Instituto Universitario Aeronáutico
Departamento de Electrónica y Telecomunicaciones

Trabajo Final de Grado

Sistema Inteligente para Cocheras

Vega Romero, Pedro Martín
D.N.I.:31.037.724
Ingeniería en Telecomunicaciones

Supervisado por
Esp. Ing. José María Ducloux

Marzo, 2017

Este trabajo final de grado está dedicado a mis padres María Elena y Pedro, que gracias a su amor, trabajo y sacrificio en todos estos años, he logrado llegar hasta aquí.

Agradecimientos

A mi familia que me brindó su apoyo incondicional en todo momento para concretar este objetivo. También a mis amigos y compañeros, quienes me alentaron a concluir con esta meta. Finalmente dar las gracias al Esp. Ing. José María Ducloux, por la orientación y ayuda que me brindó para la realización de este trabajo y a toda la comunidad de profesores del IUA quienes forjaron mi formación durante tantos años.

Resumen

El objetivo de este trabajo final de grado es desarrollar un sistema capaz de automatizar el funcionamiento de cocheras para estacionamiento mediante la detección de lugares disponibles, registro de la actividad y conexión a Internet para brindar un servicio innovador a sus usuarios y potenciales clientes.

Con esta aplicación se busca dar un valor agregado a los servicios de un estacionamiento, ofreciendo al usuario información sobre las plazas disponibles y la localización de las mismas en el lugar, así como la posibilidad de saber el tiempo medido de estacionamiento que se encuentra su vehículo. Al titular de la explotación se le brindará un servicio adicional que le permita tener un mejor control y registro de la utilización de sus instalaciones.

El sistema estará conectado a Internet y, bajo la filosofía de IoT, será capaz de mostrar información sobre disponibilidad, cantidad de ocupaciones, entre otros datos de interés al propietario y datos con menores detalles al posible usuario (tarifas, lugares libres, entre otros).

Para llevar a cabo esta prueba de concepto, se diseñará e implementará una solución que permita tanto a los usuarios finales como operadores acceder a través de un portal web a información relevante sobre el estado de la cochera y sus lugares para falicitar la toma de decisiones.

Este trabajo trata aspectos vistos a lo largo de toda la carrera, Ingeniería en Telecomunicaciones, como por ejemplo la comunicación entre diferentes medios de transmisión, diseño de redes, uso de protocolos, funcionalidades y uso de microcontroladores, el uso de actuadores y sensores, y la integración de estos componentes como un sistema único.

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

Investigación

1.1. Introducción

En un mundo conectado casi 3.500 millones de personas se conectan a Internet [1], comparten información y se comunican a través de blogs, redes sociales y muchos otros medios más. Además de ello, con "Internet de todas las cosas" los objetos se unen a la conversación de un mercado de miles de millones de dólares. Este concepto, mundialmente conocido como Internet de las Cosas (*IoT*, *Internet of things*), consiste en que los objetos puedan conectarse a Internet en cualquier momento y lugar.

IoT es una realidad muy presente que está evolucionando constantemente, donde millones de dispositivos están siendo conectados entre sí a través de distintas redes de comunicación. Pequeños sensores permiten medir desde la temperatura de una habitación hasta el tráfico de taxis en una ciudad. A diario, cámaras de vigilancia velan por la seguridad en los edificios y los paneles de una estación indican el tiempo que falta hasta la llegada del siguiente tren. Incluso en las multas de tráfico existe poca intervención humana. Cada vez más objetos están siendo integrados con sensores, ganando capacidad de comunicación, y con ello las barreras que separan el mundo real del virtual se difuminan.

En las grandes ciudades y con el aumento de la cantidad de vehículos, el tráfico de automóviles y su estacionamiento se han convertido en problemas considerables que afectan a la vida de las personas. En la ciudad de Córdoba circulan alrededor de 1.380.000 vehículos autorizados [2], registrándose un incremento de ese número año a año. La búsqueda de un lugar para estacionar implica una cantidad de esfuerzo y tiempo muy importantes.

Muchos usuarios prefieren por seguridad el uso de una cochera de alquiler temporario. Habitualmente, un automóvil ingresa a una cochera, y en ese momento el conductor necesita encontrar un lugar libre para estacionar su automóvil. Este no es un trabajo sencillo, la gran cantidad de lugares de estos establecimientos y la creciente movilidad de los mismos hacen de ésta una tarea complicada.

Ante esta problemática y valiéndose del concepto de la *IoT*, surge como motivación el desarrollar una prueba de concepto de un sistema inteligente para cocheras, teniendo el objetivo de capturar, procesar, exponer y compartir información con el resto de la comunidad a través de Internet.

El propósito de este trabajo final de grado es diseñar e implementar un sistema capaz de detectar y notificar la disponibilidad de los lugares en una cochera de manera automática, recolectar y registrar los datos para su posterior procesamiento, facilitar el

acceso a la información y la toma de decisiones. Para el desarrollo del mismo, se utilizan sensores de ultrasonido y diferentes plataformas de sistemas embebidos, de hardware y software libre, conectados localmente a un servidor central a través de Internet. Diferentes usuarios podrán acceder al servidor mediante una página web y/o una aplicación móvil para conocer remotamente la cantidad de lugares disponibles o el tiempo de estadía de su vehículo en el establecimiento, y localmente se podrá conocer la cantidad de lugares libres por nivel, mientras que el operario local podrá realizar un control de este establecimiento.

El resto del documento se organiza de la siguiente manera. En el capítulo 2 se incluyen aspectos teóricos sobre la *IoT*, conceptos de interés para facilitar el entendimiento, elementos y tecnologías involucradas. El diseño del hardware y sus funcionalidades se encuentran en el capítulo 3. El capítulo 4 muestra el sistema completo y como se implementó tanto a nivel firmware como software. Los test realizados para probar el funcionamiento del sistema se plantean en el capítulo 5, mientras que el montaje en placas de los elementos del sistema son mostrados en el capítulo 6. Los resultados obtenidos se plantean en el capítulo 7 y, por último, en el capítulo 8 se comentan los logros, posibles mejoras y trabajos a futuros.

1.2. Problema a resolver

La búsqueda de un lugar para estacionar en las grandes ciudades se ha convertido en una complicación debido a la cantidad de automóviles en circulación. Sumado a ello, ubicar una cochera de estacionamiento temporario y saber si existen lugares disponibles o no en ella, es uno de los problemas a resolver.

Una cochera de estacionamiento temporario cuenta con un gran número de plazas de estacionamiento, distribuidos en diferentes niveles, el encontrar un lugar disponible suele ser una experiencia agotadora. Recorrer niveles en busca de un lugar para estacionar, conlleva un consumo mayor de combustible, pérdida de tiempo, mayor cantidad de emisiones de gases contaminantes además del estrés que somete al usuario durante esta búsqueda.

Disponer de información relevante para facilitar la toma de decisiones, en estos establecimientos es de suma importancia, tanto para los usuarios como para los responsables de la explotación. La posibilidad de conocer la cantidad de plazas disponibles, así como también el tiempo de estadía de un vehículo, serían de suma utilidad.

1.3. Productos existentes

1.3.1. Prodelux – Sistema de estacionamiento guiado

Prodelux [3] es una empresa argentina, que ofrece un sistema de estacionamiento guiado compuesto de sensores, indicadores, carteles y un software de gestión, habiendo sido ya instalado en varias cocheras de la provincia de Buenos Aires, en lugares como supermercados, centros comerciales, edificios corporativos, hoteles, etc.

El sistema está compuesto de un sensor de ultrasonido quien se encarga de reconocer el estado del lugar, un indicador LED separado el cual enciende conforme al estado del lugar,

que mediante un bus de comunicación RS-485 y luego adaptado a TCP-IP, desemboca en un servidor donde se ejecuta el software de gestión.

Este sistema no permite ser consultado en cuanto a sus lugares disponibles, ni tiempos de estadía, no está integrado a la *IoT*.

1.3.2. Keytop – Sistema de estacionamiento guiado

KEYTOP [4] es una empresa de China, que ofrece un sistema de estacionamiento guiado conformado por elementos diferenciados y un software de gestión. Entre los elementos diferenciados se encuentran, un detector ultrasónico, indicador LED, nodo controlador, controlador central, display de LEDs para interiores y display de LEDs para exteriores.

Esta solución no se encuentra integrada a la *IoT*, por lo que sólo es de utilidad para ubicar lugares de manera local.

1.4. Solución propuesta

En la Figura 1.1 se muestra el diagrama de bloques del sistema propuesto.

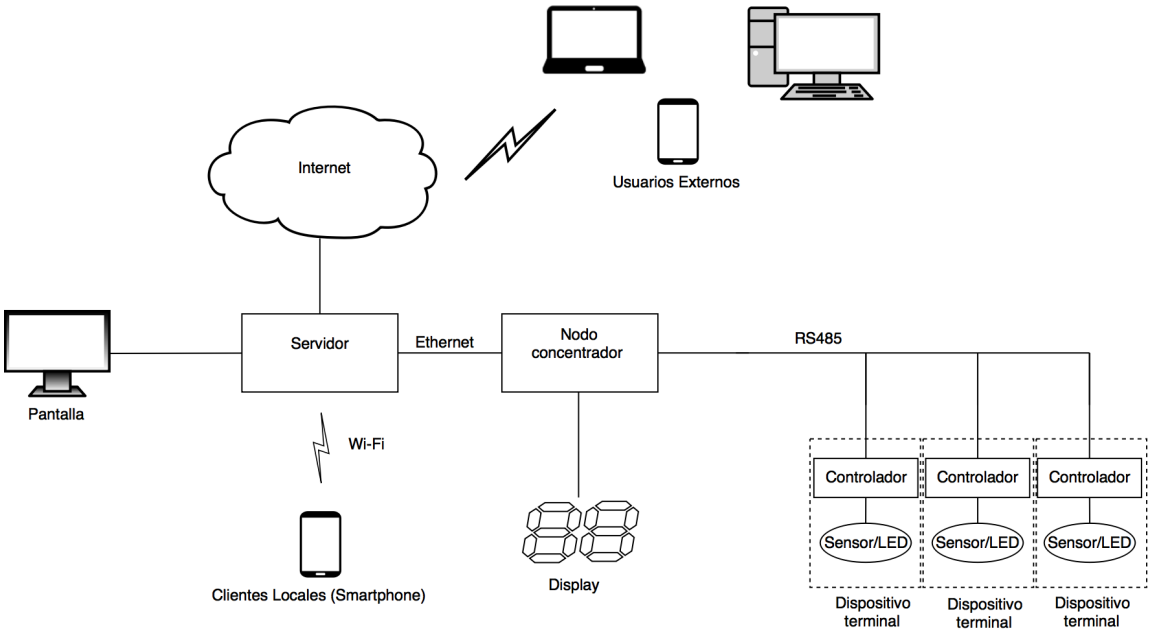


Figura 1.1: Diagrama de bloques del sistema propuesto.

Se realizará una prueba de concepto de un sistema inteligente para cocheras. El sistema estará compuesto, inicialmente, por un dispositivo terminal, un nodo concentrador y un servidor.

El dispositivo terminal será el encargado de la captura de datos por parte de los sensores de ultrasonido [5],[6]. Estos datos viajarán vía RS-485 [7] a un nodo concentrador [8], el cual cumplirá las funciones de procesar y enviar los datos obtenidos vía Ethernet al servidor [9], actualizar información de displays e interconectar los diferentes dispositivos terminales con el fin de permitir una mayor escalabilidad al sistema.

El servidor se encargará de procesar los datos permitiendo su visualización a través de una pantalla e inalámbricamente dentro del establecimiento por medio de una página web.

En este trabajo se diseñarán e implementarán los dispositivos terminales, concentrador y servidor con tecnologías basadas en sistemas embebidos. El software de aplicación que ejecutarán los dispositivos terminales [10], concentrador y servidor [11] serán de desarrollados en su totalidad en este trabajo. Para la implementación de los prototipos de dispositivos terminales y el concentrador se propone la utilización de Arduino [8], mientras que para el servidor se utilizará Raspberry Pi [9].

El hecho de disponer de la información en un servidor central a través de Internet, ofrece la posibilidad de poder ser consultada y representa una red de *IoT*. La posibilidad que ofrece *IoT* para que todos, personas y cosas, estén permanentemente conectados y se pueda recibir y procesar información en tiempo real, conduce a nuevos modos de toma de decisiones basados en esa disponibilidad de información. Estos cambios en los modelos de producción y consumo modifican las relaciones existentes entre todos los agentes del sistema. Se abre así la oportunidad de diseñar y, ofrecer un nuevo producto o servicios y explotar más eficientemente activos existentes, lo que crea un terreno fértil para emprender [12]. De esta manera un potencial usuario podrá consultar ante el interés si es que dispone de lugares libres en un estacionamiento, una vez en el mismo podrá saber qué posiciones se encuentran libres para utilizar con tan sólo entrar, y al salir poder consultar el tiempo de estadía medido en cualquier momento.

Capítulo 2

Marco teórico

En este capítulo se establece el contexto para el resto de los conceptos e ideas que se desarrollarán a lo largo del texto.

Introduciendo el concepto de la Internet de las cosas y el impacto, los principios de libertad de software y hardware libre en los que se basará esta prueba de concepto, las aplicaciones utilizadas, haciendo énfasis en las que posibilitaron el desarrollo del trabajo y una descripción de los protocolos utilizados para la comunicación así como también del hardware empleado.

2.1. Internet de las cosas

La Internet de las cosas consiste en que las cosas tengan conexión a Internet en cualquier momento y lugar. En un sentido más técnico, consiste en la integración de sensores y dispositivos en objetos cotidianos que quedan conectados a Internet a través de redes fijas e inalámbricas. El hecho de que Internet esté presente al mismo tiempo en todas partes permite que la adopción masiva de esta tecnología sea más factible.

Este concepto, mundialmente conocido como "Internet de las Cosas" (*IoT*, *The Internet of things*), consiste en que tanto personas como objetos puedan conectarse a Internet en cualquier momento y lugar. Tan simple como eso. Sin embargo, la sencillez de su definición no debe cegar la complejidad de sus implicaciones.

El término *IoT*, se le atribuye al Auto-ID Center del Instituto Tecnológico de Massachusetts (MIT) a finales de los años noventa. Sin embargo, la idea de la *IoT* ha tomado relevancia práctica gracias a la rápida evolución de la electrónica durante la última década. Esta evolución ha seguido el patrón marcado por el visionario Gordon Moore, cofundador del fabricante de microprocesadores Intel. Moore formuló su famosa predicción, conocida a nivel mundial como "Ley de Moore", en 1965, refinándola en 1975. En ella establece que el número de transistores que contiene un chip se duplica cada dos años aproximadamente. Bien sea porque Moore fue capaz de predecir el futuro o porque los fabricantes de procesadores fijaron sus palabras como un objetivo a largo plazo, la Ley de Moore se ha venido cumpliendo durante los últimos cuarenta años. Funciones que décadas atrás requerían de una computadora del tamaño de una habitación son hoy día realizadas con facilidad por simples dispositivos electrónicos del tamaño de una gota de agua. El tamaño, el costo y el consumo de energía del hardware se han reducido drásticamente, por lo que ahora es posible fabricar dispositivos electrónicos diminutos a un costo muy reducido. Estos pequeños dispositivos, junto con la expansión de las redes de comunicación, permiten incorporar inteligencia y conexión a los objetos del mundo real y están transformando lo que era una red global de personas en una red global de "todas las cosas".

2.1.1. Internet de las cosas como un sistema nervioso mundial

El Dr. Paul Horn, científico, rector de la Universidad de New York, ex vicepresidente senior de IBM y experto del Future Trends Forum, afirma que el mundo está siendo instrumentado e interconectado, a la vez que se vuelve más inteligente. Los objetos que forman parte de la vida cotidiana siempre han generado gran cantidad de información, pero esa información estaba fuera de nuestro alcance. Con la *IoT*, pequeños sensores están siendo integrados en los objetos del mundo real y son instrumentos que proporcionan información de prácticamente todo lo que es posible medir. De esta manera, cada vez se está más interconectado y las personas y objetos pueden interactuar de manera completamente distinta.

La Figura 2.1 muestra la integración de la *IoT* al mundo real.

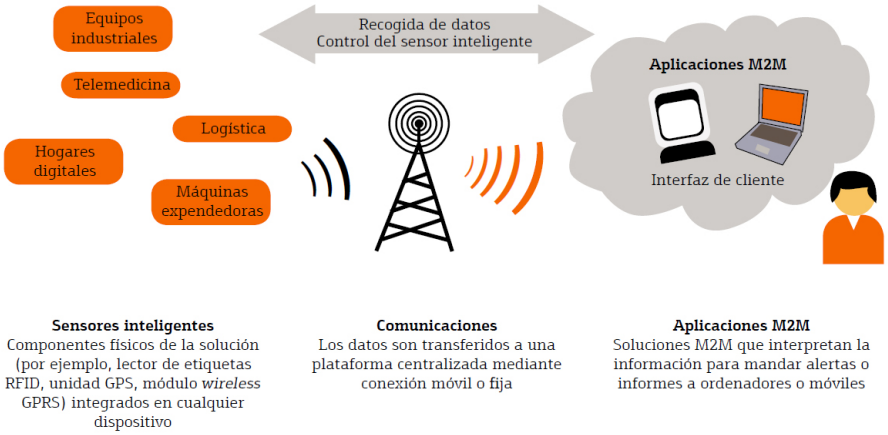


Figura 2.1: Cuando las cosas se vuelven inteligentes. Fuente: Accenture (abril 2009).

A su alrededor, se construyen entornos "inteligentes" capaces de analizar, diagnosticar y ejecutar funciones, eliminando posibles errores humanos, para bien y para mal. Por ejemplo, una red eléctrica "inteligente" es capaz de detectar sobretensiones y de dirigir la electricidad por caminos alternativos para minimizar cortes eléctricos.

2.1.2. El desafío de convertir los datos en conocimiento

Albert Einstein dijo una vez: "La información no es conocimiento". Si bien es verdad que hay más información al alcance de la que jamás se podría imaginar, a Einstein no le faltaba razón, porque separar el grano de la paja se presenta cada vez más complicado. Sólo el 5 % de la información que se crea está "estructurada", es decir, se encuentra en un formato estándar de palabras o números interpretables por computadoras. La *IoT* implica que todo objeto puede ser una fuente de datos y que el "comportamiento" de todo puede ser monitorizado a través del tiempo y el espacio. Por ello, el mundo va a estar repleto de información nueva al alcance de todos. Ante esta cantidad creciente de información, no es de extrañar que empresas y emprendedores se encuentren en una carrera por innovar en términos de almacenamiento, velocidad, acceso y métodos de análisis de datos.

Dada la importancia de procesar toda la información, el almacenamiento y la velocidad de búsqueda no constituyen los únicos retos. La habilidad para analizar mucha información en tiempo real es fundamental para las organizaciones, así como las oportunidades de mercado que puedan surgir para empresas que sean capaces de ofrecer este servicio. El nivel más básico son los "datos". Al dotar de contexto a un conjunto de datos, se obtiene "información". Esta información sólo será "conocimiento" si se sabe cómo utilizarlo. Por último, la "sabiduría" responde a por qué se está utilizando.

No cabe duda de que las empresas comienzan a invertir en servicios "inteligentes", además de las partidas más habituales de tecnologías de la información. Cada vez recurren más a proveedores externos que ofrecen soluciones potentes basadas en centros de servicios compartidos que permiten a las empresas dedicarse al *core* de su negocio, dejando a los proveedores actuar como "agregadores" de aplicaciones e infraestructuras.

Una vez establecidas las bases de la *IoT* y tras haber entendido cómo se va a instrumentar un mundo ubicuo donde personas, objetos y máquinas tienen la posibilidad de

interactuar traspasando las barreras del tiempo y espacio, a continuación se explorará el estado actual y futuro de las tecnologías.

La consultora Bosh, basándose en una base de datos de predicción, espera que haya aproximadamente 14 mil millones de dispositivos conectados en todo el mundo a finales del año 2022 [13], desde un punto de partida de más de 2 mil millones a finales de 2013. Anticipando que la mayoría de los dispositivos conectados a la *IoT* en 2022 se concentrarán en cuatro industrias: edificios inteligentes, automotriz, sanidad y servicios públicos.

Mientras que la velocidad de adopción de la *IoT* en las distintas industrias por continentes queda plasmado en la Figura 2.2.

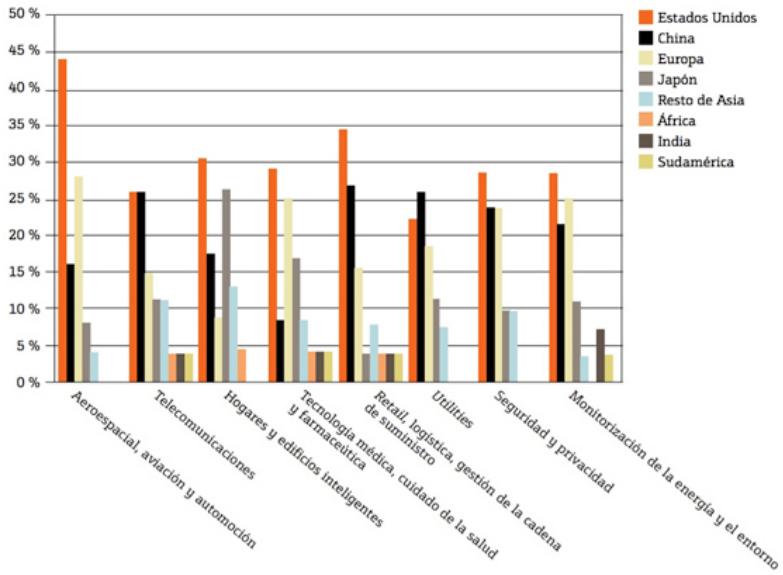


Figura 2.2: Velocidad de adopción del Internet de las cosas en las distintas industrias. Fuente: Reporte Digital [14].

2.1.3. Situación actual de una tecnología prometedora

Del mismo modo que el siglo XIX fue el escenario de la Revolución Industrial, el siglo XX fue testigo de la Revolución de la Información. Con la aparición de las tecnologías TCP/IP, HTML y Wi-Fi, se aprendió a "navegar" por la red y a utilizar contenidos web. Luego se vio nacer el *e-commerce* y la explosión del *crowdsourcing* y el concepto "2.0". Está claro que la popularización de Internet y los avances en las telecomunicaciones han facilitado que todo esté cada vez más conectado.

Los sensores y dispositivos colocados en todo tipo de objetos y conectados a Internet a través de redes fijas e inalámbricas son cada vez más pequeños y baratos. Las redes son capaces de extraer un gran volumen de datos susceptibles de ser analizados por computadoras. Un cambio importante por lo que los objetos ya no sólo interactúan con el usuario, sino que también están dotados de la inteligencia para hacerlo entre ellos.

2.1.4. El impacto en el medio ambiente con el uso de la IoT

El experto del *Future Trends Forum* Paul Horn ilustra la necesidad de progreso en el cuidado del medio ambiente con cifras: 170.000 millones de kilovatios hora se malgastan cada año por parte de los consumidores debido a la falta de información sobre el uso de energía. Habla de 3.700 millones de horas de trabajo perdidas, o de 8.700 millones de litros de combustible derrochados (sólo en Estados Unidos), porque las personas no son eficientes a la hora de trabajar o de trasladarse a su lugar de trabajo, respectivamente. También denuncia que 100 millones de personas en todo el mundo son arrastradas al umbral de la pobreza por culpa de los gastos de asistencia sanitaria personal. Por todo ello, Horn propone a la *IoT* como solución a algunos de los problemas medioambientales que nos amenazan hoy día.

Los edificios "inteligentes" constituyen el mejor ejemplo de la aplicación de Internet a un objetivo medioambiental. En Estados Unidos, los edificios consumen el 70 % de toda la electricidad, de la cual un 50 % se malgasta. Además, un 50 % del agua que consumen también es derrochada. Para subsanar este tipo de situaciones, se dota a muchos edificios de *smart grid*, una red que permite optimizar la generación y el consumo de energía gracias a una serie de medidores inteligentes que eligen las mejores franjas horarias entre empresas eléctricas y discriminan entre horarios de consumo. El resultado es un consumo más sensato y económico.

2.1.5. IoT y los consumidores

Es obvio que la *IoT* beneficia al consumidor gracias a una tarificación más transparente de los servicios que consume. Esto es así porque los sensores permiten registrar constantemente una actividad, lo que lleva a poder tomar decisiones en base a los resultados, como por ejemplo, controlar el gasto en luz o agua. Sin embargo, en algunos casos sí se traduce en un beneficio económico para determinados perfiles de consumidores. Algunas aseguradoras de automóviles de Europa y Estados Unidos están instalando sensores en los vehículos de sus clientes para cobrarles en función de su comportamiento al volante en lugar de por criterios demográficos.

La *IoT* también puede brindar información sobre algunos patrones del consumidor, lo cual resulta extremadamente valioso para las marcas y comercios.

2.1.6. La industria del software y la IoT

Allá por el siglo XVII, el matemático e ingeniero Al-Juarismi inventó el algoritmo, es decir, el conjunto ordenado y finito de pasos que permite la realización de una tarea o la resolución de un problema. Se apuntaba antes a la gran cantidad de datos que se está generando gracias a la *IoT*, dado que cualquier objeto es capaz de transmitir información.

Los algoritmos tienen aplicaciones muy valiosas. Mediante su uso en aplicaciones de software es posible producir respuestas rápidas a fenómenos físicos sobre la base de la información recogida o los patrones que siguen determinados objetos o personas. Se crean nuevas oportunidades para satisfacer los requerimientos del negocio, desarrollar nuevos servicios en tiempo real, adentrarse en procesos y relaciones complejas, gestionar incidencias, abordar la degradación del medio ambiente, supervisar las actividades humanas,

mejorar la integridad de la infraestructura y tratar de resolver cuestiones de eficiencia energética.

Existe toda una mina en el desarrollo de software que recolecte, almacene, organice e integre la información con otras fuentes de datos o que dispare alertas en otros programas o para los seres humanos. Gracias a la *IoT*, los objetos en sí, provistos de sensores y actuadores, se convierten en elementos de los sistemas de información, capaces de capturar y comunicar datos a gran escala y, en algunos casos, pueden adaptarse automáticamente a los cambios en el entorno. Estos activos "inteligentes" pueden hacer más eficientes los procesos y proporcionar nuevas capacidades a los productos. Sólo con el software adecuado será posible que la *IoT* cobre vida como se imagina, como parte integrante de la Internet del futuro. Es a través de este software que las aplicaciones e interacciones novedosas se llevan a cabo, y que la red con todos sus recursos, dispositivos y servicios distribuidos, se vuelve manejable.

