Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Propuesta de Sistema SCADA código abierto para Sistemas Embebidos

Autor: Lisvet Pino Acosta

Tutor: Dr. C. René González Rodríguez

Santa Clara

2017

" Año 59 de la Revolución "

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Propuesta de Sistema SCADA código abierto para Sistemas Embebidos

Autor: Lisvet Pino Acosta

email: lpino@uclv.cu

Tutor: Dr. C. René González Rodríguez

Dpto. de Automática y Sistemas Computacionales,

Facultad de Ing. Eléctrica, UCLV

email: voltusv@gmail.com

Consultante: Dr. C. Luis Hernández Santana

Dpto. de Automática y Sistemas Computacionales,

Facultad de Ing. Eléctrica, UCLV

email: luishs@uclv.edu.cu

Santa Clara

2017

"Año 59 de la Revolución "

PENSAMIENTO

El que no posee el don de maravillarse ni de entusiasmarse más le valdría estar muerto, porque sencillamente sus ojos están cerrados.

Albert Einstein

DEDICATORIA

A mis padres,

por el amor, comprensión y apoyo incondicional.

A mis sobrinos,

que los quiero como si fueran mis hijos.

AGRADECIMIENTOS

Especialmente a mis padres, que me han apoyado, inspirado y guiado hasta convertirme en quien soy hoy.

A mi hermana, por estar siempre a mi lado y brindarme todo su amor y cariño.

A mi tutor y amigo René González, por ser mi principal fuente de inspiración, por la ayuda que me brindó desde los inicios de mi carrera y en la realización de este trabajo, por confiar en mí.

A mi novio Otnier, por impulsarme en los momentos más difíciles, por hacerme sentir especial, por su amor y paciencia.

A mis amigos y compañeros de estudio que me han apoyado en el transcurso de mi carrera, porque sin ellos no hubiera sido posible seguir adelante, en especial a Claudia, América, Lida, Ketia, Soca, Rolando, Armando, Carlos Manuel y Diony, que siempre me apoyaron y ayudaron.

A mis suegros por escucharme, inspirarme y preocuparse por mí en todo momento.

A mi amiga Ivonne que en estos últimos años se ha convertido en una hermana, por escucharme, y ayudarme a enfrentar las adversidades.

A mi gran amiga Gretter, que ha estado a mi lado en todo momento, me ha escuchado y entendido.

A todos los profesores que durante la carrera han intervenido en mi formación profesional, por brindarme sus conocimientos.

En general quisiera agradecer a todos aquellos que de una forma u otra ayudó a la realización de este trabajo de diploma y a mi formación.

Muchas gracias.

RESUMEN

Los SCADA (Control Supervisorio y Adquisición de Datos) son sistemas de control muy utilizados en la actualidad, con gran importancia en industrias, edificios inteligentes, sistemas aeronáuticos, incluso en sistemas aeroespaciales, en fin, en cualquier sistema que se realice algún tipo de control automático o se requiera la supervisión continua de un proceso.

El presente trabajo pretende aportar una solución tecnológica viable de automatización para los hoteles de Cuba, por lo cual se propone la utilización de software y hardware abierto para el desarrollo del mismo.

La implementación de un SCADA código abierto en sistemas embebidos sería muy factible para nuestro país debido a la independencia tecnológica, soberanía nacional y reducción de costos que trae consigo ya que se ahorra en pago de licencia del software, llave de programación, así como en el uso de computadoras que según las características necesarias de hardware son costosas.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Hoy en día la automatización es un campo en continuo desarrollo. Su gran importancia radica en la implementación de procesos automáticos en disimiles aplicaciones, tanto en la esfera industrial, de servicios, como domésticos, ya sea en ciudades grandes o pequeñas.

En los procesos industriales los sistemas SCADAs son los encargados de la recolección, supervisión, monitoreo y control de las variables de procesos sin la necesidad de intervención continua de un operador humano (Moya, 2009). Comúnmente estos sistemas tienen una configuración cliente-servidor donde los servidores requieren altas prestaciones de procesamiento y comunicación lo que provoca un alto costo en la automatización del proceso y por ende limitantes de instalación.

La arquitectura de un SCADA debe ser abierta, de forma que sea capaz de crecer o adaptarse según las necesidades cambiantes de la empresa. Más aún, el SCADA debe ser capaz de satisfacer los requerimientos de los procesos industriales de los diferentes sectores, tales como: eléctrico, aluminio, hierro, petróleo, hotelero, etc. Por esta razón, para el proceso de definición de la arquitectura de un SCADA se debe involucrar a todos los sectores productivos, de manera que ningún requerimiento específico de algún proceso quede sin ser considerado (Abaffy and Lárez, 2007).

Los sistemas SCADA son softwares cotizados a altos precios. En la mayoría de los casos se ejecutan sobre Windows, lo cual eleva aún más su valor comercial, pero presentan problemas de seguridad por causa del sistema operativo. Debido a estos problemas surge la migración de la mayoría de las aplicaciones informáticas industriales a sistemas más robustos como las distribuciones del sistema GNU/Linux. Existen otros casos de sistemas SCADA basados en arquitecturas multiplataformas con lenguajes como Python y Java que permiten la ejecución

en cualquiera de estos sistemas operativos. De esta necesidad tangible a nacido la idea de crear sistemas de supervisión y control sobre plataformas de software libre. En la actualidad el uso de software libre y de código abierto cobra gran importancia gracias a las ventajas que introduce desde el punto de vista ético, económico, legal y tecnológico (López, 2009).

En el presente trabajo se propone una variante de sistema SCADA libre y de código abierto que se ejecuta en sistemas embebidos con ciertas limitaciones de procesamiento, pero con una relación costo/beneficio que facilita la instalación en muchos de los procesos industriales de bajas prestaciones como la hotelería y el control inteligente de inmuebles que es el campo de aplicación de nuestro trabajo.

La utilización de software libre y de código abierto en sistemas embebidos con hardware abierto es una solución que permite reducir las importaciones, aumentar los niveles de automatización y eficiencia de nuestras instalaciones con un bajo costo, consumo de potencia y total control de los sistemas. Es de vital importancia la utilización de sistemas SCADA código abierto para garantizar independencia tecnológica y soberanía nacional.

Como objeto de estudio a controlar y supervisar durante la ejecución del presente trabajo se utilizarán los sistemas automáticos para hoteles, casas e inmuebles, conocido como domótica o *Home Automation* en terminología inglesa.

La domótica es el conjunto de tecnologías aplicadas al control y la automatización inteligente de inmuebles, que permite una gestión eficiente del uso de la energía, aporta seguridad y confort, además de una interfaz amigable entre el usuario y el sistema (Huidobro, 2016).

El alto costo de los softwares y hardwares comerciales y la falta de mantenibilidad de los intentos de desarrollo endógeno constituye la situación problémica de este trabajo y desencadena el siguiente **problema científico**:

¿Existe alguna configuración de software y hardware que permita la implementación de sistemas SCADAs por el bajo costo, facilite el desarrollo endógeno de aplicaciones según necesidades puntuales de los clientes y cumpla con los requisitos mínimos indispensables en sistemas de supervisión y control de la Industria hotelera nacional?

Hipótesis:

Con el uso de SCADA libre, de código abierto y sistema de hardware embebido basado en ARM es posible controlar y supervisar una instalación hotelera bajando los costos de implementación y garantizando la independencia tecnológica, soberanía nacional y mantenimiento por la comunidad internacional.

Para demostrar la hipótesis en esta investigación se pretende cumplir con los siguientes objetivos:

Objetivo General:

Proponer un sistema SCADA código abierto que se pueda ejecutar en sistemas embebidos para aplicaciones de automatización de edificios que minimice gastos, facilite el desarrollo endógeno de aplicaciones según necesidades puntuales de los clientes y cumpla con los requisitos mínimos indispensables en sistemas de supervisión y control.

Objetivos Específicos:

1. Evaluar los diferentes sistemas SCADAs existentes en la actualidad con la condición de código abierto y que pueden ser utilizados en sistemas embebidos.
2. Comparar las arquitecturas de hardware basadas en hardware abierto que permitan la ejecución del sistema SCADA propuesto.
3. Comprobar el desempeño del sistema SCADA en sistemas empotrados con arquitectura ARM.
4. Implementar un sistema de supervisión y control real en la configuración software hardware propuesto.

Impacto posible:

El principal aporte de esta investigación es la propuesta de uso y configuración de SCADA código abierto en sistemas embebidos que permite la automatización de los hoteles en nuestro país a un costo muy bajo. La selección, prueba e implementación del SCADA código abierto openHAB en Raspberry Pi es una solución que permite reducir las importaciones, aumentar los niveles de automatización y eficiencia de nuestras instalaciones.

Organización del informe

El informe de tesis incluye tres capítulos, además de las conclusiones, recomendaciones, referencias bibliográficas y anexos correspondientes. Los temas que se abordan en cada capítulo se encuentran estructurados de la forma siguiente:

Capítulo I: Este capítulo se dedicará a abordar los conceptos principales relacionados con los sistemas SCADA comerciales y de código abierto y los sistemas embebidos, su evolución y desarrollo. Además, se realiza una comparación entre los diferentes softwares de código abierto para poder hacer una selección del más adecuado, así como el sistema embebido que mejor se presenta para la implementación de este proyecto. Para finalizar el capítulo se hará una justificación de la selección.

Capítulo II: Este capítulo expone detalladamente las funcionalidades del SCADA de código abierto openHAB, así como facilidades de utilización y de instalación, mostrando además manuales y métodos de programación, debido a que es un software que se encuentra en desarrollo y aún no cuenta con suficiente bibliografía para su estudio.

Capítulo III: En este capítulo se mostrarán las pruebas y resultados realizados de la aplicación desarrollada en openHAB en la Raspberry Pi, además, se hará una comparación con un sistema real montado en los hoteles del Cayo Santa María de Villa Clara, así como un análisis económico del mismo.

CAPÍTULO 1. Generalidades de los sistemas SCADA y los sistemas embebidos

El presente capítulo aborda el material teórico referente a los sistemas SCADA y los sistemas embebidos donde se encuentran aspectos generales como: definición, componentes, características, aplicaciones, comparaciones con otros sistemas, ventajas y desventajas. También se hace una pequeña descripción de los SCADA de código abierto.

1.1 Sistemas SCADA

El término SCADA proviene de las siglas en inglés "*Supervisory Control and Data Acquisition*". Se trata de un software diseñado para funcionar sobre ordenadores en el control de producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, RTUs[1] e instrumentación industrial) supervisando y controlando el proceso de forma automática desde la pantalla del ordenador. Además, provee toda la información que se genera en el proceso productivo a diversos usuarios, tanto del mismo nivel como de otros supervisores dentro de la empresa (Meza, 2007).

Los sistemas SCADA mejoran la eficacia del proceso de monitoreo y control proporcionando la información oportuna para poder tomar decisiones operacionales apropiadas (Abaffy and Lárez, 2007).

[1] Unidad Terminal Remota (*Remote Terminal Unit*)

1.1.1 Historia de los sistemas SCADA

En su surgimiento los SCADA eran simples sistemas que proporcionaban reportes periódicos de las variables de campo, muestreando las señales que representaban medidas y/o condiciones del estado de la planta desde ubicaciones remotas, en muchas ocasiones lo que se hacía era imprimir o registrar en un papel la información de las variables, para tener un histórico de los eventos que ocurrían durante la operación del proceso y gráficos de tendencia de las variables. Estos sistemas no prestaban funciones de aplicación alguna, sino que ofrecían capacidades muy simples de monitoreo y control. La visión del operador del proceso estaba basada en instrumentos y señalizaciones lumínicas montadas en paneles llenos de indicadores (Hentea, 2008).

Con el desarrollo tecnológico, las computadoras empezaron a aplicarse en el control industrial, realizando tareas de recolección y almacenamiento de datos, generación de comandos para el control, y una nueva función muy importante: la presentación de la información sobre una pantalla, que en aquel entonces eran monocromáticas (Igure et al., 2006; Ramagosa et al., 2004).

Según Izaguirre y colaboradores la mayoría de los supervisores instalados hoy en día forman parte de la estructura de dirección de cualquier organización. Estos sistemas son vistos por la gerencia como un recurso importante de información corporativa, sin el cual es imposible administrar la empresa. Además, sirven como centro de responsabilidad operacional, proporcionando datos importantes a los sistemas y usuarios que fuera del ambiente de control, dependen de dicha oportuna información para tomar sus decisiones económicas cotidianas (Izaguirre, 2008).

1.1.2 Funciones de un sistema SCADA

Las principales funciones de un SCADA se pueden resumir en los siguientes puntos (Abaffy and Lárez, 2007):

- Adquisición de datos, para recoger, procesar y almacenar la información recibida en forma continua y confiable.
- Supervisión, para observar desde un monitor la evolución de las variables de control.

- Control, para modificar la evolución del proceso, actuando bien sobre los reguladores autónomos básicos o directamente sobre el proceso mediante las salidas conectadas.
- Transmisión de información con dispositivos de campo y otros PC[2].
- Base de datos con posibilidades de gestión de datos con bajos tiempos de acceso.
- Representación gráfica y animada de variables de proceso y monitorización de éstas por medio de alarmas.
- Presentación, representación gráfica de los datos. Interfaz del Operador o HMI (*Human Machine Interface).*
- Arquitectura abierta y flexible, con capacidad de ampliación y adaptación.
- Conectividad con otras aplicaciones y bases de datos, locales o distribuidas, en redes de comunicación.
- Explotación de los datos adquiridos para gestión de la calidad, control estadístico, gestión de la producción y gestión administrativa y financiera.
- Alertar al operador de cambios detectados en la planta, tanto aquellos que no se consideren normales (alarmas) como cambios que se produzcan en la operación diaria de la planta (eventos). Estos cambios son almacenados en el sistema para su posterior análisis.
- Fácil cambio de parámetros del sistema de control como valores deseados (*set points*), ajustes de los reguladores, límites de alarmas, etc.

1.1.3 Objetivos de un sistema SCADA

Un SCADA debe cumplir varios objetivos o requisitos para que su instalación sea bien aprovechada (Muñoz, 2006):

- Debe ser un sistema de arquitectura abierta, capaz de crecer o adaptarse según las necesidades cambiantes de la empresa.
- Debe comunicar con total facilidad y de forma transparente al usuario con el equipo de planta y con el resto de la empresa.

[2] Computadora Personal

- Debe ser un programa sencillo de instalar, sin excesivas exigencias de hardware y fácil de utilizar, con interfaces amigables para el usuario.
- Ser independiente del sector y la tecnología.
- Funciones de mando y supervisión integradas.

1.1.4 Hardware de un sistema SCADA

Un sistema SCADA consiste en un computador principal o master llamado Estación Principal, más conocido por sus siglas en inglés *MTU* (*Master Terminal Unit*); una o más unidades de control obteniendo datos de campo y realizando el control directo llamadas Estaciones Remotas o por sus siglas en inglés *RTU* (*Remote Terminal Units*); una red de comunicación industrial; una instrumentación de campo y una colección de software usada para monitorear y controlar de manera remota los dispositivos de campo (Abaffy and Lárez, 2007).

En la actualidad existen diversas configuraciones de estos sistemas, la más general y utilizada se muestra en la Figura 1-1.

