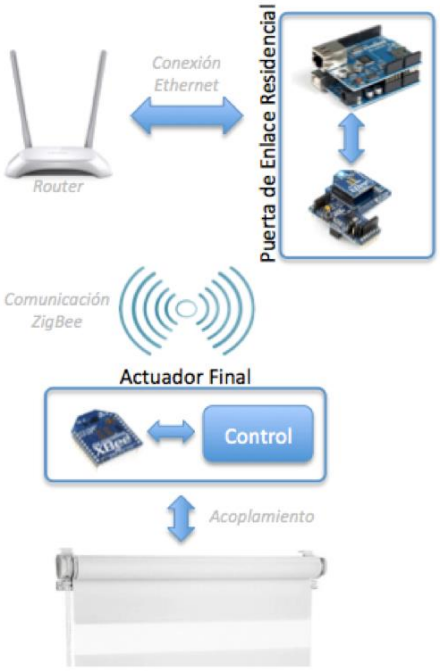


Figura 43. Esquema gráfico de elementos de hardware de prototipo domótico.

Para la parte de Control y Acoplamiento de electrónica discreta y de potencia en el Actuador Final, como prototipo se diseñará un pequeño circuito utilizando relevadores y transistores para poder enviar la señal de control de las salidas del XBee y poder actuar el motor de la persiana.

Para el diseño del circuito de control se utilizará un opto acoplador conectado a cada una de las salidas del XBee como protección y aislamiento del elemento sensible (XBee) y la de control. En la figura 44, se muestra de forma esquemática los elementos que constituirán el Actuador Final.

Figura 44. Esquema de elementos del Actuador Final.

4.4 Diseño e Implementación de sistema y protocolo de comunicación

En los capítulos anteriores se ha hablado sobre el diseño e implementación tanto de software como de hardware para el prototipo domótico pero, un elemento faltante y de mucha importancia antes de poder realizar la integración del prototipo es el protocolo y sistema de comunicación.

En la sección 4.3 de diseño de hardware, se mencionaba que el sistema estará formado por un Servidor Web el cual se integrará en la Puerta de Enlace Residencial y será conectado mediante Ethernet al *Router* que proporcionará el servicio de red Wi-Fi y al cual será conectado el dispositivo móvil para la manipulación de los actuadores finales.

Para poder lograr la comunicación del prototipo desde el dispositivo móvil hasta los actuadores finales, se deberán realizar tres tipos de conexiones en dos distintos estándares de redes de comunicación. La primera conexión será la comunicación del dispositivo móvil mediante Wi-Fi hacia el *Router* o dispositivo que proveerá el servicio de esta red en el hogar. La segunda de estas conexiones, será mediante un cable

Ethernet del *Router* hacia el Ethernet Shield. La tercera y última conexión, será la que se llevará a cabo entre el dispositivo coordinador y los actuadores finales XBee. Así, una vez definidos los tipos de conexión, se deberá definir la forma en que, a través de estas conexiones, se estará enviando la información y los protocolos y formas de comunicación que su utilizaran.

Primeramente, para la comunicación entre el dispositivo móvil y el servidor Web Arduino, se realizará mediante solicitudes de http a través de *http request* que serán enviados desde la aplicación móvil. Dentro de estos *request* se enviará el código del dispositivo que se estará actuando, así como la variable del botón que se ha presionado para actuar la persiana. En la figura 44 se expresa gráficamente el esquema que contendrá el protocolo del *request* y que será enviado al servidor Web.

Figura 45. Contenido del h*ttp request* a enviar a servidor Web de la aplicación.

Una vez que el servidor Web ha recibido la petición *http request*, decodificará y validará la información contenida en el mensaje y la volverá a codificar en formato *API* para ser enviada al módulo coordinador XBee mediante una comunicación serial con el XBee *Shield*.

El formato *API* para la comunicación entre los módulos XBee coordinador y actuadores finales contendrá la información necesaria para poder actuar el motor de la persiana que se desea mover. La figura 45 muestra el formato estándar que debe contener el protocolo *API* para comunicación entre dos XBee.

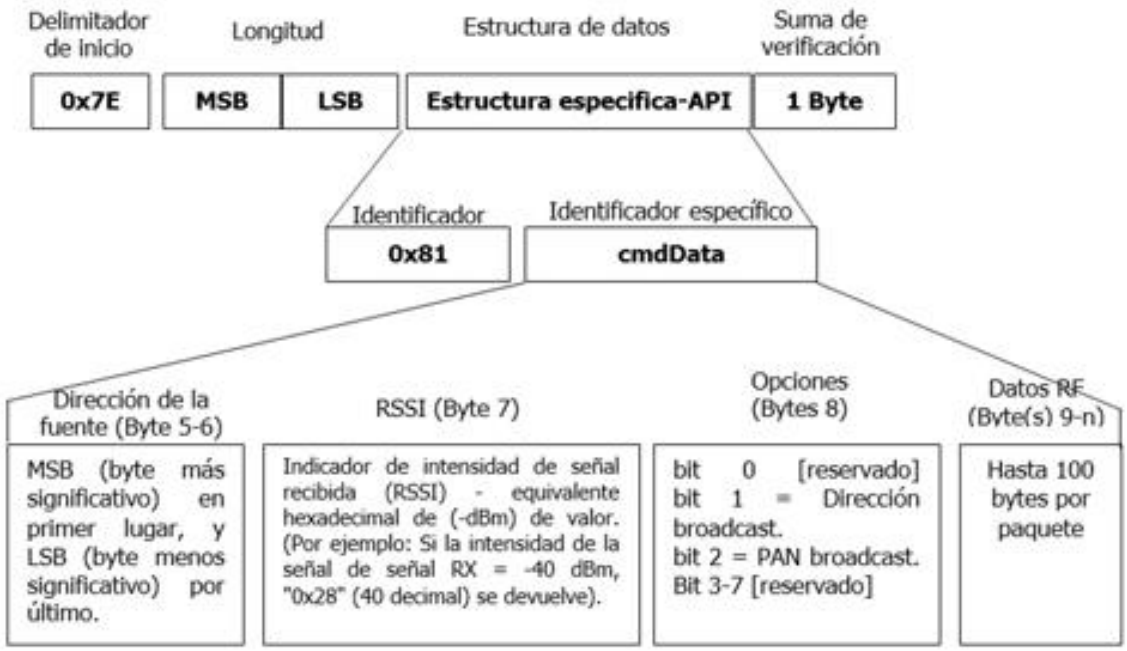


Figura 46. Estructura de datos para una cadena *API* para XBee (Digi International Inc, 2015).

Una vez definidos los protocolos de comunicación y los tipos de conexiones necesarias, se deberá realizar la integración.