2.2. *Open source*

Open source [15] es una expresión de la lengua inglesa que pertenece al ámbito de la informática. Aunque puede traducirse como "fuente abierta", suele emplearse en nuestro idioma directamente en su versión original, sin su traducción correspondiente.

Se califica como *open source* a los programas informáticos que permiten el acceso a su código de programación, lo que facilita modificaciones por parte de otros programadores ajenos a los creadores originales del software en cuestión.

Es importante distinguir entre el software *open source*, que dispone de la mencionada característica de presentar su código abierto, y el software libre (que puede descargarse y distribuirse de manera gratuita). Existe software libre que no brinda acceso al código (y que, por lo tanto, no puede considerarse como *open source*), y programas *open source* que se distribuyen de manera comercial o que requieren de una autorización para ser modificados.

Pese a que ambas nociones suelen confundirse, por lo general la idea de *open source* está vinculada a una filosofía de trabajo conjunto sobre los programas informáticos. Cuando se brinda acceso al código fuente, la comunidad de programadores puede hacer sus aportes para solucionar eventuales fallos, incrementar la usabilidad y mejorar el programa a nivel general.

2.3. Software Libre

Es el software que respeta la libertad de los usuarios y la comunidad [16]. A grandes rasgos, significa que los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Es decir, el software libre es una cuestión de libertad, no de precio.

Los usuarios (tanto individualmente como en forma colectiva) controlan el programa y lo que este hace. Cuando los usuarios no controlan el programa, se dice que dicho programa "no es libre", o que es "privativo". Un programa que no es libre controla a los usuarios, y el programador controla el programa, con lo cual el programa resulta ser un instrumento de poder injusto.

Un programa es software libre si los usuarios tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa como se desea, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y modificarlo para hacer lo que se desea (libertad 1).
- El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

2.4. Raspbian

Raspbian [17] es un sistema operativo libre basado en Debian, optimizado para el hardware de Raspberry Pi.

Este sistema operativo viene con más de 35.000 paquetes de software pre compilado. Es un proyecto comunitario en desarrollo activo, con énfasis en mejorar la estabilidad y el rendimiento de tantos paquetes de Debian como sea posible.

2.5. PHP

Pre-procesador de hipertexto (PHP, *hypertext pre-processor*) es un lenguaje de *scripting* para la programación de páginas dinámicas de servidor. Este lenguaje forma parte del software libre. Es decir que se le pueden introducir modificaciones o mejoras y ponerlas a disposición de los demás usuarios del mismo.

Otra característica importante es que se trata de un lenguaje multiplataforma, esto quiere decir que la aplicación web desarrollada en PHP puede funcionar en casi cualquier tipo de plataforma Windows, Unix/Linux (y sus diferentes versiones y distribuciones). También ofrece soporte a los motores de base de datos más populares, lenguaje de consulta estructurada (SQL, *Structured query language*), MySQL, PosgresSQL, Oracle, etc.), así como también acceso a conectividad de base de datos abierta (ODBC, *Open database conectivity*).

Una aplicación web basada en PHP necesita dos tipos de software. El primero es un servidor web que va a atender las peticiones de los usuarios y devolverá las páginas solicitadas. El servidor Apache, tanto su versión Windows como Linux es el más utilizado. El segundo software es el propio PHP, es decir el módulo que se va a encargar de interpretar y ejecutar los scripts que se soliciten al servidor.

Al utilizar una tecnología del tipo pre-procesado en el servidor es necesario visualizar las páginas generadas con PHP utilizando el protocolo HTTP. Al contrario de lo que

ocurre con las páginas de la tecnología cliente, en las que se puede visualizar mediante la opción "Archivo>Abrir" en cualquier navegador, las páginas generadas con PHP necesitan ser interpretadas por un servidor web para que sean procesadas y luego enviadas al navegador del usuario.

2.6. MySQL

MySQL es un sistema de administración de bases de datos (DBMS, *Database management system*) para bases de datos relacionales. Así, MySQL no es más que una aplicación que permite gestionar archivos llamados de bases de datos.

Existen muchos tipos de bases de datos, desde un simple archivo hasta sistemas relacionales orientados a objetos. MySQL, como base de datos relacional, utiliza multiples tablas para almacenar y organizar la información. MySQL fue escrito en C y C++ y destaca por su gran adaptación a diferentes entornos de desarrollo, permitiendo su interactuación con los lenguajes de programación más utilizados como PHP, Perl y Java y su integración en distintos sistemas operativos.

También es destacable, la condición de *open source* de MySQL, que hace que su utilización sea gratuita e incluso se pueda modificar con total libertad, pudiendo descargar su código fuente. Esto ha favorecido positivamente en su desarrollo y continuas actualizaciones, para hacer de MySQL una de las herramientas más utilizadas por los programadores orientados a Internet.

2.7. Servidor web

La definición más sencilla de servidor web es un programa especialmente diseñado para transferir datos de hipertexto, es decir, páginas web con todos sus elementos (textos, *widgets*, *banners*, etc). Estos servidores web utilizan el protocolo HTTP.

Los servidores web están alojados en una computadora que cuenta con conexión a Internet. El servidor web, se encuentra a la espera de que algún navegador le haga alguna petición, como por ejemplo, acceder a una página web y responde a la petición, enviando código HTML mediante una transferencia de datos en red.

2.7.1. Apache servidor HTTP

Apache es un poderoso servidor web, cuyo nombre proviene de la frase inglesa "*a patchy server*" y es completamente libre, ya que es un software *open source* y con licencia pública general GNU (GPL, *GNU general public license*). Una de las mayores ventajas de Apache, es que es un servidor web multiplataforma, es decir, puede trabajar con diferentes sistemas operativos y mantener su excelente rendimiento.

2.8. openHAB

OpenHAB es un software para integrar diferentes sistemas y tecnologías de automatización del hogar en una sola solución que permite gestionar reglas de automatización ofreciendo una interfaz de usuario estable.

Entre las características más notorias de openHAB se encuentran las siguientes:

- OpenHAB está diseñado para ser libre de utilizar con cualquier tipo de hardware, proveedor o protocolo orientado a la domótica.
- Puede funcionar en cualquier dispositivo que sea capaz de ejecutar una máquina virtual Java (JVM, *Java virtual machine*) (Windows, Linux, Mac).
- Permite integrar una gran cantidad de diferentes tecnologías domóticas en una sola.
- Tiene un potente motor de reglas para satisfacer todas las necesidades de automatización.
- Incorpora diferentes interfaces de usuario basadas en la web, así como también interfaces de usuario nativas para iOS y Android.
- Es completamente *open source*.
- Es fácilmente adaptable para integrarse con nuevos sistemas y dispositivos.
- Es mantenido por una comunidad apasionada y en crecimiento.
- Proporciona interfaz de programación de aplicaciones (API, *Application programming interface*) para integrase en otros sistemas.

El proyecto openHAB se divide en dos partes:

1. OpenHAB *Runtime*: este es el paquete que se ejecutará en el servidor y que hace el trabajo "real".
2. OpenHAB *Designer*: es una herramienta de configuración para openHAB *Runtime*. Viene con editores fáciles de usar para configurar las rutinas de ejecución, definir la interfaz de usuario e implementar las reglas.

2.8.1. OpenHAB *Runtime*

Es un conjunto de paquetes de iniciativa abierta de servicios de puerta de enlace (OSGi, *open services gateway initiative*) desplegados en una infraestructura Equinox, el cúal es un sistema de módulos dinámicos. Por lo tanto es una solución pura Java y necesita una máquina virtual Java para ejecutarse.

Basándose en OSGi proporciona una arquitectura altamente modular, que incluso permite agregar y quitar funcionalidades durante la ejecución sin detener el servicio. La figura 2.3 muestra una descripción de las partes principales y de cómo dependen unos de otros.

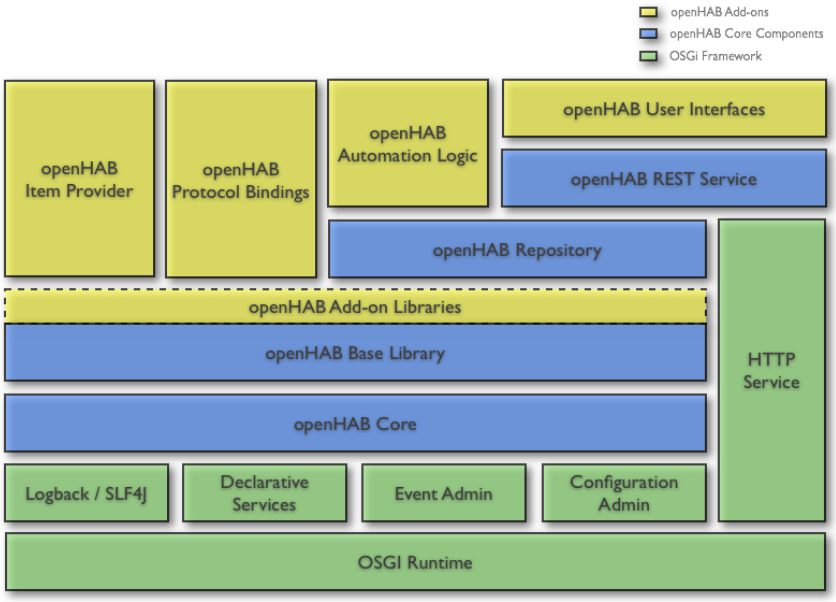


Figura 2.3: Arquitectura openHAB. Fuente: openHAB Wiki [18].

2.8.1.1. Comunicación

OpenHAB tiene dos canales diferentes de comunicación internos:

1. Bus de eventos asíncrono.
2. Un repositorio de ítems.

Bus de eventos Es un servicio base de openHAB. Todos los paquetes que no requieren comportamiento con estado deben utilizarlo para informar a otros paquetes sobre eventos y para que sean actualizados por paquetes en eventos externos.

Hay principalmente dos tipos de eventos:

1. Comandos que activan una acción o un cambio de estado de algún elemento/dispositivo.
2. Actualizaciones de estados que informan sobre un cambio de estado de algún elemento/dispositivo (a menudo como respuesta a un comando).

Todos los protocolos de enlace (*bindings*), que proporcionan el enlace a los dispositivos de hardware reales deben comunicarse a través del bus de eventos. Esto asegura que existe un acoplamiento a bajo nivel del conjunto, lo que facilita la naturaleza dinámica de openHAB.

OpenHAB sirve como un centro de integración entre dispositivos y como un mediador entre los diferentes protocolos que se utilizan entre dispositivos. En una instalación típica por lo general sólo hay una instancia de openHAB en ejecución en alguno de los servidores centrales. Sin embargo, como el servicio administrador de eventos OSGi también se puede utilizar como un servicio remoto, es posible conectar varias instancias openHAB distribuidas a través del bus de eventos.

Repositorio de ítems Los ítems representan la funcionalidad que es utilizada por las aplicaciones, como interfaces de usuario o lógica de automatización. Los ítems tienen un estado y pueden recibir comandos.

OpenHAB ofrece un repositorio de ítems, que está conectado al bus de eventos y realiza un seguimiento del estado actual de todos los elementos. El repositorio se puede utilizar cuando sea necesario acceder al estado de los elementos. También la automatización de la lógica de ejecución (*auttomation logic execution*) necesita estar siempre informada sobre los estados actuales. El repositorio de ítems evita que cada paquete se instale en el caché para su uso interno. También asegura que el estado esté sincronizado para todos esos paquetes y proporciona la posibilidad de que persistan los estados en el sistema de archivos o en una base de datos, de modo que se mantenga incluso durante un reinicio del sistema.

La Figura 2.4 muestra como se utilizan estos canales de comunicación.

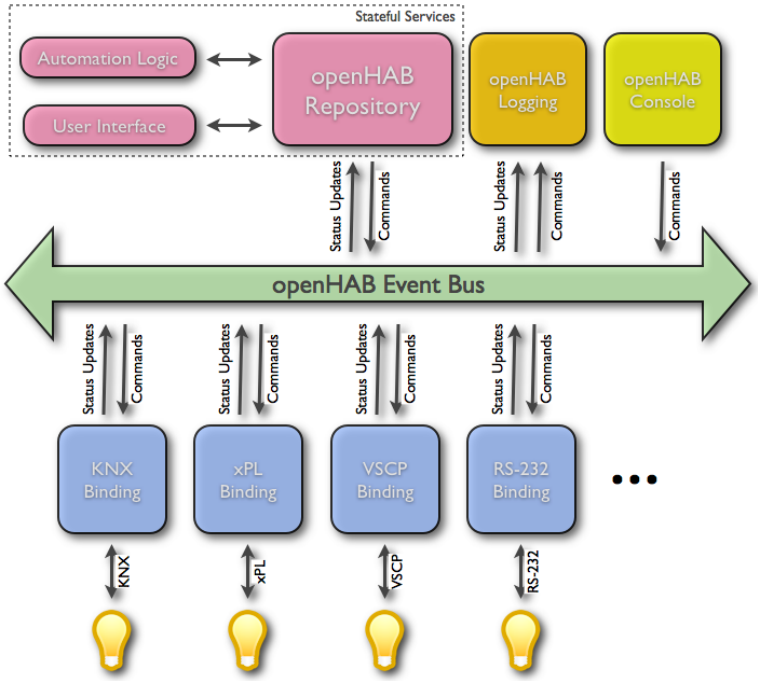


Figura 2.4: Canales de comunicación de openHAB. Fuente: openHAB Wiki [18].

2.8.1.2. Mapa de sitio

OpenHAB cuenta con una configuración genérica para sus interfaces de usuario: el mapa de sitio (*sitemap*). El mapa de sitio es una estructura de árbol de complementos o *widgets* que definen las diferentes páginas de una interfaz de usuario y su contenido. Los complementos se pueden asociar a elementos, para los que deben mostrar el estado y/o elementos de control.

La definición del mapa del sitio es bastante abstracta; se supone que es un modelo de interfaz de usuario adecuado para diferentes tipos de interfaces de usuario, de modo que el usuario no tiene que configurar cada uno de ellos en caso que configure varias interfaces de usuario.

2.8.1.3. Reglas de automatización

Las reglas se utilizan para automatizar procesos. Cada regla puede ser disparada e invocar un *script* que realiza cualquier tipo de tareas, por ejemplo, encender luces, realizar cálculos matemáticos, etc. openHAB tiene un motor de reglas bien integrado, ligero y potente.

Disparadores de reglas Para que una regla se ejecute debe ser disparada o activada. Hay diferentes categorías de disparadores de reglas:

- Ítem (basadas en disparadores de eventos): reaccionan sobre eventos en el bus de eventos openHAB, es decir, comandos y actualizaciones de los elementos.
- Disparadores basados en tiempo: reaccionan a tiempos especiales, por ejemplo, cada 20 segundos, cada medianoche, cada hora, etc.
- Disparadores basados en estados del sistema: reaccionan a ciertos estados del sistema, por ejemplo, inicio, apagado.

2.8.1.4. Persistencia: almacenamiento en base de datos

El soporte de persistencia almacena los estados de los ítems a lo largo del tiempo (una serie de tiempo). No está restringido a un solo tipo de almacenamiento de datos, diferentes tipos de almacenamientos pueden coexistir y configurarse independientemente.

Cuando existen datos persistente existen diferentes tipos de forma de almacenar estos datos: base de datos NoSQL, base de datos *round-robin*, servicio de nube *IoT*, y archivos simples de registro. OpenHAB intenta hacer todas estas opciones posibles y configurables de la misma manera.

Entre los servicios de persistencia están disponibles: db4o, JDBC, MQTT, MySQL, RRD4J, Sen.se, entre otros.

2.8.1.5. *Binding*: protocolos de enlaces

Los *bindings* o protocolos de enlaces son paquetes opcionales que se puede utilizar para extender la funcionalidad de openHAB. Mediante la ayuda de estos, los usuarios pueden, por ejemplo, acceder al software de comunicaciones de Asterisk o conectarse al bus de automatización de KNX Home[1].

2.8.2. OpenHAB *designer*

Es una aplicación de cliente enriquecida Eclipse (Eclipse RCP, Eclipse *rich client platform*) para configurar la rutina de ejecución de openHAB. Provista con editores para archivos de configuración como el mapa de sitio. Su ventaja sobre los editores de texto sencillos es el soporte de entorno de desarrollo integrado (IDE, *integrated development*

[1]KNX es un estándar (ISO/IEC 14543) de protocolo de comunicaciones de red, basado en el modelo de interconexión de sistemas abiertos (OSI, *open system interconnection*), para edificios inteligentes (domótica e inmótica)

environment), así como la comprobación de sintaxis, la finalización automática, el resaltado y la ayuda de contenido. También está destinado a implementar y desplegar reglas para acciones automáticas.

2.9. Souliss

Es una estructura conceptual y tecnológica de soporte definido para redes distribuidas basadas en Arduino para la *IoT* o casas inteligentes [19]. Está diseñado para ser liviano y escalable, se ejecuta a través de múltiples nodos y medios de comunicación.

Souliss es un proyecto *open source* que recopila códigos y personas con común interés en el campo de la *IoT* y domótica.

Está construido sobre la capa de red del modelo de interconexión de sistemas abiertos (OSI, *open system interconnection*) permitiendo construir una red completa de nodos, con lógica distribuida y funcionalidad, donde todos los nodos pueden intercambiar información en una red de pares (*peer-to-peer*) y no hay necesidad de un nodo central que coordine la lógica de las comunicaciones. Gracias a su estructura escalable, la funcionalidad de Souliss puede manejarse a través de diferentes nodos o uno solo, donde la mejor solución puede orientarse de acuerdo al tamaño de la red y los requerimientos de la misma.

2.9.1. Souliss Hardware

Existen varias plataformas disponibles en donde se puede ejecutar Souliss, todas están basadas en microcontroladores AVR [2]. Souliss se ejecuta a través de múltiples microcontroladores y transceptores de red, Arduino y otras placas compatibles.

En la Tabla 2.1 se muestra el hardware soportado.

Hardware soportado	
KMP Electronics	PRODINo Wi-Fi-ESP Wroom
	DINo II
	ProDINo
IONO Relay Board	
Industruino	
Freaklabs Chibiduino	
Moteino RFM69HW	
Arduino Boards	Arduino Duemilanove
	Arduino UNO
	Arduino Leonardo
	Arduino Mini
	Arduino Micro
	Arduino Mega 2560
	Arduino Nano
Generic Espressif ESP8266 Based Boards	
Olimex Boards	

Tabla 2.1: Tabla de hardware soportado por Souliss. Fuente: Souliss Wiki [19].

[2]Familia de microcontroladores RISC del fabricante estadounidense Atmel.

2.9.2. Arquitectura de red

Souliss permite definir una arquitectura que mezcle diferentes medios de comunicación como Ethernet, *wireless*, punto a punto (PPP, *procotol point to point*) y RS-485 según las necesidades del proyecto, reportando todos los datos a un nodo *gateway* que actúa de colector de datos para la interfaz de usuario como la aplicación openHAB.

Una característica innovadora de Souliss es el manejo automático de múltiples medios de comunicación. Es capaz, por ejemplo de tener un nodo con una interfaz Ethernet y también usar RS-485.

Hay varias maneras de tener nodos con múltiples interfaces dependiendo de las características que se estén buscando.

2.9.2.1. Arquitecturas de red compatible

La forma que tiene Souliss de comunicarse entre nodos se basa en vNet, una capa virtual de red que proporciona un solo conjunto de APIs a través de diferentes interfaces de comunicación, incluyendo enrutamiento y puente entre nodos. Las tramas que implican diferentes interfaces de comunicación o extensiones *wireless* son enviadas a destino sin ninguna acción por parte de la aplicación, creando una red de dispositivos que corren sobre diferentes medios de comunicación y que están interconectados.

Desde el punto de vista de la red, vNet se ubica en la capa 3 del modelo OSI sin utilizar la transmisión completa en la capa 2. Esta técnica construye una especie de capa 2,5, útil para dispositivos de baja performance y consumo como un microcontrolador con unos miles de bytes de RAM.

2.9.2.2. Arquitecturas de redes soportadas

Básicamente, vNet funciona con todas las arquitecturas de red que puedan soportar los medios, incluso si la arquitectura pudiera requerir diferentes configuraciones basadas en el número de nodos y sus interconexiones.

Red de un único medio Una red es definida como un único medio de comunicación o *single media network* si todos los nodos utilizan el mismo medio de comunicación, en este caso, la capa vNet realiza la transferencia entre las API y los controladores de medios.

Red puente Se define como red puente o *bridged network* a la red que posee más de una interfaz de medios de comunicación y los datos deben compartirse dentro de los nodos utilizando diferentes interfaces de comunicación. Un nodo puente tiene dos o más interfaces de comunicación y vNet se encarga de realizar el puente entre ellas.

Todos los nodos pueden comunicarse directamente con otros nodos en la misma interfaz de comunicación, las tramas se puentean cuando el destino es un nodo con una interfaz diferente. Cada interfaz tiene un rango de direcciones, por lo que a partir de esa dirección se sabe a qué interfaz se refiere.

Un nodo puede actuar como un puente si se define como súper nodo al momento de compilar.

Red enrutada Se define una red enrutada o *routed network* si hay nodos que comparten la misma interfaz de comunicación pero no pueden comunicarse directamente. Por ejemplo, dos nodos inalámbricos que están fuera de alcance de escucha y necesitan un nodo intermedio para enrutar las tramas.

Un nodo puede actuar como *router* si se define como súper nodo al momento de compilar.

Red mixta Una red es mixta sí hay más de una interfaz de medios de comunicación y los datos necesitan ser compartidos dentro de nodos utilizando diferentes interfaces de comunicación; Además, dentro de uno o más medios se requiere un encaminamiento de tramas para ampliar el rango de la red. Esta combinación despliega toda la capacidad de Souliss, donde los paquetes se encaminan en un entorno *peer-to-peer* y se entregan de forma transparente.

2.9.3. *Gateway*

Dentro de una red distribuida como la de Souliss, a veces puede ser útil poder recopilar datos de un solo punto. Esto contrasta con la arquitectura distribuida pero da libertad de aplicación.

El *gateway* se utiliza sólo para que la interfaz de usuario obtenga todos los datos sin un conocimiento previo de la red, el resultado es una red distribuida para la lógica de ejecución pero centralizada para la recopilación de datos y fluye desde un único nodo a una interfaz de usuario.

2.9.3.1. Configuración del *gateway*

Un nodo seleccionado como *gateway* es capaz de recopilar datos de otros nodos a través de MaCaco basándose en eventos, los datos no se almacenan, sólo pasan a través del *gateway* a la interfaz de usuario que se ha suscripto. Para almacenar localmente los datos de otros nodos y enviarlos a través de las interfaces se utiliza el modo *persistence* y *lastin*, de lo contrario la información sólo está disponible en el nodo.

2.9.4. vNet

Es el protocolo de transporte utilizado en Souliss, se utiliza para enrutar diferentes medios de comunicación en las distintas arquitecturas. Puede ser transportado a través de otros protocolos de transporte (como IPv4) o directamente sobre un protocolo de nivel 2 (como IEEE 802.15.4). Cuando se combina con otro protocolo de transporte se requiere un enlace entre la dirección vNet y la dirección del protocolo de transporte subyacente (o los datos se transportarán como transmitidos).

2.9.4.1. Estructura de la trama

Las tramas en la red vNet están compuestas de seis bytes de encabezado y una carga útil variable para una longitud de trama, la longitud máxima de trama depende de la implementación.

La Figura 2.5 muestra la estructura de la cabecera de la trama vNet.

Longitud (1 byte)	Puerto (1 byte)	Dirección destino final (2 byte)	Dirección origen inical (2 byte)	Carga útil (Longitud - 6 bytes)

Figura 2.5: Cabecera de 6 bytes de la trama vNet. Fuente: Souliss Wiki [19].

El encabezado en la parte superior junto con la carga útil es lo que se analiza desde la capa vNet, esto se pasa al controlador de medios que incluye información adicional y el encapsula la trama vNet en la trama de comunicación final.

Las direcciones están estructuradas en seis grupos:

- Los primeros cinco se agrupan para los medios de comunicación físicos subyacentes.
- El último grupo es para medios de comunicación multicast y broadcast.

El enlace entre las direcciones y los medios de comunicación físicos es generalmente inusual y hace que vNet sea un protocolo que está entre la capa 3 y la capa 2 del modelo OSI. Al analizar la trama, la dirección se utiliza para identificar la ruta de enrutamiento, sobrescribiendo con la ruta definida y el enrutamiento permitido.

2.9.4.2. Controladores vNet

Los controladores son parte del código que maneja la comunicación vNet sobre un medio de comunicación físico específico y un transceptor relevante.

vNet sobre UPD/IPv4 Uno de los protocolos de transporte subyacentes utilizados para vNet es UDP/IPv4, los nodos siempre están en el puerto UDP 230 y la trama vNet se incluye en la carga útil de UDP con un byte adicional para toda la longitud, como se ve en el trama de ejemplo de la Figura 2.6.

Longitud+1 (1 byte)	**Longitud (1 byte)**	**Puerto (1 byte)**	**Dirección destino final (2 byte)**	**Dirección origen inicial (2 byte)**	**Carga útil (Longitud - 6 bytes)**
0x0C	0x0B	0x17	0x11, 0x00	0x12, 0x00	0x05, 0x6D, 0x02, 0xCB, 0x08

Figura 2.6: Ejemplo cabecera de 6 bytes de la trama vNet. Fuente: Souliss Wiki [19].

El nodo asume que todos están escuchando el puerto 230, así el origen y el IP destino tienen el mismo puerto 230, esto no se aplica cuando se utiliza el modo usuario (dirección de origen entre 0x0101 y 0x64FE) donde se usa un enfoque cliente-servidor y la respuesta se envía al IP origen utilizando el puerto que se encuentra en el pedido.

2.9.5. Direccionamiento

La comunicación entre los nodos Souliss se basa en vNet que permite el manejo de la interfaz de comunicación múltiple, el enrutamiento y el puente entre ellos, por lo que cada nodo tiene una o más direcciones. Este direccionamiento es realizado de acuerdo a la Tabla 2.2.

Medio	Inicio de rango	Fin de rango
M1 – UDP/IP	0x0000	0x00FF
M1 – UDP/IP in User Mode	0x0100	0x64FF
M2 - 2.4 GHz IEEE 802.15.4	0x6500	0xAAFF
M3 – UDP/IP Broadcast	0xAB00	0xBBFF
M4 – Not Used	0xBC00	0xCCFF
M5 – USART	0xCE00	0xDFFF

Tabla 2.2: Direccionamiento Souliss.

Cada nodo puede tener una o varias direcciones vNet, los nodos Ethernet/WiFi tienen adicionalmente una dirección IP. El propio protocolo vNet puede ser transportado en tramas UDP/IP, pero tiene su propio esquema de direccionamiento.

2.9.5.1. Reglas de configuración de red

Las facilidades de los modos puente y ruteo son proporcionadas por las reglas de la configuración de red, estas permiten a los nodos identificar como enrutar o encaminar un mensaje si estos lo requieren.

