Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Módulos de administración, comunicación y adquisición para un sistema de supervisión y control de enfriadoras de agua en dispositivo empotrado.

Autora: Edelis Molina Rios

Tutor: M. Sc. Erick López González.

Santa Clara

2013

"Año 55 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Módulos de administración, comunicación y adquisición para un sistema de supervisión y control de enfriadoras de agua en dispositivo empotrado.

Autora: Edelis Molina Rios.

Email: edelis@uclv.edu.cu

Tutor: M. Sc. Erick López González.

Instructor Dpto. Automática y Sistemas Computacionales

Facultad de Ingeniería Eléctrica, Universidad Central de Las Villas.

Email: ericklglez@uclv.cu

Santa Clara

2013

"Año 55 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Autor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

Toda nuestra ciencia, contrastada con la realidad, es primitiva y pueril; y, sin embargo, es lo más valioso que tenemos.

Albert Einstein

DEDICATORIA

A mis padres, Pedro y Edelis, por guiarme hacia adelante, dándome ejemplos dignos de superación y entrega, porque el orgullo que sienten por mí fue lo que me hizo ir hasta el final para alcanzar mi meta. Va por ustedes, por lo que valen, porque admiro su dedicación y por lo que han hecho de mí.

AGRADECIMIENTOS

- ✓ *A mi mamá y a mi papá, porque son la inspiración por la que he llegado hasta aquí.*

- ✓ *A mi familia, que de varias formas ha contribuido en la culminación de esta carrera.*

- ✓ *A Luis Felipe, por su apoyo y comprensión durante estos 5 años.*

- ✓ *A todos los profesores del Departamento de Automática que han influido en mi formación como profesional.*

RESUMEN

Los sistemas de clima centralizado se componen de enfriadores de agua o *chillers*. En Cuba estos sistemas se emplean principalmente en hoteles para asegurar las condiciones de confort de los huéspedes. Estudios realizados revelan que los *chillers* consumen más de la mitad de la electricidad total de un hotel, por lo cual resulta ineludible el desarrollo de técnicas para monitorear y controlar su funcionamiento.

En el presente Trabajo de Diploma se implementan los módulos de administración, comunicación y adquisición, integrantes de una aplicación empotrada sobre la computadora en una tarjeta MBTI2440, que junto a otros módulos permite supervisar y controlar las enfriadoras.

Para la implementación de los módulos, se realizó un análisis de las herramientas y tecnologías de software libre a utilizar, seleccionándose como sistema gestor de los datos recolectados a SQLite y la biblioteca Qt, basada en C++, para aprovechar las ventajas que ofrece en el desarrollo de aplicaciones multiplataforma.

Como resultado, se obtuvo una solución totalmente nacional que integra hardware cubano y contribuye al crecimiento económico del país.

TABLA DE CONTENIDOS

PENSAMIENTO.....i
DEDICATORIA..... ii
AGRADECIMIENTOS..... iii
RESUMEN..... iv
INTRODUCCIÓN.....1
Organización del informe.....5
CAPÍTULO 1. INTRODUCCIÓN A LOS SISTEMAS EMPOTRADOS.....6
1.1 Sistemas empotrados.....6
1.1.1 Definición y características.....7
1.1.2 Computadora en una tarjeta: PC embebida.....8
1.2 Sistemas operativos empotrados.....9
1.2.1 Linux como sistema operativo empotrado.....10
1.2.2 Windows CE: El Windows más pequeño que existe.....13
1.3 Bases de datos empotradas.....14
1.3.1 SQLite.....16
1.4 Consideraciones finales del capítulo.....19
CAPÍTULO 2. DISEÑO E IMPLEMENTACIÓN DE LOS MÓDULOS.....21
2.1 Requisitos funcionales.....21
2.2 Descripción de la tarjeta MBTI2440.....23
2.2.1 Sistema operativo para la tarjeta de hardware.....24
2.2.1.1 ¿Por qué Linux?.....25
2.3 Lenguajes y herramientas.....25

2.3.1 Modelado de la aplicación. 25
2.3.2 Herramienta de modelado. 26
2.3.3 Lenguaje de programación. 26
2.3.3.1 Biblioteca Qt 26
2.3.4 El IDE Qt Creator. 28
2.4 *Toolchain* de compilación cruzada 28
2.5 Descripción de los módulos 31
2.5.1 Módulo Manejador de Datos 31
2.5.2 Módulo Administrador (Manager) 37
2.5.3 Módulo de comunicación (*Driver*) 38
2.5.3.1 Biblioteca libmodbus 39
2.5.3.2 Funcionalidades típicas 40
2.6 Consideraciones finales del capítulo. 41
CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS 42
3.1 Desarrollo de las pruebas 42
3.1.1 Prueba 1 42
3.1.2 Prueba 2 44
3.2 Análisis económico 48
3.3 Conclusiones del capítulo 49
CONCLUSIONES Y RECOMENDACIONES 50
Conclusiones 50
Recomendaciones 51
REFERENCIAS BIBLIOGRÁFICAS 52
ANEXOS 54
Anexo I. Guía para compilar la biblioteca Qt para la MBTI2440. 54
Anexo II. Tabla de variables a recolectar de los *chillers* 56

INTRODUCCIÓN

El desarrollo social en el pasado siglo XX propició un salto cualitativo y revolucionario nunca antes imaginado, que ha modificado los procesos de la comunicación e intercambio y ha dado paso a las Tecnologías de la Información y la Comunicación (TIC), a partir de la convergencia tecnológica de la microelectrónica, la informática y las telecomunicaciones (Vidal y Manrique, 2012). Las TIC se desarrollan bajo un contexto caracterizado por un agotamiento de las fuentes tradicionales de energía y acelerado deterioro del medio ambiente, lo que ha traído como consecuencia guerras por apropiación de los recursos. La utilización responsable y eficaz de la energía constituye un elemento motriz del desarrollo económico e influyente en la sostenibilidad ambiental. En la actual situación mundial, la humanidad afronta el reto de reducir el consumo energético ante la inminente aparición de una crisis energética que desembocaría irremediablemente en una crisis económica, por lo cual la energía es cada vez más parte integrante de la agenda de desarrollo de un país.

El ahorro de energía en las edificaciones es una de las cuestiones más tratadas en la actualidad debido a los elevados gastos energéticos y costos asociados necesarios para la operación de los edificios. A nivel mundial se reporta que el consumo de energía en las edificaciones representa el 40%, siendo más de la mitad atribuible a los sistemas de climatización (Pout y col., 2008). Particularmente, en los hoteles turísticos en climas tropicales, resulta imprescindible garantizar las condiciones de confort de los usuarios a partir del uso de estos sistemas.

Los sistemas de climatización utilizados en el sector turístico son generalmente del tipo centralizado, debido a que es más eficiente la producción de frío centralizada que producirlo de manera aislada por locales independientes, ya que el costo de la energía para acondicionar el aire usando otros sistemas de refrigeración sería bastante alto.

Un sistema de clima centralizado se compone, esencialmente, por una estación de máquinas enfriadoras de agua (en inglés, *water chillers*[1]). Un *chiller* es un sistema de aire acondicionado refrigerado por agua que enfría el agua que será bombeada a toda la instalación a través de un circuito hidráulico cerrado. Su principio de funcionamiento se basa en el bombeo de agua helada por la red hidráulica hasta llegar a las unidades manejadoras de aire, donde se enfría un flujo de aire que crea un ventilador/extractor de la manejadora haciéndolo pasar por una red de tuberías por las que circula el agua fría y donde físicamente ocurre un intercambio de energía, obteniendo como resultado aire frío y seco que se hace circular por las habitaciones y agua que retorna nuevamente a los *chillers* para ser enfriada.