4.5 Prototipo

En base a la descripción de los capítulos anteriores sobre los elementos de los cuales estará constituido el sistema domótico para control de persianas; en esta sección se describirá cada uno de éstos elementos como resultado de la investigación realizada para la constitución de un prototipo funcional.

4.5.1 Puerta de Enlace Residencial o Control Central

Éste es quizá el elemento fundamental para la comunicación. En la sección anterior, se hablaba sobre la utilización de un Arduino Uno como elemento principal de procesamiento en conjunto con un Ethernet Shield para la integración de comunicación como Servidor de Peticiones Web que estaría interpretando las peticiones provenientes de la aplicación Android del dispositivo móvil.

Además de la comunicación para protocolo Web era necesaria la comunicación con los Actuadores finales, los cuales, tendrán constituido un elemento XBee para la recepción de los comandos API y poder actuar la persiana. Es por esto, que se le agregó un XBee Shield. Como resultado final, se obtuvo un pequeño centro de procesamiento y comunicación constituido por un Arduino UNO, un Ethernet Shield y un XBee Shield, conectados todos a través de los pines de expansión de cada tarjeta. En el figura 47 se muestra el esquema de la Puerta de Enlace Residencial resultante, junto con una imagen del prototipo final en la figura 48, con vista superior en la figura 49 y una vista lateral en la figura 50.

Figura 47. Elementos finales de la Puerta de Enlace Residencial.

Figura 48. Prototipo de Puerta de Enlace Residencial.

Figura 49. Prototipo de Puerta de Enlace Residencial (vista superior)

Figura 50. Prototipo de Puerta de Enlace Residencial (vista lateral)

4.5.2 Actuador Final

Como ya se mencionó en la sección 4.3 de este mismo capítulo, el Actuador Final constará de cuatro bloques principales que permitirán el control del elemento a controlar, que para nuestro caso será el motor de la persiana, siendo estos:

- Comunicación con XBee
- Optoacoplador
- Etapa de potencia
- Motor de persiana (Corriente Alterna)

Para la comunicación con XBee se utiliza un módulo de la Serie 2 configurado como *Router AT* el cual recibirá mediante el protocolo API, las instrucciones para activar las entradas/salidas digitales. La entrada/salida 0 (cero) servirá para activar un pulso para subir la persiana. En cambio, la entrada/salida 1 (uno) hará lo contrario a la 0, activando un pulso para bajar la persiana. El XBee está alimentado por una fuente regulada de 3.3V, que también servirá para alimentar una sección del circuito optoacoplador.

El optoacoplador sirve como elemento de protección para el componente crítico y más importante en el Actuador Final, es decir, el XBee. El *opto* (como comúnmente se le denomina), es un circuito que cuenta internamente con un diodo emisor de luz y un fototransistor que ayudan al aislamiento o separación entre elementos sensibles y componentes de potencia. Básicamente el opto, separa el dispositivo XBee del circuito de potencia para evitar daños en caso de algún problema en la parte de control. El componente elegido es un circuito 4N25. En la figura 51 se muestra el diagrama esquemático de la distribución de las terminales.

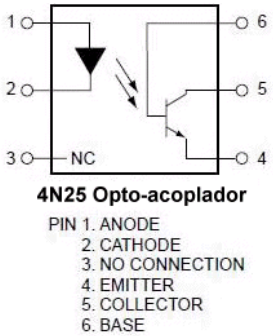


Figura 51. Diagrama esquemático de conexiones de un optoacoplador 4N25.

Debido a que se utilizaran dos entradas/salidas (I/O) del XBee, es necesaria la implementación de un optoacoplador para cada una de éstas. Para la conexión del XBee al ánodo del LED emisor, se requiere la interconexión de una resistencia limitadora de corriente para no dañar a este elemento dentro del optoacoplador. Para activar el LED, basta con activar la terminal I/O con un pulso en alto. Del lado del fototransistor, se requiere la alimentación de una fuente de voltaje diferente a la que alimenta al XBee. Esta segunda fuente será la responsable de proveer alimentación a la parte del fototransitor, así como a los elementos de potencia.

En la figura 52 se muestra el diagrama esquemático de la conexión eléctrica del optoacoplador junto con el XBee.

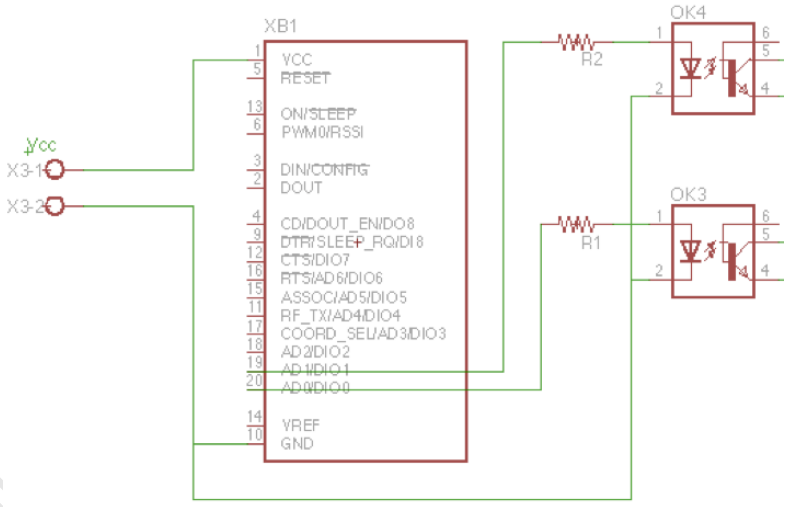


Figura 52. Diagrama esquemático de conexión eléctrica de optoacoplador con XBee.

Para la etapa de potencia, se utilizó un par de relevadores, uno de ellos para activar el motor de la persiana para subirla y el otro para bajarla. Para el prototipo se optó por utilizar un voltaje de 3.3 V en la bobina de los relevadores.

Para activar los relevadores, se tomó la salida proveniente del colector del fototransistor del optoacoplador adaptándola a través de un transistor de

acoplamiento hacia la bobina del relevador, a la cual también se le integra un diodo para eliminar los efectos inductivos de esta.

En la figura 53 se muestra el diagrama de la conexión eléctrica del optoacoplador junto con los relevadores.

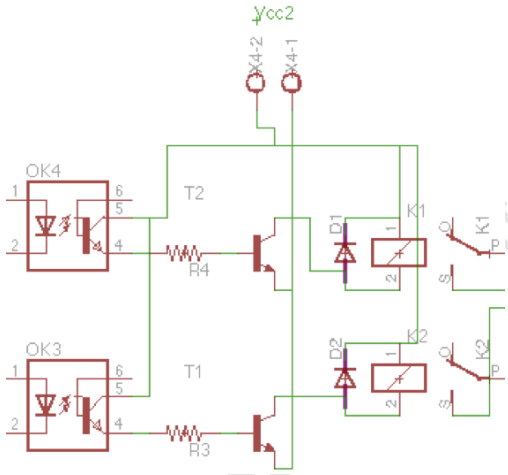


Figura 53. Diagrama esquemático de conexión eléctrica del optoacoplador con los relevadores.