2.9.5.2. Reglas de direccionamiento

Las reglas de direccionamiento utilizadas para vNet son principalmente las mismas de IPv4 con una diferencia, la longitud de la dirección. En vNet se utilizan sólo dos bytes (en lugar de 4 bytes como en IPv4).

Además, en vNet las direcciones están divididas por medio, esto facilita la identificación de qué medio tiene que ser utilizado para la comunicación. Cada medio tiene una máscara de subred asociada, utilizando una función lógica *and* bit a bit entre la dirección y la máscara de subred, se obtiene la subred del dispositivo.

2.9.5.3. Reglas de enrutamiento y puente

El dispositivo dentro de la misma subred debe estar en el rango de audición, el concepto de rango de audición debe referirse a los medios de comunicación que se utilizan para el dispositivo. Para el dispositivo que utiliza una Ethernet cableada o inalámbrica un rango de escucha es la red LAN conmutada/enrutada y no se solicita el puente.

Una trama que tiene un destino fuera de la propia subred, se envía a un dispositivo con capacidad de enrutamiento y/o puente. En este caso, este dispositivo se llama súper nodo (diferente de la puerta de enlace IP) que puenteará o enrutará el mensaje. En la configuración de cada nodo se especifica la dirección del súper nodo.

Un dispositivo compilado como súper nodo es capaz de enrutar y/o puentear tramas sin configuraciones adicionales, se evalúa la ruta de salida para una trama enrutada o puenteada suponiendo lo siguiente:

- Todos los dispositivos comparten la misma máscara de subred,
- Cada subred tiene un súper nodo con la dirección "subn & 0x0001".

Si estos requisitos no se cumplen, las tablas de enrutamiento y puente permiten el enrutamiento de tramas a través de rutas preferidas o de direccionamiento no estándar, éstas se configuran en tablas de encaminamiento y puentes.

2.9.5.4. Conexiones TCP/IP basadas en IP y vNet

La conectividad vNet IP se basa en tramas UDP/IP utilizando el puerto 230 (entrada/salida) y 23000 (en el modo de usuario) y no se puede utilizar para otras comunicaciones. En vNet están disponibles APIs para la conexión estándar de TCP/IP estándar para la capa vNet.

En caso de software IP *stack* para cada nueva conexión se requiere un controlador adicional.

2.9.6. Estructura de datos

La estructura de datos es un arreglo de información definida en el nodo e intercambiada con el protocolo MaCaco, el cual permite la comunicación entre diferentes nodos y la interfaz de usuario. Un valor en una estructura de datos asume un significado diferente dependiendo la posición en la estructura, dando una interpretación común a todos los otros dispositivos que tiene acceso a estos datos.

2.9.6.1. Área de estructura de datos

La estructura de datos (también llamada *MaCaco shared memory area*) desde el punto de vista del usuario se divide en tres sub-áreas. Siendo las mismas los elementos lógicos (*typical logic*), variables de entrada (*input*) y variables de salida (*output*), a su vez en estas estructuras se encuentran contenidos las ranuras o *slots*. La Figura 2.7 muestra la estructura del área de memoria compartida y su interacción a nivel de aplicación.

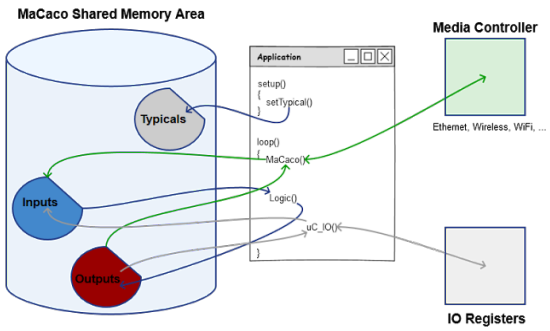


Figura 2.7: Estructura de datos Souliss. Fuente: Souliss Wiki [19].

Typical logic las variables lógicas están pre configuradas en las llamadas *typicals*. Este tipo de variable hace referencia a un conjunto predefinido de comandos de entrada y salida con un comportamiento bien conocido. Se utilizan para estandarizar la interfaz de usuario y tener un trato de las mismas sin configuración. *Typicals* es una interfaz lógica entre la rutina de ejecución del *framework* Souliss y la interfaz de usuario.

Al iniciar la interfaz de usuario, pide la lista completa de *typicals*, cada nodo responde con la lista de detallada en su configuración, de este modo la interfaz descubre las funcionalidades y comandos soportados de cada variable.

El nodo posee una estructura de datos construida en base a *slots* y cada *slot* asociado a un *typical*. Una vez asignada la lógica del *typical* lee periódicamente la entrada del *slot* y reescribe la salida relevante.

Cada *typical* tiene un conjunto predefinido de comandos de entradas aceptables y salidas relevantes, esto permite una fácil interacción ya que funciona como un acuerdo entre las capas de bajo nivel y varios clientes (Android, iOs, entre otros).

Input Es el área de la estructura de datos que contiene información entrante de otros nodos o aplicación dentro del mismo nodo. Los datos incluidos en las entradas se procesan en el nivel de aplicación con llamadas a través de la API.

Output Es el área de la estructura de datos que contiene información de salida a otros nodos o aplicación dentro del mismo nodo, los datos de salida se insertan y se manejan a nivel de aplicación.

Slot Un *slot* o ranura es una partición de la estructura de datos, cada sub-área (*typical, input, output*) se divide en *slots*. En el nivel de aplicación se asignan *slots* a las funcionalidades o variables lógicas, mapeando dispositivos a cada una de estos *slots*.
Los *typicals* son definidos por aplicación durante la llamada a `setup()`, esto declara qué dispositivos están ubicados en ese nodo y en cual ranura o *slot*. Durante el bucle principal, las *inputs* son escritas por el protocolo de comunicación (o las I/O locales), son procesadas por las lógicas y el resultado es finalmente almacenado en el *slot output*. Estas variables están disponibles a través de los protocolos de comunicación a cualquier otro nodo de la red.

La interfaz de usuario inicialmente lee los *typicals* definidos en los nodos, de esta manera conoce en cada nodo las funcionalidades disponibles, esto permite construir una lista de dispositivos y propiedades reduciendo los esfuerzos de programación. Una vez que se crea esta lista expone al usuario sólo los comandos disponibles y muestra los estados adquiridos relevantes desde el nodo.

2.9.7. Protocolo de datos MaCaco

MaCaco es un protocolo de datos basado en eventos binarios diseñado para microcontroladores con pocos bytes de RAM, el cual define la estructura de datos y el protocolo para el intercambio de datos. Tiene una estructura *peer-to-peer* y se transporta sobre una capa *peer-to-peer*, sin embargo, se puede utilizar un protocolo maestro/esclavo en el nivel de aplicación. MaCaco es casi un protocolo sin estado, significa que al recibir una trama,

incluirá toda la información para analizarla y ejecutar (si existe) una acción, no requiere almacenar el historial de interacción previa.

2.9.7.1. Estructura de la trama

El protocolo MaCaco necesita de un protocolo de transporte (vNet en caso de Souliss, pero cualquier otro es aplicable) y por lo tanto no tiene una asignación de direccionamiento o de puerto, cada trama tiene una cabecera fija y una carga útil variable. Parcialmente la estructura de MaCaco es similar a Modbus [3], pero incluye modificaciones para soportar la interacción *peer-to-peer* y basada en eventos entre nodos.

2.9.7.2. Flujo de datos de comunicación

Potencialmente cualquier nodo puede iniciar una comunicación, MaCaco es un protocolo no orientado a la conexión y cada interacción está compuesta sólo de dos mensajes: la solicitud y la respuesta. No hay confirmación (ACK) ni comprobación de coherencia, se supone que estos son llevados por la capa de transporte.

Hay tres tipos de flujo de datos en MaCaco:

- Polling: en el modo *polling*, un nodo inicia una comunicación con una petición, entonces recibirá la respuesta una vez.
- Pushing: en el modo de *pushing*, un nodo inicia una comunicación con una petición de fuerza y no recibe ninguna respuesta.
- Subscribing: en el modo de *subscribing*, un nodo inicia una comunicación con una petición de suscripción, que notifica el interés en algunos datos, después de una solicitud se sigue una respuesta inmediatamente y después cada vez que se produce un suceso en el nodo suscripto, se envía automáticamente un nuevo paquete.

La suscripción reduce el uso del canal, los datos se transmiten sólo si es necesario y libera recursos en el nodo. Al mismo tiempo, el suscriptor puede tener una larga inactividad, por lo que no se sabe si el nodo suscrito sigue siendo saludable, dado que la suscripción se realiza en tiempo de ejecución y se almacena en la RAM un restablecimiento del nodo dará una suscripción rota.

Por esta razón el nodo abonado se encargará de renovar periódicamente sus suscripciones, no hay restricciones por el tipo de método a usar, cada nodo podrá utilizar el que desee.

El modo sugerido es un sondeo adaptativo *(polling adaptive)*, para cada suscripción se define un valor saludable que comienza a partir de un valor predefinido, siendo menor ese valor más saludable es y menos es el tiempo de espera. Esta técnica permite un flujo de datos híbrido, en principio es similar a *polling* y si las respuestas se reciben en el tiempo entonces el tiempo de espera se incrementa, siendo para un nodo saludable la tasa de sondeo más baja a horas.

En el caso de un canal de comunicación “sucio”, el valor saludable será un valor medio que resulta en una tasa mayor pero que ayuda a recuperar las tramas perdidas debido a errores en el canal.

[3] protocolo de comunicaciones situado en el nivel 7 del modelo OSI, basado en la arquitectura maestro/esclavo (RTU) o cliente/servidor (TCP/IP)

2.9.7.3. MaCaco y la comunicación del *gateway*

Un *gateway* es un nodo que hace de medio de comunicación entre las interfaces de usuario y otros nodos de la red, esto se utiliza para tener una comunicación más sencilla. La interfaz del usuario no debe preocuparse por las rutas de direccionamiento y enrutamiento, sólo hace referencia a los nodos por un índice y recopila todos los datos disponibles para ese nodo.

La interacción con la interfaz de usuario no es *peer-to-peer* pero usa el clásico enfoque cliente-servidor, la interfaz de usuario es el cliente de la comunicación.

Para facilitar la comunicación el protocolo de datos MaCaco utiliza códigos funcionales definidos.

2.10. TCP/IP

Es una denominación que permite identificar al grupo de protocolos de red que respaldan a Internet y que hacen posible la transferencia de datos entre redes de computadoras. En concreto, puede decirse que TCP/IP hace referencia a los dos protocolos más trascendentes de este grupo: el conocido como protocolo de control de transmisión (o TCP) y el llamado protocolo de Internet (presentado con la sigla IP).

En este sentido, es necesario subrayar que el primero de los protocolos citados lo que hace es proporcionar un transporte fiable de los datos dentro de lo que es el nivel de transporte del modelo de referencia OSI. Y mientras, el segundo, el protocolo IP se identifica y define especialmente por el hecho de que lo que hace, en el nivel de red, ofrece la posibilidad de dirigir los citados a otras máquinas.

Asimismo, hay que destacar que dentro de lo que es TCP/IP existen varios niveles que es muy importante que sean tenidos en cuenta. En concreto son cuatro:

- **Nivel de aplicación**: Es el más alto dentro del protocolo en él se encuentran una serie de aplicaciones que tienen la capacidad de acceder a diversos servicios a los que se puede acceder vía Internet.
- **Nivel de transporte**: Es el encargado de ofrecer una comunicación entre extremos de programas de aplicación.
- **Nivel de red**: Se dedica a realizar una serie de acciones sobre la información que recibe del nivel anterior para luego acometer el envío al nivel que está por debajo de él.
- **Nivel de enlace**: Su misión más clara es transmitir la información que recibe al hardware.

El grupo de protocolos TCP/IP fue diseñado para enrutar y ofrece un nivel alto de fiabilidad, lo que permite que sea adecuado para grandes redes y que posibilite el funcionamiento de Internet a nivel global. Además resulta compatible con las herramientas estándar que analizan el funcionamiento de la red.

En cuanto a los puntos en contra del TCP/IP, suele mencionarse que es algo más complejo de configurar y de mantener bajo control que otros sistemas, y que puede funcionar con una lentitud notoria en las redes con un volumen de tráfico bajo.

2.11. RS-485

También conocido como EIA-485, que lleva el nombre del comité que lo convirtió en estándar en 1983. Está definido como un sistema en bus de transmisión multipunto diferencial, es ideal para transmitir a altas velocidades sobre largas distancias (35 Mbps hasta 10 metros y 100 Kbps en 1.200 metros) y a través de canales "ruidosos", ya que reduce los ruidos que aparecen en los voltajes producidos en la línea de transmisión.

El medio físico de transmisión es un par entrelazado que admite hasta 32 estaciones en 1 solo hilo, con una longitud máxima de 1.200 metros operando entre 300 y 19200 bps y la comunicación *half-duplex* (*semiduplex*). Soporta 32 transmisiones y 32 receptores. La transmisión diferencial permite múltiples controladores dando la posibilidad de una configuración multipunto. Al tratarse de un estándar bastante abierto permite muchas y muy diferentes configuraciones y utilizaciones.

RS-485 está diseñado para ser un sistema balanceado, como muestra la Figura 2.8, esto significa que los cables se utilizan para transmitir la señal no son referenciados a tierra.

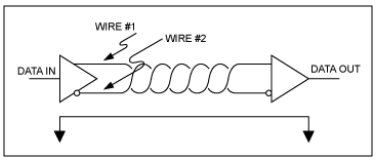


Figura 2.8: Un sistema balanceado utiliza dos cables, distintos de tierra, para transmitir datos. Fuente: Guidelines for proper wiring of an RS-485 [20].

El sistema se llama balanceado porque la señal en un hilo es idealmente la exacta opuesta de la señal en el segundo hilo. En otras palabras, si un cable está transmitiendo un alto, el otro cable transmitirá un bajo, y viceversa. Como muestra la Figura 2.9.

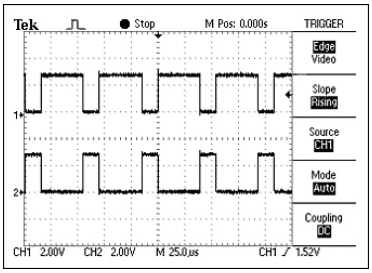


Figura 2.9: Las señales en un cable de un sistema balanceado son idealmente opuestas. Fuente: Guidelines for proper wiring of an RS-485 [20].

Aunque RS-485 se puede transmitir con éxito usando múltiples tipos de medios, debe ser utilizado con el cableado comúnmente llamado "par trenzado".

La red RS-485 más simple está compuesta por un único transmisor y un solo receptor. Aunque es útil en varias aplicaciones, el RS-485 permite una mayor flexibilidad permitiendo múltiples receptores y transmisores en un solo par trenzado. El número máximo de transceptores y receptores permitidos depende de cuánto carga cada dispositivo en el sistema. En un mundo ideal, todos los receptores y transmisores inactivos tendrán una

impedancia infinita y no sobrecargarán el sistema de ninguna manera. En el mundo real, sin embargo, este no es el caso. Cada receptor conectado a la red y todos los transmisores inactivos agregará una carga incremental.

En la norma RS-485 se utiliza el tipo de transmisión diferencial *half-duplex* que se caracteriza por:

- Utilizar dos hilos o cables, referenciado a masa.
- El dato se obtiene de la diferencia de la señal eléctrica entre los dos hilos que componen el canal de comunicación.
- El hecho de que el tipo de transmisión sea *half-duplex* indica que cada equipo puede enviar y recibir, pero no de forma simultánea.
- Esta norma RS-485 permite velocidades de hasta 10 Mbps y distancias de hasta un máximo de 1.200 metros.
- Esta norma física permite la configuración de una red con un máximo de 32 estaciones de trabajo.

La norma RS-485 incorpora un tercer estado que permite que un equipo se pueda colocar en estado de alta impedancia, y por tanto no lee nada, es como si se encontrara desconectado de la línea.

Normalmente la habilitación se encuentra en estado de recepción “0”. Si se quiere transmitir, bastará con poner un “1” en la entrada de habilitación correspondiente.

La Figura 2.10 muestra un ejemplo de conexionado para una comunicación *halfduplex*.

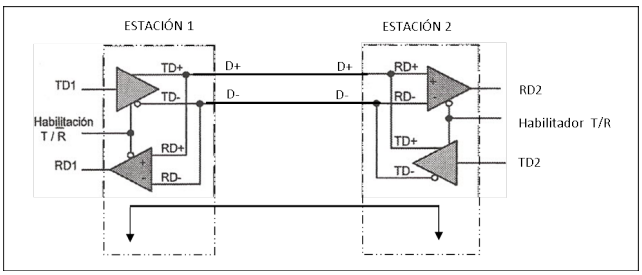


Figura 2.10: Transmisión diferencial aplicada a la norma RS-485. Fuente: Guidelines for proper wiring of an RS-485 [20].

Las principales características son que la longitud máxima es de aproximadamente 1.200 metros a una velocidad de 90 Kbps, y la velocidad máxima del enlace es de 10 Mbps. Como en cualquier sistema de comunicaciones, la velocidad y longitud del enlace están inversamente relacionadas: si deseamos obtener la máxima velocidad, el cable deberá ser de unos pocos metros y viceversa.

En la Tabla 2.3 se muestra las características del protocolo.

Parámetros	**RS-485**
Modo de trabajo	Diferencial
Número de emisiones y receptores	32 emisores y32 receptores
Longitud máxima del cable	1.200 metros
Velocidad de transmisión máxima	Hasta 10 Mbps.
Número de líneas	2 (datos y control por software).
Tipo de cable	Par trenzado (1 par).
Topología que admiten	Punto a punto
	Multipunto
	Anillo
	Bus
Simultaneidad en la transmisión	Half Duplex
Tensión de salida del emisor	Sin cargar +/- 1,5 V
	Cargado +/- 6 V

Tabla 2.3: Características del protocolo RS-485.

La norma establece que el número máximo de equipos será de 32, pero con receptores de alta impedancia se pueden alcanzar los 256 equipos. Los adaptadores RS-485 utilizan una fuente de alimentación de 5 voltios para sus circuitos.

2.12. Prueba de concepto

Es una implementación, a menudo resumida o incompleta, de un método o de una idea, realizada con el propósito de verificar que el concepto o teoría en cuestión, es susceptible de ser explotada de una manera útil. La prueba de concepto se considera habitualmente un paso importante en el proceso de crear un prototipo realmente operativo.

2.13. Hardware libre

Hardware de código abierto (*Open source hardware*, OSHW) es aquel hardware cuyo diseño se hace disponible públicamente para que cualquier persona lo pueda estudiar, modificar, distribuir, materializar y vender, tanto el original como otros objetos basados en ese diseño. Los códigos del hardware (entendidos como los archivos fuente) estarán disponibles en un formato apropiado para poder realizar modificaciones sobre ellas.

Idealmente, el hardware de código abierto utiliza componentes y materiales de alta disponibilidad, procesos estandarizados, infraestructuras abiertas, contenidos sin restricciones, y herramientas de fuentes abiertas de cara a maximizar la habilidad de los individuos para materializar y usar el hardware. Este hardware da libertad de controlar la tecnología y al mismo tiempo compartir conocimientos y estimular la comercialización por medio del intercambio abierto de diseños.

2.14. Raspberry Pi

Raspberry Pi es una computadora de placa reducida, computadora de placa única o computadora de placa simple (SBC, *Single board computer*) de bajo costo desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas, es un producto con propiedad registrada pero de uso libre. De esa forma la fundación mantiene el control de la plataforma pero permitiendo su uso libre tanto a nivel educativo como particular.

La Tabla 2.4 nos detalla las especificaciones técnicas y la Figura 2.11 muestra el modelo Raspberry Pi 2 B.

Raspberry Pi 2 Módelo B	
SOC	Broadcom BCM2836
CPU	ARM11 ARMv7 ARM Cortex-A7 4 núcleos @ 900 MHz
Overclocking	Sí, hasta arm-freq=1000 sdram-freq=500 core-freq=500 over-voltage=2 de forma segura
GPU	Broadcom VideoCore IV 250 MHz OpenGL ES 2.0
RAM	1 GB LPDDR2 SDRAM 450 MHz
USB 2.0	4
Salida de Vídeo	HDMI 1.4 @ 1920x1200 píxeles
Almacenamiento	microSD
Ethernet	Sí, 10/100 Mbps
Tamaño	85,60x56,5 mm
Peso	45 g
Consumo	5 V, 900mA, aunque depende de la carga de trabajo de los 4 cores

Tabla 2.4: Especificaciones técnicas de Raspberry Pi 2 modelo B.

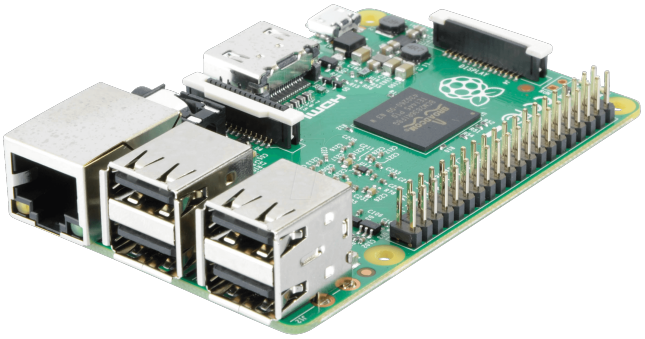


Figura 2.11: Modelo Raspberry Pi 2 B.

2.15. Arduino

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8 por su sencillez y bajo costo que permiten el desarrollo de múltiples diseños. Por otro lado el software consiste en un entorno de desarrollo que implementa

el lenguaje de programación *Processing/Wiring* y el cargador de arranque (*boot loader*) que corre en la placa.

Al ser de hardware libre, tanto su diseño como su distribución es libre. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

La plataforma Arduino se programa mediante el uso de un lenguaje propio basado en el popular lenguaje de programación de alto nivel *Processing*. Es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. También es utilizado por artistas y diseñadores como una herramienta alternativa al software propietario.

2.15.1. Arduino Mega 2560 R3

Es una placa de desarrollo basada en el ATmega2560. Tiene 54 pines de entradas/salidas digitales (15 pines pueden ser usados para señales de salida PWM). Tiene 16 entradas analógicas, 4 UARTs (hardware para puertos seriales), maneja una frecuencia de reloj de 16 MHz, una conexión USB, un conector de alimentación, una cabecera ICSP y un botón de reinicio, contiene todo lo que el microcontrolador necesita para funcionar en óptimas condiciones, tiene un conector de alimentación el cual puede ser conectado a través de cable USB, un adaptador AC-DC o una batería para que la placa pueda funcionar. Mega es compatible con la mayoría de los *shield* (hardware extraíble) para Arduino Duemilanove o Diecimila. El Mega 2560 difiere de sus predecesoras, puesto que no usa *drivers* de USB a FTDI para puerto serial, ya que esta necesidad la suple el ATMEGA16U2.

La Tabla 2.5 muestra las especificaciones técnicas de la placa Arduino Mega 2560 R3.

Arduino MEGA	
Microcontrolador	ATMEGA2560
Voltaje de operación	5V
Entrada de voltaje recomendada	7 V-12 V
Limites de entrada de voltaje	6 V-20 V
Pines digitales entradas/salidas	54 (15 pines para señal PWM a la salida)
Pines analógicos de entrada	16
Corriente de salida DC total I/O pin	40mA
Corriente DC por el pin de 3.3 V	50 mA
Memoria Flash	256 KB (8 KB para el arranque)
SRAM	8 KB
EEPROM	4 KB
Frecuencia de reloj	16 MHz

Tabla 2.5: Especificaciones técnicas Arduino Mega.

La Figura 2.12 muestra la placa Arduino Mega 2560 R3.

Figura 2.12: Modelo Arduino Mega 2560 R3.

2.15.2. Arduino Uno R3

Esta placa tiene un microcontrolador ATMEGA328, este tiene 14 pines de entradas/-salidas digitales (de los cuales 6 pines pueden ser usados para señales de salida PWM), 6 entradas analógicas, maneja una frecuencia de reloj de 16 MHz, se puede conectar mediante cable USB, posee una fuente de alimentación, cabecera ICSP, y un botón de reinicio, contiene todo lo que el microcontrolador necesita para funcionar en óptimas condiciones, tiene un conector de alimentación el cual puede ser conectado a través de cable USB, un adaptador AC-DC o una batería para que la placa pueda funcionar. Esta placa difiere de sus predecesoras, ya que no usa *drivers* de USB a FTDI para puerto serial, en lugar de ello, cuenta con el ATMEGA16U2 (ATMEGA8U2 hasta la versión R2) programado como conversor USB a serial.

La Tabla 2.6 muestra las especificaciones técnicas de la placa Arduino Uno R3.

Arduino UNO	
Microcontrolador	ATMEGA328
Voltaje de funcionamiento	5 V
Voltaje de entrada (recomendado)	7 V-12 V
Voltaje de entrada (límites)	6 V-20 V
Pines digitales entradas/salidas	14 (de los cuales 6 proporcionan PWM)
Pines analógicos de entrada	6
Corriente de salida DC I/O pin	40 mA
Corriente DC por el pin de 3.3 V	50 mA
Memoria Flash	32 KB (0.5 KB para el arranque)
SRAM	2 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz

Tabla 2.6: Especificaciones técnicas Arduino Uno R3.

La Figura 2.13 muestra la placa Arduino Uno R3.

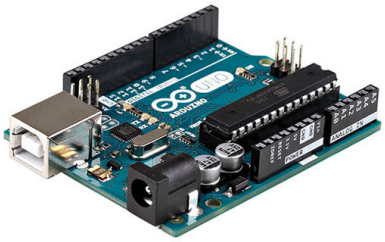


Figura 2.13: Modelo Arduino Uno R3.

2.15.3. Arduino Nano v3.0

Es una pequeña y completa placa basada en el ATMEGA328. Se usa conectándola a una *protoboard*. Tiene la misma funcionalidad que el Arduino Duemilanove, pero con una presentación diferente. No posee conector para alimentación externa, y funciona con un cable USB Mini-B en vez del cable estándar. Esta placa puede ser conectada a una computadora y así realizar programas interactivos en colaboración con software Flash, procesamiento, Max/MSP, PD, VVVV. Esta placa tiene todas las características de las placas Diecimila/Duemilanove, además de incluir más pines de entrada analógicas. Posee una cabecera ICSP para poder programar la placa.

La Tabla 2.7 muestra las especificaciones técnicas de la placa Arduino Nano v3.0.

Arduino NANO	
Microcontrolador	ATMEGA328
Voltaje de operación	5 V
Voltaje de entrada	7 V-12 V
Voltaje de entrada máximo	6 V / 20 V
Pines digitales entradas/salidas	14 (6 pines para señal PWM a la salida)
Pines analógicos de entrada	8
Corriente de salida DC I/O pin	40 mA
Memoria Flash	32 KB (2 KB para el arranque)
SRAM	2 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz

Tabla 2.7: Especificaciones técnicas Arduino Nano.