La puesta en marcha de los enfriadores de agua, implica un alto costo de inversión y operación, puesto que son responsables de alrededor del 60% del consumo total de electricidad de los hoteles (Montesino, 2005), por lo cual se impone concebir desde la etapa de diseño, una instalación que satisfaga la demanda térmica a suplir, pero que a la vez permita la secuenciación, rotación y cambio de *setpoints* horarios para cada una de las enfriadoras conectadas al sistema, para que el costo del producto final sea mínimo, definiendo este producto final como: el agua helada para la climatización de los locales.

Las enfriadoras de agua pueden ser supervisadas y controladas en todas sus funciones por medio de un software. En el mercado internacional, las empresas ofertan los enfriadores con el software propietario correspondiente y con características particulares, debido a que generalmente está diseñado para cumplir con las especificaciones comunes de la mayoría de las implementaciones que se realizan, por lo que no se puede manipular ninguna sección de código del programa para corregir errores o personalizar el sistema para casos específicos. El sistema que se compra no se puede mejorar para situaciones especiales de una implementación.

[1] Se emplea el término del idioma inglés ***chiller*** para referirse a la unidad enfriadora de agua.

Las empresas cubanas que integran a sus servicios la automatización de los *chillers* de las instalaciones hoteleras, desarrollan productos de software y estrategias de control que están teniendo una gradual evolución en el diseño con respecto a formas de operación y rendimiento energético.

Un detalle importante sobre los productos de software que se han desarrollado es que funcionan en una computadora personal (PC, del inglés *Personal Computer*) con sistema operativo Windows –generalmente en su versión XP- que debe estar encendida sin interrupciones para su acceso en cualquier momento para la supervisión, la adquisición de datos de forma fiable y el control de los *chillers*. La debilidad de este sistema operativo en cuanto a seguridad y a la proliferación cada vez mayor de virus informáticos, unido al proceso de Informatización de la Sociedad Cubana (INFOSOC), que incluye la migración a plataformas de código abierto, ha motivado a desarrollar las aplicaciones informáticas a sistemas más robustos como las distribuciones de GNU/Linux. Además, la experiencia adquirida durante la explotación de los sistemas de clima centralizado demuestra que con mucha frecuencia los operadores emplean la computadora para otras tareas o la apagan en horarios nocturnos, desconectan el sistema e instalan programas que afectan el funcionamiento de las unidades enfriadoras (Torres, 2009).

Se vislumbra entonces la necesidad de diseñar un nuevo sistema de supervisión y control para las máquinas enfriadoras, tomando en cuenta las características del software que ya existe, pero mejorando los puntos en los que estos sistemas fallan. Se pretende que el nuevo diseño emplee como plataforma de hardware un dispositivo empotrado, puesto que el software que se ejecuta sobre el mismo -a diferencia de una PC- está específicamente dedicado a la aplicación y su hardware se caracteriza por consumir bajos recursos y energía, se disminuye en costo y en tamaño y puede operar en ambientes más robustos que el de una computadora de escritorio, por lo que se puede instalar en la sala de máquinas cerca de los *chillers.*

Como la creación de productos de alta calidad, que integren hardware cubano y software para sistemas empotrados constituye un aporte trascendental en la soberanía tecnológica, seguridad nacional y sustitución de importaciones, la Empresa de Automatización Integral (CEDAI) de Villa Clara presenta un proyecto para insertarse en el desarrollo de

aplicaciones competitivas en mercados nacionales, que incluye el desarrollo de un software con herramientas de código abierto y de bajo costo de implementación para los enfriadores de agua de los hoteles, utilizando la computadora en una tarjeta MBTI2440 en sustitución de la PC que tradicionalmente se emplea. Este trabajo abarca el diseño de módulos o paquetes para dicho software, tales como, los módulos para la comunicación del empotrado con el *Gateway* de los *chillers*, adquisición y administración del sistema. Así, se lograría obtener una solución totalmente nacional, con valor agregado, que aporte al crecimiento económico del país, debido a que el desarrollo de aplicaciones nacionales abarataría los costos de adquisición, mejoraría la calidad y estarían adaptadas a las características propias del país.

Resumiendo el contenido anterior, se plantea como problema científico: "No existen aplicaciones sobre sistemas empotrados para conocer el estado de las principales variables de las enfriadoras de agua para supervisar y controlar su funcionamiento"

Para dar solución al problema anterior, se planteó como objetivo general:

- Implementar los módulos de administración, comunicación y adquisición de datos para el sistema de supervisión y control de enfriadores de agua en dispositivo empotrado para instalaciones turísticas de la cayería norte de Villa Clara.

Para lograr dicho objetivo se proponen como objetivos específicos:

- Analizar el contenido bibliográfico existente sobre sistemas empotrados, sistemas operativos y bases de datos para estos dispositivos.
- Seleccionar las herramientas de software libre y lenguajes de desarrollo a emplear.
- Implementar los módulos requeridos por la aplicación.
- Validar los resultados obtenidos poniendo en funcionamiento los módulos creados.

Con este proyecto se pretenden desarrollar varios módulos integrantes de una aplicación para un sistema empotrado que sustituya a la PC que se emplea en las labores de adquisición de datos, supervisión y control, para mejorar el funcionamiento de las enfriadoras de agua en instalaciones hoteleras cubanas con clima centralizado y con ello contribuir al ahorro de energía eléctrica en dichas instalaciones. Se ofrece una solución

económicamente factible y sin ataduras tecnológicas a empresas o compañías, debido a que el dispositivo de hardware que se emplea es de menor complejidad y valor monetario y la aplicación se implementa con herramientas de código abierto.

Organización del informe

El trabajo de diploma, posterior a esta introducción, incluye tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos. Los contenidos que se abordan en cada capítulo se estructuran de la siguiente forma:

En el primer capítulo se definen algunos términos esenciales para comprender el informe. Se presentan las características principales de los sistemas empotrados para justificar su empleo en esta investigación y la arquitectura de computadora más empleada en estos sistemas que se conoce como PC embebida. Además, se exponen los dos sistemas operativos empotrados que soporta la tarjeta de hardware seleccionada y la base de datos embebida más extendida en el mundo y que se utilizará para almacenar los datos recolectados.

En el segundo capítulo se describe el hardware de la tarjeta sobre la cual se ejecutará la aplicación y se seleccionan los lenguajes y herramientas de software para implementar los módulos integrantes del sistema de supervisión y control de los *chillers.*

En el tercer y último capítulo se validan los módulos creados mediante la realización de pruebas y se efectúa un análisis económico de la aplicación.

CAPÍTULO 1. INTRODUCCIÓN A LOS SISTEMAS EMPOTRADOS

El trabajo expuesto a continuación contribuye al diseño de software para un dispositivo empotrado fabricado en Cuba, específicamente una aplicación para la supervisión y el control de las enfriadoras de agua. Para facilitar la comprensión de la labor realizada es necesario conocer algunos conceptos relacionados con el tema que serán abordados en el presente capítulo, comenzando con los sistemas empotrados y sus características más importantes y los sistemas operativos para estos dispositivos. Se presentará la definición y características de la base de datos embebida más extendida actualmente.

1.1 Sistemas empotrados

La creciente complejidad de las aplicaciones de los sistemas electrónicos requiere el desarrollo de sistemas de propósito específico determinado por la aplicación, lo cual da origen al campo de los sistemas empotrados (en inglés: *embedded systems*).

En la actualidad, los sistemas empotrados han logrado un gran auge gracias a sus amplios campos de aplicación y a su vez, por los bajos costos comparados con los sistemas de cómputo tradicionales. Su uso es cada vez más extendido en los productos de consumo masivo, tales como dispositivos electrónicos (teléfonos móviles, reproductores mp3, cámaras digitales, video juegos portátiles, calculadoras), electrodomésticos de todo tipo, aplicaciones automotrices, equipos médicos y en general, en productos o aplicaciones donde se requiere de dispositivos inteligentes. Esto se debe a que los nuevos dispositivos no requieren instalar aplicaciones o software como las de una computadora de escritorio, sino aplicaciones de uso dedicado; los sistemas empotrados tienen sus recursos limitados, por lo que no soportarían aplicaciones grandes en tamaño o que requieran un gran número de recursos.