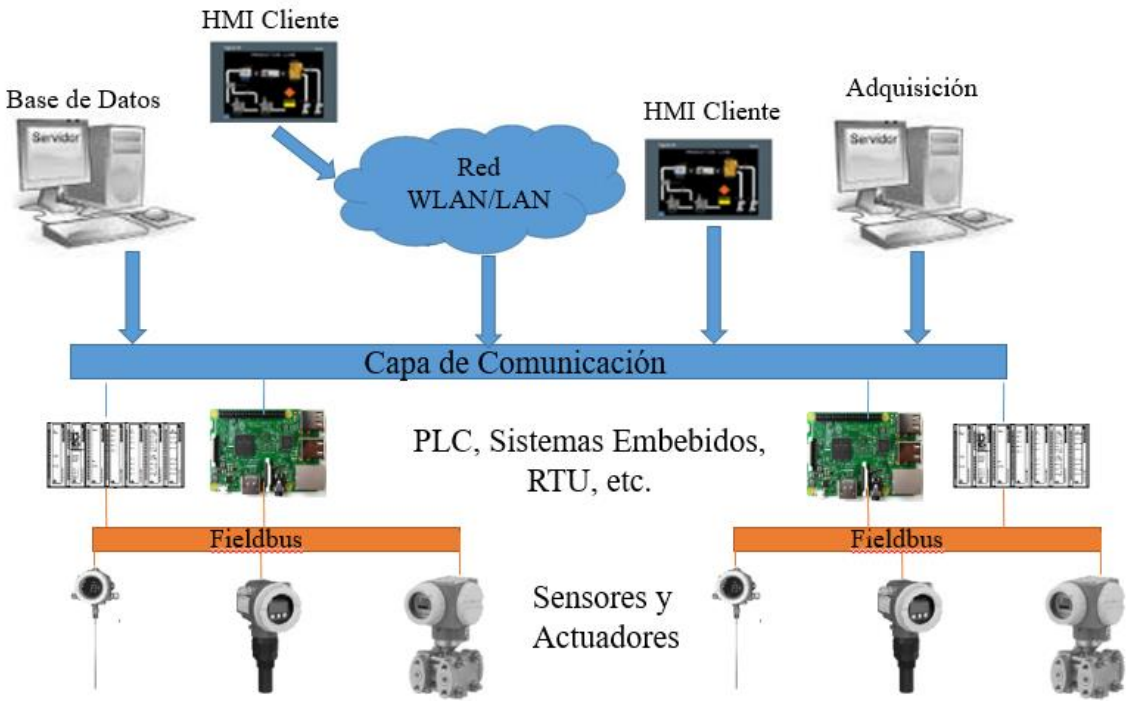


Figura 1-1: Arquitectura de un Sistema SCADA

El computador central o MTU (*Master Terminal Unit*) se encarga de supervisar y recoger la información del resto de las subestaciones, bien sean otros computadores conectados (en sistemas complejos) a los instrumentos de campo o directamente sobre dichos instrumentos. Este componente soporta el HMI (Abaffy and Lárez, 2007).

Las funciones principales del MTU son (Gamboa, 2001):

- Recuperación y depuración de los datos adquiridos.
- Supervisión y monitoreo de valores límites.
- Facilidades de diagnóstico remoto.
- Almacenamiento de información.
- Actualización de la base de datos.
- Dar información a operador, diseñadores, planificadores, etc.
- Comunicación con centros de control.
- Comunicación con las RTUs.

Las unidades de control o RTUs (*Remote Terminal Unit*) son ordenadores ubicados en el campo, los cual se encargan de procesar la información y la interacción entre el servidor SCADA y los distintos elementos de campo (Sensores, Actuadores). Las RTUs son unidades independientes, encargadas de la adquisición de datos y en la actualidad, también poseen la capacidad de realizar algoritmos de control (Moya, 2009).

Una tendencia actual es la de dotar a los PLCs[3] (en función de las E/S a gestionar) con la capacidad de funcionar como RTUs gracias a un nivel de integración mayor y CPUs[4] con mayor potencia de cálculo. Esta solución minimiza costos en sistemas donde las subestaciones no sean muy complejas sustituyendo el computador industrial que es mucho más costoso (Abaffy and Lárez, 2007). En conclusión, PLCs y RTUs compiten en la actualidad por el mismo mercado dentro de los sistemas SCADA. Sin embargo, en ocasiones se pueden usar alternativas de combinación de ambos elementos. Esta combinación se realiza

[3] Controlador Lógico Programable

[4] Unidad Central de Procesamiento

cuando las condiciones lo requieren debido a limitaciones en la comunicación, el procesamiento, o solo cuando se requiere aumentar el tamaño de la red SCADA (Moya, 2009).

La red de comunicación industrial es el nivel que gestiona la información que los instrumentos de campo envían a la red de computadores desde el sistema. El tipo de BUS[5] utilizado en las comunicaciones puede ser muy variado según las necesidades y el software escogido para implementar el sistema (Abaffy and Lárez, 2007).

Se pueden encontrar SCADA sobre formatos estándares como son:

- Medio Físico: RS-232, RS-422, RS-485, son medios de comunicación serie que pueden implementar protocolos de comunicación como Modbus serial, Modbus RTU, Profibus, etc.
- Medio Físico: Ethernet donde se implementan otros tipos de protocolos de comunicación como Modbus IP, ETHERNET/IP, entre otros.

Los instrumentos de campo son todos aquellos que permiten tanto realizar la automatización o control del sistema (PLCs, controladores de procesos industriales, y actuadores en general) como los que se encargan de la captación de información del sistema (sensores y alarmas) (Pérez, 2015).

1.1.5 Software de un sistema SCADA

Los módulos o bloques de software que permiten las actividades de adquisición, supervisión y control son los siguientes (Pérez, 2015):

- **Configuración:** Permite al usuario definir el entorno de trabajo de su SCADA, adaptándolo a la aplicación particular que se desea desarrollar y los niveles de acceso para los distintos usuarios.
- **Interfaz gráfica del operador:** Proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos

[5] Sistema Digital que transfiere datos

almacenados en el ordenador de proceso y generados desde el editor incorporado en el SCADA o importados desde otra aplicación durante la configuración del paquete.

- **Módulo de proceso:** Ejecuta las acciones de mando pre-programadas a partir de los valores actuales de variables leídas. La programación se realiza por medio de bloques de programa en lenguaje de alto nivel (como *Visual Basic for Application* o *C for Application,* entre otras).
- **Gestión y archivo de datos:** Se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.
- **Comunicaciones:** Se encarga de la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA, y entre ésta y el resto de elementos informáticos de gestión.

1.1.6 Principales aplicaciones de los sistemas SCADA

Los sistemas SCADA pueden ser simples, como en el caso del control de las condiciones ambientales en una oficina, o pueden ser complejos, como en una planta de generación hidroeléctrica o nuclear. Los SCADA son usados en la mayoría de entornos industriales complejos, o en procesos industriales que cubren un área geográfica amplia debido a que se puede adquirir información de forma rápida y desde lugares remotos, para luego ser presentada en la pantalla de un ordenador (Moya, 2009).

Los sistemas SCADA son aplicados en industrias como:

- Cervecerías
- Control de Aguas Residuales y Desperdicios
- Domótica
- Fábrica de pienso
- Generación de Energía
- Industria Alimenticia
- Industria Hotelera
- Industria Petroquímica
- Refinerías de Gas y Aceite

1.1.7 SCADA comerciales

En la actualidad muchos de los sistemas SCADA son desarrollados por poderosas compañías que son líderes en esta rama, tales como: InTouch de Wonderware (Wonderware, 1999), WinCC de Siemens (Martin, 2015), Movicon de Progea (Progea, 2015). Todos estos sistemas tienen algo en común son sistemas privativos y muy costosos (Linares, 2014).

Las principales características de estos SCADA comerciales se describen a continuación:

- **InTouch de Wonderware:** Ofrece funciones de visualización gráfica que llevan sus capacidades de gestión de operaciones, control y optimización a un nuevo nivel. Integración con controles Microsoft ActiveX y controles .NET. Migración de versiones de software sin interrupción, lo que significa que la inversión en sus aplicaciones HMI están protegidas. Una de sus principales desventajas es su alto costo de licencia.
- **WinCC:** Soporte de tecnologías Microsoft ActiveX. Permite scripts en *C for Application* (Programación en C). Comunicación con otras aplicaciones vía OPC (*OLE*[6] *for Process Control*). Comunicación sencilla mediante *drivers* (código que implementa el protocolo de comunicaciones con un determinado equipo inteligente) implementados. Programación *online*: no es necesario detener la *runtime* del desarrollo para poder actualizar las modificaciones en la misma.
- **Movicon:** Es una plataforma SCADA/HMI universal capaz de adaptarse a cualquier tipo de aplicación, con la máxima independencia del hardware. Se basa en el standard XML y sus tecnologías emergentes y abiertas como: *Web Services*, Gráfica Vectorial SVG, OPC, SQL (*Structured Query Language*), ODBC (*Open DataBase Connectivity*) y .Net, además de la tecnología Java utilizada para las soluciones Web Cliente.

[6] Incrustación y Enlazado de Objetos (*Object Linking and Embedding*)

1.2 SCADAs libres y de código abierto

En la actualidad existen dos movimientos importantes que promueven el desarrollo de software de libre distribución, estos son (Moya, 2009):

- La Fundación de Software Libre (*Free Software Foundation*)
- La Iniciativa de Código Abierto (*Open Source Initiative*)

La **Fundación de Software Libre**, desarrolló la Licencia Pública General (*GPL: General Public Licence*) GNU/GPL, la misma que promueve la libre distribución de software, incluyendo su código fuente. El objetivo principal es permitir a la comunidad de programadores realizar cambios al código fuente. De acuerdo a la licencia GNU/GPL ninguna aplicación que declare el uso de esta licencia puede ser distribuida sin su código fuente. Compartir entre los programadores el código fuente permite responder a defectos y problemas en las aplicaciones de manera más rápida y efectiva.

La **Iniciativa de Código Abierto** difiere un poco del movimiento de Software Libre, aunque comparten el objetivo principal de distribuir libremente el software. El movimiento de Código Abierto no se preocupa mucho si cierta persona obtiene ganancias de un determinado sistema, el movimiento está más preocupado de la distribución libre del mismo y su código fuente, lo que no implica que sea gratis. (Se debe garantizar que sin importar que el sistema tenga un costo inicial, su posterior distribución libre debe ser asegurada).

1.2.1 Características definidas por la Iniciativa de Código Abierto

La Iniciativa de Código Abierto (OSI *Open Source Initiative*) se encarga de mantener la definición de código abierto en beneficio de la comunidad y transmitir los beneficios del código abierto (Moya, 2009).

Código Abierto (*Open Source*) no solo significa acceso al código fuente de un determinado programa. Para que un software o aplicación sea catalogada de código abierto debe cumplir con las siguientes características:

- Redistribución libre: permite a los usuarios vender o distribuir el software de manera libre.

- Código fuente: el software debe incluir el código fuente, o que pueda ser obtenido de manera libre. El código fuente debe ser distribuido en forma adecuada, en la cual el programador pueda realizar cambios.
- Trabajos derivados: las modificaciones y los trabajos derivados, pueden ser distribuidos.
- Integridad del código fuente del autor: existe la posibilidad de que la licencia establezca que las modificaciones en el código fuente solo puedan ser distribuidas en forma de parches. Caso contrario, la licencia debe especificar el permiso de distribución del software con su código fuente modificado.
- Sin discriminación a personas o grupos: todas las personas pueden acceder al software de código abierto.
- Sin discriminación en contra de los campos de aplicación: no se puede restringir el uso en un determinado campo de aplicación. Por ejemplo, en el uso en un negocio particular.
- Distribución de la licencia: todos los usuarios a quienes se redistribuyó el programa, tienen los derechos especificados sobre el mismo.
- La licencia no debe ser específica a un determinado producto: los derechos establecidos de un determinado programa no deben depender en el hecho que el mismo este formando parte de una distribución particular de software.
- La licencia no debe restringir otro software: la licencia no puede obligar a otro software que se distribuya de manera conjunta con el software de código abierto.
- La licencia debe ser tecnológicamente neutral: no se requiere ninguna aceptación de la licencia.

1.2.2 SCADA libre y de código abierto en el mundo

En la actualidad existen varios proyectos y comunidades que se dedican a la investigación y desarrollo de los sistemas SCADAs libres, entre los que sobresale la región de Europa (Alemania) como uno de los mayores productores de estos sistemas.

Dentro de los principales sistemas SCADA libres y de código abierto se encuentran:

- **OpenHAB:** Es un proyecto de código abierto (*Open Source*) desarrollado en Java que pretende integrar múltiples sistemas de domótica que permiten automatizar

distintas funciones de la casa, por ejemplo: encender la calefacción o las luces en un determinado momento, o controlar éstas funciones desde tu teléfono móvil. Se ejecuta en dispositivos pequeños y baratos como la Raspberry Pi y permite explorar el mundo de la automatización del hogar más allá de lo ordinario (Bruce, 2015). Funciona utilizando enlaces (*bindings*), por lo tanto, es posible añadir paquetes de software para habilitar la comunicación con una variedad de sistemas como: KNX, Philips Hue, MiLight, Bluetooth, Asterisk, Modbus, Serial, por solo mencionar algunos (Correa and Trufiño, 2015; Martínez, 2014).

- **Domoticz:** Es un sistema de automatización del hogar escrito en C++ que tiene una interfaz de servidor web para los usuarios finales y que puede ejecutarse bajo Unix/Windows. Esta aplicación es ligera y permite controlar varios dispositivos como interruptores, luces, además se puede obtener la lectura de varios tipos de sensores como humedad, temperatura, lluvia o de viento. Este sistema está diseñado para operar en varios sistemas operativos y se puede instalar como servicio en sistemas embebidos como la Raspberry Pi (Correa and Trufiño, 2015).
- **HomeGenie:** Es un sistema basado en tecnologías web que permite controlar y visualizar dispositivos dentro de una casa inteligente (*smart home*) a través de modernas interfaces optimizadas para dispositivos móviles. Su principal característica es su sencillez. Puede ser instalado en varias plataformas informáticas incluido la Raspberry Pi (Correa and Trufiño, 2015).
- **OpenRemote:** Es una plataforma de integración para software de automatización mantenida por una comunidad de código abierto en el área de la automatización residencial y de edificios que promueven la utilización de estándares abiertos. Permite el manejo de varios protocolos tal como Z-Wave, Isnteon, X10, etc., además se ejecuta bajo múltiples plataformas tales como Linux, Windows, Mac, y diferentes arquitecturas como IBM, ARM, entre otras. El lenguaje de programación es el Java (Correa and Trufiño, 2015).
- **Eclipse OpenSCADA:** SCADA de código abierto desarrollado en Alemania en el 2006. Es independiente de la plataforma y se basa en un diseño moderno de un sistema que proporciona seguridad y flexibilidad al mismo tiempo. Es práctico y fácil de usar para llevar a cabo la supervisión, control y adquisición de datos. No es una

solución agrupada, sino un conjunto de herramientas (módulos) que se pueden combinar de diferentes maneras. La conectividad se realiza mediante un servidor OPC. Su estructura modular le ha permitido convertirse en un SCADA con buena perspectiva (Simó, 2012).