La Figura 2.14 muestra la placa Arduino Nano v3.0.

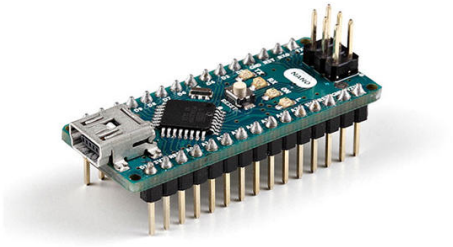


Figura 2.14: Modelo Arduino Nano v3.0

2.16. Sensor ultrasónico HC-SR04

Los ultrasonidos son señales acústicas cuyas frecuencias están por encima del rango de frecuencias sensibles al oído humano. Los sensores de ultrasonidos son capaces de medir la distancia a la que están respecto a un objeto por medio de un sistema de medición de ecos.

Estos sensores están formados por un transductor que emite un pulso corto de energía ultrasónica, cuando el pulso es reflejado por un objeto, el sensor captura el eco producido por medio de un receptor, y mediante un sistema de tratamiento de la señal, calcula la distancia a la que está de dicho objeto.

Debido a sus características se pueden encontrar sensores ultrasónicos en aplicaciones como medición de nivel (en tanques que contienen diferentes productos en forma líquida), control de colisiones en sistemas de aparcamiento, control de posición en campos como robótica, industria del plástico, control de llenado de tanques, entre otros.

Las principales ventajas de estos sensores se basan en que no necesitan contacto físico para poder detectar objetos, tienen una buena relación de calidad-precio y en comparación con otras tecnologías, los dispositivos basados en ultrasonidos son compactos y livianos.

El HC-SR04 es un dispositivo que está formado por un emisor y un receptor de ultrasonidos que opera a una frecuencia de 40 KHz. El sensor se alimenta a 5 V, por lo que se puede alimentar directamente desde Arduino, y puede llegar a detectar objetos hasta 5 metros con una resolución de 1 cm.

La Tabla 2.8 muestra las especificaciones técnicas del HC-SR04, y en la Figura 2.15 puede apreciarse el mismo.

Dispositivo Ultrasónico HC-SR04	
Modelo	HC-SR04
Voltaje de entrada	5 V
Corriente estática	<2 mA
Frecuencia de operación	20 KHz∼ 400 KHz
Angulo sensor	<15°
Distancia de detección	2 cm ∼ 450 cm
Precisión	0,3 cm
Entrada de señal de disparo	10 μs impulso TTL
Tamaño	4,5 cm x 2,0 cm x 1,3 cm

Tabla 2.8: Especificaciones técnicas del dispositivo ultrasónico HC-SR04.

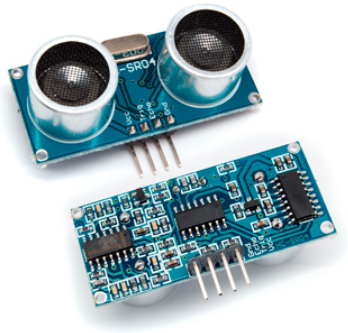


Figura 2.15: Dispositivo ultrasónico modelo HC-SR04.

2.17. *Shield* Ethernet

Este módulo permite a una placa Arduino conectarse a Internet. Está basada en el chip Ethernet Wiznet W5100. El Wiznet W5100 tiene una pila de red IP capaz de funcionar mediante los protocolos TCP y UDP. Soporta hasta cuatro conexiones de *sockets* simultáneas. Usa la librería Ethernet para escribir programas que se conecten a Internet usando el *shield*. El *shield* Ethernet dispone de unos conectores que permiten conectar a su vez otras placas encima y apilarlas sobre la placa Arduino.

Mediante esta placa y con sus respectivas librerías, se pueden implementar tanto un servidor web pequeño, como un cliente web. Incorpora un módulo de alimentación por Ethernet (PoE, *power over Ethernet*), en pocas palabras esto es para que se pueda alimentar la placa desde la red Ethernet que se tenga. La configuración de red se realiza mediante software, por lo que se puede adaptar con facilidad esta placa a una red local. Dispone de un zócalo para tarjetas de memoria micro-SD con el fin de almacenar archivos o servirlos como servidor web embebido. También incluye un controlador de reinicio automático para que el chip interno W5100 se encuentre en óptimas condiciones al momento de reiniciarlo y listo para utilizar al arranque.

La Tabla 2.9 muestra las especificaciones técnicas del *shield* Ethernet y en la Figura 2.16 se puede apreciar el modelo.

***Shield* Ethernet**	
Compatible	IEEE802.3af
Rizado de salida baja y el ruido	100 mVpp
Voltaje de entrada	36 V - 57 V
Protección	Sobrecarga y cortocircuito
Voltaje de salida	9 V
Salida de alta eficiencia convertidor DC / DC	75 %-50 % de carga
Voltaje de aislamiento (entrada a salida)	1.500 V
Conector estándar	RJ45 Ethernet

Tabla 2.9: Especificaciones técnicas del *shield* Ethernet.

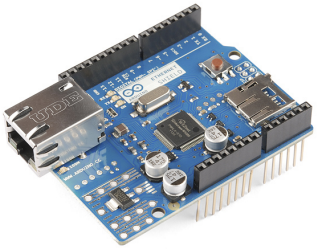


Figura 2.16: Placa *shield* Ethernet.

2.18. Módulo conversor TTL a RS-485

Esta placa utiliza el chip MAX485, el cual es un transceptor de baja potencia con una velocidad de respuesta limitada, y es usado para la comunicación RS-485.

El MAX485 exige corriente de alimentación en condiciones sin carga de alrededor de 120 μA y 500 μA con carga completa cuando el conductor está deshabilitado. El conductor está limitado por la corriente de corto-circuito y las salidas del controlador se pueden colocar en un estado de alta impedancia a través del circuito por medio de apagado térmico. La entrada del receptor tiene una característica a prueba de fallos que garantiza la salida lógica alta si la entrada está en circuito abierto. Además, tiene un buen comportamiento anti-interferencia.

La placa permite la adopción de una comunicación *half-duplex* para implementar la función de conversión de nivel de lógica transitor a transitor (TTL, *transistor-transistor logic*) a RS-485, puede alcanzar una velocidad máxima de transmisión de 2,5 Mbps.

La Tabla 2.10 muestra las especificaciones técnicas del módulo conversor TTL a RS-485.

Módulo conversor TTL a RS-485	
Modelo	MAX-485
Voltaje de operación	5 V
Corriente de operación	120 μA $\sim$ 500 μA
Potencia de operación	830 mW

Tabla 2.10: Especificaciones técnicas módulo conversor TTL a RS-485.

La Figura 2.17 muestra un módulo conversor TTL a RS-485.

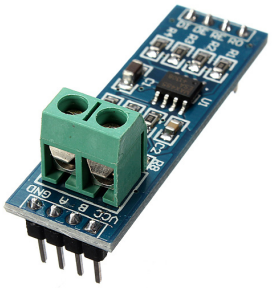


Figura 2.17: Módulo conversor TTL a RS-485.

Capítulo 3

Diseño

En este capítulo se expondrán los aspectos técnicos tenidos en cuenta para el diseño del trabajo.

Esta fase inicia con los casos de uso del sistema, seguido por la definición de los requerimientos funcionales y no funcionales. Y basándose en lo establecido anteriormente, se realiza el diseño de la arquitectura del sistema.

3.1. Casos de uso

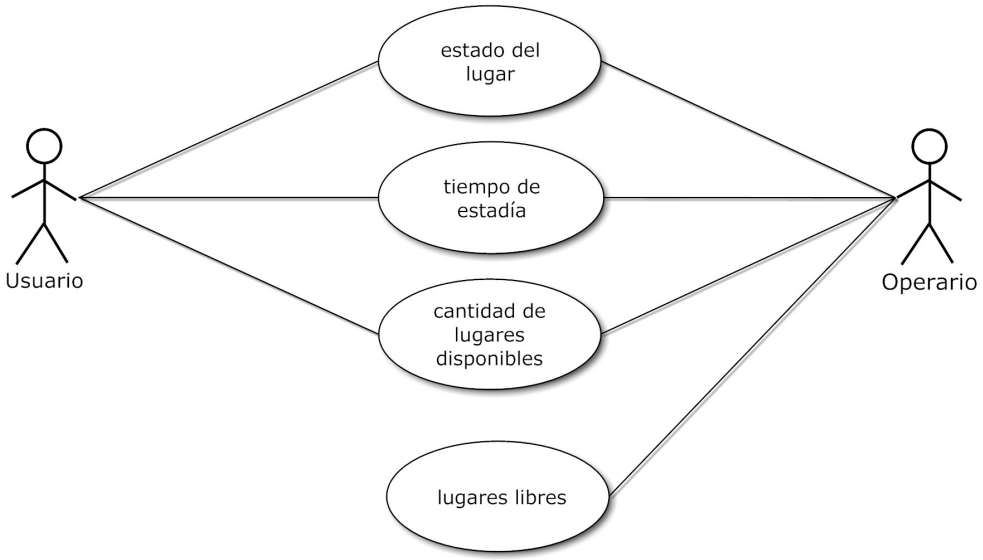


Figura 3.1: Caso de uso general.

3.1.1. CU-01

- Caso de uso: detección/notificación de presencia.
- Actor: usuario.
- Objetivo: conocer si existe un vehículo en el lugar.
- Condiciones previas: sistema energizado, estable y con conectividad.
- Escenario: el usuario ocupa un lugar bajo el sensor.
- Condiciones posteriores: el usuario verifica el cambio de estado mediante la señal lumínica e inicia el sistema el conteo del tiempo.

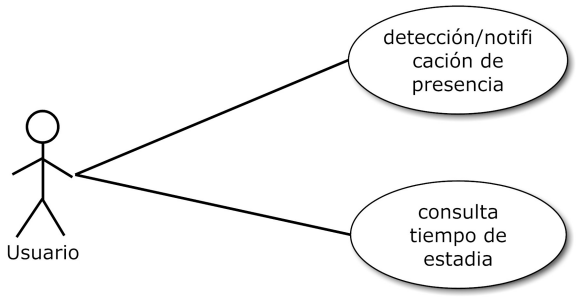


Figura 3.2: Caso de uso CU-01.

3.1.2. CU-02

- Caso de uso: notificar tiempo de estadía.
- Actor: usuario.
- Objetivo: conocer el tiempo de estadía del vehículo.
- Condiciones previas: sistema energizado, estable, con conectividad y posición ocupada.
- Escenario: el usuario consulta vía web el tiempo de estadía de la posición ocupada.
- Condiciones posteriores: el usuario verifica el tiempo de estadía mediante la pantalla de la web.

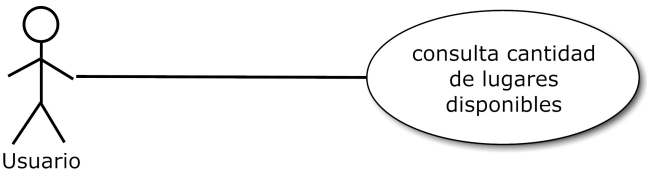


Figura 3.3: Caso de uso CU-02.

3.1.3. CU-03

- Caso de uso: cantidad de lugares disponibles.
- Actor: usuario.
- Objetivo: conocer disponibilidad de lugares disponibles para estacionar.
- Condiciones previas: sistema energizado, estable y con conectividad.
- Escenario: el usuario consulta vía web la cantidad de lugares disponibles para estacionar.
- Condiciones posteriores: el usuario verifica la cantidad de lugares disponibles a través de la web en pantalla.

3.2. Definición de requerimientos

3.2.1. Requerimientos funcionales

- Detectar de manera automática lugares disponibles.
- Identificar mediante una luz testigo el lugar disponible dentro de la playa de estacionamiento.
- Contabilizar el tiempo de permanencia del automóvil en el estacionamiento.

- Poder consultar vía web desde cualquier dispositivo conectado a Internet la información sobre lugares disponibles, tiempo de ocupación de un lugar, alertas y cualquier otra información de interés.
- Deberá ser capaz de mantener sus estados frente a micro cortes de energía o conectividad.
- Capacidad de almacenar el estado de los sensores en una base de datos.

3.2.2. Requerimientos no funcionales

- Debe poseer un sistema de energía ininterrumpido.
- Contar con conexión a Internet.
- Capacidad de funcionar en ambientes cerrados.
- Bajo consumo.
- Estable.

3.3. Arquitectura del sistema

Como se aprecia, en la Figura 3.4, el sistema está compuesto por dispositivos terminales, nodos concentradores, *gateway* y un servidor. Los dispositivos terminales son los encargados de capturar los datos provenientes de los sensores para luego ser enviados a través del nodo concentrador y el *gateway* al servidor.

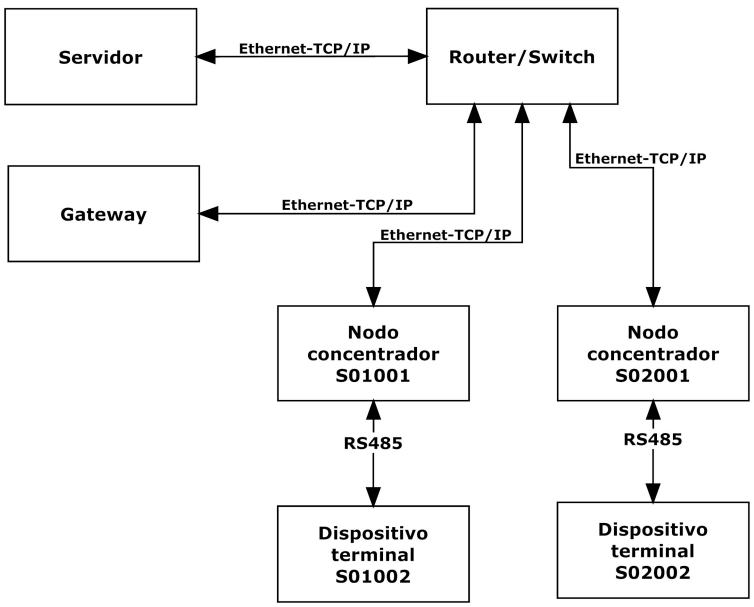


Figura 3.4: Arquitectura del Sistema.

En este trabajo se diseñaron e implementaron los dispositivos terminales, los nodos concentradores, *gateway* y el servidor central con tecnologías basadas en sistemas embebidos. Para ambos casos se utilizaron placas de desarrollo basadas en microcontrolador y microprocesador.

Tanto los dispositivos terminales, como los nodos concentradores y el *gateway* son alimentados con una fuente *switching* de 12 V. Mientras que el servidor montado en una Raspberry Pi 2 se alimenta con una fuente de 5 V.

3.3.1. Servidor

El servidor es el encargado de recibir, almacenar y procesar los datos de los dispositivos terminales, y permite el acceso de usuarios o clientes a la información a través de una página web. Se encuentra montado sobre una computadora Raspberry Pi 2 modelo B.

Para la interfaz de usuario se utiliza un servidor web mediante el cual se diseñan diferentes instancias (páginas web individuales) con el fin de satisfacer las necesidades tanto del operario o dueño de la cochera así como también la de los clientes o usuarios finales.

3.3.1.1. Interfaces de usuario

El servidor central a través de Internet y su servidor web permitirá la consulta de la información obtenida. Esta capacidad de comunicación con Internet es clave para los sistemas que conforman una red de la *IoT*.

Esta información presente en el servidor podrá ser consultada a través de una interfaz de usuario, por ejemplo, usuarios que deseen conocer la cantidad de lugares libres disponibles en ese establecimiento, o un usuario final que desea conocer cuál es el tiempo de estadía su automóvil en el establecimiento.

Para la llevar a cabo estos objetivos se deben plantear dos tipos de usuarios diferentes, un operador el cual será el encargado de administrar o controlar los tiempos de ocupación, que demandará información adicional en tiempo real y un usuario final o cliente cuya información deberá ser más acotada.

Para cumplir con tales propósitos se diseñan un conjunto de páginas web separadas en cuatro instancias diferentes. Siendo estas instancias las siguientes:

- Control de operario.
- Capacidad por nivel.
- Consulta de estadía.
- Consulta de capacidad.

La Figura 3.5 muestra la estructura de página web y las cuatro instancias.

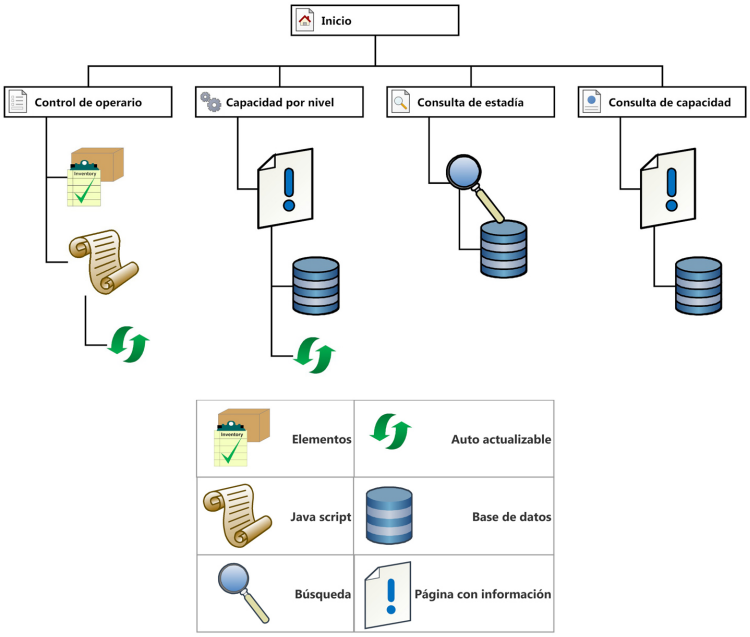


Figura 3.5: Diagrama de estructura web.

Control de operador Esta pantalla será de uso exclusivo del operador del establecimiento, mostrará los elementos de interés tanto de los nodos como los dispositivos terminales, permitiendo observar en tiempo real el estado de los mismos y detalles de interés, tales como, los minutos de activación, calidad del enlace, entre otros.

Para realizar esta instancia se utilizará la interfaz de usuario generada por openHAB y el *framework* Souliss, que estableciendo los elementos de interés a través de *scripts* en Java permitirán desde el navegador web realizar un control de los elementos de cada nodo concentrador y dispositivo final del establecimiento.

Capacidad por nivel La instancia capacidad por nivel será una página auto actualizable, la cual mediante un *script* PHP realizará la consulta a la base de datos del servidor mostrando la cantidad de lugares libres sobre la capacidad por nivel.

Consulta de estadía Instancia dedicada exclusivamente a los usuarios finales los cuales a través de un menú simple accionará un *script* PHP que consulta la base de datos para mostrar el tiempo de estadía de una posición de interés.

Consulta de capacidad Esta instancia será realizada para los posibles usuarios finales, los cuales a través de Internet, en cualquier lugar del mundo, podrán consultar la cantidad de lugares disponibles en el establecimiento mostrados en una página web a través de un *script* PHP que se encargará de realizar una consulta a la base de datos del servidor obteniendo la capacidad del mismo.

3.3.2. Gateway

El *gateway* se encuentra montado sobre una placa de desarrollo Arduino Mega 2560 R3 con un *shield* Ethernet a través de la interfaz de periféricos serie (SPI, *Serial peripheral interface*) para brindar conectividad con la red.

La Figura 3.6 muestra el diagrama de bloques del *gateway*.

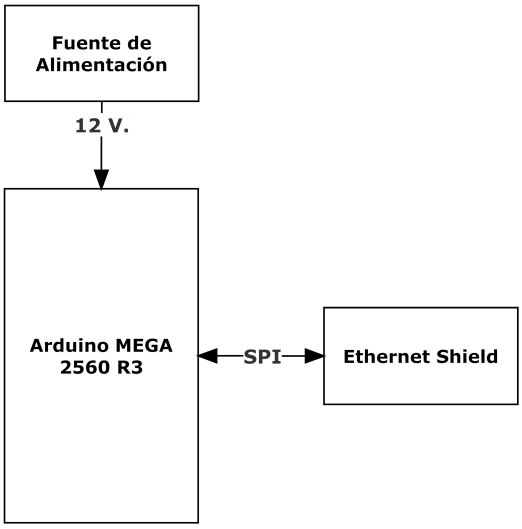


Figura 3.6: Diagrama de bloques del *gateway*.

3.3.3. Nodo concentrador

Este elemento de la red montado sobre un Arduino Uno, posee dos interfaces de comunicación, un *shield* Ethernet a través de SPI y un módulo conversor TTL a RS-485 a través de un receptor/transmisor síncrono/asíncrono universal (USART, *universal synchronous/asynchronous receiver/transmitter*), en sus pines digitales se encuentran conectados los LEDs que harán de luces testigos de los estados y el dispositivo ultrasónico HC-SR04 el cual será el encargado de medir la distancia para su posterior tratamiento.

En la Figura 3.7 se aprecia el diagrama de bloque del nodo concentrador.

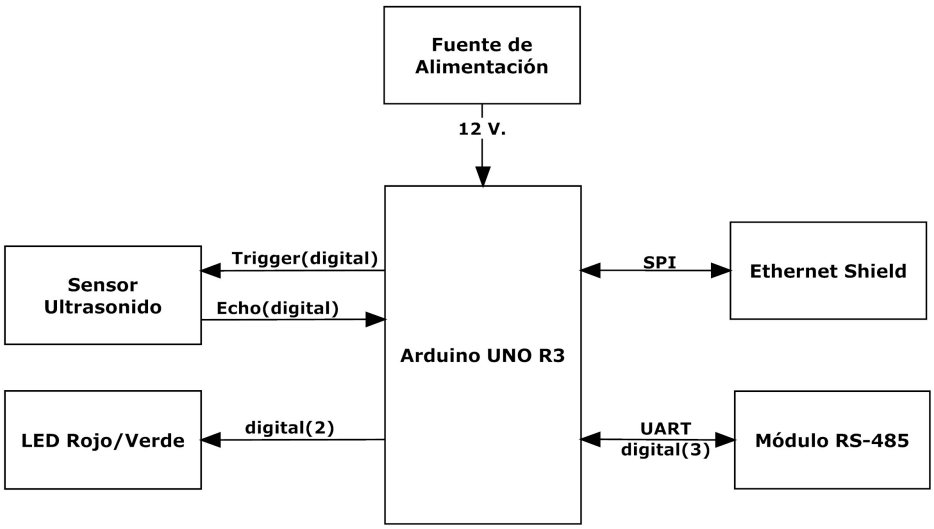


Figura 3.7: Diagrama de bloques del nodo concentrador.

En la Figura 3.8 se aprecia el diagrama de flujo del funcionamiento del nodo concentrador.

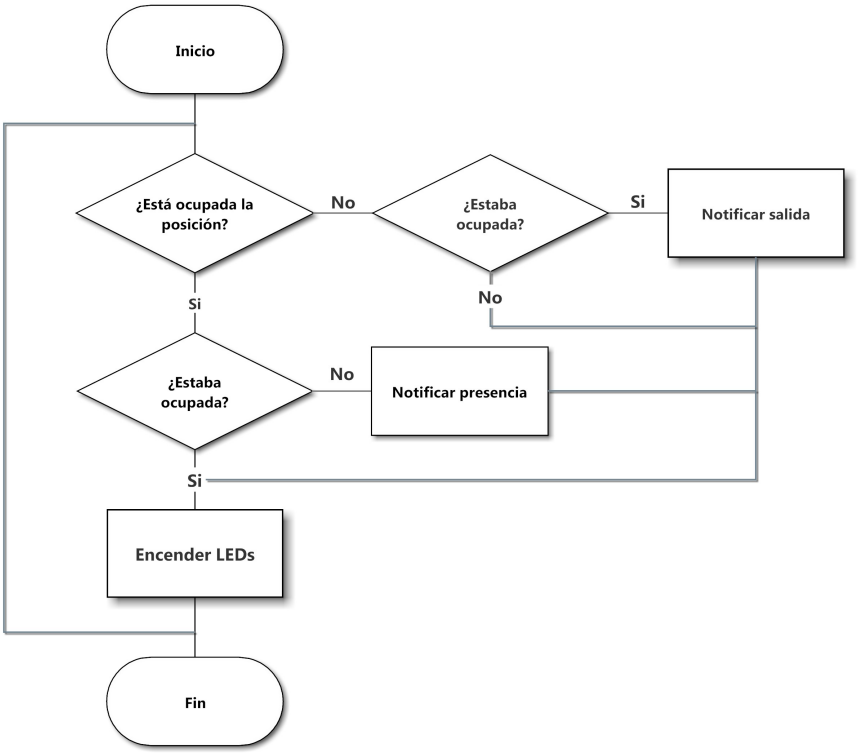


Figura 3.8: Diagrama de flujo del nodo concentrador.

3.3.4. Dispositivo terminal

Los dispositivos terminales se encuentran diseñados sobre Arduino Nano v3.0, posee una interfaz de comunicación con su módulo conversor TTL a RS-485 a través de USART,

a sus pines digitales se encuentran conectados los LEDs que hacen de luces testigo de los estados y el sensor ultrasónico HC-SR04.

Para su funcionamiento utilizará la misma lógica del diagrama de flujo del concentrador, Figura 3.8.

En la Figura 3.9 se aprecia el diagrama de bloque del dispositivo terminal.

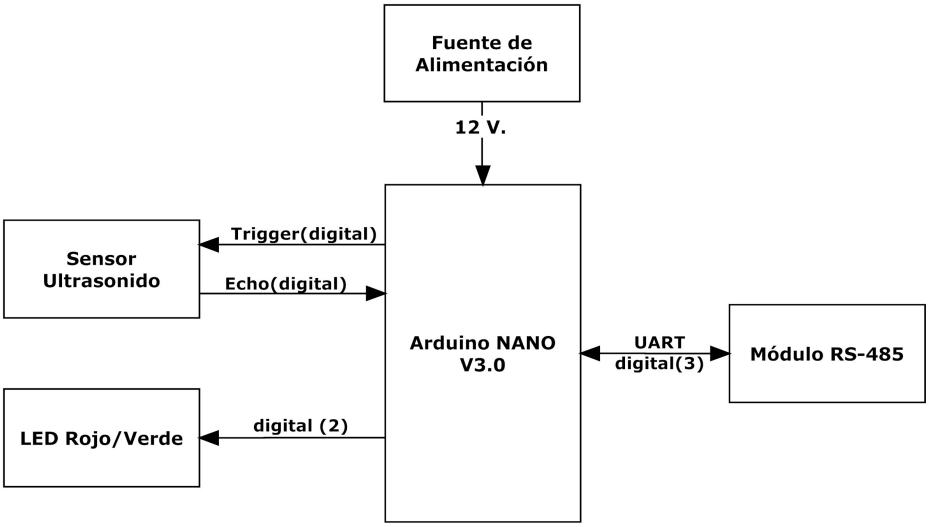


Figura 3.9: Diagrama de bloques del dispositivo terminal.