Finalmente, para la conexión del motor de la persiana, éste requiere alimentación a 120 V de Corriente Alterna (CA) para su funcionamiento. El motor que se utilizó es de tipo tubular especialmente diseñado para la automatización de toldos y persianas en hogares y negocios. Este dispositivo consta de una estructura rígida tubular donde en uno de sus extremos se tiene la parte móvil, donde se encuentra fijo el rotor del motor y el otro extremo, hace función de un cojinete o balero para permitir el libre movimiento de la parte móvil del eje principal.

El motor tubular, como ya se mencionó en el párrafo anterior, requiere de alimentación de 120 V en corriente alterna para activarlo. Para el correcto funcionamiento de este dispositivo, se debe realizar la conexión eléctrica de acuerdo a lo especificado por el fabricante. Básicamente, se cuentan con cuatro cables a conectar: el cable verde (o

verde/amarillo) se deberá conectar a Tierra de la fuente AC, el cable azul deberá ser conectado al Neutro, el cable café y negro deben de conectarse a la toma de Línea o Activo ya que estas dos conexiones son las que permiten que el motor suba o baje. En la figura 54, se muestra el esquema con la configuración eléctrica de las terminales del motor tubular para la persiana que se presenta como prototipo.

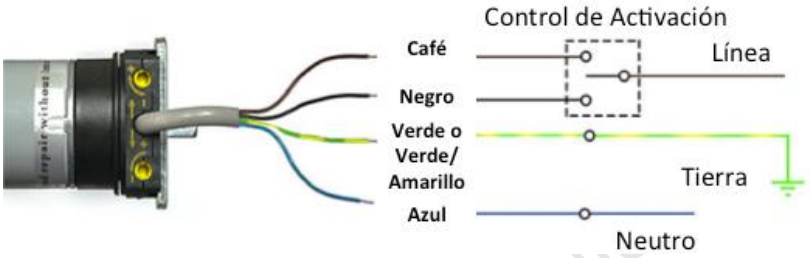


Figura 54. Esquema de conexión para motor tubular de persiana a fuente de alimentación CA.

Como se muestra en la Figura 54, en los cables café y negro del motor, se encuentran el Control de Activación, es aquí donde la etapa de potencia con los relevadores debe conectarse. Es decir, hay que conectar el cable café al circuito Normalmente Abierto (NA) del primer relevador y el cable negro al circuito NA del segundo relevador para que cuando queramos subir la persiana, se active el primer relevador y si en cambio se quiere bajar, se deberá activar el segundo relevador. En la figura 55 se muestra un diagrama esquemático de la conexión de las terminales del motor tubular con la etapa de potencia o los relevadores.

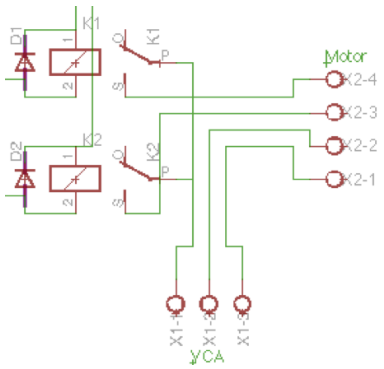


Figura 55. Diagrama de conexiones de relevadores con motor tubular.

Los elementos antes mencionado, en conjunto, conforman el Actuador Final. En la figura 56 se puede apreciar el diagrama de conexiones eléctricas de este elemento en su totalidad junto con la figura 57 donde se muestra el diseño del esquema de PCB que se trazará en las placas y la figura 58 donde se ilustra una imagen del ensamble de la tarjeta electrónica (PCBA) resultante.

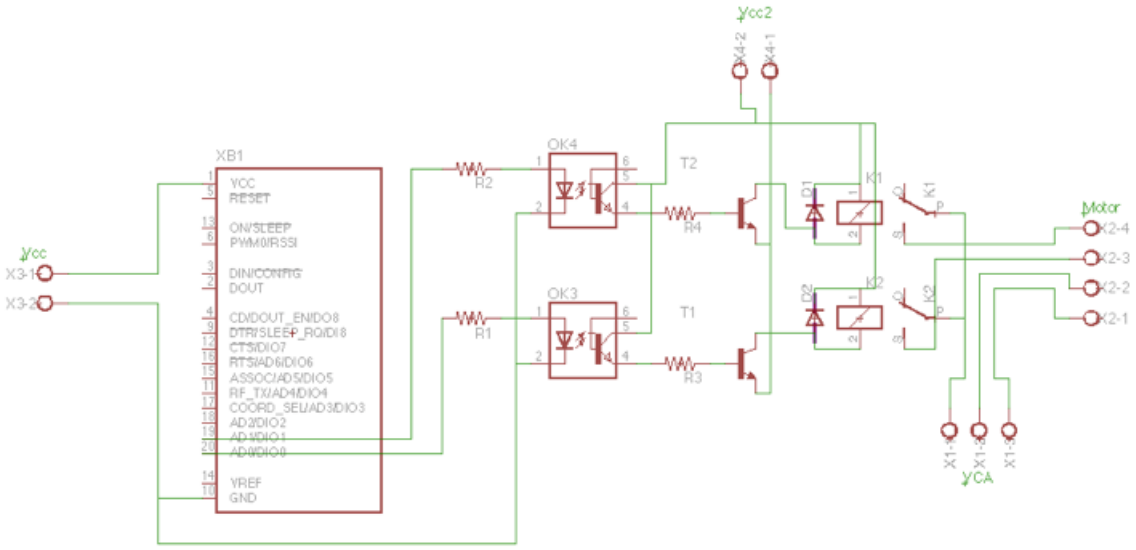


Figura 56. Diagrama esquemático de conexiones del Actuador Final.

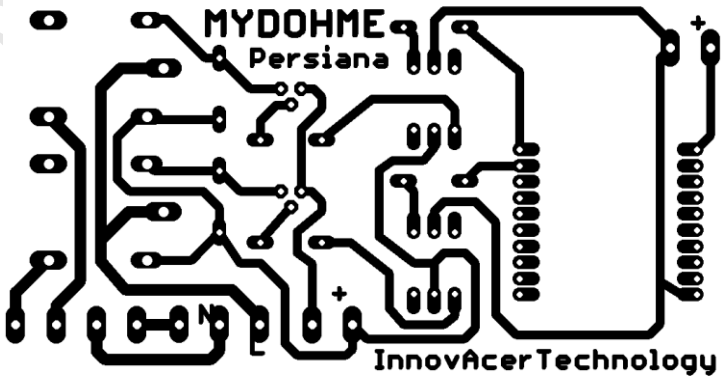


Figura 57. Diseño del esquema de PCB de Actuador Final.