En la Tabla 1-1 se muestra una comparación con diferentes indicadores que permite establecer una comparativa clara entre los diferentes proyectos antes mencionados.

Tabla 1-1: Comparativa entre los proyectos libres y de código abierto

	OpenSCADA	**Domoticz**	**HomeGenie**	**OpenRemote**	**OpenHAB**
Protocolo de Comunicación	OPC, PLC	X10, Z-Wave	X10, Z-Wave	X10 Z-Wave, 1-Wire, Insteon	Z-Wave, Modbus, Serial, Bluetooth, KNX, 1-Wire, GPIO, HTTP, TCP/UDP
Lenguaje de Programación	Java	C++, Javascript, HTML	C#, Javascript, HTML	Java	Java, Javascript, HTML
Plataformas de softwares y hardware	Windows, Linux, Raspberry Pi	Windows, Linux, Raspberry Pi	Windows, Linux, Mac, Raspberry Pi	Windows, Linux, Mac, Raspberry Pi	Windows, Linux, Mac, Raspberry Pi
Soporte para dispositivos móviles	IOS, Android	-	Android	IOS, Android	IOS, Android
Licencia	EPL	GPL V3	CC BY-NC V2.0	Affero GNU Public License	EPL V1

De los sistemas SCADA antes mencionados el openHAB es el proyecto que mejor se ajusta a los requerimientos de los sistemas de supervisión y control en automatización de inmuebles debido a sus facilidades de programación, su pequeño requerimiento de hardware lo que lo hace muy factible en sistemas embebidos, además de que su uso es libre de costo y reglas.

Realmente en cuanto a funcionalidad es el sistema de los estudiados que más posibilidades ofrece. El diseño visual, sin ser espectacular, es moderno y la documentación es la más completa de las consultadas (Vega, 2016).

De cualquier modo, cuenta con una amplia comunidad que en muchos casos es la referencia oficial para solucionar los errores con los que se encuentran y que resaltan las carencias de la documentación en algunos apartados.

Con esta información se puede afirmar que openHAB es el proyecto más grande de los consultados, el más estable, flexible y el que más opciones aporta por lo que éste es el software base sobre el que se desarrolla la aplicación en este proyecto.

1.2.3 Código abierto frente al código cerrado

El software libre y de código abierto tiene muchas ventajas, no solo económicas, frente al uso de software propietario o de código cerrado. A continuación, se mencionan estas ventajas y desventajas.

Ventajas

Utilizar software de código abierto trae múltiples ventajas para sus usuarios, ya sean estos personas o empresas. Algunas de estas ventajas son (Asesores, 2012):

- La disponibilidad del código fuente hace posible que usuarios, programadores y empresas se involucren en el desarrollo de las aplicaciones. De esta forma, el proceso de detección y corrección de errores se lleva a cabo de forma eficiente, así como la implementación de nuevas características.
- Es posible realizar modificaciones a los programas con el fin de adaptarlos a las necesidades específicas de una empresa.
- Con el software de código abierto no existe un gasto de dinero en la compra de licencias, sino una inversión en la capacitación del personal.
- Al utilizar programas de código abierto no se depende de una empresa específica para las tareas de mantenimiento, sino que puede contratarse a cualquiera que tenga la habilidad y el conocimiento necesario.

Desventajas

Aunque las ventajas suelen anular a las desventajas, también existen algunas desventajas de usar este tipo de software en la empresa (Pérez, 2016).

- Falta de garantías al ser comunidades altruistas o sin ánimo de lucro, aunque no siempre es así y aunque no tienen garantía, en muchas ocasiones, el software de código abierto cumple mejor de lo que lo hace un software propietario con garantías.
- No siempre hay soluciones únicas o suites integradas (*best of suite*) que satisfacen todas las áreas de negocios y en algunos casos hay que disponer de diversos paquetes de software (estrategia *best of breed*).
- No maduran muy rápido al estar tan fragmentadas, y esto es un inconveniente que debe subsanarse, concienciando a la comunidad a unirse en vez de separarse en pequeños grupos de desarrollo y crear muchos programas redundantes o *forks* sin sentido en vez de centrarse en mejorar lo que ya existe.

1.3 Sistemas Embebidos

Se trata de un sistema de computación diseñado para realizar una o algunas funciones dedicadas en un sistema de computación en tiempo real. Al contrario de lo que ocurre con los ordenadores de propósito general (como por ejemplo una computadora personal o PC) que están diseñados para cubrir un amplio rango de necesidades, los sistemas embebidos se diseñan para cubrir necesidades específicas. En un sistema embebido la mayoría de los componentes se encuentran incluidos en la placa base (la tarjeta de vídeo, audio, módem, etc.) y muchas veces los dispositivos resultantes no tienen el aspecto de lo que se suele asociar a una computadora (Asesores, 2012).

Los sistemas embebidos se pueden programar en el lenguaje ensamblador del microcontrolador o microprocesador incorporado sobre el mismo, o también, utilizando los compiladores específicos, pueden utilizarse lenguajes como C o C++; en algunos casos, cuando el tiempo de respuesta de la aplicación no es un factor crítico, también pueden usarse lenguajes como Java (Asesores, 2012).

Su principal beneficio es su facilidad de uso, ya que mantienen la flexibilidad de una PC, con respecto a la variedad de procesos que pueden realizar, pero su pequeño tamaño, costo y bajo

consumo de potencia los hace ideales para aplicaciones donde el uso de un PC regular es excesivo (Martínez, 2014).

1.3.1 Comparación de Ordenadores de Placa Reducida

Actualmente en el mercado se encuentra una gran variedad de ordenadores de placa reducida. La mayoría de los procesadores de los *SBC*[7] (*Single Board Computer*) están basados en la arquitectura de procesadores ARM y algunos de ellos ofrecen un mejor rendimiento y más memoria que otros, algunos tienen una gran variedad de interfaces de entrada salida GPIO[8] (*General Purpose Input Output*), mientras que otros solo tienen el mínimo necesario. Con el fin de elegir el SBC adecuado para este proyecto, a continuación, en la Tabla 1-2 se presenta una comparación de los principales *SBC* disponibles en la actualidad (Pérez, 2014).

Tabla 1-2: Comparación de Ordenadores de Placa Reducida

	BeagleBone Black	**Raspberry Pi 3 Modelo B**	**Orange Pi Plus 2**	**Banana Pi M3**	**PandaBoard ES**
Frec. CPU (MHz)	1000	1200	1600	1800	1200
RAM (MB)	512	1000	2000	2000	1000
Precio (USD)	56	35	30	100	182
Arquitectura	ARM	ARM	ARM	ARM	ARM
Sistemas operativos soportados	Linux (Debian, Ubuntu), Android	Linux (Raspbian), RISC OS, Android	Linux (Ubuntu, Debian), Android	Linux (Debian, Lubuntu), Android, Bananian	Linux (Ubuntu), Android

[7] Ordenador de Placa Reducida

[8] Entrada Salida de Propósito General

Almacenamiento externo	MicroSD	MicroSD	MicroSD	MicroSD	MicroSD
Puerto HDMI	Si	Si	Si	Si	Si
Puerto Ethernet	Si	Si	Si	Si	Si
Wi-Fi interno	No	Si	Si	Si	Si
Entradas USB	2	4	4	4	4
Pines GPIO	Si	Si	Si	Si	Si

El ordenador de placa reducida más económico es la Orange Pi, mientras que el más costoso es PandaBoard. La BeagleBone es el ordenador de placa reducida más robusto, pero la Raspberry Pi con menor costo mejora sus características como: aumento de memoria RAM y frecuencia de la CPU.

La Orange Pi y la Banana Pi son una versión de la Raspberry Pi 2 Modelo B que mejoran sus características, como la velocidad del procesador, la memoria RAM e integra Wi-Fi, pero con la salida de la Raspberry Pi 3 ya se integran todas estas opciones por lo que se hace más factible su uso.

Una característica importante, pero difícil de cuantificar, es el soporte de la comunidad de desarrolladores. En este aspecto la Raspberry Pi se destaca, en parte debido a su fama, lo que incrementa el número de usuarios proficientes, y en parte también a que el proyecto Raspberry Pi tiene una inclinación educativa lo que hace que los creadores se preocupen por resolver los problemas que puedan surgir en los proyectos de los usuarios. La Raspberry Pi combina características muy positivas de hardware, como un buen rendimiento, buena memoria RAM y bajo consumo de potencia (12.5W) lo que lo hace ideal para este proyecto. Además, los pines GPIO permiten conectividad con hardware personalizado para desarrollos futuros (Martínez, 2014).

1.3.2 Selección de la Raspberry Pi

La Raspberry Pi es un dispositivo de bajo costo que permite a las personas de todas las edades explorar la computación para aprender a programar en lenguajes como Scratch y Python. Es

capaz de hacer todo lo que se espera de un ordenador de escritorio, desde navegar por Internet, reproducir vídeos en alta definición, crear hojas de cálculo y procesar textos (Suárez Pinzón, 2015).

La Raspberry Pi 3 Modelo B (Pérez, 2014) mostrada en la Figura 1-2 combina características muy positivas de hardware y costo lo que influyó en su selección para la realización de este proyecto.

Figura 1-2: Raspberry Pi 3 Modelo B

Especificaciones (Pérez, 2014):

- **CPU:** Broadcom BCM2837 QUAD ARM Cortex-A7, 1.2 GHz de 64 bits de 4 núcleos
- **GPU:** VideoCore IV 400 MHz
- **RAM:** 1 GB
- **Conectores:** USB, HDMI, Jack de audio 3.5mm, 10/100 Ethernet, micro USB, Wi-Fi, Bluetooh 4.1

1.4 Conclusiones del capítulo

Después de la búsqueda bibliográfica realizada y el estudio comparativo sobre SCADAs abiertos y plataformas embebidas se puede concluir que:

- Existe una diversidad de sistemas SCADAs de código abierto que se encuentran en desarrollo, aunque en muchos casos la documentación aún no es suficiente.
- El openHAB resulta entre los sistemas estudiados el que más se acerca al campo y objeto de estudio del presente trabajo.
- De los sistemas embebidos reportados en la literatura, la tarjeta Raspberry Pi no es la más potente en prestaciones, pero sí la que mejor relación costo/beneficio presenta con características suficientes para solucionar el problema de investigación del presente proyecto.

CAPÍTULO 2. SCADA de código abierto openHAB

En este capítulo se abordan las características principales del software openHAB, así como una detallada descripción de la filosofía de su funcionamiento, manuales de instalación, métodos de programación y las potencialidades que expone como plataforma SCADA para la automatización del hogar y el sector de servicios.

2.1 OpenHAB

OpenHAB es un sistema domótico de código abierto (*open source*) totalmente agnóstico en cuanto a tecnología se refiere. Se ejecuta en dispositivos pequeños y baratos como la Raspberry Pi y permite explorar el mundo de la automatización del hogar más allá de lo ordinario (Bruce, 2015). A través de su arquitectura modular de OSGi (*Open Services Gateway Initiative*) es extensible, de modo que puede comunicarse a una multitud de diferentes sistemas (Bueno, 2014).

Es un software desarrollado en Java que pretende integrar múltiples sistemas de domótica que permiten automatizar distintas funciones de la casa, como por ejemplo encender la calefacción o las luces en un determinado momento, o controlar éstas funciones desde un teléfono móvil (Soler, 2016).

Funciona utilizando enlaces (*bindings*), por lo tanto, es posible añadir paquetes de software para habilitar la comunicación con una variedad de sistemas como: KNX, Z-Wave, Serial, Bluetooth, Modbus, por solo mencionar algunos (Vega, 2016).

El proyecto openHAB se divide en dos partes (Correa and Trufiño, 2015):

Runtime: es el paquete basado en el *framework* OSGi que se ejecuta en el servidor. Permite la comunicación entre los dispositivos que se desea controlar y la plataforma.

D*esigner:* es una aplicación basada en Eclipse RCP[9] para configurar en tiempo de ejecución el *Runtime*. Permite editar archivos de configuración, generación de interfaces y de reglas.

2.2 Arquitectura

2.2.1 Arquitectura software openHAB

Al estar basado en OSGi, openHAB proporciona una arquitectura altamente modular, lo que permite incluso agregar y/o quitar una funcionalidad en tiempo de ejecución sin detener el servicio. OpenHAB le proporciona al usuario diferentes interfaces web basadas en HTML, capaces de funcionar en cualquier dispositivo móvil o de escritorio, así como aplicaciones nativas para iOS y Android. Además, si es necesario integrarlo con otros sistemas, puede interactuar con ellos mediante un interfaz de servicio REST API, permitiendo el acceso en lectura a los elementos, así como actualizaciones de su estado o el envío de comandos hacia los dispositivos (Martínez, 2014).

La Figura 2-1 muestra un resumen de los principales componentes de openHAB y la forma en que dependen unos de otros:

[9] Plataforma Eclipse de Cliente Enriquecido (*Rich Client Platform)*

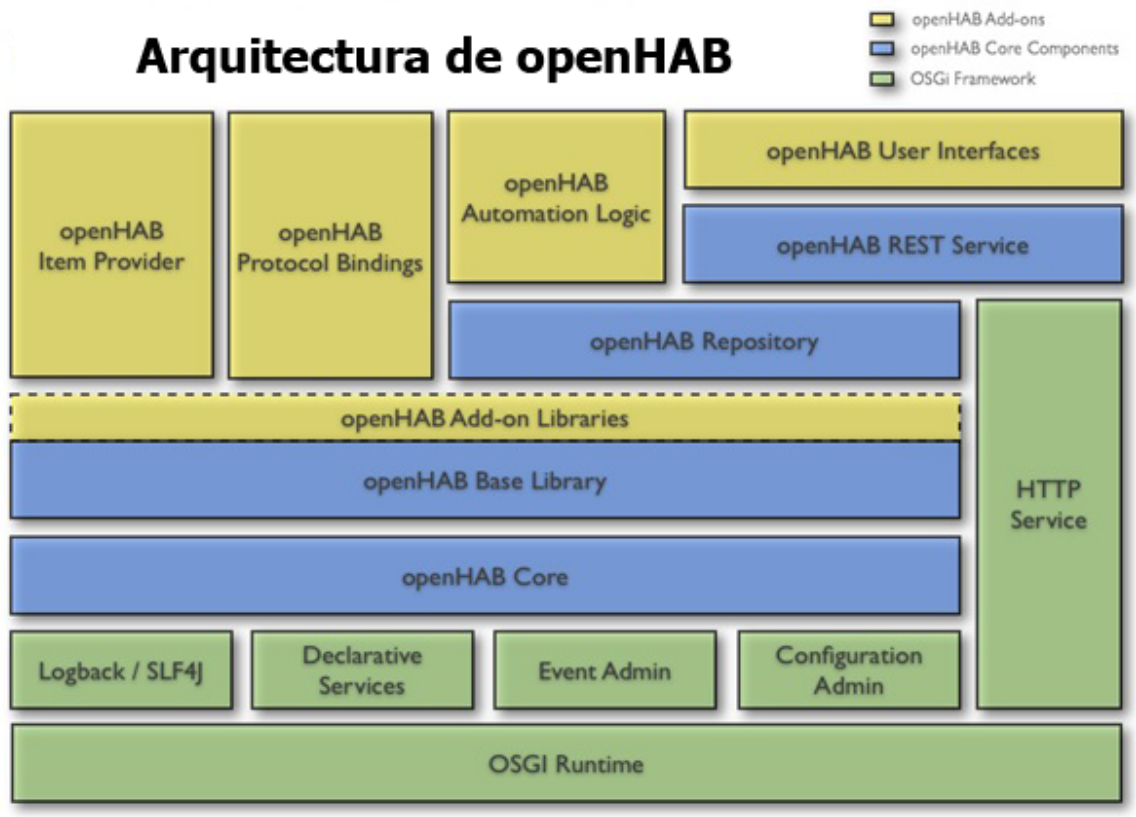


Figura 2-1: Arquitectura software openHAB

2.2.2 Arquitectura del Software

En la Figura 2-2 se muestra la arquitectura de software donde se identifica las tres secciones definidas e independientes del sistema de domótica. A lo largo del capítulo se profundiza más sobre cada uno de los componentes que lo conforman.