Capítulo 4

Implementación

En este capítulo se explica cómo se realizó la implementación de la prueba de concepto, desglosando la arquitectura del sistema en dos partes bien diferenciadas, una plataforma de software compuesta principalmente por el servidor y sus funciones, y una plataforma de hardware integrada por el *gateway*, nodos concentradores y dispositivos terminales.

4.1. Arquitectura detallada del sistema

La arquitectura del sistema quedará conformada por la plataforma de hardware y la plataforma de software, la integración de las mismas se basa en una comunicación TCP/IP entre el servidor montado en la computadora Raspberry Pi y el *gateway* sobre una placa Arduino Mega. Esta comunicación se realiza en un nivel aplicación entre el *framework* Souliss que se ejecutan sobre openHAB y las funcionalidades Souliss que se ejecutan sobre la placa Arduino Mega. Mientras que la comunicación entre el *gateway* y el nodo concentrador será TCP/IP y la comunicación del concentrador y los dispositivos terminales será a través de RS-485.

En la Figura 4.1 se muestra la interacción entre los elementos que componen el sistema.

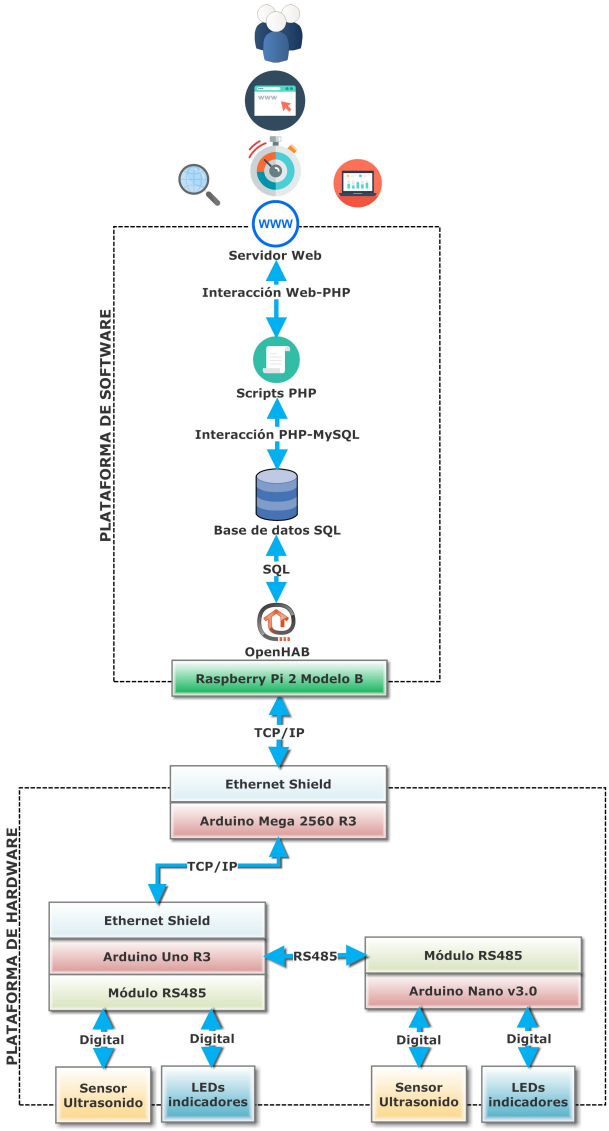


Figura 4.1: Arquitectura del sistema donde se muestra la interacción entre los elementos.

4.2. Plataforma de hardware

La plataforma de hardware, compuesta por el *gateway*, nodos concentradores y dispositivos terminales, es la encargada de recolectar los datos obtenidos por los sensores y de notificar el estado de los mismos al *gateway* para su posterior envío de información al servidor. En esta plataforma conviven diferentes medios físicos de comunicación, así como también diferentes protocolos de comunicación.

A continuación se explica el rol que ocupa cada elemento de esta plataforma en el sistema.

4.2.1. Sensor ultrasónico HC-SR04

El sensor basa su funcionamiento en simplemente medir el tiempo entre el envío y la recepción de un pulso ultrasónico. Sabiendo que la velocidad del sonido, en condiciones de temperatura a 20 ºC, 50 % de humedad y presión atmosférica a nivel del mar, es de 343 m/s y transformando unidades resulta

$$343\frac{m}{s} \cdot 100\frac{cm}{m} \cdot \frac{1}{1000000}\frac{s}{\mu} = \frac{1}{29,2}\frac{cm}{\mu s}$$

Es decir, el sonido tarda 29,2 microsegundos en recorrer un centímetro. Por tanto, se puede obtener la distancia a partir del tiempo entre la emisión y recepción del pulso mediante

$$Distancia(cm) = \frac{Tiempo(\mu s)}{29,2 \cdot 2}$$

El motivo de dividir en dos el tiempo (además de la velocidad del sonido en las unidades apropiadas, que se ha calculado antes) es porque se mide el tiempo que tarda el pulso en ir y volver, por lo que la distancia recorrida, como muestra la Figura 4.2, por el pulso es el doble de la que se pretende medir.

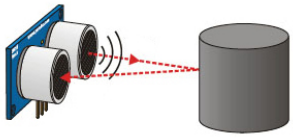
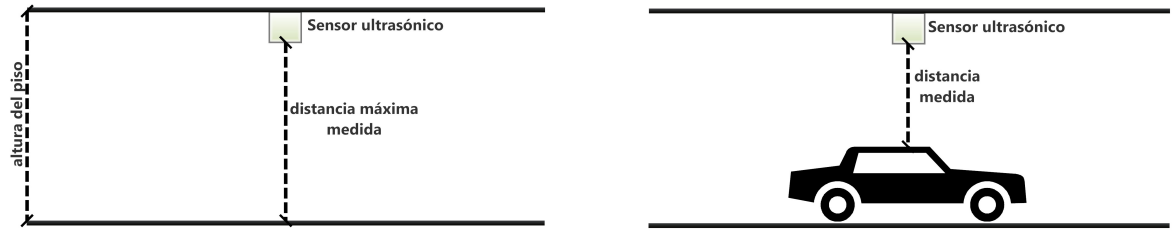


Figura 4.2: Para el funcionamiento del sensor ultrasónico HCSR-04 se mide el tiempo que tarda el pulso en ir y volver.

Este sensor será el encargado de detectar la presencia o no del automóvil. Dado que la altura donde se instalará el sensor es fija y conocida, como se puede observar en la Figura 4.3 (a), el sensor tendrá una altura de referencia, al estacionarse un auto bajo el sensor produce una disminución significativa en la medición de distancia realizada lo que se traduce en que un automóvil ha ocupado el lugar, como se puede observar Figura 4.3 (b).



(a) Medición sin automóvil

(b) Medición con automóvil

Figura 4.3: Utilización del sensor para conocer la presencia del automóvil.

4.2.2. *Shield* Ethernet

Este módulo permite a una placa Arduino conectarse a Internet, basado en el chip Ethernet Wiznet W5100 es capaz de funcionar con los protocolos de control de transmisión (TCP, *Transmission control protocol*) y protocolos de datagrama de usuario (UDP, *User datagram protocol*).

El *shield* Ethernet dispone de unos conectores que permiten conectar a su vez otras placas encima y apilarlas sobre la placa Arduino. La configuración de red se realiza mediante software, por lo que se puede adaptar con facilidad esta placa a la red local.

Siendo compatible con las placas Arduino Mega y Arduino Uno, resulta indispensable su uso para permitir la conectividad del *gateway* y de los nodos concentradores en la red.

4.2.3. *Gateway*

El *gateway* del sistema es el elemento encargado de la comunicación entre los nodos concentradores y dispositivos terminales, y la aplicación openHAB con el *binding* Souliss.

Informando la arquitectura de la red a openHAB y a Souliss en Arduino con el direccionamiento de los nodos correspondientes.

Una vez subscripto a openHAB, se encarga de comunicar los cambios de estados o otra información que se requiera.

Este elemento de la red demanda mayor procesamiento y memoria disponible, haciendo que una placa Arduino Uno R3 se vuelva inestable y presente reinicios automáticos. Es por esto que el *gateway* es montado sobre una placa Arduino Mega 2560 R3 con un *shield* Ethernet, este modelo de placa posee mayor capacidad de procesamiento y memoria por lo que el *gateway* correrá sin inconvenientes.

La Figura 4.4 muestra el esquema del *gateway*.

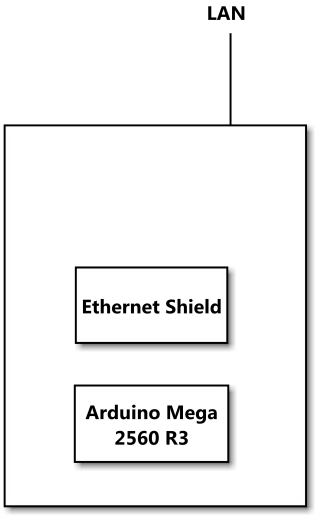


Figura 4.4: Esquema *gateway*.

4.2.4. Nodo concentrador

Montado sobre un Arduino Uno con un *shield* Ethernet y un módulo conversor TTL a RS-485 es el encargado de administrar los nodos adyacentes a él y enviar las actualizaciones al *gateway*.

La Figura 4.5 muestra el esquema del nodo concentrador, mientras que la Figura 4.6 muestra el conexionado del mismo.

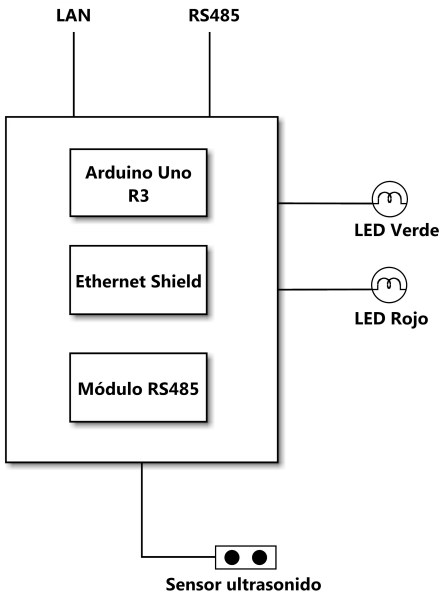


Figura 4.5: Esquema nodo concentrador.

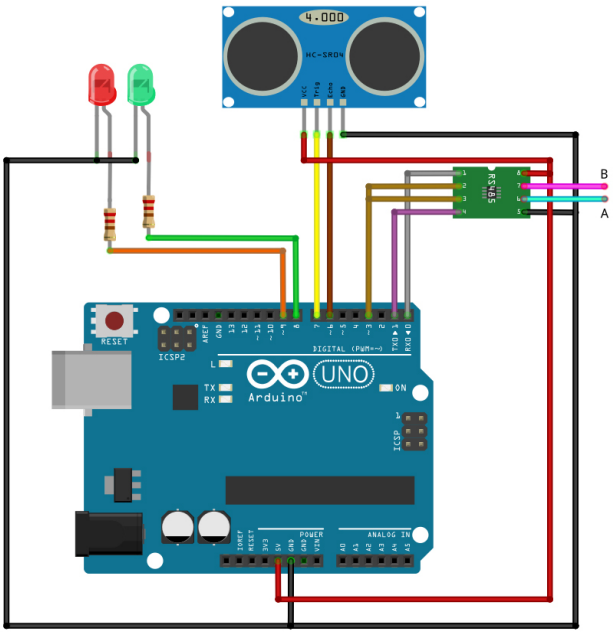


Figura 4.6: Conexionado del nodo concentrador.

Este dispositivo permite la conectividad entre una red Ethernet TCP/IP y una red RS-485, como muestra la Figura 4.7

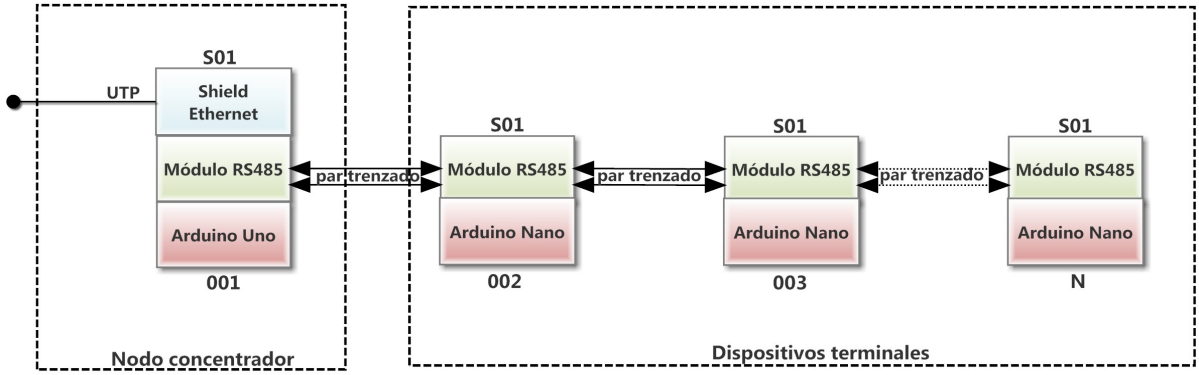


Figura 4.7: Interfaces y arquitectura de nodo concentrador y dispositivos terminales.

A través de su interfaz Ethernet se conecta a la red local identificándose con una IP fija de conocimiento en la arquitectura de red del *gateway*, y haciendo a su vez de *gateway* RS-485 de los dispositivos terminales adyacentes conectados a el mediante un par trenzado al módulo RS-485.

Posee también conectada una pequeña placa accesoria con un sensor ultrasónico HC-SR04 y LEDs, por lo que es también un nodo de detección/notificación de la posición donde esté ubicado.

4.2.5. Dispositivo terminal

Este elemento del sistema se encuentra montado sobre una placa Arduino Nano v3.0 debido a que no requiere demasiado procesamiento, posee un sensor de ultrasonido que se encarga de detectar/notificar con sus LEDs la presencia de un vehículo. Como se ve en la Figura 4.7 forma parte de la red cableada RS-485 de un par cruzado a través del módulo TTL a RS-485.

Este dispositivo terminal reporta sus cambios de estados al nodo concentrador para que posteriormente a través de la red Ethernet lo notifique al *gateway*.

La Figura 4.8 muestra el esquema del dispositivo terminal, mientras que la Figura 4.9 muestra el conexionado del mismo.

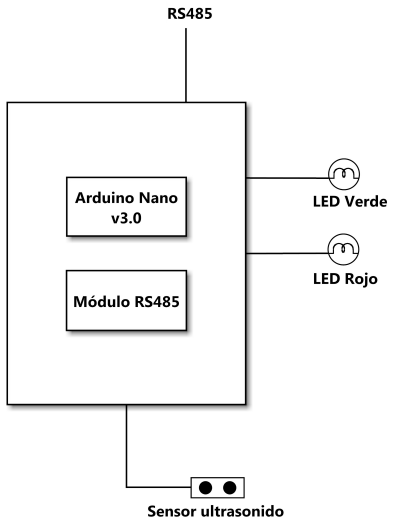


Figura 4.8: Esquema dispositivo terminal.

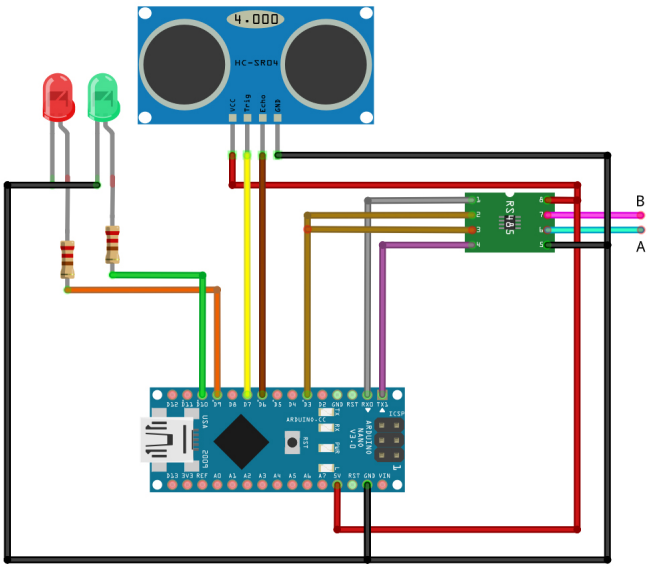


Figura 4.9: Conexionado dispositivo terminal.

4.2.6. Test del hardware

4.2.6.1. Sensor ultrasónico HC-SR04

Para utilizar este sensor se usan como elementos de prueba una placa Arduino Uno y un sensor HC-SR04 haciendo uso de la librería Ultrasonic HC-SR04 [21] con el código de prueba, véase código fuente 4.1.

```
#include <Ultrasonic.h>
Ultrasonic ultrasonic(6,5); // (Trig PIN,Echo PIN)
void setup() {
Serial.begin(9600);
}
void loop()
{
Serial.print(ultrasonic.Ranging(CM)); // CM or INC
Serial.println(" cm" );
delay(100);
}
```

Código fuente 4.1: Código fuente para test de librería Ultrasonic HC-SR04.

En primera medida se conecta el *trigger* del sensor al pin 6 de Arduino y el *echo* al pin 5, como muestra la Figura 4.10.

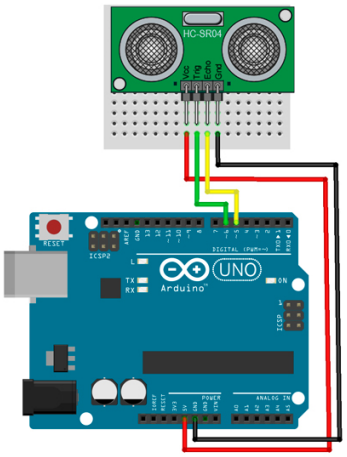


Figura 4.10: HC-SR04 conexionado en Arduino Uno.

Luego de realizar mediciones a diferentes distancias se concluye que el sensor responde con inconvenientes para distancias mayores a 310 cm y menores a 8 cm.

Para distancias mayores a 320 cm se debe de realizar una modificación en el *timeout*, para así definir una distancia máxima a medir. Debido que dicho sensor se encontrará ubicado a alturas no mayores a 300 cm se opta poner un *timeout* de 18000 microsegundos, que resulta medir hasta 310 cm como máximo.

Para las distancias menores a 8 cm oscila dejando incertidumbre, esto se debe a la separación entre el generador de ondas ultrasónicas y el receptor, que generan un triángulo con el objeto, como se observó en la Figura 4.2, al bajar la distancia del objeto a una distancia umbral las ondas ya no son capaces de llegar al receptor.

Realizando distintas mediciones, se ubica el objeto a 10 cm y la medición se torna estable, por lo que a la hora de realizar las mediciones se pedirá que el automóvil se

encuentre a mas de 10 cm del sensor, y por lo tanto siendo menor a esta distancia se descartará debido a las variaciones anteriormente explicadas.

La Figura 4.11 muestra la prueba con *protoboard* y el objeto a 10 cm de distancia.

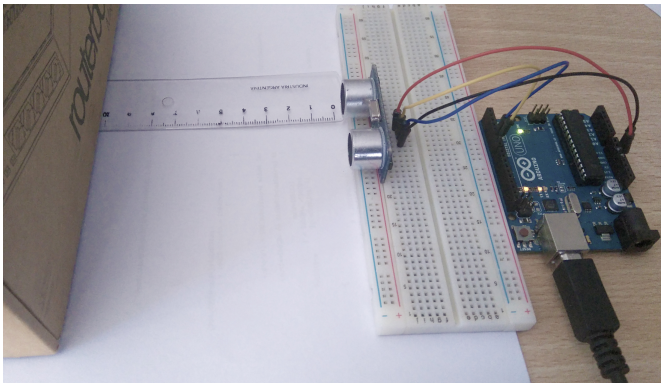


Figura 4.11: Prueba con *protoboard* y objeto a 10 cm.

En la Figura 4.12 se puede apreciar la captura desde IDE de la medición con el objeto a 10 cm de distancia.

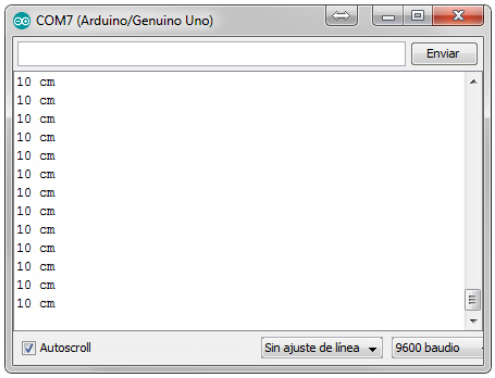


Figura 4.12: Captura de IDE de medición con objeto a 10 centímetros.

4.2.7. Firmware

La mayoría de estos conceptos se encuentran explicados en detalle en el capítulo marco teórico, sólo se referirá a ellos y refrescará conceptos para facilitar la comprensión del firmware y su funcionamiento.

4.2.7.1. Librerías

4.2.7.2. Souliss *framework* y Souliss

La librería Souliss *framework* informa a Arduino IDE la ruta específica donde se encuentra la carpeta con los archivos necesarios para la configuración de Souliss, mientras que la librería Souliss incorpora las definiciones necesarias para el uso de sus funcionalidades y protocolos.

Funcionalidades de Souliss

- **Typicals:** son variables lógicas predefinidas. Cada *typical* tiene un conjunto definido de comandos de entradas y salida relevantes. Se almacenan en *slot*.
- **vNet:** es el protocolo de transporte de datos. Se utilizará para enrutar los diferentes medios de comunicación.
- **Protocolo de datos MaCaco:** es protocolo no orientado a la conexión y cada interacción está compuesta de sólo de dos mensajes: la solicitud y la respuesta. No hay confirmación (ACK) ni comprobación de coherencia, se supone que estos son llevados por la capa de transporte.

  En el modo de *subscribing*, un nodo inicia una comunicación con una petición de suscripción, que notifica el interés en algunos datos, después de una solicitud se sigue una respuesta inmediatamente y después cada vez que se produce un suceso en el nodo suscripto, se envía automáticamente un nuevo paquete.

  Este modo es elegido debido a que reduce el uso del canal, los datos se transmiten sólo si es necesario y libera recursos en el nodo.

4.2.7.3. StandardArduino

Librería que adapta el *framework* Souliss para ser grabado en placas de desarrollo Arduino.

4.2.7.4. ethW5100

Librería que define la comunicación para el *shield* Ethernet basado en el *chipset* W5100.

4.2.7.5. SPI

Librería encargada de la comunicación serie.

4.2.7.6. USART

Librería de Souliss para el uso de RS-485. Define parámetros estandarizados para el uso de RS-485, como la velocidad de transmisión, cantidad de bits, paridad, etc. También permite habilitar/deshabilitar la función para evitar colisiones o habilitar/deshabilitar el control de redundancia cíclica.

4.2.7.7. Gateway

Librería que habilita la función de *gateway* del nodo.

4.2.7.8. SuperNode

Esta librería es necesaria para el manejo de más de una interfaz de medio de comunicación. Junto con vNet permiten que un nodo tenga más de un direccionamiento, uno en cada red a la que pertenezca.

4.2.7.9. Ultrasonic

Librería utilizada para el cálculo de la distancia desde el sensor al objeto.

4.2.8. Direccionamiento

Para el funcionamiento del sistema es necesario utilizar dos interfaces de comunicación diferentes: Ethernet y RS-485.

La red LAN creada con el nombre de "myvNet_supern" tiene como medio físico de comunicación cable par trenzado sin blindaje (UTP, *unshielded twisted pair*), y como interfaz de comunicación utiliza el *shield* Ethernet. De esta red local forman parte el *gateway* y los nodos concentradores, donde cada uno de ellos tiene una dirección IP estática. Formando una topología de red tipo estrella.

Teniendo como medio físico de comunicación un cable de par trenzado y como interfaz de comunicación el módulo conversor TTL a RS-485, se crea una subred física tipo bus llamada "myvNet_supernCNX", perteneciendo a la subred donde los integrantes de la misma serán el nodo concentrador y los dispositivos terminales que se encuentren en el mismo bus de comunicación.

El intercambio de datos entre ambas redes, RS-485 y Ethernet, se logra gracias al protocolo de transporte vNet, cada nodo concentrador y dispositivo terminal posee una dirección identificadora del tipo 0xCE00 en la red RS-485.

La Tabla 4.1 muestra el significado de esta dirección.

0x	**Segmento de red RS-485**	**Nº Dispositivo**
0x	CE	01-FF
0x	CF	01-FF

Tabla 4.1: Direccionamiento RS-485.

Analizando la Tabla 4.1 es posible direccionar 255 dispositivos en cada segmento de red, siendo CE segmento inicial para la definición de RS-485, sin embargo según

especificaciones del *framework* Souliss el número máximo de nodos en la red puede ser de 127, incluyendo el *gateway*.

La Figura 4.13 muestra la red de manera lógica, con su direccionamiento.

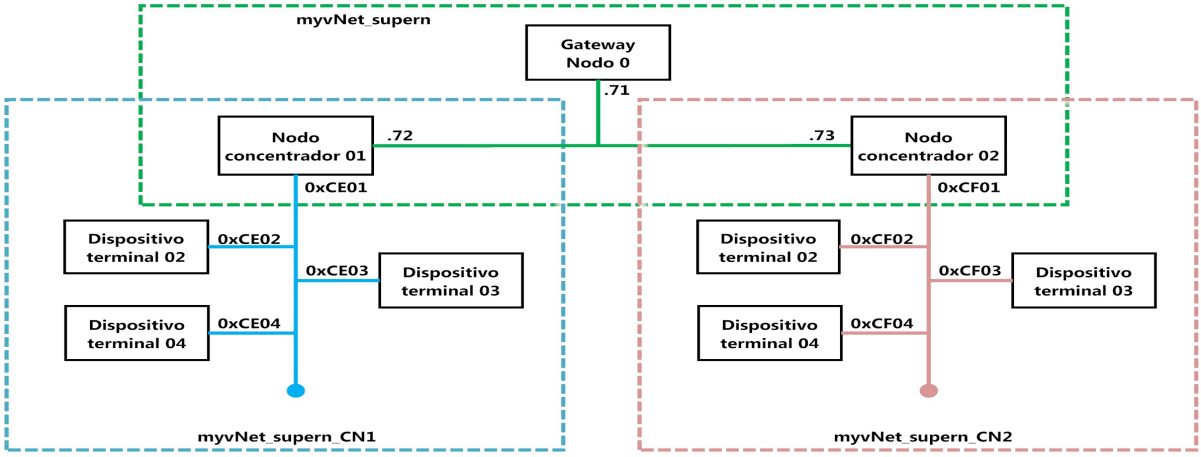


Figura 4.13: Direccionamiento de la red.