1.1.1 Definición y características

Un sistema empotrado (también llamado sistema embebido, incrustado o integrado) es un sistema basado en un microprocesador cuyo hardware y software están específicamente diseñados y optimizados para realizar una tarea específica o un rango de funciones dedicadas (Raghavan y col., 2006).

El término "empotrado" hace referencia al hecho de que la electrónica o el sistema electrónico de control es una parte integral del sistema en que se encuentra (Pérez, 2009). Los sistemas empotrados poseen ciertas características que los distinguen de otros sistemas de cómputo, según Vahid y Givargis (2002) las más importantes son:

- Funcionamiento específico. Un sistema empotrado usualmente ejecuta un programa específico de manera repetitiva. En contraste, un sistema de escritorio ejecuta una amplia variedad de programas.
- Fuertes limitaciones. Generalmente estos sistemas se utilizan para reducir el gasto de implementación para las tareas específicas que se les da, por lo que suelen emplear procesadores muy básicos, relativamente lentos y memorias pequeñas para minimizar los costos y poseen un tamaño reducido pues suelen estar implementados en una sola placa madre.
- Robustez. Al sistema empotrado se le impondrá la necesidad de la máxima robustez ya que las condiciones de explotación no tienen por qué ser las adecuadas.

Debido a la inmensa diversidad de aplicaciones disponibles de dispositivos empotrados a continuación se muestran los tres componentes principales que tienen en común.

1. Hardware.
2. Software primario o aplicación principal. Este software o aplicación lleva a cabo una tarea en particular, o en algunas ocasiones una serie de tareas.
3. Un sistema operativo que permite supervisar la(s) aplicación(es), además de proveer los mecanismos para la ejecución de procesos. En muchos sistemas empotrados es requerido que el sistema operativo posea características de tiempo real.

Además de esto, normalmente un sistema empotrado cuenta con una serie de entradas y salidas necesarias para comunicarse con el mundo exterior (Sancho, 2012).

El software que se ejecuta en un sistema empotrado es diseñado bajo algunas restricciones importantes tales como:

- Cantidades pequeñas de memoria, generalmente en el orden de los kb, lo cual da lugar a una limitación en el almacenamiento de los datos y en el tamaño de las aplicaciones de software.
- Capacidades limitadas de procesamiento, usualmente los procesadores poseen velocidades que no superan los MHz.

En la figura 1.1 se muestra la estructura de un sistema empotrado típico propuesta por Hallinan (2006). Este diagrama muestra la arquitectura de hardware a alto nivel de un punto de acceso inalámbrico. Básicamente el sistema se centra en un procesador ARM (*Advanced RISC Machine*) de 32 bits. La memoria *flash* se utiliza para almacenamiento de datos y programas de forma persistente. La memoria principal típicamente cuenta con algunos megabytes o cientos de megabytes, en los cuales se almacenan valores temporales para la ejecución de programas. En este ejemplo también se observa una interfaz Ethernet y una USB, un puerto serie y un chip que contiene la implementación del estándar IEEE 802.11.

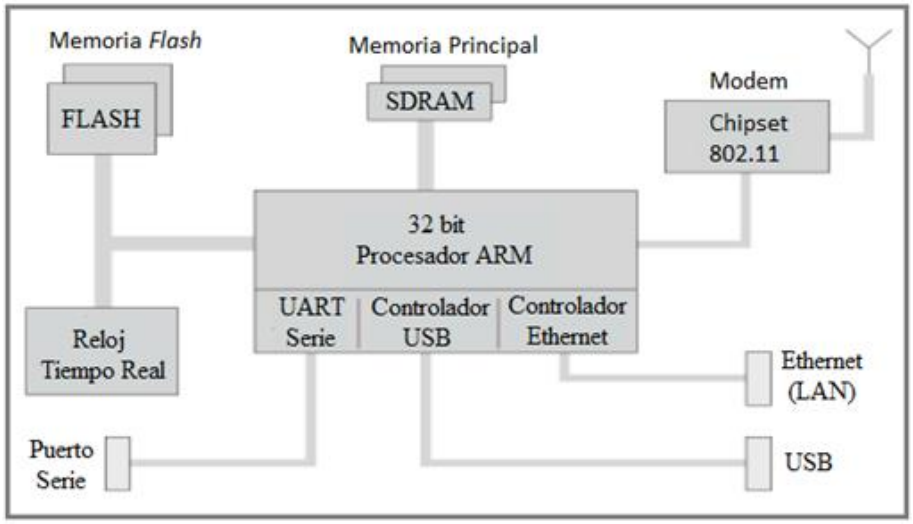


Figura 1.1. Ejemplo de un Sistema Empotrado

1.1.2 Computadora en una tarjeta: PC embebida

Una PC embebida posee una arquitectura semejante a la de una PC de escritorio, con la salvedad de que el diseño se centra en integrar en una sola placa computadora (en inglés: *Single Board Computer o SBC*) todas las características de una computadora personal.

Contiene un solo microprocesador con la RAM, entradas/salidas y todas las demás características de un computador funcional en una sola tarjeta que suele ser de tamaño reducido y bajo coste y que tiene todo lo que necesita en la placa base.

Esta arquitectura no se usa tanto en las computadoras tradicionales -aunque las tendencias indican que esto puede cambiar- sino en entornos industriales o en sistemas empotrados. Debido a los grandes niveles de integración y reducción de componentes y conectores, las computadoras en una tarjeta suelen ser más pequeñas, livianas y con un mejor manejo de la potencia eléctrica que los computadores de múltiples tarjetas.

1.2 Sistemas operativos empotrados

El uso de los sistemas empotrados en los últimos años ha aumentado considerablemente, algunos se han ido integrando para no solo realizar una tarea específica, sino varias tareas; para estos casos, un sistema empotrado necesita el uso de un sistema operativo (SO) que brinde el soporte necesario para las diferentes tareas que se ejecuten.

Un sistema operativo empotrado usualmente es diseñado para una aplicación específica, por lo tanto es más estático que un sistema operativo de propósito general; es un software pequeño en cuanto a consumo de recursos para adaptarse al hardware que se tiene, pero brindan control sobre el sistema de forma eficiente para los programas que se requieran. En ocasiones puede ser una versión reducida de algún sistema operativo que se utiliza en una computadora tradicional (Puente y col. 2012).

Una peculiaridad de los sistemas operativos para empotrados es que son un todo con las aplicaciones que ejecutan, lo que quiere decir que en muchos casos no es posible instalar en ellos ningún tipo de software adicional.

Los sistemas operativos para que un dispositivo empotrado pueda funcionar y ejecutar programas suelen ser específicos, modulares (inclusión sólo de los elementos necesarios del sistema operativo para el sistema empotrado concreto) y consumen bajos recursos de memoria. Actualmente existen varios sistemas operativos que se pueden usar en dispositivos empotrados y de reconocimiento a nivel mundial como QNX, Windows CE, eCos, Palm OS, Symbian y Linux Embebido, este último es claramente el líder en los sistemas empotrados. Cada uno de estos SO puede soportar una o varias de las tecnologías

empotradas que existen en el mercado, destacándose las basadas en procesadores INTEL y ARM (González y col., 2011). A continuación se describirán dos de los sistemas operativos empotrados más comunes.

1.2.1 Linux como sistema operativo empotrado

Linux está presente en muchas partes, quizás aún no haya ganado la batalla en los ordenadores personales, pero es el sistema operativo de código abierto más extendido en los sistemas empotrados. Su desarrollo es uno de los ejemplos más prominentes de software libre; todo su código fuente puede ser utilizado, modificado y redistribuido libremente bajo los términos de la Licencia Pública General de GNU (en inglés: GNU GPL, *GNU General Public License*).