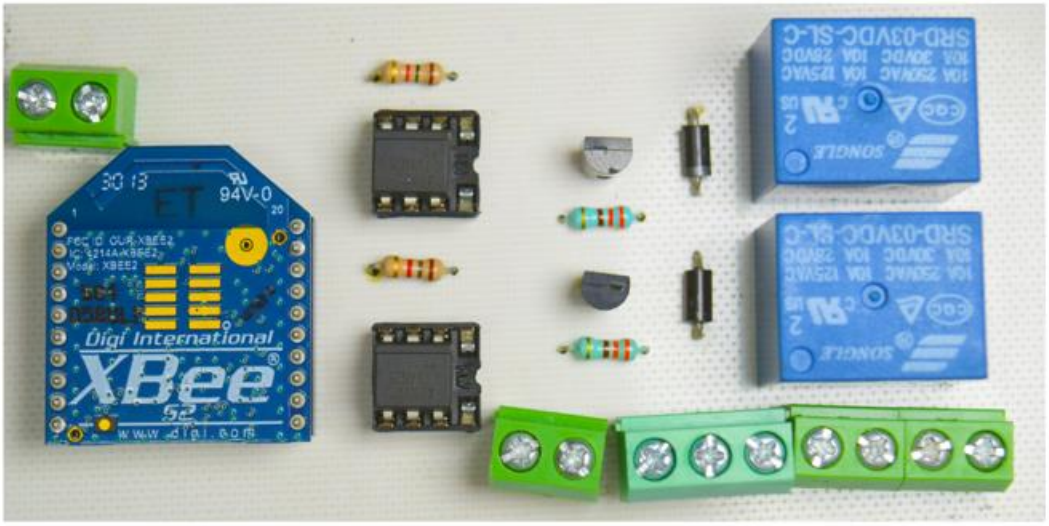


Figura 58. PCBA de Actuador Final.

Como dato adicional, para la alimentación eléctrica del Actuador Final se requiere la utilización de dos fuentes de voltaje de corriente directa, la primera de 3.3 V para alimentar al circuito XBee y la etapa emisora de los optoacopladores, la segunda, de por lo menos 3.3V para poder alimentar los transistores de acoplamiento y la bobina de los relevadores. En la figura 59 se muestran las fuentes utilizadas en el prototipo.

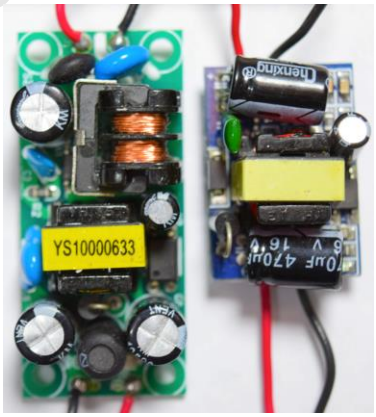


Figura 59. Fuentes de voltaje de corriente directa usadas en el Actuador Final.

4.5.3 Configuración general del prototipo

Lo anterior, en conjunto, conforman el prototipo de Sistema domótico para el Control de Persiana mediante dispositivo móvil y comunicación ZigBee. En la Figura 60 se muestra todos los elementos que conforman el prototipo junto con el motor tubular usado en la persiana en la figura 61.

Figura 60. Elementos y prototipo final.

Figura 61. Motor tubular usado en la persiana.

Capítulo 5.
Resultados

En este capítulo se exponen los resultados del trabajo de investigación. El capítulo se estructura en dos secciones. En la sección 5.1 se analizan los resultados de la prueba de consumo eléctrico. En la sección 5.2 se resumen los costos finales del prototipo.

5.1 Prueba de consumo eléctrico

Obtenido el prototipo del sistema domótico para control de persianas, se ha procedido a la realización de las pruebas de funcionamiento y mediciones de valores de consumo de corriente y potencia en el motor tubular de la persiana con la finalidad de proporcionar información que pueda complementar el dimensionamiento y la aplicación del prototipo en un entorno real en el hogar.

El procedimiento utilizado para la realización de las pruebas consistió en monitorear ciertos valores (a continuación mencionados) durante el funcionamiento del prototipo en el ciclo de subida y el ciclo de bajada del motor tubular de la persiana. Los valores a monitoreados fueron:

- Consumo de corriente del motor tubular y tiempo del ciclo de subida
- Consumo de corriente del motor tubular y tiempo del ciclo de bajada

Adicional a los dos datos monitoreados, se realizó el cálculo para 5 valores adicionales para complementar el análisis de los resultados:

- Potencia consumida durante el ciclo de subida
- Potencia consumida durante el ciclo de bajada
- Cálculo de Wh consumido durante ciclo de subida
- Cálculo de Wh consumido durante ciclo de bajada
- Estimación de costo de consumo eléctrico mensual

5.1.1 Consumo de corriente del motor tubular y tiempo de ciclo.

Para la realización de esta prueba se debe considerar que: el motor tubular para su activación es mediante corriente alterna de 120 V a 60 Hz y éste actuador consume potencia al momento de actuar ya sea en el ciclo de subida o en el ciclo de bajada. También se debe mencionar que la longitud a recorrer de la tela de la persiana es de 120 cm.

La prueba consiste en realizar 5 ciclos, tanto de subida como de bajada, para obtener los valores de tiempo y corriente consumida durante éstos periodos. La tabla 3 muestra los valores medidos durante esta prueba.

Ciclo	Corriente en subida (A)	Corriente en bajada (A)	Tiempo subiendo (seg.)	Tiempo bajando (seg.)
1	1.32	1.3	10	10
2	1.33	1.3	10	10
3	1.34	1.3	10	10
4	1.35	1.3	10	10
5	1.35	1.3	10	10

Tabla 3. Resultados de prueba de consumo de corriente y tiempos de ciclo de subida y bajada de la persiana.