En todos los nodos se realiza declaración de direcciones según el Código fuente 4.2.

```
// -------------INICIO de la configuracion de RED ---------------
#define ip_Router      1
#define  ip_Gateway       71        // Direccion IP Gateway
#define ip_Concentrador_N1      72        // Direccion IP 1 Concentrador
#define  ip_Concentrador_N2        73        // Direccion IP 2 Concentrador
//agregar forma ip_Concentrador_NX 7X

uint8_t ip_address_Gw[4]  = {192, 168, 1, ip_Gateway};      // Direccionamiento Gateway (ARDUINO MEGA)
uint8_t ip_address_CN1[4]  = {192, 168, 1, ip_Concentrador_N1};    // Direccionamiento Concentrador 1  (Arduino UNO)
uint8_t ip_address_CN2[4]  = {192, 168, 1, ip_Concentrador_N2};    //  Direccionamiento Concentrador 2  (Arduino UNO)
uint8_t subnet_mask[4]    = {255, 255, 255, 0};
uint8_t ip_gateway_Router[4] = {192, 168, 1, ip_Router}; // Direccionamiento Gateway ROUTER

#define myvNet_address     ip_address_Gw[3]     // ultimo byte de la direccion IP del Gateway es tambien la direccion de Vnet
#define  myvNet_subnet       0xFF00
#define myvNet_supern        0x0000
#define  myvNet_supern_CN1  CN01_Sensor01_RS485 //red Concentrador 1 - Master RS485
#define  myvNet_supern_CN2  CN02_Sensor01_RS485  //red Concentrador 2 - Master RS485

#define CN01_Sensor01_RS485         0xCE01   // Direccion Concentrador 1 Sensor 01  con RS485 MASTER
#define CN01_Sensor02_RS485         0xCE02   // Direccion Concentrador 1 Sensor 02  con RS485
#define CN01_Sensor03_RS485         0xCE03   // Direccion Concentrador 1 Sensor 03  con RS485

#define CN02_Sensor01_RS485         0xCF01   // Direccion Concentrador 2 Sensor 01  con RS485 MASTER
#define CN02_Sensor02_RS485         0xCF02   // Direccion Concentrador 2 Sensor 01  con RS485
#define CN02_Sensor03_RS485         0xCF03   // Direccion Concentrador 2 Sensor 01  con RS485
#define CN02_Sensor04_RS485         0xCF04   // Direccion Concentrador 2 Sensor 01  con RS485
// ------------- FIN de la configuracion de RED ------------
```

Código fuente 4.2: Declaración y direccionamiento de la red.

4.2.9. Gateway

El *gateway* se utiliza sólo para que la interfaz de usuario obtenga todos los datos sin un conocimiento previo de la red. El resultado es una red distribuida para la lógica de ejecución pero centralizada para la recopilación de datos y que fluye desde un único nodo a una interfaz de usuario.

Para su mejor entendimiento se desglosa el código fuente en partes.

1. Librerías.
2. Direccionamiento y configuración.
3. Bucle principal.

4.2.9.1. Librerías

La primera parte es la inclusión de las librerías que permiten la utilización de Souliss, Código fuente 4.3, *shield* Ethernet y Gateway.

```
1 // -- Librerias para el funcionamiento de Souliss --
2 #include "SoulissFramework.h"       // Funcionalidades Souliss
3 #include "bconf/StandardArduino.h"    // Especifica que es un Arduino compatible
4 #include "conf/ethW5100.h"            // Habilita Ethernet Shield
5 #include "conf/Gateway.h"             // Habilita el gateway
6
7 #include <SPI.h>
8 #include "Souliss.h"          // Funcionalidades Souliss
```

Código fuente 4.3: Declaración de librerías del gateway.

4.2.9.2. Configuración de la red

Una vez enunciados los direccionamientos se proceden a hacer efectivos los mismos. Definiendo la dirección IP del router en la red, y luego los nodos involucrados en la red.

La línea 4 del Código fuente 4.4 realiza la conexión entre el *gateway* y el *router* de la red.

La línea 5 lo establece como *gateway*, registrando la dirección IP del *gateway* en donde openHAB encontrará la arquitectura de red, al definirse como *gateway* pasa a ser el "nodo 0".

Una vez definido como *gateway*, se debe definir todos los nodos involucrados en la red, ennumerandolos con la sentencia `SetAsPeerNode(peer_address, index)`, donde *index* será el valor de nodo que tendrá. En este caso el nodo concentrador 1 será el nodo 1 y el dispositivo terminal conectado a él será el nodo 2.

```
1  void setup()
2  {
3  Initialize();
4  Souliss_SetIPAddress(ip_address_Gw, subnet_mask, ip_gateway_Router);
5  SetAsGateway(myvNet_address);
6  SetAsPeerNode(ip_Concentrador_N1, 1);       // Concentrador 1 - Sensor 01 RS485
7  SetAsPeerNode(CN01_Sensor02_RS485, 2);      // Concentrador 1 - Sensor 02 RS485
8  SetAsPeerNode(ip_Concentrador_N2, 3);       //Concentrador 2 - Sensor 01 RS485
9  SetAsPeerNode(CN02_Sensor02_RS485, 4);      //Concentrador 2 - Sensor 02 RS485
10 }
```

Código fuente 4.4: Asignación de los nodos a la red.

Conocer el identificador de cada nodo es de vital importancia para la correcta configuración en openHAB.

4.2.9.3. Bucle principal

Debido a que el *gateway* sólo es el nexo entre OH y los nodos controladores, en su bucle principal, Código fuente 4.5, se encuentra solo el comando `FAST_GatewayComms();` el cual se utiliza para iniciar el proceso de comunicación para el *gateway*, esto incluye recuperar estados y *typicals*.

```
1 void loop()
2 {
3   EXECUTEFAST()
4         {
5     UPDATEFAST();
6         FAST_GatewayComms();
7         }
8 }
```

Código fuente 4.5: Bucle principal del gateway.

4.2.10. Nodo concentrador

Este nodo es un súper nodo, ya que necesita poder realizar el enrutamiento y puente de dos interfaces de comunicación diferentes.

4.2.10.1. Librerías

Las librerías que utiliza son las siguientes:

- SoulissFramework
- Souliss
- StandardArduino
- thW5100
- USART
- SPI
- Ultrasonic
- SuperNode

A la inclusión de librerías se debe agregar la definición de los pines involucrados en la placa Arduino para el manejo del RS-485.

De acuerdo a la Tabla 4.2, los pines RE y DE se puentean con el pin 3 debido a que la configuración utilizada para la transmisión es *half-duplex*, permitiendo así la transmisión de datos bidireccional pero no al mismo tiempo.

RS-485 Pin	Descripción	Uso USART.h	Arduino Pin
RO	Salida receptor	RX	pin 0
RE	Receptor habilitado (activo cuando este pin es BAJO)	CS	pin 3
DE	Controlador habilitado (activo cuando este pin es ALTO)	CS	pin 3
DI	Ingreso controlador (pin de transmisión)	TX	pin 1

Tabla 4.2: *Pinout* RS-485

El Código fuente 4.6 muestra la declaración de los pines a utilizar para la transmisión.

```
#include "SoulissFramework.h"
#include "bconf/StandardArduino.h"
#include "conf/ethW5100.h"
#include "conf/usart.h"
#include "conf/SuperNode.h"
#include <SPI.h>
#include <Ultrasonic.h>
#include "Souliss.h"

#define USARTDRIVER_INSKETCH
#define USART_TXENABLE          1
#define USART_TXENPIN           3
#define USARTDRIVER             Serial
/*************/
```

Código fuente 4.6: Definición de librerías del nodo concentrador.

4.2.10.2. Lógica de detección

La lógica de detección de presencia se basa en el diagrama de la Figura 4.14. Este diagrama establece que una vez iniciada la secuencia de detección, el sensor obtiene la medición haciendo uso de la librería Ultrasonic, esta medición obtenida debe ser mayor a 10 cm para que continúe el programa. Se estableció este valor debido que a distancias menores de 10 cm el sensor presenta variaciones significativas en las mediciones. Superada esta instancia, la medición obtenida se compara con el umbral establecido. Este umbral se fijará (dejando un porcentaje de más por las variaciones en los tamaños) de acuerdo a la ecuación que resulta de la Figura 4.15.

$$umbral \simeq alturadelpiso - (alturaautopromedio - 15\,\%alturaautopromedio)$$

La lógica de detección de presencia se representa en el diagrama de flujo de la Figura 4.14.

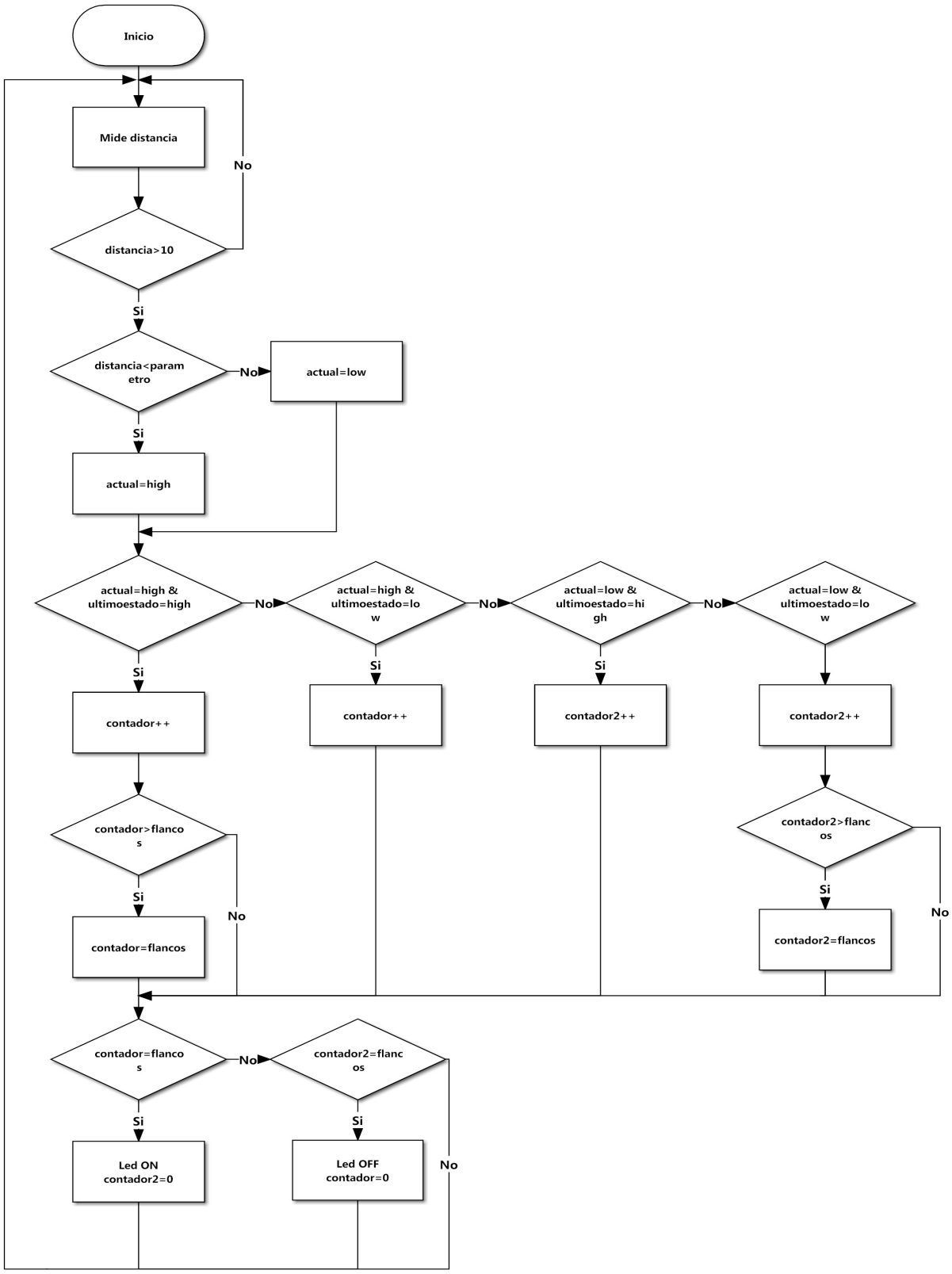


Figura 4.14: Diagrama de flujo del nodo concentrador.

La Figura 4.15 muestra las variables a tener en cuenta para la definición del umbral de detección.

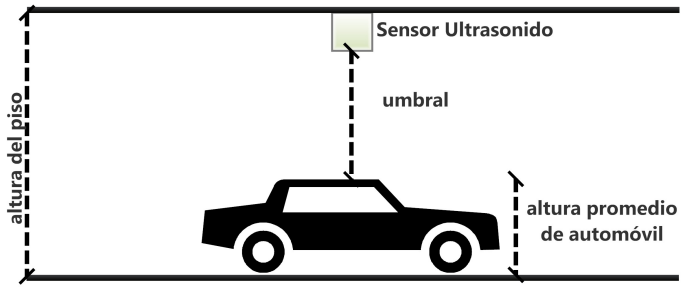


Figura 4.15: Figura con las variables a tener para la definición del umbral de detección.

Para esta prueba de concepto y para los test se establece este umbral en 50 cm.

Si la medición obtenida es menor que el umbral indica que existe un automóvil bajo el sensor, con esta afirmación se inicia una lógica de chequeo a través de contadores

Para notificar como ocupado el lugar y se active el LED rojo, deberá el contador alcanzar el valor de flancos, de esta manera se evita que el lugar se active por transeúntes o eventuales movimientos de otros automóviles.

Con la lógica de chequeo está establecida en 5 flancos y siendo el tiempo de la rutina de ejecución aproximadamente 2.110 ms, tardará aproximadamente 10,5 segundos, de mantenerse la medición por debajo del umbral, para indicar que el lugar está ocupado. De la misma manera una vez ocupado el lugar demorará la misma cantidad de flancos en desocuparse.

4.2.10.3. Declaración de variables

En esta parte del Código fuente 4.7, la declaración de variables establece también cuales son los pines usados para la utilización del sensor ultrasónico, así como también las variables involucradas en la lógica de detección.

```
1  #define LUZNODE1           0
2
3  Ultrasonic ultrasonic(7,6); // (Trig PIN,Echo PIN)
4
5  int distancia;
6  boolean ultimoestado=LOW;
7  boolean actual=LOW;
8  int contador=0;
9  int contador2=0;
10 int flancos=5;
11 int parametro=50;
12 #define LUZROJA 9
13 #define LUZVERDE 8
```

Código fuente 4.7: Declaración y direccionamiento de la red.

4.2.10.4. Setup y configuración de la red

La línea 4 del Código fuente 4.8 realiza la conexión entre el *gateway* y el *router* de la red, y además se establece como un nodo perteneciente a la subred myvNet_supern.

En la línea 7 definimos como *typical* T11 en el *slot* 0, siendo este el manejo de un dispositivo de salida digital con los comandos disponibles *ON/OFF*. Luego se definen los dos pines de salida a utilizar con los LEDs.

```
void setup()
{
Initialize();
Souliss_SetIPAddress(ip_address_CN1, subnet_mask, ip_gateway_Router);
Souliss_SetAddress(CN01_Sensor01_RS485, myvNet_subnet, myvNet_supern);

Set_T11(LUZNODE1);
pinMode(LUZROJA, OUTPUT);
pinMode(LUZVERDE, OUTPUT);
}
```

Código fuente 4.8: Configuración nodo concentrador.

4.2.10.5. Bucle principal

El bloque principal, Código fuente 4.9, ejecuta la lógica de detección cada 2110 ms, sí el contador alcanza el valor de flancos ejecuta una serie de comandos cuya acción es mantener en alto la salida LUZROJA y en bajo la salida LUZVERDE estableciendo contador2 en 0.

De esta manera se enciende la luz roja de ocupado el lugar y se establece el cambio en el *typical*.

```
void loop()
{
EXECUTEFAST()
{
UPDATEFAST();
FAST_2110ms()  {
chequeo: distancia=ultrasonic.Ranging(CM);
if (distancia<10) { goto chequeo;}
Serial.println(distancia);
if (distancia<=parametro)
{
actual=HIGH;
}
else actual=LOW;
if ((actual==HIGH) && (ultimoestado==HIGH))
{
contador++;
if (contador>flancos) contador=flancos;
}
if ((actual==HIGH) && (ultimoestado==LOW))  { contador2++; }
if ((actual==LOW ) && (ultimoestado==HIGH)) { contador++; }
if ((actual==LOW) && (ultimoestado==LOW))
{ contador2++;
if (contador2>flancos) contador2=flancos;
}
ultimoestado=actual;
if (contador==flancos)
{
mInput(LUZNODE1) = Souliss_T1n_OnCmd;
Logic_T11(LUZNODE1);
DigOut(LUZROJA, Souliss_T1n_OnCoil, LUZNODE1);
digitalWrite(LUZVERDE, LOW);
contador2=0;
}
if (contador2==flancos)
{
mInput(LUZNODE1) = Souliss_T1n_OffCmd;
Logic_T11(LUZNODE1);
DigOut(LUZROJA, Souliss_T1n_OnCoil, LUZNODE1);
digitalWrite(LUZVERDE, HIGH);
contador=0;
}
        }
```

```
44 FAST_PeerComms();
45 START_PeerJoin();
46 }
47 EXECUTESLOW()   {
48 UPDATESLOW();
49 SLOW_PeerJoin();
50 }
51 }
```

Código fuente 4.9: Bucle principal nodo concentrador.

El comando `FAST_PeerComms()` se encarga de procesar la comunicación estándar. Mientras que con `START_PeerJoin()` envía una trama de difusión para suscribirse al *gateway*.

La interacción entre la red Ethernet y la RS-485 se ve aplicada cuando una trama que tiene un destino fuera de la propia subred RS-485, es enrutada por el nodo controlador hacía la subred destino.

4.2.11. Dispositivo terminal

Este dispositivo ejecuta la misma lógica de detección de la Figura 4.14 que el nodo concentrador.

Para su funcionamiento sólo necesita saber cuál es su dirección de vNet y cuál es la dirección de vNet del nodo controlador. A través de la línea de código `Souliss_SetAddress (CN01_Sensor02_RS485, myvNet_subnet, CN01_Sensor01_RS485)` se define la dirección del nodo, a qué subred pertenece y cuál es el nodo concentrador a quien debe enviar la información.

Debido a que la conexión RS-485 puede llegar a fallar con el comando `SLOW_PeerJoin()` se busca que el nodo reconstruya la suscripción y comunicación en caso de reinicio del mismo.

4.3. Plataforma de software

4.3.1. Servidor

El servidor es uno de los elementos claves en la arquitectura de este sistema. Siendo la Raspberry Pi 2 modelo B, una computadora compacta, de software libre, de bajo costo y consumo, que además cuenta con una gran capacidad de procesamiento, es ideal por utilizar como servidor del sistema diseñado.

Su capacidad de procesamiento permite correr los servicios necesarios para el funcionamiento del sistema.

Teniendo un almacenamiento basado en una tarjeta microSD de 8 GB, proporciona el espacio suficiente para albergar el sistema operativo, Raspbian, los servicios y la base de datos del sistema.

Este pequeño ordenador posee además una interfaz Ethernet de 10/100 Mbps permitiendo la conectividad a Internet, por lo que en él se monta un servidor web, sobre Apache con PHP y SQL con el fin de poder gestionar las consultas de los usuarios a través de Internet.

También, esta computadora es la encargada de correr openHAB con el *binding* Souliss, permitiendo así la conectividad con el *gateway* del sistema y obteniendo los estados de los

sensores en los nodos concentradores y dispositivos terminales, prácticamente en tiempo real.

La salida HDMI y sus puertos USB permiten que esta computadora pueda ser utilizada tanto como una máquina de uso del operador o simplemente para mostrar las pantallas de los lugares disponibles en la cochera.

4.3.2. OpenHAB: configuraciones y definiciones

OpenHAB es la interfaz de usuario, que se encarga de interpretar y obtener la información obtenida de los sensores. Para lograr este cometido se debe en primer lugar definir en el archivo "openhab.cfg" la dirección IP del *gateway* para realizar la suscripción entre la interfaz de usuario y el *gateway*. De esta manera al iniciar la aplicación reconocerá al *gateway* como tal en la red y obtendrá el direccionamiento de los nodos en la red que se encuentran definidos en el *gateway*. También se define el tiempo de actualización del estado salud, el cual en esta prueba de concepto será establecido cada 6 segundos.

Una vez conocida la arquitectura de red y cuáles son los nodos que lo integran, openHAB utiliza a través de su archivo "default.items" las variables de interés del sistema, por lo que en él habrá que definir qué variables interesan de cada nodo y qué posición ocupan en él.

Se muestra el Código fuente 4.10 de los ítems definidos para el primer nodo.

```
1 Switch  Luz01001  "Posicion 1"  (grp1,TechnicView_Piso1)    { souliss="T11:1:0",
      autoupdate="true" }
2 DateTime  ingreso01001  "Activado [%1$tm.%1$td.%1$tY %1$tr]"
3 DateTime  salida01001 "Desactivado [%1$tm.%1$td.%1$tY %1$tr]"
4 Number  tiempo01001 "Tiempo en minutos [%.0f]"
5 Number  Salud01001  "Se\'nal C1S01 [%1d]" <keyring> (Diagnostic,TechnicView_Piso1,Health)   {
      souliss="D98:1:998" }
6 String  Respuesta01001  "Ultima conexion C1S01  [%1$td.%1$tm.%1$tY %1$tk:%1$tM:%1$tS]"
      <keyring> (Diagnostic,TechnicView_Piso1)    { souliss="D99:1:999" }
```

Código fuente 4.10: Código fuente sobre los ítems de interes declarados del primer nodo.

Se definen 6 ítems de interés por cada nodo, el más importante de ellos es el *typical* T11, el cual se define como un *switch* y cambiará su posición entre ACTIVADO/DESACTIVADO de acuerdo al estado *ON/OFF* obtenido. Las variables de ingreso y salida se establecen para sobrescribirlas con los valores de activación y desactivación del *typical*. A su vez se fija una variable tiempo, la cual mostrará la cantidad de minutos de ocupación.

Las variables salud y respuesta son propias de Souliss, la primera mide la salud de la conexión mediante la medición del tiempo de respuesta, siendo un valor entre 0 y 255, cuanto más alto es este valor más confiable es el enlace. La respuesta está fijada por defecto cada 10 segundos y se trata del envío de un *ping* hacía el nodo, estableciendo así la última conexión.

El archivo "sitemap", parte del Código fuente 4.11, se encarga de establecer la estructura a mostrar en la aplicación web, es decir cómo estarán ordenadas en pantalla a la hora de mostrar.

```
1 Switch item=Luz01001 mappings=[ON="OCUPADO", OFF="VACIO"]
2 Text item=ingreso01001 valuecolor=[ingreso01001>300="black", >0="green", <0="null"]
3 Text item=salida01001 valuecolor=[salida01001>300="red", >240="purple", >120="orange",
      >0="green", <0="null"]
4 Text item=tiempo01001 icon="clock-on"
5 Text item=Salud01001 icon="signal2" valuecolor=[Salud01001>190="green", >125="orange",
      >60="red", <0="null"]
```

```
Text item=Respuesta01001 icon="lastseen" valuecolor=[Respuesta01001>180="red", >120="purple",
    >60="orange", >0="green", <0="null"]
```

Código fuente 4.11: Código fuente de la forma de mostrar la información del primer nodo.

El campo *valuecolor*, mostrado en el Código fuente 4.11, cumple la función de modificar el color del texto de acuerdo a las condiciones establecidas. Las variables cambiarán de color según el tiempo transcurrido o los valores adquiridos para facilitar la visualización y entendimiento de la misma.

4.3.2.1. Reglas: automatización de los procesos

Las reglas se utilizan para automatizar procesos y es por esto que openHAB es una herramienta ideal para esta tarea. Estas reglas son disparadas por tiempos o cambios de estados, según se establezca.

Para el funcionamiento del sistema se establece un conjunto de reglas las cuales permiten llevar a cabo lo propuesto.

Cuando el estado del sensor cambia a ACTIVADO, el *item* cambia de estado de APAGADO a ENCENDIDO y se dispara la primera regla, cuyo diagrama de flujo se puede observar en la Figura 4.16, consiste en preguntar si es que el sensor estuvo anteriormente activado, si estuvo anteriormente activado entonces se debe restablecer el tiempo de activación anterior, de lo contrario será un ingreso nuevo.

El bloque “ingreso nuevo” de la Figura 4.16 ejecuta el Código fuente 4.12. El mismo guarda el tiempo actual en una variable, activa una bandera y sobrescribe dos variables, ingreso con el tiempo actual del sistema, y tiempo inicializando en 0.

```
ingreso01001.postUpdate(new DateTimeType())
salida01001.postUpdate('0 0 0 0 0 0' )
t01001  = now.millis
a01001= true
postUpdate(tiempo01001, 0)
```

Código fuente 4.12: Código fuente de ingreso nuevo.

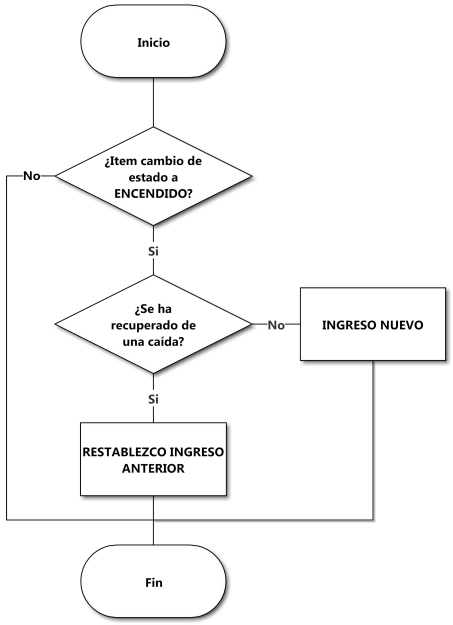


Figura 4.16: Diagrama de flujo de la regla de ingreso.

Mientras que el bloque “restablezco ingreso anterior” de la Figura 4.16, a través del Código fuente 4.13, sobrescribe la variable de tiempo con el tiempo anterior de activación y actualiza el estado de ingreso por dicho tiempo.

```
t01001 = t01001b
ingreso01001.postUpdate(t01001)
salida01001.postUpdate('0 0 0 0 0 0' )
```

Código fuente 4.13: Código fuente de restablecer ingreso.