Este SO distingue las versiones estables de su núcleo o *kernel* de las versiones que se encuentran en desarrollo con un simple esquema de numeración. Cada versión corresponde a tres números separados por puntos. Los dos primeros números sirven para indicar la versión, el tercer número identifica el lanzamiento tal como se muestra en la figura 1.2.

Figura 1.2. Numeración de las versiones del kernel de Linux

El *kernel* de Linux es el componente principal del SO, es un puente entre las aplicaciones y el procesamiento de datos a nivel de hardware. Es el encargado de que el software y el hardware puedan trabajar juntos en el sistema y cuenta con las siguientes características principales (Yaghmour, 2003):

- Fiabilidad en su código. Se expresa como consecuencia directa de la filosofía que lleva implícita, la aportación altruista de miles de programadores de todo el mundo observando el código, mejorándolo, cambiándolo y probándolo en miles de configuraciones posibles del sistema.

- Portátil y con gran soporte de hardware. Muchas arquitecturas son manejadas por el sistema operativo.
- Código disponible. Cada lanzamiento del *kernel* de Linux viene con su código fuente para ser modificado, corregido o estudiado por otros. Además, las herramientas para compilación y configuración pueden obtenerse libremente.
- Soporte de hardware. Linux tiene soporte para muchas plataformas y dispositivos existentes, y aunque muchos vendedores todavía no ofrecen soporte para algunos dispositivos, la comunidad que se encarga de mantener a Linux desarrolla muchos *drivers* con el fin de dar soporte a tal hardware.
- Seguridad. El hecho de que Linux sea mantenido por muchas personas, hace que este sea corregido a todo momento siendo fiable y seguro.
- Modularidad. A pesar de que Linux es un *kernel* monolítico, cuenta con la facilidad de poder cargar módulos en tiempo de ejecución, permitiendo su extensión sin tener que ser recompilado.

Linux Embebido se refiere al uso del *kernel* Linux combinado con un conjunto de algunas otras utilidades de software libre, adaptadas dentro del limitado espacio de hardware de los sistemas empotrados. Si bien Linux ha sido desarrollado para funcionar en servidores y computadoras de escritorio, en los últimos años se ha ido diversificando su campo de aplicación. A pesar de que Linux es un SO que originalmente se diseñó para equipos de sobremesa, su alto nivel de modularidad hace que sea fácilmente configurable para trabajar sobre una gran variedad de hardware. Su sistema de configuración permite elegir solo aquellos elementos que sean necesarios para el sistema particular, por ejemplo, para un sistema en el que no se necesiten funciones de red basta con deshabilitar estos componentes en la configuración del núcleo y mantener el resto. Así fue como en 1996 surgió un proyecto de investigación denominado RTLinux para implementar Linux en sistemas de tiempo real, utilizando un *kernel* pequeño y compacto. Un año más tarde, los esfuerzos para implementar el *kernel* de Linux en arquitecturas de menor rango que la x86 y similares, y que no cuentan con unidades de administración de memoria (MMU[2], del inglés *Memory*

[2] Dispositivo de hardware formado por un grupo de circuitos integrados, responsable del manejo de los accesos a la memoria por parte del microprocesador.

Management Unit), dieron como resultado el sistema operativo µClinux (pronunciado "you-see-linux" y significa Linux para Microcontroladores), un derivado del *kernel* de Linux 2.0. Es a partir de esta fecha que se comienza a utilizar ampliamente Linux en sistemas empotrados.

Linux Embebido tiene algunas ventajas en relación a otros sistemas operativos empotrados, como pueden ser: el código abierto, pequeño (una instalación típica de un Linux empotrado ocupa en promedio 2 MB y puede ofrecer las mismas aplicaciones o equivalentes con un procesador la mitad de potente que para Windows CE 7.0, que aunque puede ocupar menos de 1MB sin funcionalidades apenas, en una instalación media con interfaz gráfica no bajará de unos 20MB.), puede no tener costos por derechos, maduro y estable (más de 20 años de uso en muchos dispositivos) y con respaldo.

Al contrario de lo que pueda parecer, Linux Embebido no es una versión reducida de Linux. El calificativo «embebido» realmente hace referencia a la funcionalidad de la aplicación, no a la funcionalidad de Linux, es decir, este término se refiere a las variantes del sistema operativo Linux adaptadas a los dispositivos empotrados.

Debido a la gran heterogeneidad de los sistemas empotrados, no existe una versión estándar de Linux, sino que hay que hablar de muchas "distribuciones" de Linux que cubren una o varias arquitecturas procesadoras. Todas ellas tienen en común los mismos componentes básicos, incluyendo el *kernel* de Linux, *drivers*, comandos, entornos de ventanas, utilidades básicas y librerías. Las diferencias entre las distribuciones se centran normalmente en cuál de los cientos de utilidades existentes se incluyen, en los módulos añadidos (tanto en código fuente como propietario), en las modificaciones del *kernel* y en la gestión de los procesos de instalación, configuración, mantenimiento y actualización.

El sistema operativo Linux encaja a la perfección en el mercado siempre cambiante de los sistemas empotrados. La avanzada tecnología de su *kernel*, la disponibilidad del código fuente y su amplio soporte de los micros y tecnologías modernas y seguimiento en todo el mundo lo convierten en la elección más habitual y de futuro como sistema operativo interno para estos dispositivos.

1.2.2 Windows CE: El Windows más pequeño que existe.

Windows CE (conocido oficialmente como *Windows Embedded Compact* y anteriormente como *Windows Embedded CE*, también abreviado como WinCE) fue creado por Microsoft, empresa que redujo Windows a su mínima expresión y lo lanzó a la conquista del mercado electrónico de los sistemas empotrados.

WinCE es muy diferente del sistema operativo que se tiene en los ordenadores. Windows está orientado a los ordenadores personales de arquitectura x86 y a una interfaz de usuario vistosa, valiéndose para ello de grandes cantidades de recursos (disco duro, procesador, memoria RAM). Sin embargo, los sistemas empotrados pueden llevar procesadores diferentes a los x86, casi nunca disponen de un dispositivo de almacenamiento permanente (mucho menos de un disco duro de varios gigabytes) y cuentan con escasa memoria RAM.

Para adaptar la filosofía de Windows a los sistemas empotrados se hizo necesario crear un sistema operativo independiente de la arquitectura del procesador de destino. Pero tratar de hacer un sistema operativo recortado de otro que estaba destinado para PCs -Windows 95- era una auténtica locura, así que WinCE fue creado desde cero y su primera versión apareció en los PCs de bolsillo o *handhelds* de Casio y Compaq en noviembre de 1996.

Entre las principales características que ofrece WinCE se encuentran (Palacios, 2003):

- Sistema operativo de 32 bits, multitarea y multihilo
- Tecnologías sin cable (*bluetooth*, IPv6) que permiten al dispositivo conectarse a infraestructuras ya existentes.
- Pequeño tamaño (el *kernel* mínimo ocupa unos 500 kB).
- Amplia gama de arquitecturas (la versión actual de WinCE funciona en procesadores Intel x86 y compatibles, además de los tipos MIPS y ARM).
- Es en tiempo real

Además WinCE provee algunas aplicaciones embebidas clásicas como Pocket Word, Pocket Excel (versiones minimizadas del paquete Office).WinCE es compacto, ofrece alto rendimiento en configuraciones limitadas de memoria, posibilita a los fabricantes optar por un microprocesador en particular y posee administración de potencia incorporada.

A modo de resumen en la Tabla1.1 se muestran las características de interés de los sistemas operativos abordados anteriormente.

Tabla 1.1. Información general de los sistemas operativos embebidos Linux y WinCE.