En base a los resultados obtenidos, se puede determinar que, los valores promedio de consumo de corriente y el tiempo ciclo del motor tubular son los siguientes:

$$Valor\ de\ corriente\ en\ ciclo\ de\ subida\ \cong \mathbf{1.34\ A}$$
$$Valor\ de\ corriente\ en\ cilo\ de\ bajada = \mathbf{1.3\ A}$$
$$Tiempo\ ciclo = \mathbf{10\ seg}.$$

5.1.2 Potencia consumida durante el ciclo

Teniendo los valores de corriente consumida por el motor durante el ciclo de subida y bajada y conociendo que la alimentación del motor es de 120 V de corriente alterna, se pueden obtener los valores de potencia consumida los cuales son:

$$Potencia\ durante\ ciclo\ de\ subida = 120\ VCA * 1.34\ A = \mathbf{160.8\ W}$$
$$Potencia\ durante\ ciclo\ de\ bajada = 120\ VCA * 1.3A = \mathbf{156\ W}$$

5.1.3 Cálculo de Wh consumido durante el ciclo

Con base a los valores de potencia consumida obtenida y la duración de cada ciclo, se puede proceder al cálculo de los Watt-Hora (Wh) consumidos por el motor tubular de la persiana. Para esto, se debe convertir el valor del ciclo de segundos a horas, obteniendo:

$$\frac{Tiempo\ del\ ciclo\ (seg)}{Segundos\ en\ 1\ hora} = \frac{10}{3600} = \mathbf{0.0027\ hrs}.$$

Tomando el valor del tiempo del ciclo en horas, bastará con multiplicarlo por el valor de la potencia obtenida dando como resultado el valor de Watt-Hora de consumo de energía.

$$Consumo\ de\ subida = 160.8W * 0.0027 = \mathbf{0.4341\ Wh = 0.0004341\ KWh}$$
$$Consumo\ de\ bajada = 156W * 0.0027 = \mathbf{0.4212\ Wh = 0.0004212\ KWh}$$

5.1.4 Estimación de costo de consumo eléctrico mensual

Tomando como referencia el valor de la tarifa 2 de consumo eléctrico de baja tensión por KWh de la Comisión Federal de Electricidad para el mes de noviembre de 2016 (CFE, 2016; tarifa representativa para un usuario doméstico con un mediano consumo), teniendo el dato de Watt-Hora y estimando que la persiana estará funcionando un promedio de 4 veces por día, se puede determinar que el costo estimado mensual por uso de la persiana es el siguiente:

$$Costo\ mensual\ a\ la\ subida = \big((0.0004341KWh * 4) * 2.386\big) * 30 = \mathbf{0.124\ pesos}$$
$$Costo\ mensual\ a\ la\ bajada = \big((0.0004212KWh * 4) * 2.386\big) * 30 = \mathbf{0.12\ pesos}$$

$$Consumo\ mensual = Costo\ mensual\ a\ la\ subida + Costo\ mensual\ a\ la\ bajada$$
$$Costo\ mensual = 0.124 + 0.12 = \mathbf{0.244\ pesos/mensuales}$$

Realmente el costo estimado mensual de consumo eléctrico del uso del prototipo para una persiana considerando que se usa en promedio 4 veces al día sería de 0.244 pesos. Es importante recalcar que, la Puerta de Enlace Residencial será un elemento que estará funcionando las 24 horas del día y, por ende, estará generando un consumo eléctrico pero, realmente este valor se puede despreciar debido a que el valor es muy bajo.

5.2 Costo del prototipo

Se mencionaba en capítulos anteriores que el prototipo de sistema domótico también tendría la característica de que sería de bajo costo. En la tabla 4 se muestra el costo del prototipo describiendo los elementos principales que lo componen. Es importante mencionar que sólo se están describiendo los precios de los componentes como tal, no se incluyen costos de programación, instalación o mano de obra del sistema.

Elemento	Costo estimado (MXP)	Costo Total (MXP)
Puerta de Enlace Residencial (Arduino UNO + Ethernet Shield + XBee Shield + XBee S2)	$2200	$7000
Actuador final (PCBA + XBee S2 + Fuentes de alimentación)	$800	
Motor de persiana	$4000	

Tabla 4. Costo de prototipo domótico

En la tabla 5 se muestra una comparativa entre el costo del prototipo domótico contra el costo de sistemas comerciales existentes. Cabe mencionar que los costos que se muestran son estimativos y de referencia, ya que sólo se considera el precio de la Puerta de Enlace Residencial de cada sistema comercial y algún controlador para 1 persiana o 1 lámpara.

Sistema (Puerta de Enlace Residencial + Actuador Final)	Costo Estimado (MXP)
Prototipo Domótico	$7000
Control 4 Home	$13000
Somfy	$20000

Tabla 5. Comparación de costos de prototipo domótico y sistemas comerciales actuales

Capítulo 6.
Conclusiones y trabajos futuros

6.1 Conclusiones

La domótica hoy en día está jugando un papel muy importante en la vida y confort de las personas volviéndose parte integral de las necesidades y cotidianeidad en el hogar.

Hoy en día existen diversos sistemas comerciales que brindan la oportunidad de automatizar algunos elementos o servicios del hogar pero, una de las grandes limitaciones con las que cuentan es que están diseñados con tecnologías propias de la marca comercial o empresa que los distribuye, dejando cerradas las posibilidades de crecimiento a sólo los productos existentes.

La bondad con la que cuenta este prototipo de Sistema Domótico diseñado y descrito en este documento, es que cuenta con elementos de tecnología de uso libre como lo son las plataformas Arduino, respaldadas por una gran comunidad de desarrolladores y gente experta en la materia. Este tipo de plataformas están constantemente en actualización y revisión, además de que suelen tener cambios que son avalados por miles de usuarios.

Aunado al hecho de utilizar tecnologías de uso libre, cuenta con un sistema de comunicación estandarizado y diseñado para protocolos de transferencia de información en ambientes tanto industrial como dentro del hogar y que, de igual manera, cuenta con el respaldo de una Alianza avalada por más de 400 empresas a

nivel mundial que promueven el uso y la estandarización de los sistemas ZigBee para la implementación en elementos de IoT y Smart Homes.

Las tecnologías móviles están jugando un papel importante en la optimización y automatización de muchas actividades cotidianas, siendo la Domótica una de estas. Como ya se analizó, el dispositivo móvil es una parte importante dentro del prototipo de Sistema Domótico ya que puede ser considerado como el centro de mando o control para actuar los elementos del sistema.

Gracias a la diversidad y uso de estas nuevas tecnologías, el prototipo de Sistema Domótico cuenta con los elementos necesarios para competir con los sistemas de automatización en el hogar comerciales actuales teniendo la posibilidad que este puede tener un crecimiento y controlar tantos Actuadores Finales como el protocolo ZigBee lo permita.

En general, se puede determinar que en base al análisis y las investigaciones realizadas, se pudo corroborar que es posible realizar un prototipo de Sistema Domótico para control de persianas con el uso de tecnologías móviles, tecnologías de uso libre y teniendo como protocolo principal de comunicación la plataforma ZigBee.

6.2 Trabajos Futuros

El prototipo de sistema domótico en su primera etapa cuenta solamente con el control de persianas motorizadas. Debido a la gran versatilidad de los elementos que componen el sistema, la escalabilidad de éste es muy grande a tal grado que se tiene visualizado en una segunda etapa integrar un control de iluminación en el hogar. A esta segunda etapa se le podrán añadir etapas posteriores integrando la implementación de diversos sensores y actuadores para el control de riego, fugas de

gas o nivel de flujo de agua en la vivienda y, cómo no mencionar la parte de la seguridad en el hogar a través del monitoreo de cámaras y sensores que puedan elevar la calidad de vida del usuario, su confort y su nivel de seguridad.