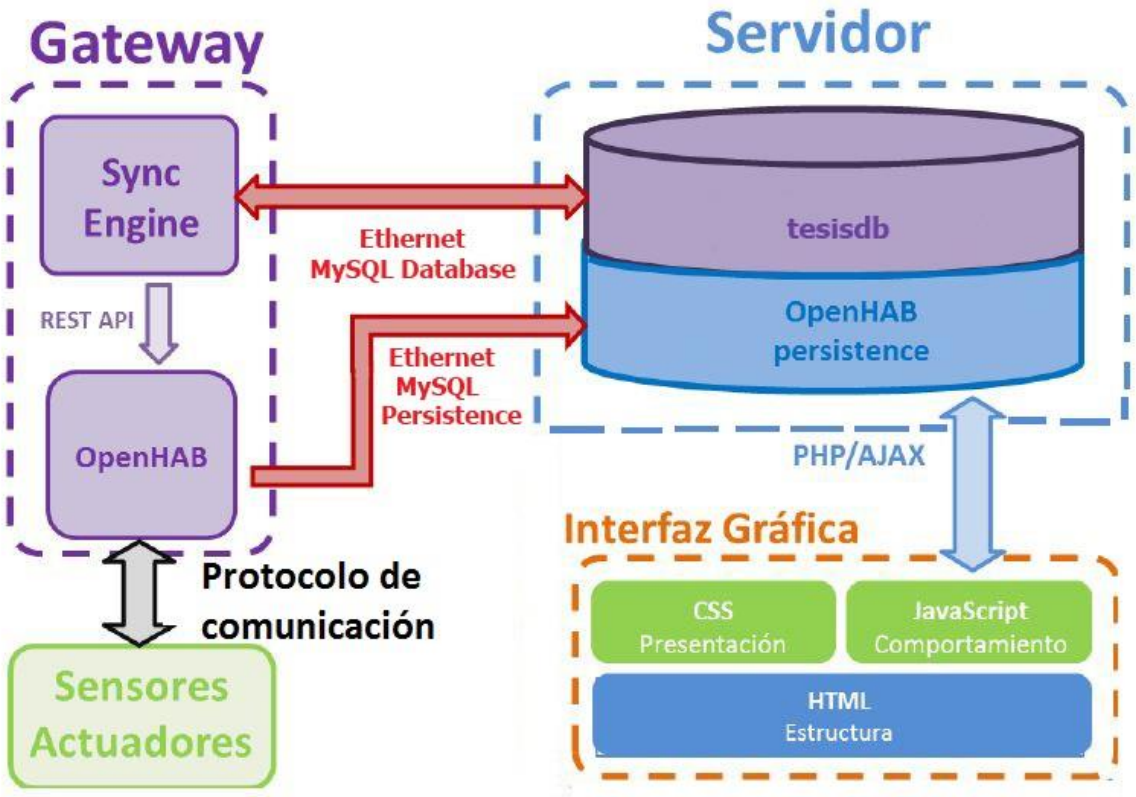


Figura 2-2: Arquitectura del software

La primera de estas secciones es el servidor en se encuentra la base de datos, que almacena los estados de los dispositivos, usando la persistencia de openHAB y que guarda además los comandos que el usuario realiza.

La segunda sección es el Gateway, componente hardware que toma el papel de centro de control del sistema y dónde está instalado y configurado openHAB, que es el encargado de administrar los dispositivos a través del protocolo de comunicación. En este dispositivo es ejecutado además el motor de sincronización encargado de mantener alineada la información real, la información en el servidor y los datos mostrados en la interfaz de usuario.

La última sección que conforma la arquitectura es la interfaz gráfica, en ésta el usuario puede supervisar el estado de los sensores e interactuar con ellos. Tiene tres componentes básicos: el HTML, el CSS y el JavaScript, donde este último se encarga de comunicar los estados entre la base de datos y la interfaz gráfica usando el método AJAX.

2.2.3 Canales de comunicación en openHAB

OpenHAB tiene dos canales diferentes de comunicación internos, un bus de eventos asíncronos y un repositorio de estados que puede ser consultado. En la Figura 2-3 se muestra como son usados los canales de comunicación (del Sol, 2016; Martínez, 2014):

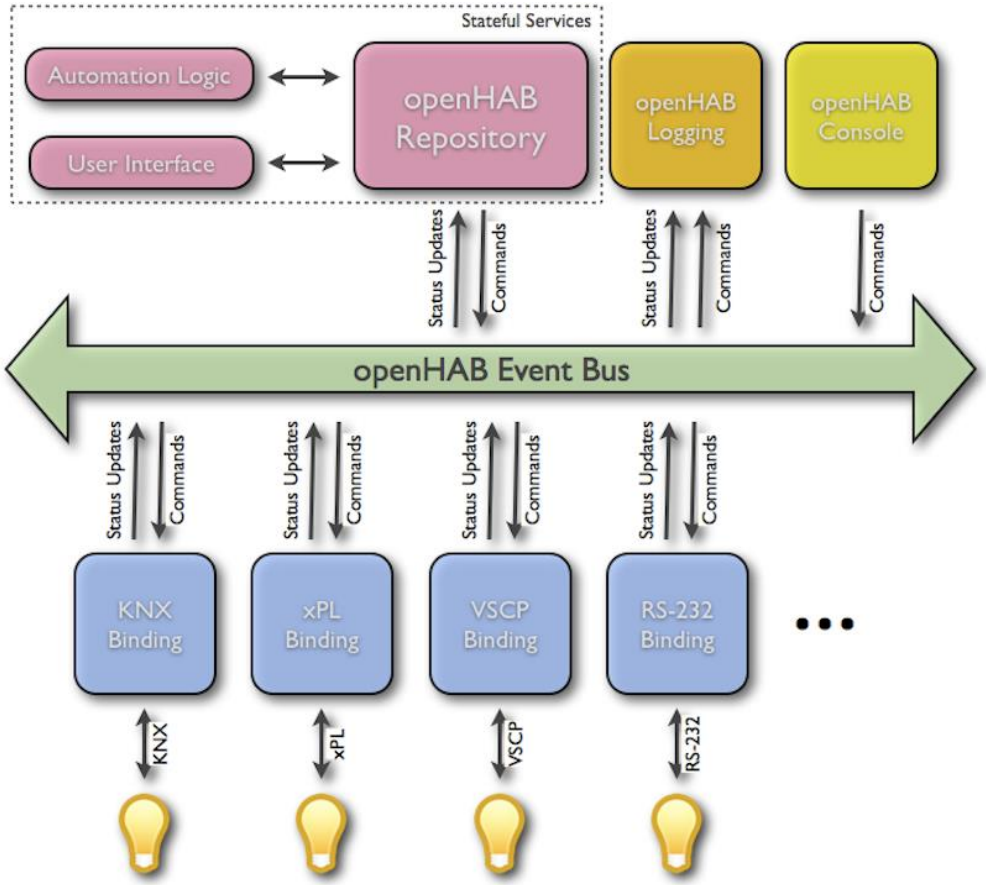


Figura 2-3: Canales de comunicación openHAB

El bus de eventos es el servicio de base de openHAB y todos los paquetes que no requieren un comportamiento basado en estados deben usarlo para informar a otros sobre los eventos y para ser actualizados por eventos externos. Hay dos tipos de eventos:

- Comandos que activan una opción o un cambio de estado de algún elemento/dispositivo.
- Actualizaciones de estado que informan sobre un cambio de estado de algún elemento/dispositivo.

Un *binding* es un paquete opcional que se puede usar para extender la funcionalidad de openHAB, además, permiten la conexión con elementos que usan tecnologías diferentes. Todos los *binding* deben comunicarse a través del bus de eventos, lo que asegura que existe un acoplamiento muy bajo entre los paquetes y facilita la dinámica de openHAB.

Es importante tener en cuenta que openHAB no está destinado a residir en dispositivos hardware que luego tengan que comunicarse de manera remota con otras instancias de openHAB distribuidas. En su lugar, openHAB sirve como centro de integración entre estos dispositivos y como un mediador entre los diferentes protocolos que se comunican entre estos dispositivos.

2.3 Programación

La programación de openHAB se realiza teniendo en cuenta cada uno de los elementos mostrados en la Tabla 2-1.

Tabla 2-1: Descripción de la función de las carpetas en la configuración de openHAB

Carpeta	Descripción
transforms	Acá es posible definir a través de archivos .map una "traducción" para los estados retornados por los diferentes dispositivos. Por ejemplo, se desea que para un interruptor encendido valor que se muestra en la interfaz gráfica de openHAB sea "ENCENDIDO" en vez de "ON".
sitemaps	El *sitemap* corresponde a la distribución lógica y gráfica de las interfaces de usuario nativas de openHAB. Esta carpeta contiene archivos con extensión .sitemap, los cuales contienen descripciones de la diagramación de los *items* en la interfaz.
scripts	Un ***script*** es la manera de crear código reutilizable para las reglas. Un archivo .script contiene instrucciones que pueden ser utilizadas simplemente incluyendo este archivo en la lógica de la regla.
rules	Con las reglas es posible definir respuestas automatizadas a diferentes eventos, como cambios en el estado de algún dispositivo, condiciones de tiempo o cambios en el sistema. Las reglas son escritas en un pseudo lenguaje similar a JAVA.

persistence	Esta carpeta contiene archivos con extensión .persist en los que es posible configurar como se desea almacenar la historia del estado de los *items* en bases de datos, archivos, etc.
items	Contiene archivos .item que son usados para definir los *items*. Un *item* es un objeto que se puede leer o escribir con el fin de interactuar con ellos, pueden estar vinculados a un dispositivo real o corresponder a dispositivos virtuales.
icons	En esta carpeta se puede incluir los iconos que se deseen utilizar para la programación. Estos iconos pueden ser .png o .svg. Luego según el panel de usuario que se vaya a utilizar se debe configurar para que estos iconos sean aceptados.

2.3.1 Items

OpenHAB tiene una separación estricta entre el mundo físico y la aplicación que se construye alrededor del concepto de "los *items*" (también llamada la capa virtual). Los *items* representan la funcionalidad que es usada por la aplicación (interfaces del usuario o lógica de automatización). Estos tienen un estado y se usan a través de los eventos. Un *item* es un objeto que puede ser leído o escrito para interactuar con él y deben estar definidos en un archivo con la extensión .items (OpenHAB, 2015a). En general los *items* se definen de la siguiente manera:

itemtype itemname "labeltext" <iconname> (group1, group2, ...) {bindingconfig}

Por ejemplo:

Switch Ligth_Juegos "Luces de la Sala de Juegos" (Juegos, Light) {gpio="pin:27"}

En la

Tabla *2-2* se nombran los diferentes tipos de *item* que pueden ser empleados en openHAB.

Tabla 2-2: Descripción de los tipos de *item* que se pueden utilizar en openHAB

Itemname	**Descripción**	**Tipos de Comandos**
Color	Información del color (RGB)	On/Off, Increase/Decrease, Percent, HSB
Contact	*Item* que guarda el estado del contacto por ejemplo puerta/ventana	Open/Closed
DateTime	Almacena un dato de tiempo	
Dimmer	Acepta porcentajes para establecer el valor	On/Off, Increase/Decrease, Percent
Group	Colección de *items*	Acepta el tipo de comando de los *item* a los que hace referencia
Number	Puede ser usado para cualquier tipo de sensor numérico como temperatura, brillo, viento, etc o como un contador.	Decimal
Player	Permite controlar audio/vídeo	Play/Pause, Next/Previous, Rewind/Fastforward
Rollershutter	Muy utilizada para persianas	Up/Down, Stop/Move, Percent
String	Guarda textos	String
Switch	Usado para luces	On/Off

En el Anexo 1 se muestran ejemplos de los *items* usados en este proyecto.

2.3.2 Sitemaps

Un *sitemaps* es un modelo visual o textualmente organizado del contenido de un sitio web que permite a los usuarios navegar a través de una página web para encontrar la información que buscan. En openHAB, el término *sitemaps* se utiliza para referirse al documento que enumera el contenido de su sistema de automatización que se muestra en la pantalla del usuario (OpenHAB, 2015b). En la Tabla 2-3 se muestra los elementos que pueden ser usados.

Tabla 2-3: Descripción de los elementos del *sitemap*

Elementos	Descripción
Chart	Agrega un objeto gráfico para mostrar los datos registrados
Colorpicker	Permite al usuario elegir el color
Frame	Área con otros elementos de *sitemap* o marcos anidados adicionales
Group	Agrupa los elementos definidos en un grupo
Image	Muestra una imagen
Selection	Da acceso a una nueva página donde el usuario puede elegir entre los valores definidos.
Setpoint	Muestra un valor y permite al usuario modificarlo. Se pueden especificar valores máximos y mínimos.
Slider	Control deslizante
Switch	Interruptor
Text	Texto
Video	Muestra un vídeo

Webview	Muestra una página web

En el Anexo 2 se muestra la sintaxis del *sitemap* desarrollado para esta aplicación.

2.3.3 Reglas (rules)

Las reglas en openHAB son usadas para automatizar procesos mediante la ejecución de un *script* y un motor de reglas integrado y potente.

Las reglas deben estar en un archivo con extensión .rules en el directorio de openHAB. Un archivo de reglas puede contener varias reglas y todas ellas comparten un contexto de ejecución común, es decir, pueden acceder e intercambiar variables entre sí. Por este motivo puede tener sentido tener diferentes archivos de reglas para diferentes casos de uso.

La sintaxis que debe seguir un archivo de reglas es: primero las importaciones, seguido de las variables y por último la definición de la lógica de la regla como se muestra en la Figura 2-4. Las importaciones se realizan de igual forma que en Java y hacen que se disponga más fácil de los tipos importados. La declaración de variables se usa para declarar todas las variables que deben ser accesibles para todas las reglas de este archivo; pueden ser declaradas con o sin valores iniciales. Las reglas pueden tener cualquier número de condiciones, pero al menos debe tener una que, cuando se cumpla, permita que se ejecute el script que se desea (OpenHAB, 2015c).

```
[Imports]

[Variable Declarations]

[Rules]
```

Figura 2-4: Estructura de las reglas

En la Figura 2-5 se muestra la sintaxis que debe seguir cada una de las reglas:

```
rule "rule name"
when
    <TRIGGER CONDITION1> or
    <TRIGGER_CONDITION2> or
    <TRIGGER_CONDITION3>
    ...
then
    <EXECUTION_BLOCK>
end
```

Figura 2-5: Sintaxis de las reglas

En el Anexo 3 se muestra ejemplos de la sintaxis de las reglas empleadas en este proyecto.