	Linux Embebido	**Windows CE**
Desarrollador	Consorcio de Linux Embebido y la comunidad de colaboradores alrededor del mundo	Microsoft
Modelo de desarrollo	Software de código abierto	Software propietario
Gratuito	Sí	No
Última versión estable	Linux *kernel* 3.2.1	7.0 (1ro. de marzo de 2011)
Tipo mercado	PDA, teléfonos inteligentes, robots, dispositivos electrónicos y aplicaciones industriales con microprocesadores	PDA, teléfonos inteligentes
Servicio de tiempo real	No (Existen soluciones para hacer de Linux un SO con servicios de tiempo real, como RTLinux y RTAI).	Sí
Plataformas soportadas	x86, MIPS, ARM, PowerPC	x86, MIPS, ARM
Footprint	2MB	540kB (versión básica *) 20MB (versión media **)

* En su versión básica, Windows CE 7.0 incluye únicamente lo indispensable para su funcionamiento, no incluye soporte para pantalla.

** En una instalación media se incluye interfaz gráfica. La versión más completa (40MB) ofrece además una versión adaptada de office, el reproductor Windows Media Player, mensajería instantánea y soporte para VoIP.

1.3 Bases de datos empotradas

Con el desarrollo constante de las tecnologías, el uso de las bases de datos (BD) se ha extendido de los servidores hacia los dispositivos móviles y empotrados.

Las bases de datos tradicionales que funcionan bajo una arquitectura cliente-servidor como: MySQL, Microsoft SQL Server y Oracle son muy lentas y consumen mucha memoria de la computadora, debido a que necesitan un proceso independiente con el que la aplicación se comunica y requieren estar establecidos en una máquina que actúe como servidor –la práctica habitual consiste en que el cliente (aplicación) y el servidor (sistema gestor) estén en dispositivos distintos–. En cambio, una base de datos embebida es aquella que se integra con una aplicación específica que requiere acceder a los datos almacenados; no inicia un servicio en una máquina independiente de la aplicación, pudiéndose enlazar directamente al código fuente o bien utilizarse en forma de biblioteca, tal que el sistema de base de datos está "oculto" para los usuarios finales de la aplicación. También se puede colocar como un fichero dentro de la carpeta de aplicación, logrando que solo el servicio que lanza la base de datos pueda acceder a su contenido (Junyan y col., 2009).

En la tabla 1.2 se presenta información general de las principales bases de datos embebidas, ordenadas alfabéticamente.

Tabla 1.2. Bases de datos embebidas.

Base de datos embebida	**Desarrollador**	**Lanzamiento**	**Licencia de software**
Advantage Database Server	Sybase	1992	Propietaria
Berkeley DB	Oracle	1996	Dual
Empress Embedded Database	Empress Software	1979	Propietaria
Firebird	Firebird Project	2000	IPL y IDPL
HSQLDB	HSQL Development Group	2001	BSD*
Informix Dynamic Server	IBM	1980	Propietaria
RDM Embedded	Raima Inc.	1984	Propietaria
SQLite	D. Richard Hipp	2000	Dominio Público
SQL Server Compact	Microsoft	2000	Propietaria

* Licencia de software libre permisiva (muy cercana al dominio público) otorgada para los sistemas BSD (*Berkeley Software Distribution*).

En la tabla anterior se puede observar que existen varios motores de bases de datos empotrados, incluso variantes embebidas de Oracle y de SQL Server de Microsoft, pero de todas, la más desplegada en el mundo es SQLite debido a que ha atraído fuertemente los diseñadores por ser una herramienta de software de dominio público[3].

1.3.1 SQLite

SQLite es una pequeña biblioteca de software escrita en lenguaje C que implementa un motor de base de datos SQL (*Structured Query Language*) autocontenido, sin servidor y sin necesidad de configuración. Es un proyecto en constante evolución porque su código fuente es de dominio público en base a la licencia GPL, por lo que puede ser modificado por cualquier persona. Gracias a esto, SQLite ha sido mejorada por un gran número de colaboradores y también ha sido migrada a diversas plataformas de sistemas operativos y dispositivos. Dado que SQLite es una base de datos relativamente desconocida, cabe destacar que es utilizada por Android para almacenar datos estructurados y Mozilla Firefox para almacenar *cookies*, XBMC Media Center para administrar contenido multimedia, entre otros.

La biblioteca de SQLite se integra en la aplicación, pasando a ser parte integral de la misma, es decir, el cliente y el servidor se ejecutan juntos en el mismo proceso o aplicación. El código de SQLite está embebido como parte del programa que utiliza su funcionalidad a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos (Owens, 2006).

SQLite tiene como características principales:

- Biblioteca autocontenida (sin dependencias externas) e implementada en ANSI-C. Toda la biblioteca ocupa menos de 400 kB y comparada con la mayoría de los motores de base de datos SQL es muy pequeña. Por ejemplo, la biblioteca de Firebird ocupa solo 350 kB, eso es tan grande como SQLite y ni siquiera contiene el motor de base de datos SQL (la tarea principal de un motor de base de datos es

[3] Software que no requiere de licencia, pues sus derechos de explotación pertenecen a todos por igual.

evaluar las declaraciones de SQL) y la biblioteca Berkeley DB de Oracle tiene 450 kB y omite la compatibilidad de SQL.

- No necesita un proceso separado funcionando como servidor independiente, de aquí se deriva su ventaja principal y es que no necesita instalar, configurar, inicializar o administrar un servidor, razón por la cual es un motor de base de datos de configuración mínima o nula. Con SQLite, el proceso que quiere acceder a la base de datos lee y escribe directamente desde el archivo de base de datos que se encuentra en el disco.
- Implementa el estándar SQL92. Permite que soporte desde las consultas más básicas hasta las más complejas del lenguaje SQL.
- La base de datos completa (definiciones, tablas, índices, y los propios datos) se almacena en un solo fichero a diferencia de otros SGBD que hacen uso de varios archivos.
- Soporta bases de datos de hasta 2 Terabytes de tamaño.
- Mucho más rápida que otros motores de base de datos populares para realizar operaciones comunes.
- API[4] simple, fácil de usar y multiplataforma. Se compila y ejecuta en Windows, Linux, Mac OS X y sistemas comerciales de Unix como Solaris, así como muchas plataformas embebidas como QNX, Palm OS, µCLinux y WinCE. Funciona en arquitecturas de 32 y 64 bits.
- Soporta importantes lenguajes de programación como C/C++, PHP y Java, los cuales se comunican con el archivo de base de datos a través de la API.
- Completa portabilidad y pocos recursos de memoria. Los archivos de bases de datos son tan portátiles como su código, se puede crear una base de datos SQLite en una estación de trabajo en Linux y utilizarlo en una máquina Mac o Windows, un iPhone u otro dispositivo sin ninguna conversión o modificación.

[4] Interfaz de programación de aplicaciones (API, por sus siglas en inglés *Application Programming Interface*), hace referencia al conjunto de funciones que posee una biblioteca para ser utilizada por otro software.

- Es de dominio público, y por tanto, es libre de utilizar para cualquier propósito sin costo y se puede redistribuir libremente.
- Integración directa con Qt4.

En el sitio web http://www.sqlite.org se encuentran varias herramientas gráficas, de código abierto y comercial para gestionar la base de datos y para crear una nueva desde cero. Entre ellas: SQLiteman y la extensión de Firefox, SQLite Manager, que transforma el explorador Firefox en un gestor de bases de datos SQLite con la ventaja de que al instalarse en el explorador funciona exactamente igual en todos los sistemas operativos, ver figura 1.3. Este programa usa la biblioteca SQLite compilada dentro de su aplicación *host* –Firefox– y por lo tanto, soporta versiones de archivos de base de datos soportadas por la biblioteca de este navegador.