Otro ámbito de mejora en el prototipo sería la optimización de la Pasarela Residencial o Control Central mediante la implementación de una placa Arduino Ethernet que podría propiciar la reducción de tamaño del controlador ya que se estaría eliminando la utilización del Ethernet Shield, funcionalidad integrada en la tablilla Arduino antes mencionada.

Para los actuadores Finales, se podría mejorar el diseño y tamaño de la PCBA a través de la miniaturización con componentes electrónicos de montaje superficial o SMT (Surface Mounting Technology) que prácticamente cumplirían con la misma función pero en elementos de menor tamaño.

Un área muy importante a considerar para ser mejorada es la de la aplicación móvil, para la cual, se pretende optimizar el entorno, diseño y usabilidad de la aplicación Android para extenderlo a equipos como Phablets o Tablets, así como la actualización a plataformas y versiones más recientes de acuerdo a los avances del sistema operativo. Dentro de este mismo ámbito, se pretende realizar la expansión e inclusión de la plataforma iOS para dispositivos Apple, abarcando así, dos de las principales plataformas que rigen el mundo de los móviles en la actualidad.

Bibliografía y Referencias.

Amariei, C. (2015). *Arduino Development Cookbook.* Birmingham, UK: Packt Publishing Ltd.

Barrera Durango, M., Londoño Ospina, N., Calvajal, J., & Fonseca, A. (29 de Mayo de 2012). Análsis y diseño de prototipo domótico de bajo costo. *Análsis y diseño de prototipo domótico de bajo costo*. Medellín, Colombia, Colombia.

Bayle, J. (2013). *C Programming for Arduino.* Birmingham, UK: Packt Publishing Ltd.

Bensky, A. (2004). *Short-range Wireless Communication.* Burlington, USA: Elsevier.

Blum, J. (2013). *Exploring Arduino: Tools and Techniques for Engineering Wizardry.* Indianapolis, USA: John Wiley & Sons, Inc.

Carballar Falcón, J. A. (2010). *Wi-Fi. Lo que se necesita conocer.* Madrid, España: RC Libros.

Carbou, R., Diaz, M., Exposito, E., & Roman, R. (2011). *Digital Home Networking.* Londres. Hoboken, Gran Bretaña. USA: ISTE Ltd John. Willey & Sons, Inc.

Cruz Zapata, B. (2013). *Android Studio Application Development.* Birmingham, UK: Packt Publishing.

Digi International Inc. (2015). Obtenido de Digi: http://knowledge.digi.com/articles/Knowledge_Base_Article/What-is-API-Application-Programming-Interface-Mode-and-how-does-it-work/?l=en_US&fs=Search&pn=1

Dobkin, D. M. (2005). *RF Engineering for Wireless Networks.* San Diego, USA: Elsevier.

Du, K.-L., & Swamy, M. (2010). *Wireless Communication Systems. From RF Subsystems to 4G Enabling Technologies.* New York, USA: Cambridge University Press.

Electricidad, C. F. (2016). *Comision Federal de Electricidad*. Obtenido de Tarifas Generales de baja tensión: http://app.cfe.gob.mx/Aplicaciones/CCFE/Tarifas/Tarifas/Tarifas_industria.asp?Tarifa=CMABT&Anio=2016

Evans, B. (2011). *Beginning Arduino Programming.* New York, USA: Technology in Action.

Federal, S. H. (s.f.). *México Gobierno Federal*. Obtenido de Demanda de Vivienda 2015: https://www.gob.mx/shf/documentos/demanda-de-vivienda-2015

Gallardo Vázquez, S. (2013). *Técnicas y procesos en instalaciones domóticas y automáticas.* Madrid, España: Ediciones Paraninfo.

Garg, V. (2010). *Wireless Communications & Networking.* San Francisco, USA: Elsevier Inc.

Google, I. (07 de 03 de 2016). *Dashboards* . Recuperado el 23 de 03 de 2016, de Android Platform Versions: http://developer.android.com/about/dashboards/index.html

Gupta, N. (2013). *Inside Bluetooth Low Energy.* Boston, USA: Artech House.

Guvenc, I., Gezici, S., Sahinoglu, Z., & Kozat, U. C. (2011). *Reliable communications for Short-range Wireless Systems.* New York, USA: Cambridge University Press.

Harke, W. (2010). *Domótica para viviendas y edificios.* (J. M. Moya, Trad.) Barcelona, España: Marcombo.

Harke, W. (2013). *Domótica para viviendas y edificios.* México: Alfaomega Grupo Editor, S.A. de C.V.

Hellman, E. (2014). *Android Programming. Pushing the Limits.* West Sussex, UK: John Wiley & Sons Ltd.

Huidobro Moya, J. M. (2006). *Redes y servicios de telecomunicaciones.* Madrid, España: Thomson - Paraninfo.

Huidobro Moya, J. M. (2011). *Radiocomunicaciones.* España, España: Creaciones Copyright.

Huidobro Moya, J. M., & Millan Tejedor, R. J. (2010). *Manual de Domótica.* España, España: Creaciones Copyright, S.L.

Huidrobro Moya, J. M., & Millán Tejedor, R. J. (2009). *Domótica. Edificios Inteligentes.* España: Creaciones Copyright, S.L.

J. Drake, J., Oliva Fora, P., Lanier, Z., Mulliner, C., A. Ridley, S., & Wicherski, G. (2014). *Android Hacker's Handbook.* Indianapolis, USA: Jhon Wiley & Sons, Inc.

Junestrand, S., Passaret, X., & Vázquez, D. (2005). *Domótica y Hogar Digital.* España: Thomson Ediciones Spain.

Klimczak, E. (2013). *Design for Software : A Playbook for Developers.* UK: John Wiley & Sons Ltd.

Lee, W.-M. (2011). *Beginning Android Application Development.* Indianapolis, USA: Wiley Publishing, Inc.

Lee, W.-M. (2013). *Android Application Development Cookbook: 93 Recipies for Building Winning Apps.* Indianapolis, USA: Jhon Wiley & Sons, Inc.

Maestre Torreblanca, J. M. (2015). *Domótica para Ingenieros.* Madrid, España: Ediciones Paraninfo.

Margolis, M. (2011). *Arduino Cookbook.* Sebastopol, USA: O'Reilly Media Inc.

Martín Castillo, J. C. (2009). *Instalaciones domóticas.* Madrid, España: Editorial Editex, S.A.

Martín Castillo, J. C. (2014). *Instalaciones eléctricas y domóticas.* Madrid: Editorial Editex S.A.

McRoberts, M. (2010). *Beginning Arduino.* New York, USA: Springer Science + Business Media, LLC.