Antes de que una regla empiece a funcionar debe ser activada. La activación de las reglas se hace por disparadores y hay tres tipos:

- Disparadores basados en *items*: reaccionan sobre eventos en el bus de eventos de openHAB, es decir, comandos y actualizaciones de estado de los *items*.
- Disparadores temporales: reaccionan en momentos especiales que sean interesantes, horarios habituales (mediodía, noche), en una hora concreta o cada cierto tiempo.
- Disparadores basados en el estado del sistema: reaccionan cuando el sistema se encuentra en un estado determinado.

2.3.3.1 Reglas basadas en *items*

Este tipo de reglas puede procesar comandos de un elemento específico, actualizaciones de estado o cambios de estado (una actualización puede dejar el estado sin cambio). Se puede decidir si se desea capturar solo un comando/estado específico o cualquiera.

```
Item <item> received command [<command>]
Item <item> received update [<state>]
Item <item> changed [from <state>] [to <state>]
```

Figura 2-6: Sintaxis de las reglas basadas en *items*

La primera opción con la que nos encontramos en la Figura 2-6 es que un *item* reciba un comando, otra opción es que reciba una actualización y, por último, otro tipo de disparador es que haya un cambio en el estado del *item* o que cambie de un estado “A” a un estado “B”.

2.3.3.2 Reglas basadas en el tiempo

Para los disparadores basado en el tiempo se pueden usar expresiones ya predefinidas en el sistema o utilizar una expresión tipo ‘cron’ en su lugar. Como expresiones predefinidas podemos encontrar medianoche o mediodía.

Respecto a las expresiones de tiempo son expresiones que tienen siete campos: segundos, minutos, horas, día del mes, mes, día de la semana y año. Con estas expresiones podemos establecer cualquier tramo horario, fecha concreta, etc.

En la Figura 2-7 se muestra la sintaxis para la declaración de una regla usando la expresión ‘cron’.

```
rule "basadas en el tiempo"
    when
        Time cron "0 0/5 * * * ?"
    then
    ...
end
```

Figura 2-7: Ejemplo de regla disparada por expresión temporal

2.3.3.3 Reglas basadas en el estado del sistema

Las reglas más importantes, en cuanto a planificación se refiere, siempre se realizan en el inicio o apagado del sistema. Estas reglas solo se ejecutan una vez, cuando el sistema detecta que está en uno de los dos estados descritos. A continuación en la Figura 2-8 se muestra la sintaxis para los disparadores del sistema.

```
System started
System shuts down
```

Figura 2-8: Sintaxis de la regla basadas en el estado del sistema

2.3.4 Bindings

OpenHAB cuenta con una variedad de *binding* muy potentes que se encargan de conectar los dispositivos de la casa inteligente y tecnologías al openHAB.

En la Tabla 2-4 se muestra algunos de los posibles *binding* que se pueden implementar en openHAB.

Tabla 2-4: Algunos de los *bindings* soportados por openHAB

Binding	**Descripción**
Exec	Este *binding* integra la posibilidad de ejecutar comandos desde la base de cualquier sistema operativo
GPIO	Este *binding* es para el subsistema local de GPIO
HTTP	Pide a un URL un comando cuando se envía los estados de los *item*, o encuesta un URL dado y actualiza los *item*
KNX	El *binding* KNX permite administrar este tipo de dispositivos
Modbus	Este *binding* soporta configuración para TCP y esclavo serie. Además permite configuración para Modbus RTU, ASCII, BIN y variantes de Modbus serie.
MQTT	Este *binding* permite a openHAB actuar como un cliente de MQTT, para que los *item* del openHAB puedan enviar y recibir mensajes *to/from* al servidor de MQTT.
Serial	El *binding* serie permite a openHAB comunicar en ASCII a través de los puertos serie vinculados al servidor de openHAB.

2.3.5 Exec

En este trabajo se hizo necesario la utilización del *binding Exec* para poder realizar la lectura de la temperatura con el sensor LM75 (Ventura, 2016) conectado a la Raspberry Pi a través de los pines GPIO.

Exec es un *binding* de gran potencialidad que ofrece openHAB el cual permite ejecutar comandos desde la base de cualquier sistema operativo para persistir los estados de los *items*.

Configuración de los items

Este *binding* puede ser utilizado para ejecutar comandos, devolviendo a la salida el estado del *item* enlazado o en respuesta a los comandos enviados.

Actualización del estado del *item*

Cuando se actualiza el estado de un *item* se ejecuta el siguiente comando:

```
exec="<[<commandLine to execute>:<refreshintervalinmilliseconds>:(<transformationrule>)]"
```

Donde:

<*commandLine to execute*>: es la línea del comando para ejecutar.

<*refreshintervalinmilliseconds*>: es la frecuencia con la que se ejecuta la línea del comando.

<*transformationrule*>: es opcional, y se puede usar para transformar el *string* devuelto del comando antes de poner al día el estado del *item*.

Enviando comandos

Al ejecutar una línea del comando en respuesta al *item* recibe el comando:

```
exec=">[<openHAB-command>:<commandLine to execute>] (>[<openHAB-command>:<commandLine to execute>]) (>[...])"
```

Donde:

<*openHAB-command*>: es el comando del openHAB que activa la ejecución de la línea del comando.

<*commandLine to execute*>: es el comando a ejecutar.

En el Anexo 4 se muestra un ejemplo de la utilización del *binding* E*xec* en este trabajo.

2.3.6 Base de Datos

El apoyo de persistencia guarda los estados del elemento con el tiempo. OpenHAB no se restringe a una sola base de datos. Las diferentes bases de datos pueden co-exitir y configurarse de manera independiente.

Al persistir los estados, hay muchas posibilidades en las que se pueden pensar: bases de datos correlativas, bases de datos de NoSQL, bases de datos del *round-robin*, Internet de las cosas (IoT) con los servicios de la nube, un simple archivo de registro, etc. OpenHAB intenta hacer todas estas opciones posible y configurable de la misma manera.

En openHAB estas diferentes opciones son llamadas "servicios de persistencia". En la Tabla *2-5* se muestran algunos de los posibles servicios de persistencia que están en la actualidad disponibles en openHAB (OpenHAB, 2015d).

Tabla 2-5: Algunos de los servicios de persistencia disponibles en openHAB

Servicio de persistencia	Descripción
db4o	Base de datos 100% Java
Exec	Persiste ejecutando los órdenes en la base de cualquier sistema operativo
IFTTT	*if-this-then-that* servicio de la nube
JDBC	Conexión de base de datos Java soportada por MySQL y otras bases de datos *JDBC*-habilitados (una tabla por *item*)
Logging	Escribe los estados del *item* para registrar los archivos con un formato flexible
MapDB	Sólo salva el último estado del *item*; útil para la estrategia del restoreOnStartup
MQTT	guarda los estados del *item* en un servidor de MQTT
openHAB Cloud Connector	Envía los estados del *item* al servicio de la nube de openHAB, y usa los *item* del openHAB en las recetas de IFTTT
MySQL	Una tabla de SQL por *item*

RRD4J	La versión poderosa de Java de la base de datos del *round-robin* RRDtool. Sólo almacena los estados numéricos.
Sen.Se	Envía los estados del *item* al sitio web Sen.Se

En este trabajo la base de datos utilizada es MySQL, un poderoso sistema de gestión de base de datos, multihilo, multiusuario y de código abierto, que es muy popular al ser parte de la plataforma LAMP[10].

La creación de una base de datos hospedada en el servidor de phpMyAdmin es necesaria para almacenar el comportamiento de las variables y generar históricos como los de tendencias.

La configuración realizada en el archivo mysql.cfg se encuentra en el Anexo 5.

2.4 Instalación de openHAB

A la fecha, la distribución oficial de openHAB es la 2.0.0 y se puede descargar en la página oficial (http://www.openhab.org/).

Para completar la instalación fue suficiente descomprimir el archivo descargado "distribution-2.0.0-runtime.zip" en una carpeta a elección, que en este caso se nombró openhab. Después de realizar este procedimiento, al interno de la carpeta mencionada se encuentra la serie de archivos y carpetas mostrada en la Figura 2-9.

[10] Acrónimo usado para denominar al conjunto de soluciones de servicios web que consiste de los siguientes softwares: Linux, Apache, MySQL y PHP.

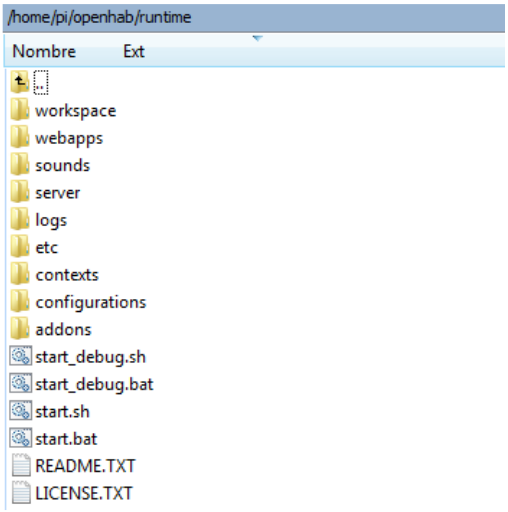


Figura 2-9: Árbol de carpetas del *runtime* de openHAB

El siguiente paso consistió en copiar los *addons* utilizados para el desarrollo de este proyecto en la carpeta "*addons*" del directorio *runtime*; éstos se encuentran en el archivo descargado denominado "distribution-1.8.3-addons.zip".

Los *addons* que se utilizaron fueron:

org.openhab.binding.modbus-1.8.3.jar

org.openhab.persistence.mysql-1.8.3.jar

org.openhab.binding.exec-1.8.3.jar

Una vez configurado como anteriormente se explica, se inicia el *runtime* ejecutando desde una terminal el archivo ./start.sh en Linux o start.bat en Windows. Luego se apunta el navegador web a la dirección http://<openHAB *address or hostname*>:8080 y se puede observar la pantalla principal que ofrece los diferentes paneles web UIs (Anexo 6).

2.5 Protocolos domóticos

Existen numerosos protocolos domóticos disponibles para tareas como la que nos proponemos para este proyecto, por lo que parece interesante mencionar en este punto dichas opciones. Dentro de estos protocolos, los más importantes, conocidos y que pueden ser implementados en openHAB son:

KNX: sistema descentralizado, es decir, no requiere de un controlador central de la instalación. En él, todos los dispositivos que se conectan al bus de comunicación de datos tienen su propio microprocesador y electrónica de acceso al medio.

KNX está basado en el modelo OSI[11] (*Open System Interconnection*) y a día de hoy presume de ser "el único ESTÁNDAR abierto para todas las aplicaciones de control de la vivienda y el edificio, como por ejemplo: el control de la iluminación y las persianas, así como variados sistemas de seguridad, calefacción, ventilación, aire acondicionado, monitorización, alarma, control de agua, gestión de energía, contador, así como electrodomésticos del hogar, audio/vídeo y mucho más" según la organización que lo gestiona, la KNX Association (Vega, 2016).

X10: es un protocolo de comunicación para el control remoto de dispositivos eléctricos que utiliza la línea eléctrica (220V o 110V) preexistente para transmitir señales de control entre equipos de automatización del hogar (domótica) en formato digital. Fue desarrollado en 1978 por *Pico Electronics of Glenrothes*, Escocia, para permitir el control remoto de los dispositivos domésticos (Vega, 2016).

Los productos compatibles con X10 son: alarmas, televisores, contestadores, interfaces de ordenador, etc. A pesar de que sólo tiene seis funciones, ha cubierto un hueco muy importante en el mercado y se ha consolidado como una buena y barata línea de productos, lo que ha acercado la domótica a presupuestos menores.

Modbus: protocolo de comunicación que ocupa el nivel 7 del modelo de referencia OSI y que proporciona comunicaciones maestro/esclavo entre recursos inteligentes. Fue desarrollado por Modicon (actualmente *Schneider Automation*) en 1979. Es una especificación abierta muy extendida en el mundo industrial debido a su simplicidad. Usado en dispositivos como PLC, HMI, drivers, sensores o actuadores remotos.

Este protocolo se puede implementar de las siguientes tres formas:

- TCP: Utilizando Ethernet como enlace de datos y acceso al medio.
- Transmisión serie asíncrona sobre diversos medios: RS232/422/485.

[11] Modelo de interconexión de sistemas abiertos

- RTU: conexión de un ordenador de supervisión con una unidad remota en sistemas de supervisión y control (SCADA).

Para el desarrollo de la aplicación ejemplo que se realiza durante esta investigación se utiliza el protocolo Modbus ya que es un protocolo abierto, estándar de facto, además su implementación es fácil y requiere poco desarrollo.

A continuación, se muestra como puede ser configurado este protocolo para su implementación en el software openHAB en el archivo modbus.cfg (OpenHAB, 2015e).

La configuración de los parámetros específicos para cada esclavo tiene el modelo siguiente y en la Tabla 2-6 se muestran los parámetros válidos:

```
<slave-type>.<slave-name>.<slave-parameter-name>=<slave-parameter-value>
```

Donde:

<slave-type> puede ser cualquiera TCP o Serie, depende del tipo de esclavo de Modbus.

<slave-name> es un único nombre por esclavo al que se puede conectar.

<slave-parameter-name> identifica el parámetro para configurar.

<slave-parameter-value> es el valor del parámetro.

Tabla 2-6: Parámetros válidos del esclavo

Propiedades	Requerimiento	Definición
poll	No	Frecuencia de registro en milisegundos. Por defecto es 200.
connection	Obligatorio	Para TCP esclavo use host_ip:[port] por ejemplo: 192.168.1.55:511. Si omite el puerto se usa 502 por defecto. Para conexión Serie use port:[:baud:[dataBits:[parity:[stopBits:[encoding]]]]] por ejemplo: dev/ttyS0:38400:8:none:1:rtu (lectura desde dev/ttyS0 usando la tasa de baudio

		38400, 8 bit de datos, ninguna paridad, 1 bit de stop y codificación de rtu).
id	Opcional	Por defecto el id del esclavo es 1.
type	Obligatorio	Tipo de objeto. Puede ser cualquiera de estos coil, discrete, holding o input.
start	Opcional	La dirección del primero coil/discrete input/register para leer. Por defecto es 0.
length	Obligatorio	Es el número de datos de los *items* que se van a leer. Los datos del *items* aquí se refieren a *register*, *coil* o *discrete input*, dependiendo del tipo de esclavo. Por ejemplo, si la meta es leer un *items* con *valuetype*=int32, necesita leer 2 registros (2*16 bit=32 bit).
valuetype	Opcional	Dice cómo interpreta los datos del registro.

Tipos de objetos soportados por Modbus:

- *coils*, también conocido como Salidas Digitales (*DO)* (lectura y escritura)
- *discrete inputs*, también conocido como Entradas Digitales *(DI)* (solo lectura)
- *input registers* (solo lectura)
- *holding registers* (lectura y escritura)

Función de lectura y escritura:

Modbus tiene diferentes especificaciones de operaciones para lectura y escritura de los diferentes tipos de *items*. Estos tipos de funcionamientos son identificados por el código de la función.