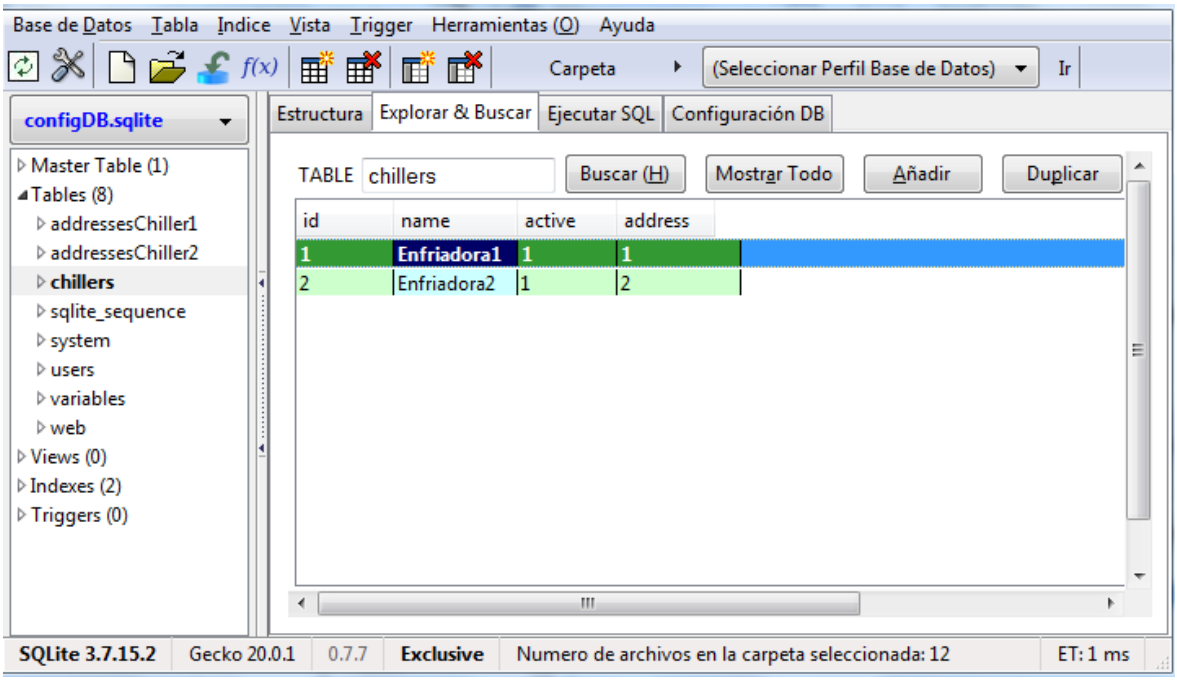


Figura 1.3. Extensión de Firefox, SQLite Manager

La biblioteca SQLite cuenta también con una utilidad pequeña de línea de comando llamada sqlite3 Command-Line Program (CLP) que permite ejecutar comandos SQL contra una base de datos SQLite, debido a que tiene el motor de la base de datos compilado estáticamente con las dependencias, actuando como un programa autónomo. Desde aquí se pueden crear bases de datos, realizar consultas e insertar datos, basta instalar la aplicación

(está disponible tanto en Linux como en Windows) y teclear “sqlite3” seguido del nombre de la base de datos a la que se quiere acceder y seguidamente la sentencia SQL que se quiera ejecutar.

SQLite no está diseñada para competir con bases de datos como Oracle o Microsoft SQL Server, según el criterio de sus desarrolladores, la regla básica para determinar cuándo es apropiado utilizar esta base de datos es la siguiente: usar SQLite en situaciones donde la simplicidad de administración e implementación son más importantes que las complejas características que proporcionan los motores de base de datos tradicionales.

1.4 Consideraciones finales del capítulo

El desarrollo tecnológico de los sistemas empotrados presenta un avance acelerado desde hace varios años con la tendencia de continuar aumentando la complejidad, capacidad de procesamiento multitarea, gestión de memoria, comunicación entre tareas y procesos y acceso a redes de comunicación entre otros factores que han hecho posible el surgimiento de sistemas operativos para estos sistemas.

Del proceso de investigación llevado a cabo y expuesto en este capítulo se evidencia que entre los sistemas operativos empotrados, las distribuciones de Linux son claramente líderes para este tipo de dispositivos, debido a que sus características lo hacen un sistema casi perfecto para esta rama tan cambiante, al ser un ejemplo eminente de software libre con el mayor soporte de arquitecturas –no está diseñado para un hardware específico–, mantenido y enriquecido por la comunidad mundial, por lo que no demora mucho en que los proyectos que surgen le den soporte inmediatamente.

Las bases de datos para sistemas empotrados difieren mucho de las bases de datos tradicionales con arquitectura cliente-servidor. Producto del limitado espacio de memoria de estos sistemas, las bases de datos embebidas no necesitan un servidor con el que el programa principal se comunique, en cambio se integran con la aplicación. Dentro de la gran variedad de bases de datos embebidas que han surgido en respuesta a la diversidad de sistemas empotrados, se destaca una, producto de su licencia pública y código abierto, sencilla configuración, soporte multiplataforma y de lenguajes de programación modernos: SQLite.

A pesar de que en Cuba existen proyectos que desarrollan algo parecido a lo que se necesita, la razón principal por la cual se deciden implementar el sistema de supervisión y control para máquinas enfriadoras sobre un dispositivo empotrado y específicamente los módulos para la comunicación entre el *Gateway* Carel –que contiene la información de las variables principales de los *chillers*– y el sistema empotrado, adquisición y administración es por la inestabilidad que presenta un software para tal motivo funcionando sobre una computadora de escritorio con sistema operativo Windows, vulnerable en cuanto a seguridad y en varias ocasiones se emplea para otros fines que no responden al objetivo por el cual existe.

CAPÍTULO 2. DISEÑO E IMPLEMENTACIÓN DE LOS MÓDULOS

En el presente capítulo se seleccionan los lenguajes para el modelado y programación de los módulos requeridos por la aplicación, así como las herramientas o programas empleados para ello, considerando siempre que posean licencia de software libre y se describen las principales funcionalidades de los módulos implementados.

2.1 Requisitos funcionales.

Para el diseño de los módulos de la aplicación hay que destacar que la información de las variables de interés de las diferentes enfriadoras se recibe de un *Gateway* Carel que se conecta a la tarjeta MBTI2440 a través del protocolo Modbus en su variante RTU vía RS232, puesto que los *chillers* emplean un protocolo de comunicación propietario llamado CarelNet y se hace necesario traducir la información de este protocolo por medio del *Gateway* a un estándar de comunicación más utilizado como Modbus (protocolo de comunicación abierto, lo que significa que es gratuito, no hay que pagar patentes), ver figura 2.1.

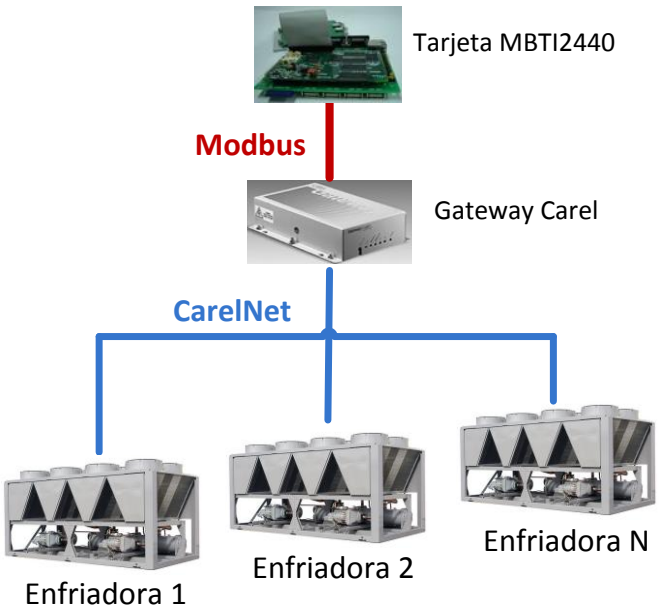


Figura 2.1. Arquitectura del sistema de comunicación de las enfriadoras

El empleo de la tarjeta de hardware MBTI2440 asegura una implementación económica, robusta, de bajo consumo de recursos y de fabricación nacional, por tanto se debe notar que para garantizar la completa independencia tecnológica las herramientas de desarrollo deben tener licencia de software libre o de dominio público. Además de seleccionar las posibles herramientas a usar, es necesario definir lo que se quiere lograr funcionalmente, es por ello que los módulos diseñados en este trabajo y que formarán parte del proyecto para la supervisión y el control de las enfriadoras deben cumplir los siguientes requisitos:

- El sistema modular debe contar con un módulo principal que maneje la ejecución del resto de los módulos.
- Comunicarse a través del protocolo Modbus RTU para línea serie con un *gateway* que contiene la información de las principales variables de las enfriadoras.
- Recolectar y almacenar los datos de las variables para una base de datos en el dispositivo empotrado.
- El módulo de comunicación debe tener un *driver* simulado que genere valores aleatorios con fines de prueba de la aplicación.
- Permitir que la parte gráfica y otros módulos del sistema tengan acceso a los datos almacenados en la base de datos.

Una vez definidas las funcionalidades que deben tener los módulos integrantes de la aplicación, se inició el proceso de diseño con la descripción del hardware y la selección de las herramientas de software.

2.2 Descripción de la tarjeta MBTI2440.

Para la concepción de la aplicación se presenta como solución de hardware la computadora en una tarjeta MBTI2440 diseñada por el Instituto Central de Investigación Digital (ICID), basada en el microprocesador S3C2440A de SAMSUNG, el cual emplea un núcleo ARM920T de 32 bits (Jiménez, 2009).

ARM es una arquitectura de procesadores RISC que permite combinar un potente núcleo de procesamiento con diversos periféricos integrados para desarrollar procesadores de diversos usos, entre ellos los sistemas empotrados que requieren de alta integración y bajo consumo.

La MBTI2440 fue diseñada con el objetivo de obtener una tarjeta de procesamiento de bajo costo y consumo de potencia, pequeño tamaño y elevada conectividad, sobre la cual se pueden instalar los sistemas operativos Linux y Windows CE.

Con el objetivo de minimizar el tamaño, esta tarjeta no posee conectores estándar para la interconexión con los periféricos y dispositivos de interfaz, sino todas las líneas de entrada/salida están conectadas a tres conectores de 50x2 contactos. De esta forma, se eliminaron los conectores estándares que llevarían cada uno de los periféricos y dispositivos de interfaz, por lo que es necesario adquirir una tarjeta base de desarrollo que se suministra con la MBTI2440 y que se conecta a dicha tarjeta, en la cual se colocarán los conectores requeridos, como muestra la figura 2.2.

Esta tarjeta tiene las siguientes características técnicas:

- Procesador S3C2440A de SAMSUNG a 400MHz (con un núcleo ARM920T).
- Memoria
    1. 32MB de SDRAM
    2. 1MB de NOR FLASH
- 64MB de NAND FLASH
- Interfaz para LCD

- Interfaz Ethernet 10 Mbit/s
- 3 puertos serie (útiles en la comunicación a través del protocolo Modbus)
- 1 puerto USB *device*
- 4 puertos USB *host*.
- Interfaz para tarjeta de memoria SD/MMC
- Reloj de tiempo real con respaldo de batería

Figura 2.2. Tarjeta MBTI2440 conectada con los dispositivos

2.2.1 Sistema operativo para la tarjeta de hardware.

Pensando en la implementación de este trabajo se debe seleccionar un SO adecuado para la aplicación, que es la encargada de darle vida al sistema empotrado, accede al hardware y le saca el máximo provecho gracias a la abstracción de bajo nivel que el sistema operativo realiza sobre todo el sistema, por lo cual es necesario hacer uso de un SO que cumpla con las siguientes características:

- Robusto y moderno, con la facilidad de poderse configurar para ser lo más sencillo posible cuando se implemente sobre una arquitectura funcional.
- Permitir que muchas aplicaciones existentes puedan funcionar en él con poca o ninguna modificación en su código.
- Que sea de código abierto y a la vez haya conocimiento de su estructura.
- Lo más importante, ser gratuito.

Como se vio anteriormente, sobre la tarjeta MTBI2440 se pueden montar los sistemas operativos Linux y WinCE y las características anteriores, en especial las dos últimas excluyen a WinCE.

2.2.1.1 ¿Por qué Linux?

La alternativa que más se ajusta a los requerimientos antes dados es sin dudas el sistema operativo Linux, el cual los cumple a cabalidad por ser un SO moderno, que se impone cada vez más sobre sistemas propietarios como Windows. Al ser de código abierto, gratuito y estar en permanente evolución y desarrollo gracias al gran número de personas que continuamente le brindan mejoras y soluciones, resulta ideal para la implementación de aplicaciones avanzadas y muy funcionales que requieren soporte para las últimas tecnologías, además para potenciar la utilización se software libre, la tarjeta MBTI2440 se suministra con Linux Embebido preinstalado para procesadores ARM (kernel 2.6.29).

2.3 Lenguajes y herramientas.

Cuando se decide realizar un proyecto de cualquier magnitud o complejidad se debe hacer un análisis previo de las posibles herramientas y materiales disponibles para la implementación del mismo, para seleccionar los más adecuados de acuerdo a los requerimientos que debe cumplir. A continuación se describen los lenguajes para el modelado y programación de la aplicación así como las herramientas empleadas para ello.

2.3.1 Modelado de la aplicación.

Un modelo es una abstracción que sirve para describir todo un sistema o un subsistema que forma parte del mismo y el modelado es el diseño de un software antes de su codificación, es la visualización de lo que se quiere construir, por lo que para el desarrollo apropiado de los módulos de la aplicación se utilizó el Lenguaje Unificado de Modelado (UML, *Unified Model Language*). UML es el lenguaje de modelado de sistemas de software más conocido en la actualidad; es el estándar internacional aprobado por la OMG (*Object Managment Group*), consorcio creado en 1989 responsable de la creación, desarrollo y revisión de especificaciones para la industria del software. Este lenguaje permite a los creadores generar modelos que capturen las ideas en formas convencionales y fáciles de comprender

para comunicarlas a otras personas (Schmuller, 2000), es un lenguaje gráfico estándar que sirve para describir los “planos del software”, puede utilizarse para visualizar, especificar, construir y documentar un sistema, en otras palabras, es el lenguaje en el que está descrito el modelo de un sistema.

2.3.2 Herramienta de modelado.

Como herramienta para modelar con UML se utilizó el software libre Umbrello que se distribuye en el módulo kdesdk de KDE, aunque funciona en otros entornos de escritorio (Hensgen, 2003). Umbrello fue desarrollado por Paul Hensgen y maneja gran parte de los diagramas estándar UML pudiendo crearlos, además de manualmente, importándolos a partir de código en C++, Java, Python, IDL, Pascal/Delphi, Ada, o también Perl (haciendo uso de una aplicación externa). Así mismo, permite crear un diagrama y generar el código automáticamente en los lenguajes antes citados, entre otros.

2.3.3 Lenguaje de programación.

Como lenguaje de programación se empleó el C++ producto de la gran cantidad de documentación y proyectos existentes, pero principalmente se emplea con el objetivo de aprovechar el conjunto de herramientas del *framework* Qt (cuya traducción aproximada sería "marco de trabajo") para el desarrollo rápido de aplicaciones multiplataforma en C++. Con Qt se puede reutilizar el código eficientemente en varias plataformas con una sola base de código, su biblioteca modular de clases librera a los desarrolladores de la escritura de código de bajo nivel (Digia Qt, 2013) y cuenta con un módulo de base de datos llamado QtSql que proporciona integración mediante *drivers* con las bases de datos más importantes, incluyendo la base de datos embebida SQLite que es la empleada en este proyecto.