Mednieks, Z., Blake Meike, G., Dornin, L., & Pan, Z. (2014). *Enterprise Android: Programming Android Database Applications for the Enterprise.* Indianapolis, USA: Jhon Wiley & Sons, Inc.

Meier, R. (2012). *Professional Android 4 Application Development.* Indianapolis, USA: John Wiley & Sons, Inc.

Monk, S. (2010). *30 Arduino Projects for the Evil Genius.* New York, USA: McGraw-Hill.

Monk, S. (2012). *Arduino + Android Projects for the Evil Genius.* New York, USA: McGraw-Hill.

Moro Vallina, M. (2011). *Instalaciones domóticas.* Madrid, España: Ediciones Paraninfo, SA.

Nussey, J. (2013). *Arduino for Dummies.* West Sussex, England: John Wiley & Sons.

Pellejero, I., Andreu, F., & Lesta, A. (2006). *Fundamentos y aplicaciones de seguridad en redes WLAN.* Barcelon, España: Marcombo S.A.

Pérez Pérez, V. R. (1 de Noviembre de 2010). Contribución al Diseño de Sistemas Domóticos y de entretenimiento utilizando hardware libre y software de cógigo abierto. Tijuana, Baja California, México.

Rackley, S. (2011). *Wireless Networking Technology.* Oxford, UK.

Romero, C., Vázquez, F., & de Castro, C. (2011). *Domótica e inmótica. Viviendas y edificios inteligentes.* México: Alfaomega Grupo Editor, S.A. de C.V.

Simon, M. (2012). *Arduino + Android Projects for the Evil Genius.* New York, USA: McGraw-Hill.

Torrente Artero, Ó. (2013). *ARDUINO. Curso práctico de formación.* San Fernando de Henares, Madrid: RC Libros.

Valentin Labarta, J. L. (2012). *Instalaciones Solares Fotovoltaicas.* Navarra, España: Editorial Donostiarra, S.A.

Wallace, J. (2014). *Android App for Absolutely Beginners.* USA, USA: Apress.

Wei, J. (2012). *Android Database Programming.* Birmingham, UK: Packt Publishing.

Wesoloswski, K. (2009). *Introduction to Digital Communication Systems.* West Sussex, United Kingdom: John Wiley and Sons.

Wilson, J. (2013). *Creating Dinamic UI with Android Fragments.* Birmingham, UK: Packt Publishing Ltd.

Anexos

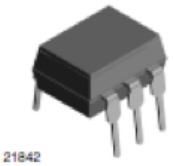
a) Datasheet Optoacoplador 4N25

4N25, 4N26, 4N27, 4N28

Vishay Semiconductors

VISHAY

Optocoupler, Phototransistor Output, with Base Connection



21842

FEATURES

- Isolation test voltage 5000 V_{RMS}
- Interfaces with common logic families
- Input-output coupling capacitance < 0.5 pF
- Industry standard dual-in-line 6 pin package
- Compliant to RoHS directive 2002/95/EC and in accordance to WEEE 2002/96/EC

e3

RoHS COMPLIANT

APPLICATIONS

- AC mains detection
- Reed relay driving
- Switch mode power supply feedback
- Telephone ring detection
- Logic ground isolation
- Logic coupling with high frequency noise rejection

AGENCY APPROVALS

- UL1577, file no. E52744
- BSI: EN 60065:2002, EN 60950:2000
- FIMKO: EN 60950, EN 60065, EN 60335

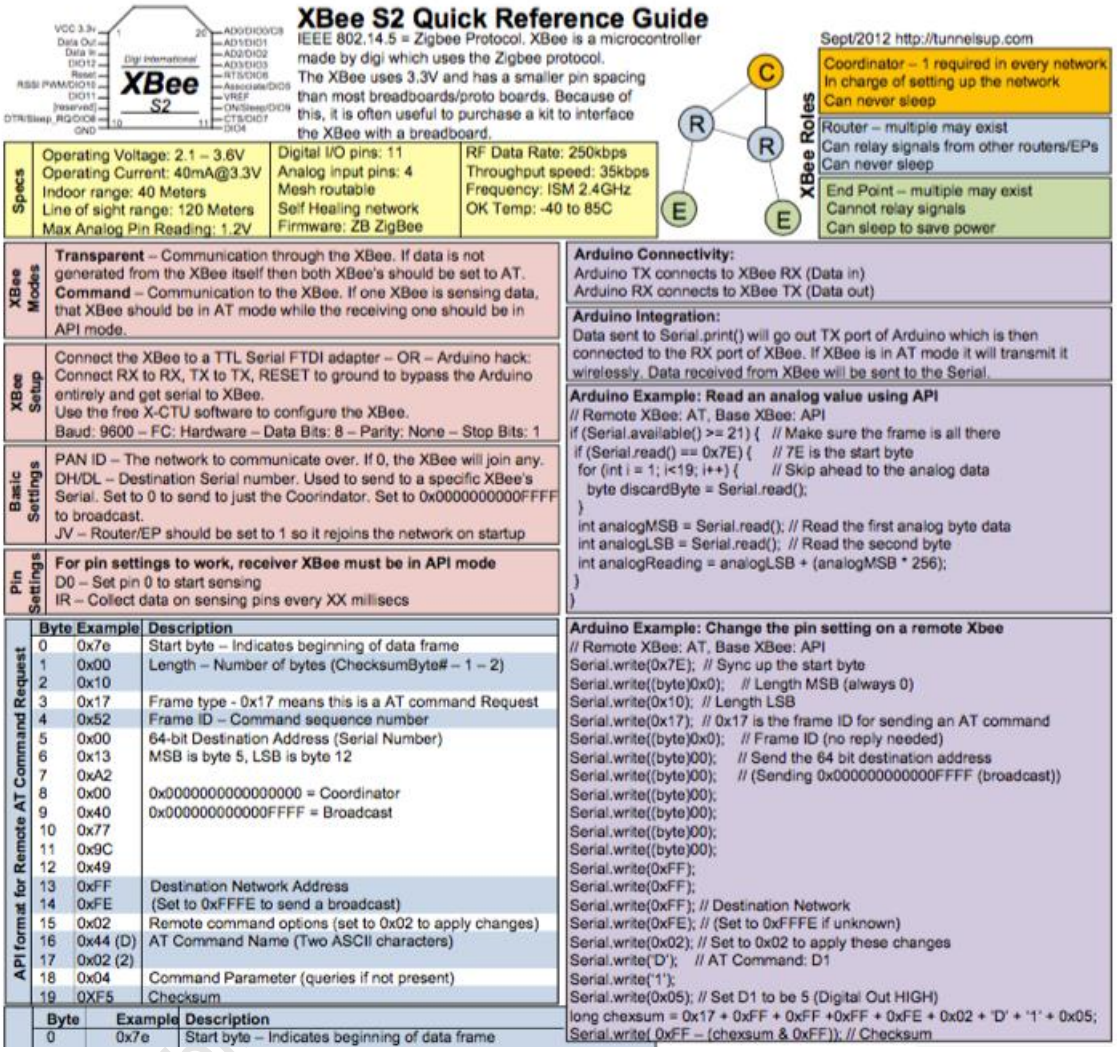
DESCRIPTION

The 4N25 family is an industry standard single channel phototransistor coupler. This family includes the 4N25, 4N26, 4N27, 4N28. Each optocoupler consists of gallium arsenide infrared LED and a silicon NPN phototransistor.