De lo contrario, cuando el sensor cambia a DESACTIVADO, el ítem cambia su estado de ENCENDIDO a APAGADO, disparando una regla de salida, como se muestra en la Figura 4.17. Esta regla de salida pregunta cuánto tiempo estuvo ausente el sensor para determinar si es una salida valida o el sensor se está recuperando de una caída. En caso de ser una salida valida, a través del Código fuente 4.14, sobrescribe la fecha de salida con el tiempo actual y baja la bandera de activación para que no siga contando los minutos.

```
if (ab01001>0 && ab01001<4) { }
else
{
salida01001.postUpdate(new DateTimeType())
a01001 = false
}
```

Código fuente 4.14: Código fuente de salida.

La Figura 4.17 muestra el diagrama de flujo de la regla de salida.

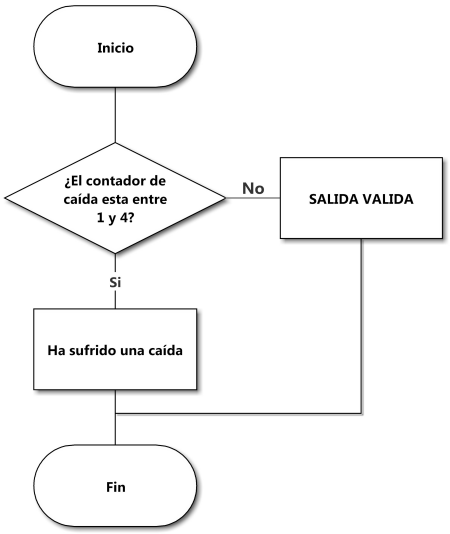


Figura 4.17: Diagrama de flujo de la regla de salida.

Cuando la bandera de activación se encuentra levantada la regla control de tiempo, que se ejecuta cada 30 segundos, efectúa el cálculo del tiempo transcurrido entre la hora de activación guardada y la hora actual, actualizando la variable tiempo. Este diagrama de flujo se puede observar en la Figura 4.18 y en el Código fuente 4.15 las sentencias de ejecución.

```
1 if (a01001== true)
2 {
3 val long ahora = now.millis
4 val double resultado01001 = ((ahora - t01001)/1000)/60
5 postUpdate(tiempo01001, resultado01001)
6 }
```

Código fuente 4.15: Código fuente de rutina de control de tiempo.

La Figura 4.18 muestra el diagrama de flujo de la regla de control.

Figura 4.18: Diagrama de flujo de la regla de control.

Para poder controlar que los sensores se mantengan conectados a la red y en el caso de que hubiera un nodo desconectado poder informarlo aplicamos dos reglas, la regla activo ejecuta la lógica del diagrama de flujo de la Figura 4.19, el cual guarda en una variable del tiempo actual cada vez que la interfaz de usuario recibe una respuesta del nodo, a través del Código fuente 4.16.

```
1 Item Respuesta01001 received update
2 then
3 update01001=now.millis
4 end
```

Código fuente 4.16: Código fuente de rutina de nodo activo.

La Figura 4.19 muestra el diagrama de flujo de nodo activo.

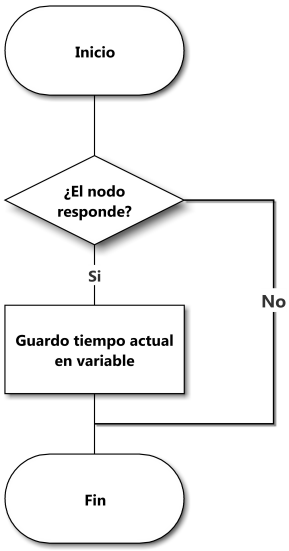


Figura 4.19: Diagrama de flujo de nodo activo.

Con esta variable de tiempo guardada se realiza mediante el diagrama de flujo de la Figura 4.20, el cálculo entre la hora actual y la almacenada. Sí este cálculo es menor a 15 segundos, siendo el tiempo de respuesta promedio de los nodos 10 segundos, se considera que el nodo se encuentra activo y disponible, iniciando un contador que se llamará nodo-arriba hasta 5, y se activa una bandera para indicar que el nodo está disponible.

De lo contrario sí el cálculo del tiempo es mayor a 15 segundos se considera que el nodo no está activo, y se inicializa un contador llamado nodo-abajo hasta 5. Sí este contador llega a 5 y la bandera de que el nodo estuvo disponible está en alto, habrá transcurrido alrededor de 50 segundos y se considerará que el nodo se desconecto. Por lo que si estaba ocupado enviará al servidor una sentencia de código, basado en el Código fuente 4.17. Para identificar al nodo como desconectado, luego de notificar al nodo se cambia de estado la bandera para indicar que el nodo ya no está disponible, la misma se volverá a levantar cuando vuelva a obtener respuesta.

```
Time cron "0/5 * * * * ?"
then
var String disp
var String not
disp="DISPONIBLE"
not = "NODISPONIBLE"
val long ahora = now.millis
val double aresultado01001 = ((ahora - update01001)/1000)
if (aresultado01001<15 && a01001==true)
  { ls01001=true
    ar01001= ar01001 + 1
  if (ar01001>=5)
    { ar01001=5
      ab01001=0 }
  postUpdate(valor01001, disp)
  }
else
  { if (aresultado01001>15 && a01001==true)
    { t01001b=t01001
      ab01001= ab01001 + 1
    if (ab01001>=5)
      {
      ab01001=5
      ar01001=0
      if (ls01001==true)
        { sendHttpGetRequest("http://192.168.1.25/ausente.php?piso=01&posicion=01")
          ls01001==false }
      }
    }
  postUpdate(valor01001,not)
  }
```

Código fuente 4.17: Código fuente de rutina de control activo

La Figura 4.20 muestra el diagrama de flujo de la regla de control de nodo activado.

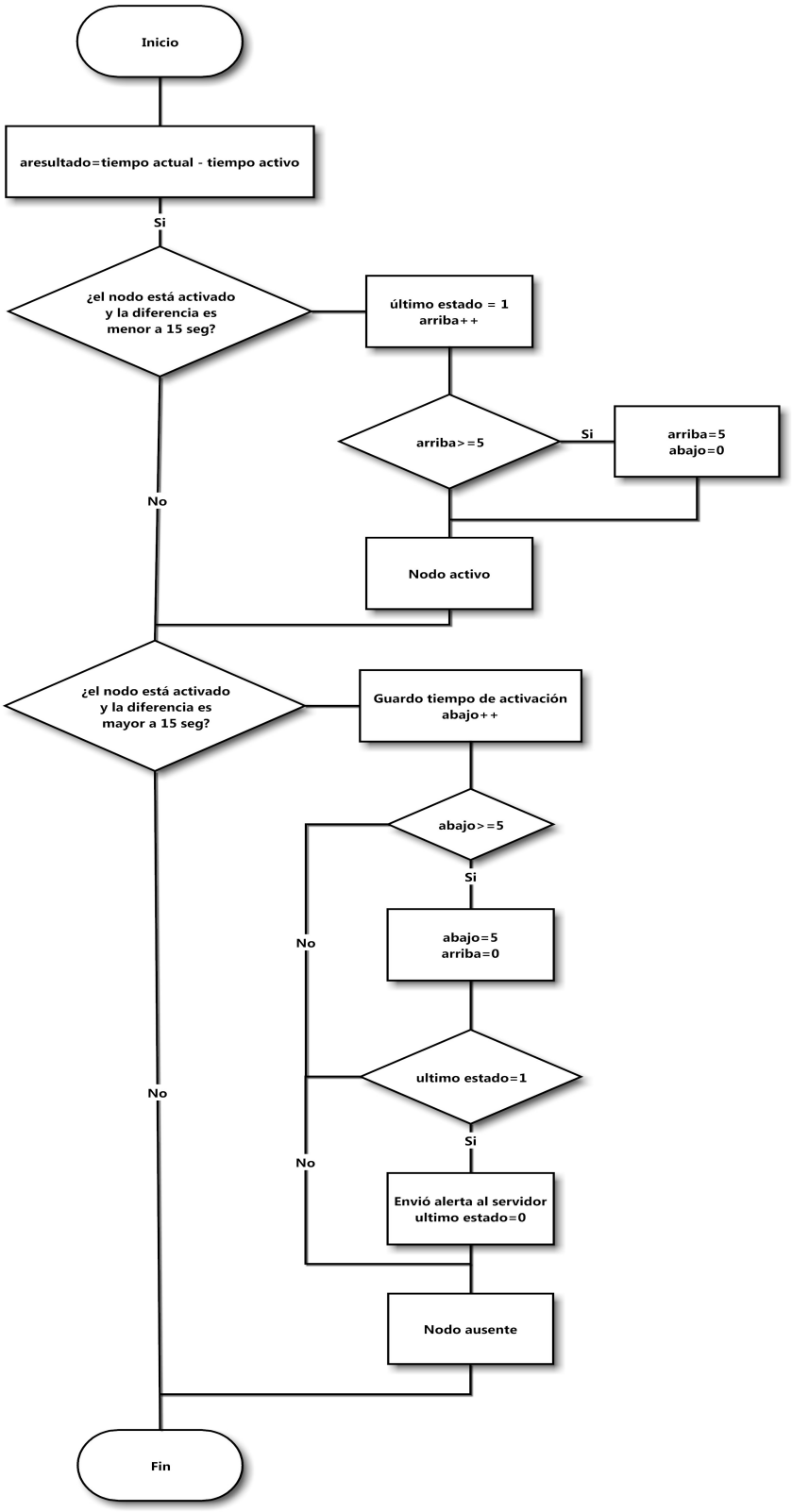


Figura 4.20: Diagrama de flujo de control nodo activo

4.3.2.2. Persistencia: almacenamiento en base de datos

El soporte de persistencia de openHAB permite almacenar los estados de los ítems a lo largo del tiempo (una serie de tiempo). En esta prueba de concepto se utiliza el modo de persistencia de base de datos tipo SQL, por lo que en las configuraciones de "openhab.cfg" se debe de especificar los datos inherentes a la conexión de la base de datos. Al realizar el guardado de los elementos, en este caso, configurado para que se guarden todos los cambios ocurridos en los ítems, se van guardando en diferentes tablas, como se puede observar en la Figura 4.21, de manera secuencial incorporando la hora de escritura que luego se puede acceder de manera simple a través de PHP u otro método.

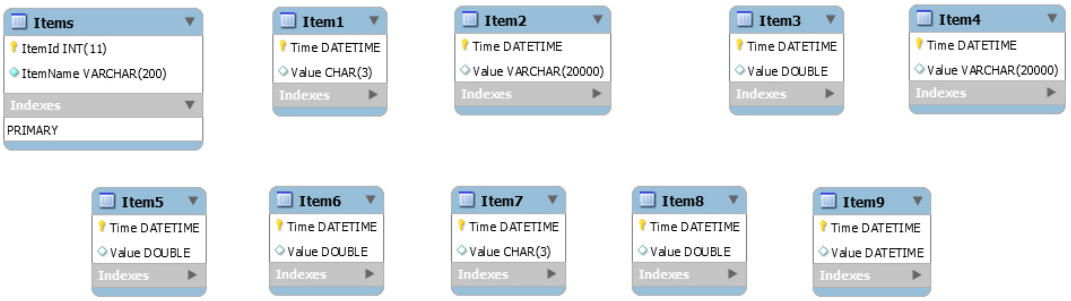


Figura 4.21: Modelado de base de datos SQL

Como se puede observar en la Figura 4.21, las tablas poseen nombres de campo muy similares, lo que hace prácticamente imposible identificar a que ítem corresponden. Para poder solucionar este inconveniente, la tabla "items" contiene la referencia entre el número de tabla y el ítem al cual corresponde.

4.3.2.3. Interfaz de usuario nativa

La interfaz de usuario nativa de openHAB sirve de control para el operario, mostrando los ítems especificados en el archivo "items" de la forma configurada en "sitemap". Por lo que el sistema para su control es multiplataforma, ya que desde cualquier dispositivo en el cual pueda ejecutarse un navegador web se podrá observar esta pantalla de control.

La Figura 4.22 muestra la interfaz de usuario nativa de openHAB.

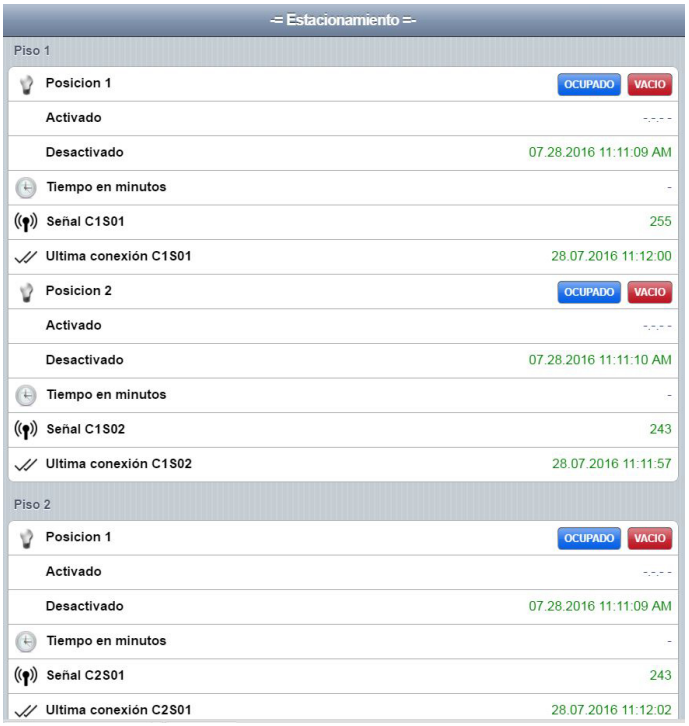


Figura 4.22: Captura de la interfaz de usuario nativa de openHAB.

4.3.3. Servidor web

El servidor web será el encargado de gestionar las consultas entre los usuarios finales y la base de datos.

La interacción con la base de datos se logra a través de un servidor HTTP Apache, PHP y PHPmyAdmin. Con el lenguaje de programación PHP, adecuado para el desarrollo web, podemos realizar consultas a la base de datos requiriendo la información necesaria para su posterior tratamiento.

Se divide el servidor web en cuatro módulos:

- Módulo 1. Configuraciones necesarias.
- Módulo 2. Pantalla con lugares disponibles por nivel.
- Módulo 3. Consulta de capacidad total del establecimiento.
- Módulo 4. Consulta de tiempo de estadía.

Los módulos 1 y 2 son de uso operativo, mientras que los módulos 3 y 4 están enfocados principalmente en el usuario.

4.3.3.1. Módulo 1: Configuraciones necesarias

A partir de que los ítems son agregados a la base de datos de manera no secuencial, se debe realizar una interpretación entre los mismos y el significado para la aplicación web, por lo que a través del archivo “inicio.php” se realiza una lectura de la tabla “items” en la base de datos openHAB y mediante un script en PHP se realiza la correspondencia necesaria generando el archivo “items.php” para el posterior tratamiento en los módulos.

4.3.3.2. Módulo 2: Pantalla con lugares disponibles por nivel

Este módulo es de uso para el operador, es el encargado de mostrar la cantidad de lugares disponibles por cada nivel.

Siguiendo el razonamiento del diagrama de flujo de la Figura 4.23, se obtiene la información de la base de datos, el último tiempo de respuesta y el último estado. A través de una función en un script PHP, quien se encarga de comparar el último tiempo de respuesta con la hora actual, si éste tiempo es menor a 15 segundos significará que el nodo se encuentra disponible, y se aumenta un contador, que se llama CAPACIDAD (por nivel). Luego se pregunta si éste nodo tiene el estado de "DESACTIVADO", de ser así se incrementará otro contador que se define como DISPONIBLE. De esta manera se tendrá en CAPACIDAD la cantidad de nodos con respuesta menor a 15 segundos y en DISPONIBLE la cantidad de lugares disponibles para ocupar. Estos datos son los mostrados en la página "pantalla.php", captura de ejemplo de la Figura 4.24, la cuál se auto actualiza cada 5 segundos mostrando los valores por nivel.

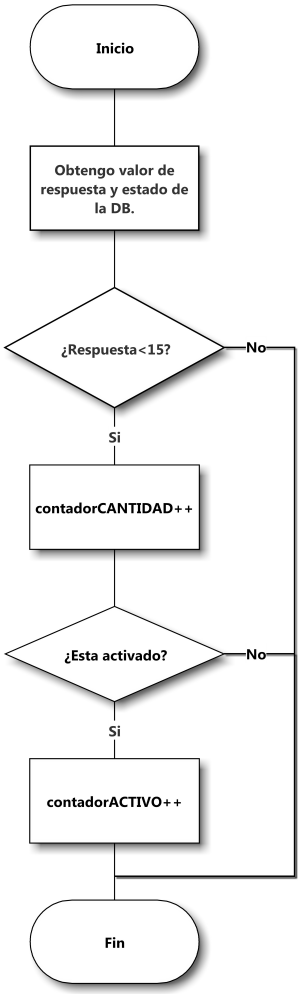


Figura 4.23: Diagrama de flujo para el conteo de lugares disponibles.

La Figura 4.24 muestra la captura de pantalla de consulta.

Figura 4.24: Captura de pantalla de consulta de capacidad del estacionamiento por nivel.

4.3.3.3. Módulo 3: Consulta de capacidad total del establecimiento

Esta página muestra la cantidad de lugares disponibles sobre la capacidad total del establecimiento, pensada para el usuario final, quien a través de Internet puede consultar cuantos lugares disponibles para estacionar existen.

El funcionamiento es similar al módulo 2, diagrama de flujo de la Figura 4.23, sólo que al contabilizar los contadores, lo toma como un total y no lo discrimina por nivel.

Al ingresar a la página “capacidad.php”, como en la Figura 4.25, se muestran los valores de los nodos activos y contando los nodos disponibles.

Figura 4.25: Captura de pantalla de consulta de capacidad total del estacionamiento.

4.3.3.4. Módulo 4: Consulta de tiempo de estadía

Este módulo permite al usuario final consultar el tiempo de estadía del automóvil a través de Internet.

Para realizar esta consulta, la Figura 4.26 tiene dos cuadros de textos que se deben completar, **posición** que ocupa el automóvil en el establecimiento y **piso** en el que se

encuentra.

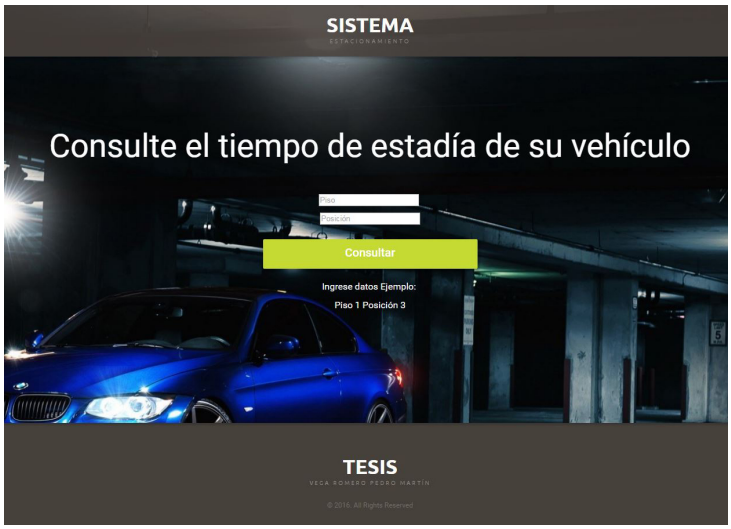


Figura 4.26: Captura de pantalla de consulta de tiempo.

De esta manera, siguiendo el razonamiento del diagrama de flujo de la Figura 4.27, con los datos ingresados, se realiza la consulta mediante un script en PHP, obteniendo dos variables “estado” y “tiempo”, dependiendo el resultado de la variable estado puede realizar las siguientes acciones:

- **ON**. Indica que el lugar está ocupado, por lo que se realiza el cálculo del tiempo de ocupación con la variable tiempo y luego se muestra en pantalla, como en la Figura 4.28.

- **OFF**. Indica que el lugar está desocupado, sólo se muestra en pantalla el texto “Sensor Desactivado”.

- **WAR**. Indica que el sensor no está en funcionamiento, se obtiene el tiempo de ingreso de la variable WAR para saber en qué momento perdió conectividad, y se muestra en pantalla “Sensor Desactivado. Hora de desactivación”.

La Figura 4.27 muestra el diagrama de flujo del funcionamiento de la instancia de consulta de tiempo.

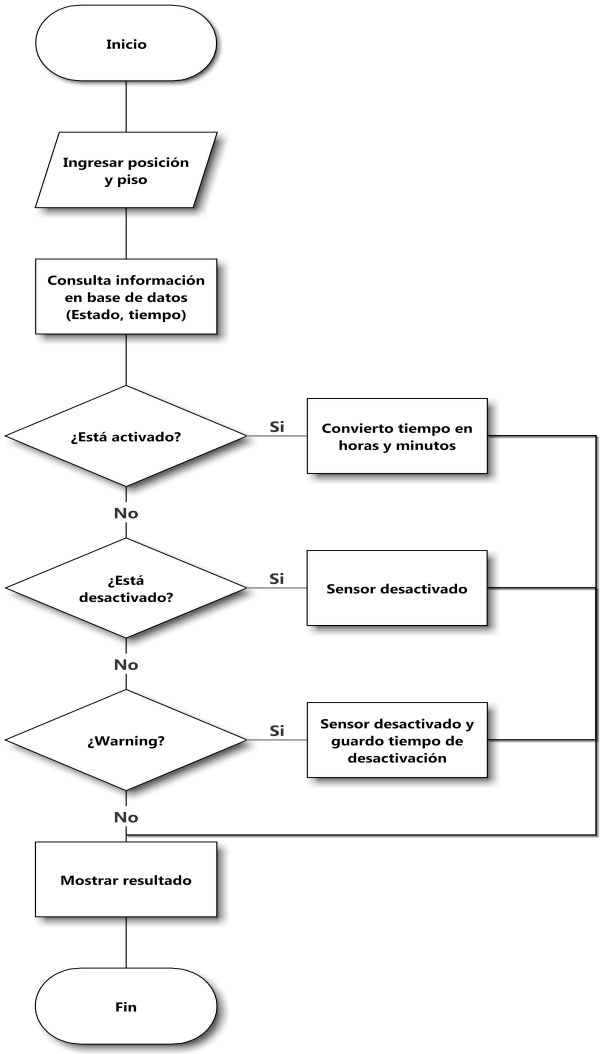


Figura 4.27: Diagrama de flujo de consulta de tiempo.

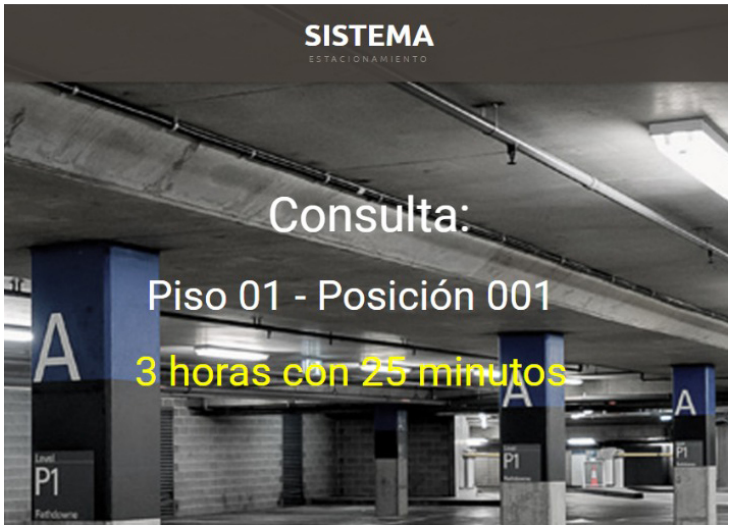


Figura 4.28: Captura de pantalla de consulta de tiempo medido.

Capítulo 5

Test

Un *test case* (caso de prueba), en ingeniería, es un conjunto de condiciones bajo las cuales se pretende determinar si una aplicación, sistema de software o una de sus características está funcionando como se estableció originalmente.

El mecanismo para determinar si un programa de software o sistema ha pasado o no una prueba de este tipo es conocido como un oráculo de pruebas. En algunas configuraciones, un oráculo podría ser, por ejemplo, un caso de requerimiento o la respuesta del sistema a cierta condición a la cual es sometido.

Normalmente, los *test cases* son utilizados para determinar que un programa de software o el sistema se considera suficientemente robusto y listo para ser lanzado.

En este proyecto se realizaron dos tipos de *test cases* al prototipo: de conectividad y de funcionalidad. A continuación se describen con mayor detalle cada uno de ellos.

5.1. Casos de test

5.1.1. Conectividad

Estos test cases pueden dividirse en dos clases bien diferenciadas respecto al tipo de señal y protocolo que se está evaluando.

Por un lado están la prueba de señal digital, que pretende asegurar el conexionado entre el sensor y el resto del hardware a las que se denominan de tipo-1 y por otro lado están las realizadas sobre la conectividad a nivel lógico de la arquitectura de red, entre los nodos y el servidor, denominadas de tipo-2.

5.1.1.1. Conectividad del sensor

Teniendo en cuenta las recomendaciones hechas por los fabricantes y las notas capturadas en el sitio oficial de Arduino denominado Playground [22], donde se exponen ejemplos, librerías, etc., de cómo interactuar con las placas de de desarrollo Arduino y los posibles periféricos, se lleva adelante la prueba de conectividad tal como se observa a continuación en la Figura 5.1.

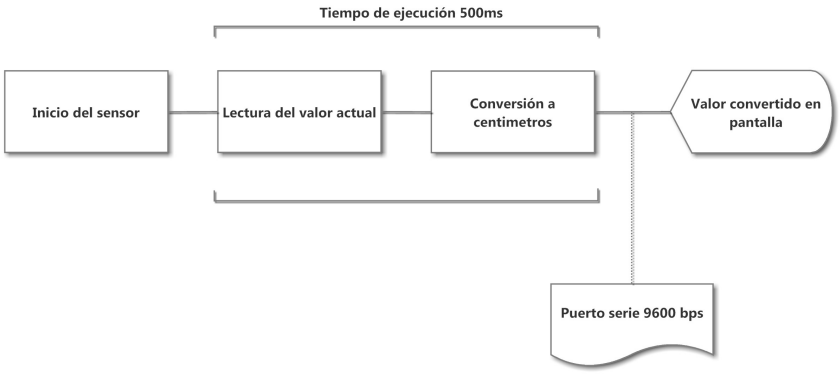


Figura 5.1: Proceso de medición para el sensor HC-SR04.

Una vez obtenidos el dato de la medición de distancia capturada por el sensor presente en el emulador de puerto serie provisto por Arduino IDE. Los datos son contrastados con los físicos obteniendo la misma medida dando por finalizada la prueba.

5.1.1.2. Conectividad del *gateway*

Para corroborar los siguientes casos de test, se hace uso de una utilidad de Souliss llamada SoulissApp, la cual es una aplicación diseñada para plataforma Android, que emula una interfaz de usuario autodidacta, sin necesidad de definir nada más que la dirección donde se encuentra el *gateway*.