Este *binding* usa los siguientes códigos de función cuando se comunican con los esclavos:

- *read coils*: código de la función (FC) 1 (Lectura)

- *write coil*: FC 5 (Solo escritura)
- *read discrete inputs*: FC 2 (Lectura de entradas discretas)
- *read input registers*: FC 4 (Lectura de entradas de registro)
- *read holding registers*: FC 3 (Lectura múltiple del registro de retención)
- *write holding register*: FC 6 (Solo escritura del registro de retención) o FC 16 (Escritura múltiple del registro de retención).

En el Anexo 7 se muestra una de las posibles configuraciones de Modbus que se realiza en este trabajo.

2.6 Conclusiones del capítulo

Con el desarrollo de este capítulo donde se realiza una detallada descripción de la configuración de los parámetros necesarios para la implementación de una aplicación con el software openHAB en Raspberry Pi se puede concluir que:

- Es una herramienta muy útil para el control de una casa o edificio.
- En la actualidad cuenta con soporte para una gran variedad de protocolos de comunicación y dispositivos.
- Su programación y configuración es relativamente fácil.
- Cuenta con una comunidad siempre creciente de desarrolladores por lo que es un claro ejemplo de lo que se puede lograr a través de proyectos de software de código abierto.

CAPÍTULO 3. OpenHAB como propuesta de automatización en una instalación hotelera

En este capítulo se presenta una propuesta de automatización para los hoteles del Cayo Santa María basada en el sistema de supervisión y control openHAB ejecutado en una Raspberry Pi como una solución de automatización más económica para el país. Además, se muestran pruebas y resultados obtenidos, así como un análisis económico del sistema propuesto.

3.1 Arquitectura del sistema de automática

La solución presentada relacionada con el sistema de automática para el pueblo Dunas 5 ubicado en el Cayo Santa María abarca los siguientes lugares: Restaurantes Especializados, Bolera, Galería de Tiendas, Snack Bar, Sala de Fiestas y Oficinas de Gaviota Tur. El sistema prevé un puesto de control central (PC) enlazado con todos los autómatas desde donde se realiza la supervisión y accionamiento (cambio de *set point*) de los diferentes sistemas tecnológicos y equipos. El sistema está concebido de manera tal que cada autómata puede trabajar de manera autónoma, aún si falla el enlace entre ellos y el PC.

En la Figura 3-1 se puede observar cómo se encuentra la arquitectura montada en el pueblo Dunas 5.

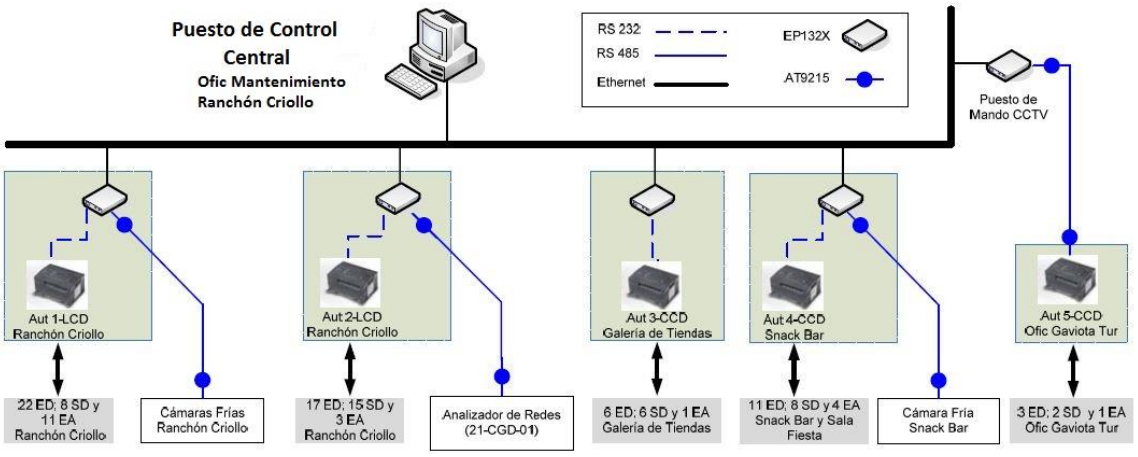


Figura 3-1: Arquitectura del sistema de automática montado actualmente

El enlace (la comunicación) entre el PC y los todos los autómatas se realiza por Ethernet utilizando la red de datos del sistema de gestión hotelera. Para lograr el enlace se utiliza convertidores Ethernet /puerto serie EP132X.

Todos los enlaces con los dispositivos de campo con posibilidad de comunicación por BUS (analizador de redes y cámaras frías) se establece por medio de BUSES RS-485 y protocolo de comunicación Modbus. La utilización de los equipos EP132X garantiza que los dispositivos de campo estén enlazados, directamente y por Ethernet, con el PC sin que medie ningún autómata. Esto garantiza inmunidad en el enlace ante fallas en el autómata correspondiente (los dispositivos de campo –analizador de redes y cámaras frías- con posibilidad de comunicación por BUS tienen el mismo nivel jerárquico que los autómatas).

En la Figura 3-2 se muestra una posible configuración más económica dónde solo se sustituye la computadora del puesto de control central por una Raspberry Pi desde la cual se puede realizar la supervisión y accionamiento de los diferentes sistemas tecnológicos y equipos.

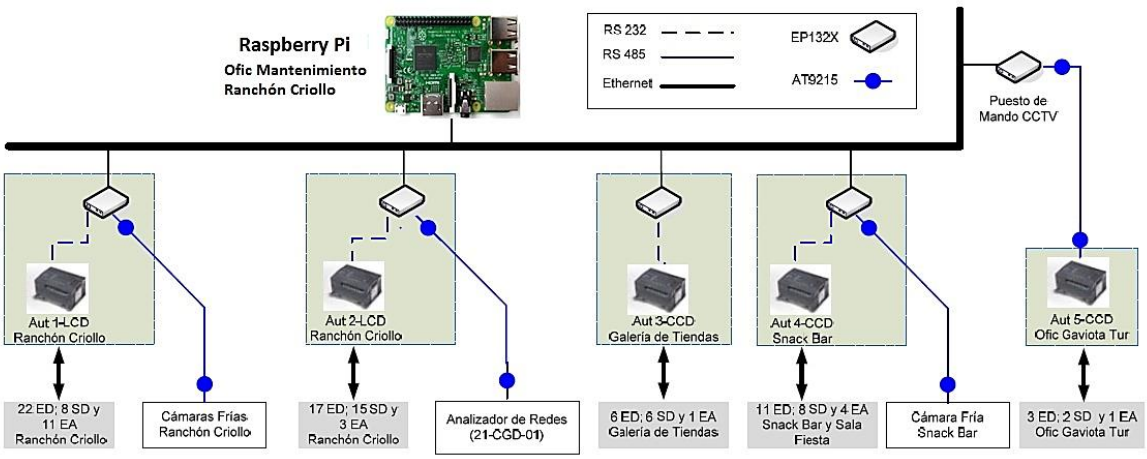


Figura 3-2: Arquitectura del sistema de automática con Raspberry Pi como servidor principal

Existe otra configuración que permite disminuir los costos de automatización manteniendo la robustez y seguridad del sistema. En la Figura 3-3 se muestra la arquitectura en la cual se utiliza la Raspberry Pi como servidor y computador central y la tarjeta de expansión IO Pi Plus de AB *Electronics* basada en el *I/O expander* de 16-bit con la cual se puede sustituir todos los PLC ya que permite conectar hasta 128 dispositivos, lo cual es suficiente según las necesidades del proyecto.

Especificaciones de la tarjeta (AB Electronics, 2016)**:**

- 32 entradas/salidas digitales
- Control mediante el puerto I2C de Raspberry Pi
- Acopla hasta 4 tarjetas en una sola Raspberry Pi
- Salidas configurables como *active-high*, *active-low* o *open-drain*
- Alimentación 5V y 3.3V

Figura 3-3: Arquitectura basada en Raspberry Pi con tarjeta de expansión IO Pi Plus

Los dispositivos que están presentes en la aplicación son mostrados en la Tabla 3-1:

Tabla 3-1: Dispositivos presentes en la aplicación

Dispositivos	**Cantidad**
Extractor	63
Extractor-Ventilador	16
Roof Top	2
Manejadoras de aire	10
Alumbrado Exterior	3
Cisterna	4
Pareja de Bombas	8

3.2 HMI para el sistema de automática Dunas 5

Como se ha explicado en otros puntos de este trabajo, la cualidad de una página web de ser *responsive* significa que de manera automática se adapta a la pantalla según la resolución de esta por lo que, independientemente del dispositivo que se emplee para su visualización (PC, tablet, *smartphone*, etc), el contenido se puede observar bien adaptado a su pantalla.

En la Figura 3-4 podemos apreciar el aspecto que presenta la pantalla principal visto desde un navegador web (en este caso Mozilla Firefox) en una PC. Desde esta pantalla el usuario puede acceder a cada una de las instalaciones del pueblo y desde allí controlar y accionar los diferentes dispositivos instalados en las habitaciones. Además, tiene la posibilidad de las pantallas auxiliares que le permite accionar y supervisar a la vez todos los dispositivos y acceder al sinóptico de los gráficos de tendencia para observar el comportamiento de las variables.

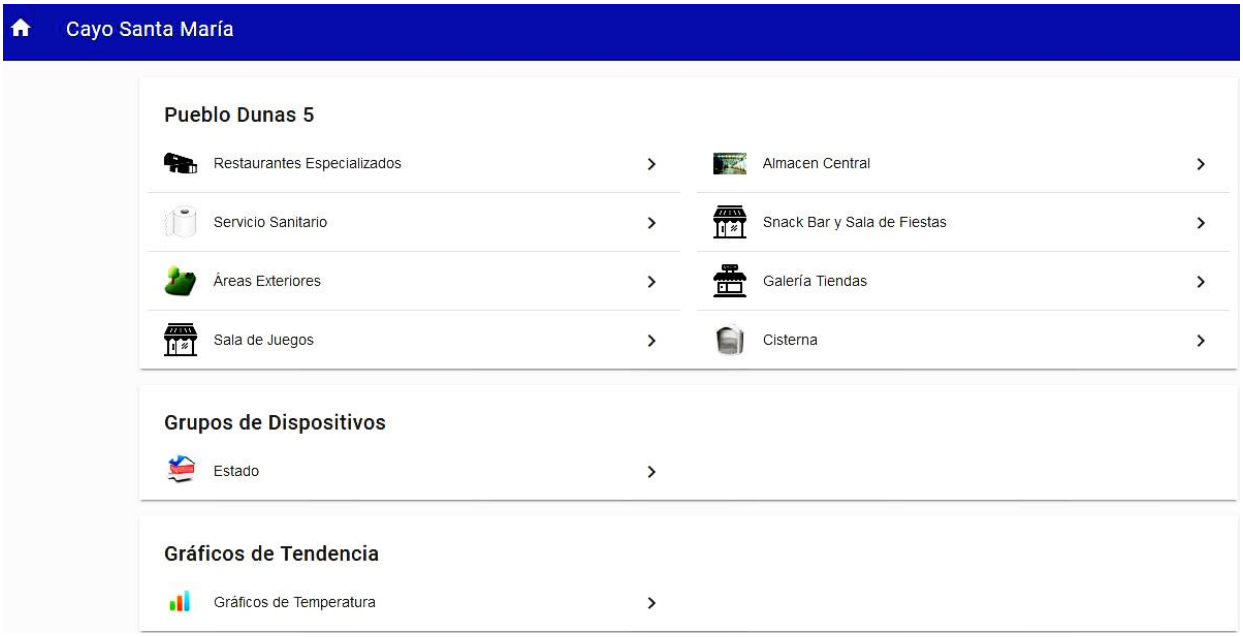


Figura 3-4: Apariencia de la pantalla principal

En la Figura 3-5 se muestra el nivel de la cisterna y el estado de la bomba, además existe la posibilidad de encenderla/apagarla de manera manual a través de un *switch*. La bomba es activada de manera automática cuando el nivel de la cisterna este por debajo de 80 m³ y se apaga cuando sobrepase un límite de histéresis.

Figura 3-5: Cisterna y Bomba

3.2.1 Pantallas auxiliares

Las pantallas auxiliares son cargadas desde la pantalla de inicio. A continuación, se muestra la apariencia de cada una de ellas.

3.2.1.1 Grupos de dispositivos

En esta pantalla se muestran los grupos de dispositivos instalados. Desde aquí el usuario puede tener un mejor control de todos los equipos sin necesidad de acceder a cada uno de manera independiente. Además, existe la posibilidad de apagar todos los dispositivos a la vez, con la utilización de los grupos de *item* como se explica en el subepígrafe 2.3.1, en caso de urgencias o cuando el usuario lo decida. En la Figura 3-6 se muestran los tipos de grupos de dispositivos que el usuario puede accionar.

Figura 3-6: Grupo de dispositivos

3.2.1.2 Estado de las luces

En esta pantalla como se muestra en la Figura 3-7 se puede tener acceso total de las luces de todas las instalaciones del pueblo y accionarlas o supervisarlas sin necesidad de acceder a ellas de manera independiente.

Figura 3-7: Estado de las luces

3.2.1.3 Temperatura de los locales climatizados

Esta pantalla brinda información de la temperatura que hay en los diferentes locales climatizados para así poder tomar decisiones en el estado de los *roof top* de cada uno de ellos. En la Figura 3-8 se muestra la apariencia de esta pantalla.

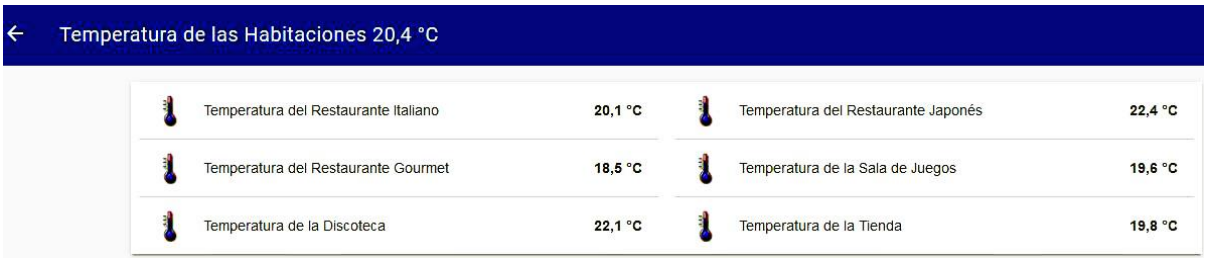


Figura 3-8: Temperatura de los locales climatizados

3.2.1.4 Estado de los roof top y de los extractores

En la Figura 3-9 y Figura 3-10 se muestran las vistas de las pantallas de los *roof top* y extractores respectivamente en donde el usuario puede accionar o supervisar el funcionamiento de estos dispositivos.