ORDER INFORMATION	
PART	REMARKS
4N25	CTR > 20 %, DIP-6
4N26	CTR > 20 %, DIP-6
4N27	CTR > 10 %, DIP-6
4N28	CTR > 10 %, DIP-6

ABSOLUTE MAXIMUM RATINGS (1)				
PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
INPUT				
Reverse voltage		V_R	5	V
Forward current		I_F	60	mA
Surge current	t ≤ 10 µs	I_{FSM}	3	A
Power dissipation		P_{diss}	100	mW
OUTPUT				
Collector emitter breakdown voltage		V_{CEO}	70	V
Emitter base breakdown voltage		V_{EBO}	7	V
Collector current		I_C	50	mA
	t ≤ 1 ms	I_C	100	mA
Power dissipation		P_{diss}	150	mW

b) Referencia programación API para ZigBee



XBee S2 Quick Reference Guide

IEEE 802.14.5 = Zigbee Protocol. XBee is a microcontroller made by digi which uses the Zigbee protocol. The XBee uses 3.3V and has a smaller pin spacing than most breadboards/proto boards. Because of this, it is often useful to purchase a kit to interface the XBee with a breadboard.

Sept/2012 http://tunnelsup.com

XBee Roles

- Coordinator – 1 required in every network; In charge of setting up the network; Can never sleep
- Router – multiple may exist; Can relay signals from other routers/EPs; Can never sleep
- End Point – multiple may exist; Cannot relay signals; Can sleep to save power

Specs

Operating Voltage: 2.1 – 3.6V	Digital I/O pins: 11	RF Data Rate: 250kbps
Operating Current: 40mA@3.3V	Analog input pins: 4	Throughput speed: 35kbps
Indoor range: 40 Meters	Mesh routable	Frequency: ISM 2.4GHz
Line of sight range: 120 Meters	Self Healing network	OK Temp: -40 to 85C
Max Analog Pin Reading: 1.2V	Firmware: ZB ZigBee	

XBee Modes

Transparent – Communication through the XBee. If data is not generated from the XBee itself then both XBee's should be set to AT.
Command – Communication to the XBee. If one XBee is sensing data, that XBee should be in AT mode while the receiving one should be in API mode.

XBee Setup

Connect the XBee to a TTL Serial FTDI adapter – OR – Arduino hack: Connect RX to RX, TX to TX, RESET to ground to bypass the Arduino entirely and get serial to XBee.
Use the free X-CTU software to configure the XBee.
Baud: 9600 – FC: Hardware – Data Bits: 8 – Parity: None – Stop Bits: 1

Basic Settings

PAN ID – The network to communicate over. If 0, the XBee will join any.
DH/DL – Destination Serial number. Used to send to a specific XBee's Serial. Set to 0 to send to just the Coorindator. Set to 0x000000000FFFF to broadcast.
JV – Router/EP should be set to 1 so it rejoins the network on startup

Pin Settings

For pin settings to work, receiver XBee must be in API mode
D0 – Set pin 0 to start sensing
IR – Collect data on sensing pins every XX millisecs

API format for Remote AT Command Request

Byte	Example	Description
0	0x7e	Start byte – Indicates beginning of data frame
1	0x00	Length – Number of bytes (ChecksumByte# – 1 – 2)
2	0x10	
3	0x17	Frame type - 0x17 means this is a AT command Request
4	0x52	Frame ID – Command sequence number
5	0x00	64-bit Destination Address (Serial Number)
6	0x13	MSB is byte 5, LSB is byte 12
7	0xA2	
8	0x00	0x0000000000000000 = Coordinator
9	0x40	0x000000000000FFFF = Broadcast
10	0x77	
11	0x9C	
12	0x49	
13	0xFF	Destination Network Address
14	0xFE	(Set to 0xFFFE to send a broadcast)
15	0x02	Remote command options (set to 0x02 to apply changes)
16	0x44 (D)	AT Command Name (Two ASCII characters)
17	0x02 (2)	
18	0x04	Command Parameter (queries if not present)
19	0XF5	Checksum

Byte	Example	Description
0	0x7e	Start byte – Indicates beginning of data frame

Arduino Connectivity:
Arduino TX connects to XBee RX (Data in)
Arduino RX connects to XBee TX (Data out)

Arduino Integration:
Data sent to Serial.print() will go out TX port of Arduino which is then connected to the RX port of XBee. If XBee is in AT mode it will transmit it wirelessly. Data received from XBee will be sent to the Serial.

Arduino Example: Read an analog value using API

```
// Remote XBee: AT, Base XBee: API
if (Serial.available() >= 21) {   // Make sure the frame is all there
 if (Serial.read() == 0x7E) {     // 7E is the start byte
  for (int i = 1; i<19; i++) {    // Skip ahead to the analog data
   byte discardByte = Serial.read();
  }
  int analogMSB = Serial.read(); // Read the first analog byte data
  int analogLSB = Serial.read(); // Read the second byte
  int analogReading = analogLSB + (analogMSB * 256);
 }
}
```

Arduino Example: Change the pin setting on a remote Xbee

```
// Remote XBee: AT, Base XBee: API
Serial.write(0x7E);  // Sync up the start byte
Serial.write((byte)0x0);    // Length MSB (always 0)
Serial.write(0x10);  // Length LSB
Serial.write(0x17);  // 0x17 is the frame ID for sending an AT command
Serial.write((byte)0x0);    // Frame ID (no reply needed)
Serial.write((byte)00);     // Send the 64 bit destination address
Serial.write((byte)00);     // (Sending 0x000000000000FFFF (broadcast))
Serial.write((byte)00);
Serial.write((byte)00);
Serial.write((byte)00);
Serial.write((byte)00);
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); // Destination Network
Serial.write(0xFE); // (Set to 0xFFFE if unknown)
Serial.write(0x02); // Set to 0x02 to apply these changes
Serial.write('D');    // AT Command: D1
Serial.write('1');
Serial.write(0x05); // Set D1 to be 5 (Digital Out HIGH)
long chexsum = 0x17 + 0xFF + 0xFF +0xFF + 0xFE + 0x02 + 'D' + '1' + 0x05;
Serial.write( 0xFF – (chexsum & 0xFF)); // Checksum
```