Para la prueba de este caso de test, se realiza la conexión del *gateway* a la red previamente configurado con 5 ítems a modo de prueba. Con este elemento en la red, se puede a través de la aplicación SoulissApp comprobar si el elemento se encuentra en la red.

Como se observa en la Figura 5.2 que el *gateway*, nodo 0, se encuentra presente y su salud valor de salud en máximo, dando por satisfactoria y finalizada esta prueba.

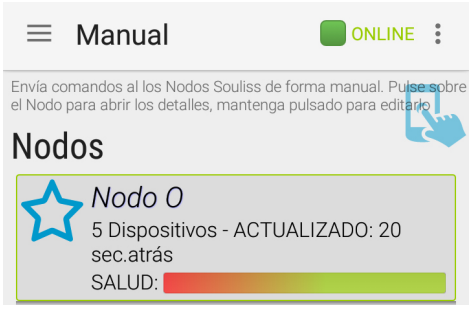


Figura 5.2: Captura de pantalla de la aplicación SoulissApp mostrando que el *gateway* se encuentra presente en la red.

5.1.1.3. Conectividad *gateway*, nodo concentrador y dispositivo terminal

Siendo la conexión del *gateway* y el nodo concentrador a través del *shield* Ethernet e independientes una de la otra, se da por probado que sí funciona la conectividad del *gateway*, la del nodo concentrador lo hará de igual forma.

Ahora se debe comprobar la conectividad entre el nodo concentrador y el dispositivo terminal a través del *gateway*.

Para ello se configura la arquitectura de red en los nodos y se utiliza el módulo conversor TTL a RS-485 como muestra la Figura 5.3, conectando el nodo concentrador y el dispositivo terminal a través de un cable de par cruzado.

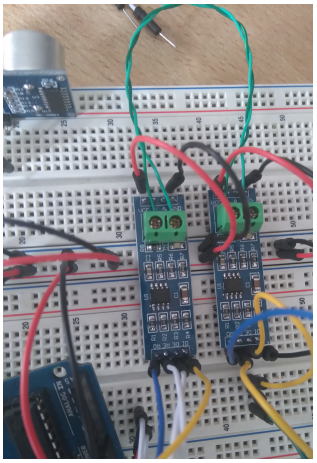


Figura 5.3: Conexionado de módulo RS-485 en *protoboard*.

La configuración de elementos en esta prueba estará compuesta por dos nodos concentradores declarados como nodo 1 y nodo 3, y dos dispositivos terminales definidos como nodo 2 y nodo 4. Se utiliza nuevamente la herramienta SoulissApp para conocer si los nodos se encuentran presente en la red. La Figura 5.4, muestra la captura de pantalla con los nodos presentes y conectados al *gateway*, siendo estos los nodos anteriormente descriptos y su valor de salud de conexión óptima.

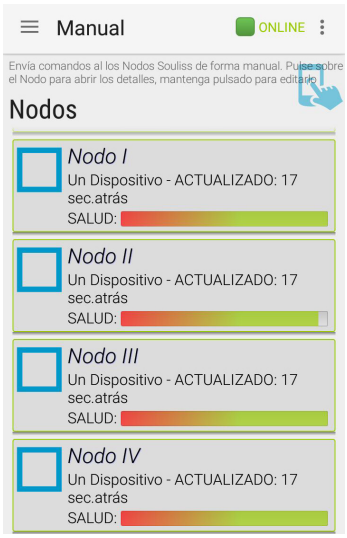


Figura 5.4: Captura de pantalla de la aplicación SoulissApp mostrando los nodos conectados a la red.

Comprobando que el nodo 2 y 4, en este caso los de interés, se encuentran conectados a la red se da por satisfactoria y finalizada esta prueba.

5.1.1.4. Conectividad entre el servidor y el *gateway*

Se desea comprobar la conectividad entre el servidor y el *gateway*, para ello se configura la aplicación openHAB, en su archivo "openhab.cfg" ingresando la IP del *gateway*, así como también se declaran los ítems y un mapa del sitio para que la aplicación pueda realizar el vínculo entre las variables y su funcionamiento. Se realiza el conexionado en *protoboard*, véase Figura 5.5, a fin de poder comprobar esta conectividad.

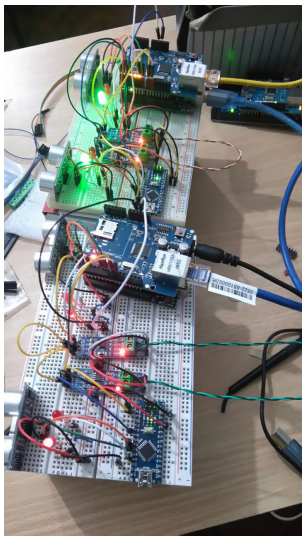


Figura 5.5: Conexionado del sistema en *protoboard* para la prueba de conectividad entre el servidor y el *gateway*.

Una vez inicializados todos los elementos se puede comprobar en la interfaz de usuario, como se muestra en la Figura 5.6, la respuesta de los nodos en la red.

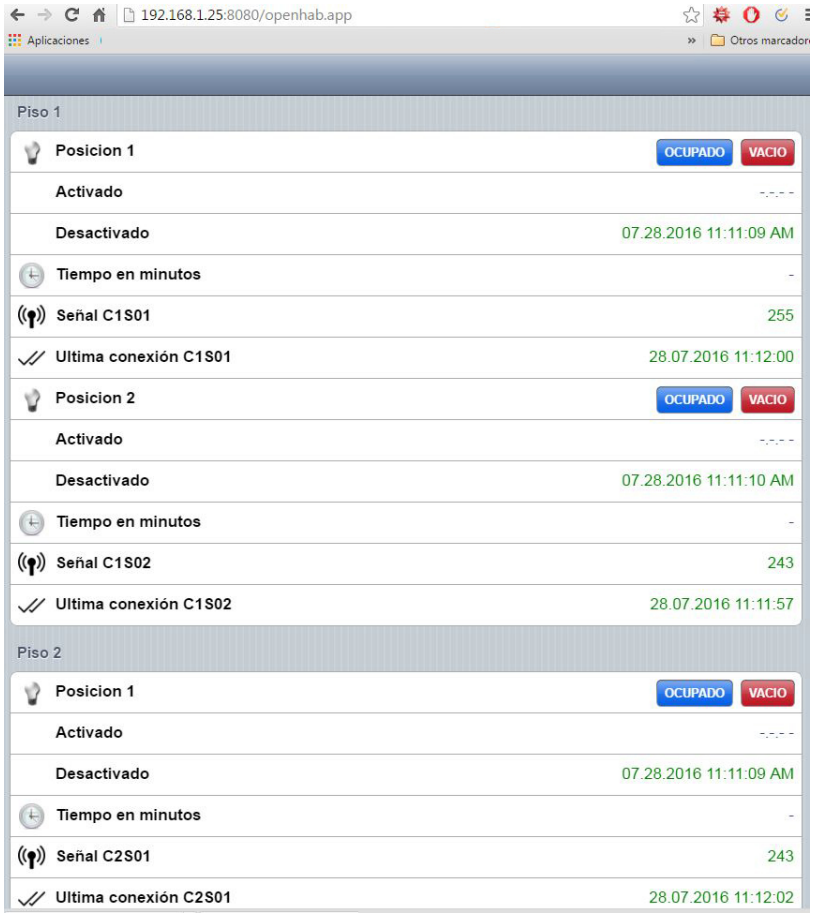


Figura 5.6: Captura de pantalla de la interfaz de usuario mostrando los elementos conectados y funcionando.

5.1.2. De Funcionalidad

Una vez aceptados y concluidos los test de conectividad garantizando el correcto acople de las partes, se procede a realizar los test de funcionalidad.

Este tipo de pruebas consiste básicamente en determinar qué tanto se asemeja el desempeño del dispositivo y su capacidad de poder cumplir con los objetivos propuestos.

Para la realización de estas pruebas se optó por realizar el conexionado a partir de placas multipropósito. De esta manera se evitan errores que puedan provenir de una mala conexión.

Los casos de funcionalidad propuestos son:

- Detectar/Notificar presencia de automóvil.
- Contabilizar tiempo de estadía.
- Mantener estados frente a desconexiones.
- Guardar estados en base de datos.

5.1.2.1. Notificación de lugar ocupado o disponible

Para probar el correcto funcionamiento de la notificación/detección de presencia, se monitorea el comportamiento tanto de un nodo concentrador, como un dispositivo terminal.

Se enciende el nodo de interés, luego de inicializar, en aproximadamente 5 segundos, inicia el proceso de detección, el cual demora alrededor de 12 segundos. Sin ningún objeto al frente dentro del rango de detección, el nodo muestra, como se observa en la Figura 5.7, a través de su LED verde encendido que se encuentra disponible.

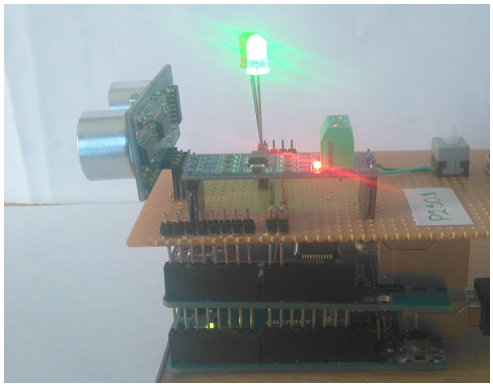


Figura 5.7: Nodo concentrador sin presencia.

Ahora colocando un objeto por abajo de la distancia definida como umbral, el sensor cambia de estado a ocupado, con la activación del LED rojo e inicializa las variables correspondientes para el conteo de tiempo en openHAB. La Figura 5.8 muestra el nodo concentrador detectando presencia.

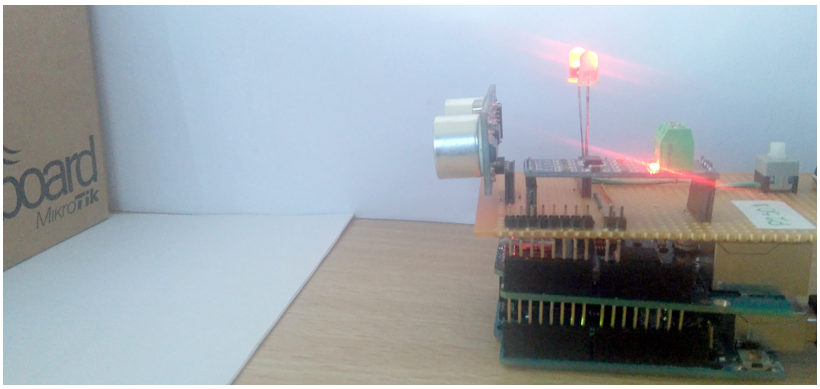


Figura 5.8: Nodo concentrador detectando presencia.

La lógica del nodo hace que el tiempo de espera mínimo para el cambio de estado entre ocupado y libre o viceversa sea de 12 segundos, debido a que está realizado así para evitar falsas activaciones o desactivaciones del mismo.

De esta manera se comprueba el correcto funcionamiento de esta función del sistema.

5.1.2.2. Tiempo de estadía medido y almacenamiento de variables en base de datos

En esta fase de prueba se probaran los nodos para establecer si realizan el conteo conforme pasa el tiempo y la escritura en la base de datos.

El escenario armado es el de un nodo concentrador con un objeto delante, superando la distancia de guardia (10 cm) y menor al umbral de medición (50 cm), éste cambia así su estado a "ocupado".

Se comprueba el correcto funcionamiento del tiempo medido a través de la tabla escrita en la base de datos, Figura 5.9, donde se observa que cada un minuto el valor se incrementa y es grabado en la base de datos, con este resultado se da por concluida la prueba.

Time	Value
2016-10-17 20:03:11	0
2016-10-17 20:04:33	1
2016-10-17 20:05:30	2
2016-10-17 20:06:35	3
2016-10-17 20:07:30	4
2016-10-17 20:08:33	5
2016-10-17 20:09:34	6
2016-10-17 20:10:30	7
2016-10-17 20:11:30	8
2016-10-17 20:12:30	9
2016-10-17 20:13:30	10
2016-10-17 20:14:30	11
2016-10-17 20:15:30	12
2016-10-17 20:16:30	13
2016-10-17 20:17:30	14
2016-10-17 20:18:30	15
2016-10-17 20:19:31	16
2016-10-17 20:20:30	17

Figura 5.9: Captura de pantalla de la tabla de tiempo en la base de datos.

5.1.2.3. Consulta web

Este test es una consecuencia del test anterior, por lo que se utilizará el mismo escenario, un dispositivo terminal o concentrador con un objeto delante, superando la distancia de guardia y menor al umbral de medición, su estado será ocupado.

Se realiza el control del tiempo medido a través de la consulta a la página web en el servidor. Monitoreando por períodos a lo largo de más de tres horas, siendo una captura ejemplo la Figura 5.10, se da por concluida la prueba.

Figura 5.10: Captura de pantalla de consulta web de tiempo medido.

5.1.2.4. Consulta pantalla de muestra de lugares disponibles por nivel

Para la realización de este test, se debe tener en cuenta que existen dos sensores por nivel, se plantean 5 casos diferentes.

- Todos los nodos ocupados.
- Nodos desconectados.
- Nodos ocupados y disponibles.
- Nodos libres.
- Nodos libres y desconectados.

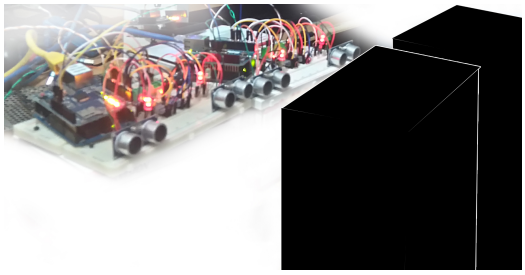


Figura 5.11: Nodos ocupados.

La Figura 5.11 muestra el primero de los escenarios con todos los sensores bloqueados, obteniendo como resultado lo mostrado en Figura 5.12.

Figura 5.12: Pantalla web disponibilidad por nivel.

Luego de las pruebas realizadas, para todos los casos involucrados, el sistema responde mostrando en pantalla la información correcta. De esta manera se da por finalizada la prueba.

Capítulo 6

Prototipo del sistema

La realización del prototipo del sistema se llevó a cabo de acuerdo a las necesidades finales y aprovechando la arquitectura de las placas de desarrollo. Teniendo en cuenta que el sistema está compuesto por elementos que requieren montaje en placa para su correcto funcionamiento, el mismo se desarrolló sobre placas de circuito impreso (PCB, *Printed circuit board*) multipropósito.

Se realizó el montaje en placa de los nodos concentradores y de manera similar de los dispositivos terminales, mientras que el servidor, montado sobre una Raspberry Pi, y el *gateway*, Arduino Mega con un *shield* Ethernet no necesitan de elementos adicionales para su funcionamiento.

6.1. Nodo concentrador

Aprovechando la arquitectura de Arduino Uno, que permite la colocación de placas accesorias, se fabricó una placa accesoria con el fin de incorporar los elementos necesarios para el funcionamiento del nodo.

La Figura 6.1 muestra como queda constituido el nodo concentrador.

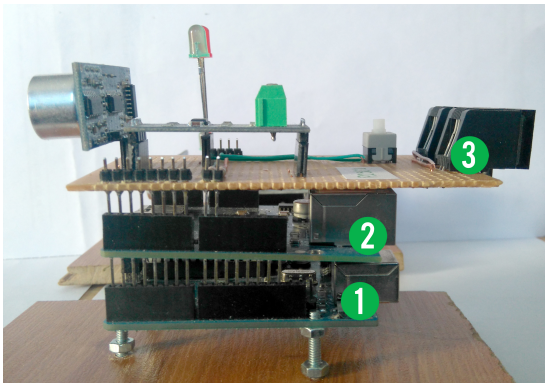


Figura 6.1: 1- Arduino Uno. 2- *Shield* Ethernet. 3- Placa accesoria.

La placa accesoria fue realizada sobre una placa PCB multipropósito, con el fin de simplificar el conexionado entre nodos, se optó por añadir dos puertos RJ11, los cuales se encuentran puenteados sus pines, para poder transportar la energía y los datos por un mismo medio físico. En este caso usando 4 hilos de un cable UTP con una ficha RJ11 macho, se consigue interconectar los nodos y alimentarlos.

Los elementos, como se muestra en la Figura 6.2, que contiene el nodo concentrador son:

1. Sensor ultrasónico HC-SR04: encargado de realizar las mediciones, alimentado por la placa Arduino.

2. LEDs: LED rojo y verde, encargados de notificar el estado del lugar, alimentados por la placa Arduino a través de una resistencia para cada uno.

3. Módulo conversor TTL a RS-485: encargado de interfaz RS-485, es alimentado por la placa Arduino, se encuentra puenteado sus pines A y B a los puertos RJ11.

4. Interruptor: permite encender y apagar el nodo.

5. Puerto RJ11 entrada: permite la alimentación del nodo, con un cable UTP armado con RJ11 macho que proviene de la fuente de energía.

6. Puerto RJ11 salida: permite la conexión con el dispositivo terminal de manera simple, con un cable UTP utilizando 4 hilos a través de una ficha RJ11 macho, de esta manera al tener puenteados los pines de energía y datos se realiza el conexionado entre nodos de manera simple y practica.

La Figura 6.2 muestra los elementos de la placa accesoria.

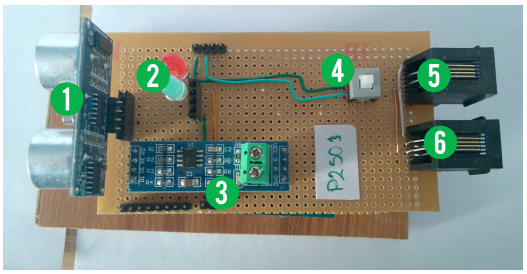


Figura 6.2: Placa accesoria del nodo concentrador.

6.2. Dispositivo terminal

El montaje de los elementos de electrónica necesarios del dispositivo terminal fueron realizados sobre una placa PCB multipropósito, basándose en las dimensiones de la placa Arduino Nano v3.0 se realiza una pequeña placa, como se muestra en la Figura 6.3, compuesta de los siguientes elementos:

1. Sensor ultrasónico HC-SR04: encargado de realizar las mediciones, alimentado por la placa Arduino.
2. LEDs: LED rojo y verde, encargados de notificar el estado del lugar, alimentados por la placa Arduino a través de una resistencia para cada uno.
3. Módulo conversor TTL a RS-485: encargado de interfaz RS-485, es alimentado por la placa Arduino, se encuentra puenteado sus pines A y B a los puertos RJ11.
4. Puerto RJ11 salida: permite la conexión con el nodo concentrador de manera simple, con un cable UTP utilizando 4 hilos a través de una ficha RJ11 macho, de esta manera al tener puenteados los pines de energía y datos se realiza el conexionado entre nodos de manera simple y practica.
5. Arduino Nano: encargada del procesamiento del nodo. Se encuentra alimentada a través del voltaje obtenido a la entrada del puerto RJ11, proveniente del nodo concentrador.
6. Interruptor: permite encender y apagar el nodo.

La Figura 6.3 muestra los elementos de la placa realizada.

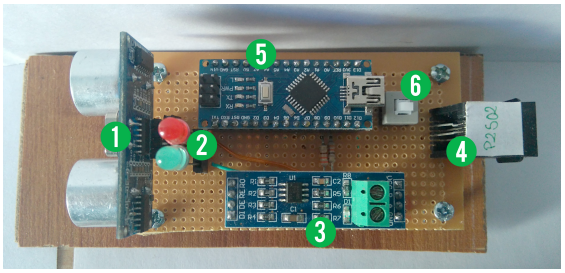


Figura 6.3: Dispositivo terminal y los elementos que lo componen.

Capítulo 7

Análisis de resultados

Luego de la implementación, pruebas y puesta a punto, del prototipo, se obtuvieron resultados satisfactorios.

Se consiguió un prototipo mejorado cumpliendo con los objetivos principales de este trabajo, se logró detectar, notificar y medir en tiempo la presencia de un automóvil, y obteniendo a través del servidor web esta información integrada a Internet para la consulta de los usuarios finales.

El prototipo final concuerda con lo planteado en los objetivos del proyecto, cumplió con las expectativas difiriendo levemente del diseño propuesto, con la inclusión de un elemento más a la red, el *gateway* montado en Arduino Mega, pero permitiendo así mejores prestaciones para el escalamiento o extensión de la red de nodos.

Entre los aspectos más relevantes de este proyecto se encuentra la coexistencia de dos tipos de interfaces de red diferentes para un uso en común. Siendo éste el punto más complicado de lograr, debido a la problemática que significa el envío de paquetes entre dos redes de diferentes medios físicos. Para poder abordar este problema se hizo uso íntegro de la potencialidad y flexibilidad del *framework* Souliss.

El uso del software libre openHAB facilitó el desarrollo del proyecto, permitiendo incluso prescindir de una Raspberry Pi como servidor y montarlo en cualquier otra computadora, ya que el software involucrado es multiplataforma. Por otro lado, utilizando un servidor web para las consultas de los servicios ofrecidos por el sistema, se logra acceder a la información desde diferentes tipos de dispositivos, por lo que desde cualquier navegador, sin importar la plataforma donde se encuentre, se puede acceder a la información.

La utilización de este prototipo junto al gran potencial de la IoT para el acceso y el procesamiento de la información, brindan un servicio diferente a las cocheras, permitiendo ofrecer un valor agregado al propietario e información relevante al consumidor. Mencionando aparte la contribución que este podría significar en disminución de emisiones de gases, gasto de combustible y tiempo invertido.

Capítulo 8

Conclusiones

Con el desarrollo de la presente prueba de concepto, se ve corroborada la veracidad de lo planteado, lo cual indica que es factible diseñar un sistema prototipo capaz de detectar, notificar y medir en tiempo la presencia de un automóvil, integrando la información obtenida a Internet. Como resultado de la investigación se dio cumplimiento a los objetivos específicos propuestos.

Si bien el *framework* Souliss utilizado para el desarrollo presenta limitaciones en la cantidad de nodos, es posible superar esta limitación utilizando dos o más *gateways* para crear instancias de openHAB, permitiendo así escalarlo e incrementar la cantidad de nodos.

El uso de este prototipo brindaría una solución al usuario final y un valor agregado al establecimiento donde se instale. La tecnología asociada a la *IoT* permite una mejor toma de decisiones. Instalando el sistema en diferentes cocheras de estacionamiento, se podría crear una red para saber los lugares disponibles en un determinado radio.

A nivel personal, la realización de este proyecto me sirvió para poner en práctica los conocimientos adquiridos en la carrera, descubrir las posibilidades que ofrece el hardware libre y despertar un mayor interés en los sistemas embebidos.

Como trabajo futuro de esta propuesta, buscando producirlo en serie a nivel producto, sería de vital importancia aislar las placas mediante un gabinete o carcasa para dicho propósito, evitando así problemas vinculados a la humedad y susceptibilidad a interferencias, entre otros. Para su comercialización es importante disminuir los costos, una posible mejora sería la de implementar más de un sensor por nodo y, de esta manera, con un mismo nodo poder abarcar más de un lugar de estacionamiento. También crear una interfaz de usuario final para *smartphone* mejorando así las prestaciones y un software propio que le daría flexibilidad al operario. Por otro lado, la inclusión de un *shield* Wi-Fi de Arduino permitiría una instalación más simple, lo cual resultaría de gran interés.

Bibliografía

[1] U. S. C. Bureau. Estadísticas de usuarios mundiales de internet. [En línea]. Disponible en: www.census.gov

[2] D. N. R. de la Propiedad del Automotor. Estadística anual de parque activo. [En línea]. Disponible en: http://www.dnrpa.gov.ar/

[3] Prodelux. Lider en estacionamiento guiado en argentina. [En línea]. Disponible en: http:/www.prodelux.com.ar/

[4] KEYTOP. Parking guidance system. [En línea]. Disponible en: http://www.ikeytop.com/

[5] *Hoja de Datos HC-Sr04*, 2013. [En línea]. Disponible en: http://www.micropik.com/PDF/HCSR04.pdf

[6] S. Corporation, *Hoja de datos del Sensor GP2Y0A21YK*, 2005. [En línea]. Disponible en: http://www.sharpsma.com

[7] *Referencias para RS485*. [En línea]. Disponible en: http://www.rs485.com/rs485spec.html

[8] *Plataforma Arduino*. [En línea]. Disponible en: www.arduino.cc

[9] *Raspberry Pi*. [En línea]. Disponible en: www.raspberrypi.org

[10] *Manual de Programación Arduino*, 2007. [En línea]. Disponible en: http://arduinobot.pbworks.com/f/Manual+Programacion+Arduino.pdf.

[11] *The Raspberry Pi Education Manual*, 2012. [En línea]. Disponible en: http://downloads.raspberrypi.org/Raspberry_Pi_Education_Manual.pdf.

[12] F. Bankinter, "El internet de las cosas." 2011. [En línea]. Disponible en: www.fundacionbankinter.org

[13] Bosch, "Market size and connected devices." 2014. [En línea]. Disponible en: https://www.bosch-si.com/media/bosch_software_innovations/documents/iot_2/201404-bosch-software-innovations-iot-infographic.pdf

[14] F. Santamaría, "Reporte digital: El internet de las cosas." 2015. [En línea]. Disponible en: http://reportedigital.com/iot/internet-cosas-revolucion-dispositivos-conectados-2015/

[15] Definición.de. Definición de open source. [En línea]. Disponible en: http://definicion.de/open-source/

[16] F. S. Foundation. Definición de software libre. [En línea]. Disponible en: https://www.gnu.org/philosophy/free-sw.es.html

[17] Raspbian. Raspbian. [En línea]. Disponible en: https://www.raspberrypi.org/documentation/raspbian/

[18] OpenHAB. Openhab wiki. [En línea]. Disponible en: https://github.com/openhab/openhab/wiki

[19] Souliss. Souliss wiki. [En línea]. Disponible en: https://github.com/souliss/souliss/wiki

[20] I. Maxim Integrated Products. Guidelines for proper wiring of an rs-485 (tia/eia-485-a) network. [En línea]. Disponible en: https://www.maximintegrated.com/en/app-notes/index.mvp/id/763

[21] JRodrigo. Librería ultrasonic hc-sr04. [En línea]. Disponible en: https://github.com/JRodrigoTech/Ultrasonic-HC-SR04

[22] Arduino. Arduino playground. [En línea]. Disponible en: http://playground.arduino.cc/Main/HomePage

Bibliografía Complementaria

- *William Stallings*, Comunicaciones y Redes de Computadores, 7ma Edición, 2004.
- *Gustavo Galeano*, Programación de Sistemas Embebidos en C, 2009.
- *Andrew S. Tanenbaum*, Redes de Computadoras, 5ta Edición, 2012.