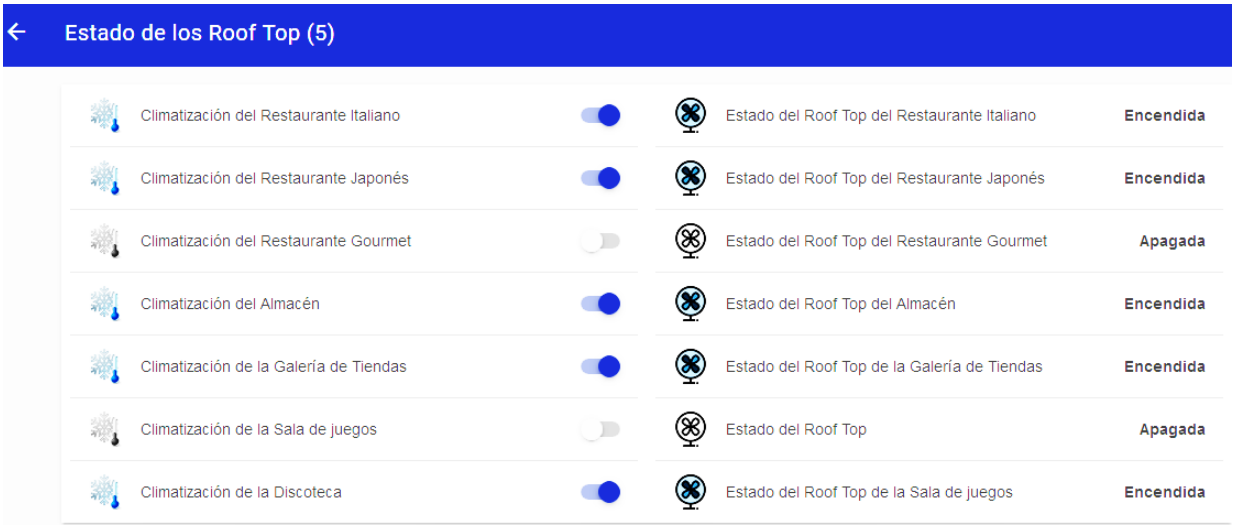


Figura 3-9: Estado de los *roof top*

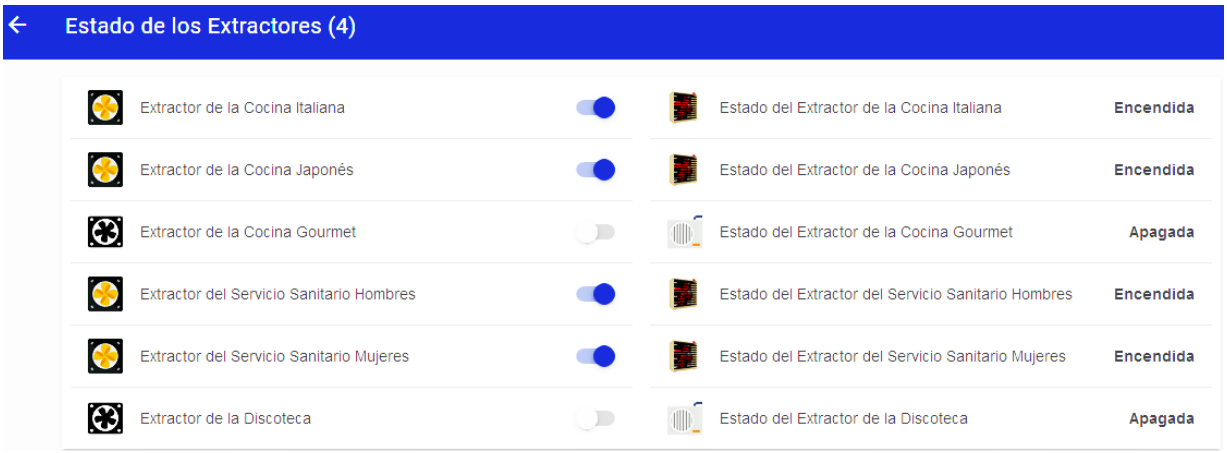


Figura 3-10: Estado de los extractores

3.2.1.5 Gráficos de tendencia

En el sinóptico de los gráficos de tendencia se puede observar el comportamiento de la temperatura en los locales climatizados para tener un registro de las variaciones de estas variables. En la Figura 3-11 se muestran algunos de los gráficos que se pueden obtener.

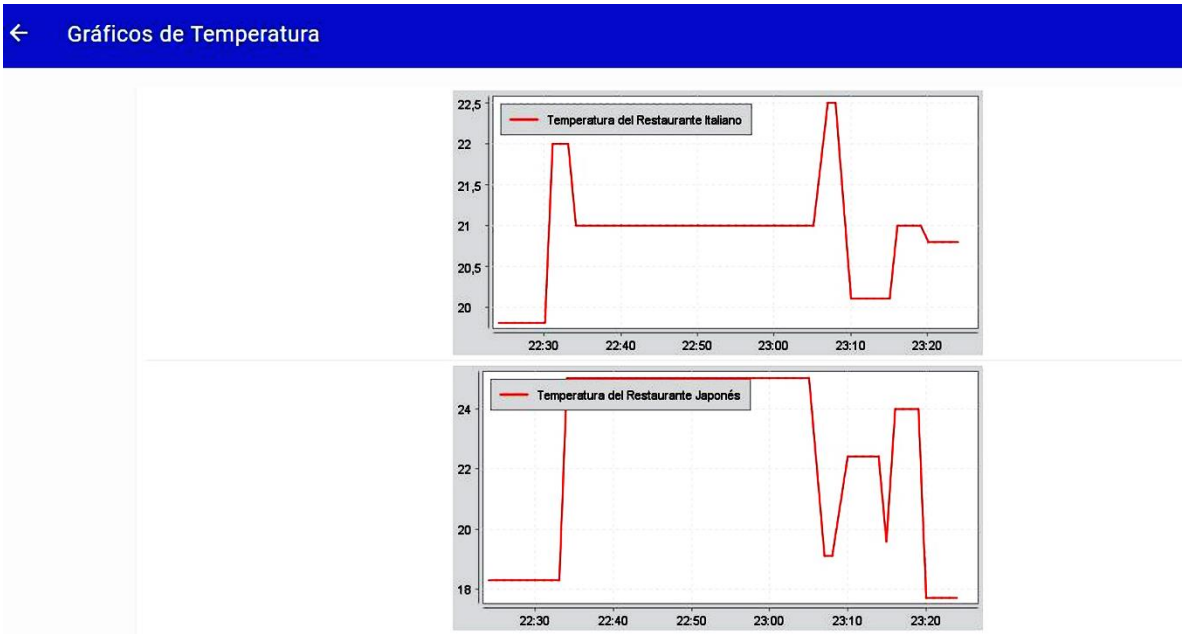


Figura 3-11: Gráficos de temperatura

OpenHAB cuenta con diferentes interfaces gráficas desde las cuales el usuario puede interactuar con la aplicación. En la Figura 3-12 se muestra otro tipo de gráficos de tendencia que se pueden obtener en el panel HABmin2. En el Anexo 8 se muestra la pantalla principal de la interfaz gráfica del usuario HABmin2.

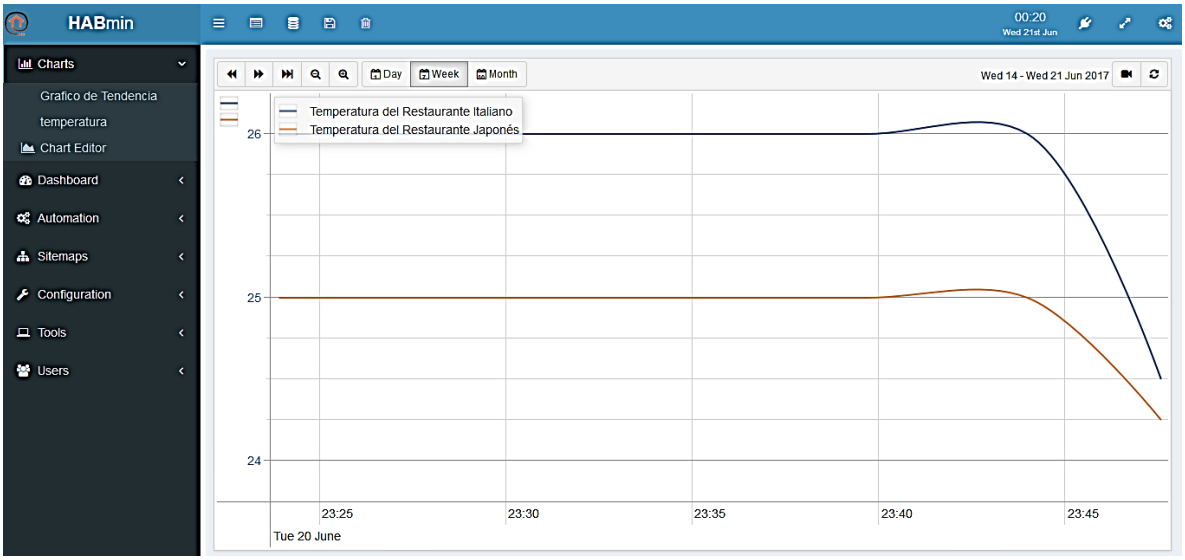


Figura 3-12: Gráfico de temperatura visto desde el panel HABmin2

3.3 Análisis Económico

Se presenta un análisis económico de lo que puede ahorrar al país el uso de alternativas de software y hardware abierto en comparación con software propietario.

Un proyecto de supervisión y control basado en software propietario tiene un costo total de $ 4065,40 CUC en el que se incluyen gastos por concepto de licencia de supervisión, llave de programación y un computador con características necesarias para su ejecución.

Con el uso de openHAB como SCADA libre se evita pagar por concepto de propiedad, licencia de software y actualizaciones. En este caso solo se necesita personal calificado para desarrollar el proyecto. La implementación de este software en sistemas embebidos es una opción económicamente factible con un costo total de $ 69,99 CUC.

En la Tabla 3-2 se muestra una comparación de los costos de la implementación de un SCADA basado en el uso de software propietario como es el caso de MOVICON y un SCADA basado en software libre como openHAB.

Tabla 3-2: Comparación entre SCADA basado en software propietario y software libre

	MOVICON	**OpenHAB**
Licencia de Supervisión	$ 2744	$ 0,00
Llave de programación	$ 78,00	$ 0,00
Computadora ASUS Core i5-7400 3GHz, 8GB DDR4-SDRAM, Disco 1 TB, Monitor 20¨ Led	$ 1243,40	__
Raspberry Pi 3 Modelo B, Pi Box y Fuente de alimentación 5V DC y 2.5A	__	$ 69,99
Costo Total CUC	$ 4065,40	$ 69,99

El uso de SCADA basado en software libre con prestaciones que abarquen las necesidades de las industrias locales y regionales, representa un ahorro económico considerable para nuestro país.

3.4 Análisis Medioambiental

El impacto ambiental es un aspecto de gran importancia en la realización de cualquier proyecto. En este caso se contribuye al mejoramiento del medio ambiente a través del ahorro recursos naturales como el agua y la energía.

Con la implementación de este proyecto es posible administrar correctamente la mayoría de los dispositivos de una instalación hotelera, evitando que estos estén encendidos innecesariamente por largos períodos de tiempo, por lo que se necesita procesar y consumir menos combustible, lo que significa una disminución de emisiones tóxicas al medio ambiente.

Además, este sistema permite dar un uso racional al agua, evitando el derroche de este vital recurso a través de la supervisión y el control de los niveles de las cisternas.

3.5 Conclusiones del capítulo

Mediante un análisis de los resultados obtenidos, se puede llegar a las siguientes conclusiones del capítulo:

- La aplicación obtenida cumple con los requerimientos iniciales del proyecto.
- Los resultados de este trabajo pueden ser utilizados en trabajos similares.
- Las pantallas del supervisor diseñado permiten el correcto control y supervisión del pueblo Dunas 5.
- Esta es una propuesta económicamente factible debido a la reducción de costos por el uso de software y hardware abierto.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Como resultado de esta investigación se propone un sistema SCADA de código abierto que puede ser ejecutado en sistemas embebidos basados en ARM. Se comprueba que, a pesar del bajo costo, el sistema tiene las prestaciones necesarias para la supervisión y control de instalaciones hoteleras.

Finalmente podemos concluir:

1. De los principales SCADAs de código abierto evaluados, openHAB es el que mejores prestaciones presenta para ser utilizado en plataformas embebidas y cumple con todas las funcionalidades necesarias de un sistema supervisor para la automatización de instalaciones hoteleras en Cuba.
2. Con la comparación de las arquitecturas de hardware basadas en hardware abierto se propone la Raspberry Pi 3 como la tarjeta de mejores resultados para ejecutar el openHAB.
3. Se comprueba mediante ejecución de varios proyectos de openHAB en la Raspberry Pi que el sistema es robusto, eficiente y cumple con todas las exigencias para sistemas de supervisión y control de instalaciones hoteleras.
4. Finalmente, se implementa el sistema de supervisión y control para la extrahotelera Dunas 5 del Cayo Santa María, con la utilización de openHAB en Raspberry Pi, como propuesta de automatización económicamente viable por su bajo costo para los hoteles de nuestro país.

Recomendaciones

Como recomendación de la investigación científica se propone:

1. Continuar el estudio e investigación de las potencialidades que puede brindar el uso del sistema openHAB y contribuir al desarrollo del mismo con *bindings* y herramientas que faciliten el uso en el territorio nacional.
2. Divulgar este proyecto para incrementar la motivación e información especializada a los usuarios sobre el uso de este sistema SCADA libre y de código abierto.
3. Instalar el openHAB en los hoteles de Cuba.
4. Proponerlo para generalizar en otras instalaciones.

REFERENCIAS BIBLIOGRÁFICAS

AB Electronics, 2016. IO Pi Plus [WWW Document]. URL http://IO Pi Plus - AB Electronics.html (accessed 10.5.17).

Abaffy, C., Lárez, J., 2007. Desarrollo de SCADA en una Plataforma de Software Libre. CITEG Rev. Arbitr. 84–97.

Asesores, A., 2012. ¿Por qué Código Abierto? [WWW Document]. URL http://abaxasesores.com/codigoabierto (accessed 2.3.17).

Bruce, J., 2015. Getting Started with OpenHAB Home Automation on Raspberry Pi.

Bueno, M., 2014. OpenHAB Domótica libre [WWW Document]. URL http://redFP.org

Correa, E., Trufiño, R., 2015. Aplicaciones en Software Libre como plataforma de enlace para Smart Home.

del Sol, V., 2016. Sistema para la monitorización y asistencia en el hogar usando openHAB y robótica móvil (Trabajo de Diploma). Universidad de Málaga, Málaga, España.

Gamboa, A.C., 2001. Diseño de Comunicaciones SCADA para la interconexión de las Unidades Terminales Maestras con las Unidades Terminales Remotas de Petroproducción para los sectores de Lago Agrio, Shushufindi y Sacha (Trabajo de Diploma). Escuela Politécnica Nacional, Quito, Ecuador.

Hentea, M., 2008. Improving Security for SCADA Control Systems. Interdiscip. J. Inf. Know Ledge Manag. 3.

Huidobro, J.M., 2016. Edificios inteligentes y domótica [WWW Document]. URL http://www.monografias.com/trabajos14/domotica/domotica.html.

Igure, V.M., Laughter, S.A., Williams, R.D., 2006. Security issues in SCADA networks. Comput. Secur. 25, 498–506.

Izaguirre, E., 2008. Sistemas de Control Supervisorio y Adquisición de Datos (SCADA). Santa Clara, Villa Clara, Cuba.

Linares, R., 2014. Evaluación de Sistemas SCADA libres (Trabajo de Diploma). Universidad Central ¨Marta Abreu¨ de Las Villas (UCLV). Departamento de Automática y Sistemas Computacionales, Santa Clara, Villa Clara, Cuba.

López, E., 2009. Interfaz Genérica para Sistemas de Adquisición basada en Software Libre (Trabajo de Diploma). Universidad Central ¨Marta Abreu¨ de Las Villas (UCLV).

Departamento de Automática y Sistemas Computacionales, Santa Clara, Villa Clara, Cuba.

Martin, A., 2015. Driving Total Control [WWW Document]. URL http://www.siemens.co.uk/industry (accessed 3.31.17).

Martínez, D.E., 2014. Sistema de Domótica para Control y Supervisión de una Habitación de manera remota (Trabajo de Diploma). Pontifica Universidad Javeriana, Bogotá D.C., Colombia.

Meza, L.E.C., 2007. SCADA System´s & Telemetry (Trabajo de Diploma). Atlantic International University.

Moya, C., 2009. Propuesta de sistema HMI/SCADA libre y de código abierto (Tesis de Maestría).

Muñoz, J.L., 2006. Diseño e implementación de una interfaz gráfica de usuarios para Sistemas SCADA bajo paradigma de Software Libre. Universidad de los Andes, Venezuela.

OpenHAB, 2015a. Items - openHAB 2 - Empowering the Smart Home [WWW Document]. URL http://docs.openhab.org/configuration/items.html (accessed 4.23.17).

OpenHAB, 2015b. Sitemaps - openHAB 2 - Empowering the Smart Home [WWW Document]. URL http://docs.openhab.org/configuration/sitemaps.html (accessed 4.23.17).

OpenHAB, 2015c. Rules - openHAB 2 - Empowering the Smart Home [WWW Document]. URL http://docs.openhab.org/configuration/rules-dsl.html (accessed 4.23.17).

OpenHAB, 2015d. Persistence - openHAB 2 - Empowering the Smart Home [WWW Document]. URL http://docs.openhab.org/persistence/openhab/openhab1/addons/wiki.html (accessed 4.23.17).

OpenHAB, 2015e. Sitemaps - openHAB 2 - Empowering the Smart Home [WWW Document]. URL http://docs.openhab.org/configuration/sitemaps.html (accessed 4.23.17).

Pérez, E., 2015. Los sistemas SCADA en la automatización industrial. Tecnol. En Marcha 3–14.

Pérez, I., 2016. Software empresarial de código abierto para sistemas GNU/Linux. Linux Adictos.

Pérez, I., 2014. Comparativa y Análisis: Raspberry Pi vs competencia.

Progea, 2015. Software SCADA/HMI Progea Movicon [WWW Document]. Softw. SCADAHMI Progea Movicon. URL http://www.automationwarehouse.com.au/movicon/ http://progea.us/scada-hmi-movicon11/ (accessed 6.2.17).

Ramagosa, J., Gallego, D., Pacheco, R., 2004. Sistemas SCADA (Trabajo de Diploma). Universidad Politécnica de Cataluña, Barcelona, España.

Simó, J.M.M., 2012. SCADAs Libres: Estado del Arte y Evaluación.

Soler, A., 2016. Connected Home (Trabajo de Diploma). Universidad Politécnica de Cataluña, Cataluña, Barcelona, España.

Vega, R., 2016. Instalación Domótica basada en OpenHAB y Raspberry Pi (Trabajo de Diploma). Universidad de Valladolid. Escuela de Ingenierías Industriales, Valladolid, España.

Ventura, V., 2016. Sensor de temperatura I2C LM75 [WWW Document]. URL https://polaridad.es/sensor-temperatura-i2c-lm75-termometro-arduino/ (accessed 5.20.17).

Wonderware, C., 1999. Wonderware FactorySuite InTouch [WWW Document]. URL (accessed 12.15.17).

ANEXOS

Anexo 1 Sintaxis de los *items*

```
Group All
Group Restaurante           (All)
Group Tiendas               (All)
Group Almacen               (All)
Group Sanitario             (All)
Group AExt                  (All)
Group Juegos                (All)
Group BarDisco              (All)
Group Cisterna              (All)

Group Restaurante_Italiano              "Restaurante Italiano"      <italiano>      (Restaurante)
Group Restaurante_Japones               "Restaurante Japonés"       <japones>       (Restaurante)
Group Restaurante_Gourmet               "Restaurante Gourmet"       <gourmet>       (Restaurante)

/* active groups */
Group:Number:AVG            Temperature     "Temperatura de las Habitaciones"   <climate>   (All)
Group:Switch:OR(ON, OFF)    Light           "Estado de las Luces [(%d)]"        <slider>    (All)
Group:Switch:OR(ON, OFF)    Roof_Top        "Estado de los Roof Top [(%d)]"     <fan1>      (All)
Group:Switch:OR(ON, OFF)    Extractor       "Estado de los Extractores [(%d)]"  <extractor> (All)

/*Light */
Switch Light_Restaurante_Italiano   "Luces del Restaurante Italiano"    (Restaurante_Italiano, Light)
Switch Light_Restaurante_Japones    "Luces del Restaurante Japonés"     (Restaurante_Japones, Light)
Switch Light_Restaurante_Gourmet    "Luces del Restaurante Gourmet"     (Restaurante_Gourmet, Light)

/*Temperature */
Number Temperature_Juegos   "Temperatura de la Sala de Juegos [%.1f °C]"    <temperature>   (Juegos, Temperature)    [mysql]


/*Roof_Top */
Switch Roof_Top_Juegos          "Climatización de la Sala de juegos"    <climate>   (Juegos, Roof_Top)
Contact Roof_Top_Juegos_Estado  "Estado del Roof Top [MAP(en.map):%s]"  <fan1>      (Juegos, Roof_Top)


/*Nivel */
Number Nivel_Cisterna_Agua              "Nivel de la Cisterna [%.1f m3]"        <cistern>   (Cisterna, Nivel)
Switch Nivel_Cisterna_Bomba             "Bomba"                                 <pump>      (Cisterna, Nivel)
Contact Nivel_Cisterna_Bomba_Agua       "Estado de la Bomba [MAP(en.map):%s]"   <pump>      (Cisterna, Nivel)
```

Anexo 2 Sintaxis del *sitemap*

```
sitemap cayo label="Cayo Santa María"
{
Frame label="Pueblo Dunas 5"
{
        Group item=Restaurante  label="Restaurantes Especializados" icon="edificio_restaurante"
        Group item=Almacen      label="Almacen Central"             icon="almacen"
        Group item=Sanitario    label="Servicio Sanitario"          icon="toilet"
        Group item=BarDisco     label="Snack Bar y Sala de Fiestas" icon="tienda_318"
        Group item=AExt         label="Áreas Exteriores"            icon="garden"
        Group item=Tiendas      label="Galería Tiendas"             icon="edificio_tienda"
        Group item=Juegos       label="Sala de Juegos"              icon="tienda_318"
        Group item=Cisterna     label="Cisterna"                    icon="cistern"
}

Frame label="Grupos de Dispositivos" {
        Text label="Estado" icon="firstfloor" {
                Switch item=Light           mappings=[OFF="Apagar Todos"]
                Switch item=Roof_Top        mappings=[OFF="Apagar Todos"]
                Switch item=Extractor       mappings=[OFF="Apagar Todos"]
                Group item=Temperature
                Group item=Light
                Group item=Roof_Top
                Group item=Extractor
                }
                }
Frame label="Gráficos"{
        Text label="Gráficos de Temperatura" icon="chart" {
        Chart item=Temperature_Restaurante_Italiano     period=h refresh=100
        Chart item=Temperature_Restaurante_Japones      period=h refresh=100
        Chart item=Temperature_Restaurante_Gourmet      period=h refresh=100
}
}
}
```

Anexo 3 Sintaxis de las reglas

```
rule "Luces de las Areas Exteriores"
    when
        Time cron "0 0 7 1/1 * ?"
    then{
        sendCommand (Light_AExt, ON)
        }
end

rule "Estado del Roof Top del Restaurante Italiano"
    when
        Item Roof_Top_Restaurante_Italiano received command
    then
    if(receivedCommand==ON){
        sendCommand(Roof_Top_Restaurante_Italiano_Estado, OPEN)
    }
else
    if(receivedCommand==OFF){
        sendCommand(Roof_Top_Restaurante_Italiano_Estado, CLOSED)
    }
end

rule "Bomba cisterna"
    when
        Item Nivel_Cisterna_Bomba changed or
    then
    if(Nivel_Cisterna_Agua.state < 80){
    sendCommand(Nivel_Cisterna_Bomba_Agua, OPEN )
    }
else
    if(Nivel_Cisterna_Agua.state > 81){
    sendCommand(Nivel_Cisterna_Bomba_Agua, CLOSED)
    }
end
```

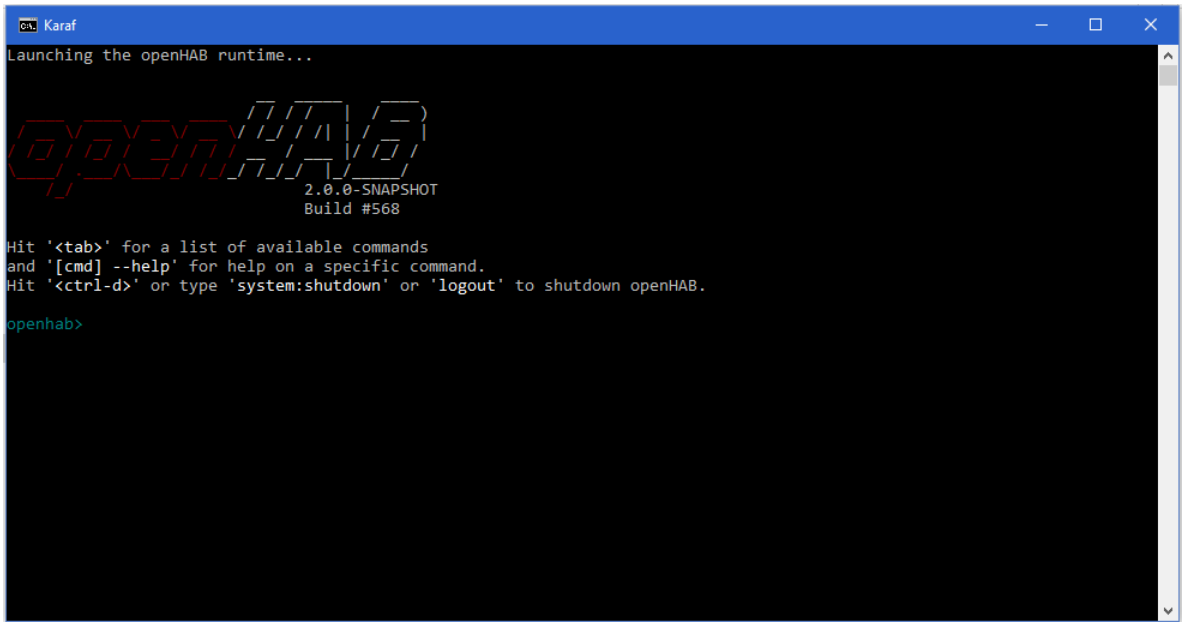
Anexo 4 **Declaración del *binding Exec* para la ejecución de un *script* que se encarga de leer los datos recibidos por el sensor.**

```
Number Temperature "Temperatura [%.1f C]" <temperature> {exec="<[/usr/bin/python
/opt/openHAB/configurations/scripts/temp.py:1000:REGEX((.*?))]"}
```

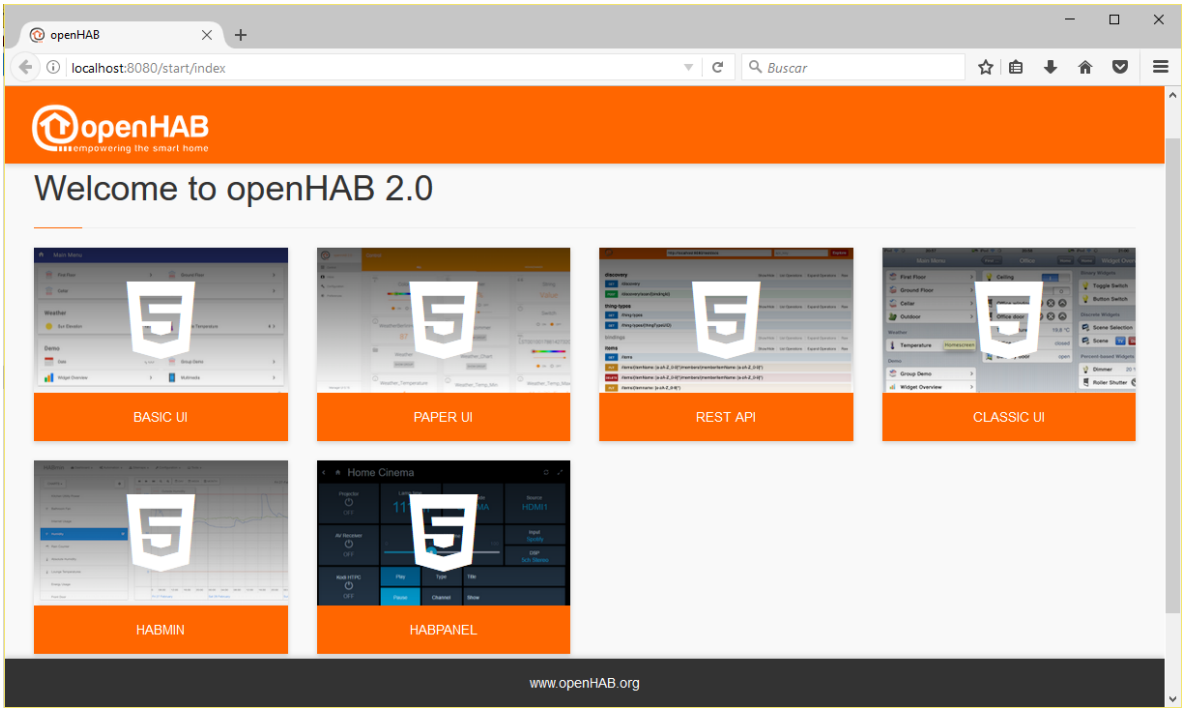
Anexo 5 Configuración de la base de datos en el archivo mySQL.cfg

```
url=jdbc:mysql://localhost:3306/cayo

# the database user
user=root

# the database password
password=xxxxxx

# the reconnection counter
reconnectCnt=1
```

Anexo 6 OpenHAB



Iniciando el *runtime*



Pantalla principal que ofrece los diferentes paneles web UIs

Anexo 7 Configuración de Modbus en el archivo modbus.cfg

```
poll=200
#    (slave name)      (host or IP)
#    |                 |             (port)
#    |                 |             |   (interTransactionDelayMillis, in milliseconds)
#    |                 |             |   |  (reconnectAfterMillis, in milliseconds)
#    |                 |             |   |  | (interConnectDelayMillis, in milliseconds)
#    |                 |             |   |  | | (connectMaxTries)
#    |                 |             |   |  | | | (connectTimeout)
#    |                 |             |   |  | | | |
tcp.slave1.connection=192.168.1.50:502:60:0:0:3:100
tcp.slave1.type=coil
tcp.slave1.id=41
tcp.slave1.start=0
tcp.slave1.length=32
```

Anexo 8 Interfaz gráfica del usuario HABmin2