2.3.3.1 Biblioteca Qt

Qt es una biblioteca multiplataforma que en su forma originaria trabaja con el lenguaje de programación C++, pero existen módulos que ofrecen soporte para otros lenguajes de

programación gracias a una adaptación (*binding*[5]), entre los cuales se encuentran Python mediante el PyQt, Java (QtJambi), C# (Qyoto), Perl (PerlQT) como los más reconocidos. Algunas de sus características son:

- Compatibilidad multiplataforma con un sólo código fuente. (Se encuentra disponible en todas las principales plataformas -GNU/Linux, Windows, Mac OS X, Windows CE, Linux Embebido-)
- Soporte para C++
- Amplio repertorio de métodos y funciones.

Su primera versión fue desarrollada en 1992 por Trolltech, una compañía de software noruega. En 2008 fue adquirida por la empresa finlandesa Nokia, que durante unos años hizo una fuerte inversión en este producto. Sin embargo, a comienzos de 2011, el negocio de licenciamiento comercial fue vendido a otra empresa finlandesa, Digia, aunque Nokia tiene intención de seguir respaldando el proyecto. En la fecha en que se confecciona este proyecto, la última versión estable de la biblioteca Qt es la 5.0.0 y se distribuye con tres tipos diferentes de licencias: GPL v2/v3 (para el desarrollo de software libre y de código abierto), QPL (licencia propietaria) y LGPL, pudiendo ser gratuitas o de pago y para fines comerciales o no comerciales.

Como referencia podemos enmarcar la presencia del uso de Qt en el entorno de escritorio KDE, en el simulador de mapas Google Earth, Skype -realiza llamadas sobre internet-, en el reproductor multimedia VLC media player y en el entorno de desarrollo integrado que se empleó para realizar la aplicación del proyecto, Qt Creator.

Aunque la función más conocida de Qt es la creación de interfaces gráficas de usuario, cuenta con módulos que permiten la aplicación de las tecnologías más productivas y aceptadas como: SQL (QtSql), XML (QtXml), SVG (QtSvg), OpenGL (QtOpenGL), entre otros, para de una manera sencilla e intuitiva facilitar ciertas tareas de programación y crear tanto aplicaciones de escritorio como para sistemas empotrados.

[5] Adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquél en el que ha sido escrita.

2.3.4 El IDE Qt Creator.

El programa es implementado usando el Qt Creator, versión 2.6, un entorno de desarrollo integrado (IDE, *Integrated Development Environment*) multiplataforma, que como indica su nombre, está basado en la biblioteca Qt. Un IDE es un software compuesto por un conjunto de herramientas para programar aplicaciones, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario (GUI).

Algunas características de Qt Creator son:

- Editor de código avanzado para C++, con completado automático y navegación.
- Soporte para el control de versiones. Se integra con varios de los sistemas de control de versiones más populares como Git, Subversion, Bazaar, Perforce, CVS y Mercurial.
- Integra dos editores de interfaz gráfica. *Qt Designer* (para la creación de interfaces gráficas a partir de los *widgets* de Qt) y *Qt Quick Designer* (para la creación de interfaces gráficas animadas con el lenguaje QML)
- Depurador visual

2.4 *Toolchain* de compilación cruzada

Como la tarjeta MBTI2440, que albergará y ejecutará la aplicación y que es conocida como "objetivo" o *target* tiene recursos limitados que hacen que el proceso de compilación sea incómodo o imposible, se decide utilizar una computadora auxiliar, conocida como huésped o *host*, en la que se escribe el código fuente de la aplicación y se hace la compilación. Normalmente el *host* es una computadora de propósito general, por lo que no tendrá la misma arquitectura que el *target*. El código fuente debe ser compilado en la arquitectura *host*, pero de acuerdo a las reglas de compilación de la arquitectura *target*, para crear un fichero binario ejecutable por esta última. Al proceso de compilar un código fuente bajo una determinada arquitectura para generar binarios para una arquitectura diferente se le denomina compilación cruzada (del inglés, *cross-compiling*). Esto significa que la máquina en la que se compila el software no puede ejecutar, de forma nativa, el código compilado.

En este proyecto el código fuente se compila en una PC con arquitectura x86, pero el resultado es un archivo binario que se ejecutará sobre la tarjeta MBTI2440 con arquitectura ARM como muestra la figura 2.3.

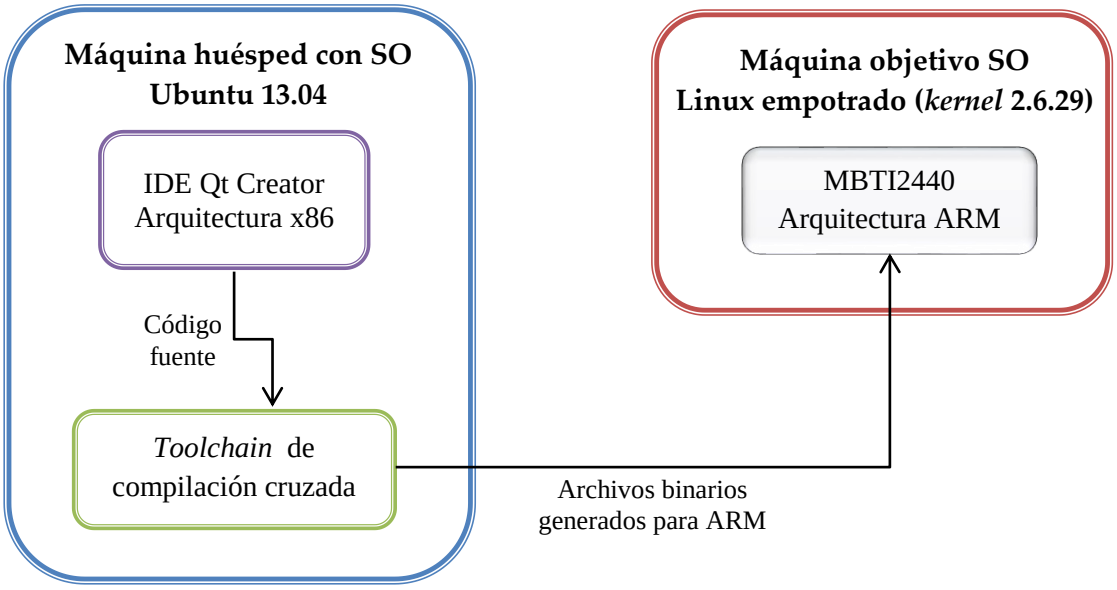


Figura 2.3. Proceso de compilación cruzada

Un *toolchain*, término en inglés que significa cadena de herramientas, es un conjunto de programas informáticos (herramientas) que se usan para obtener un determinado producto (normalmente otro programa o sistema informático) para una plataforma concreta de hardware. Los distintos programas se suelen usar en una cadena, de modo que la salida de cada herramienta sea la entrada de la siguiente (Petazzoni y Opdenacker, 2009). Una simple cadena de herramientas de desarrollo de software consiste de un editor de texto para editar código fuente, un compilador y enlazador para transformar el código fuente en un programa ejecutable, bibliotecas para proveer una interfaz al sistema operativo y un depurador.

Aunque es posible construir un *toolchain*, el proceso sería largo y difícil, se necesitan compilar varios componentes, descargar las cabeceras del núcleo y la biblioteca de C, estar familiarizado con las herramientas de configuración y compilación, por lo que suele ser más conveniente y cómodo obtener un *toolchain* pre compilado (González y Vázquez, 2011).

Para este trabajo, se ha descargado la versión 4.8.4 de la biblioteca Qt del enlace de Internet http://qt-project.org/downloads y que corresponde con el fichero `qt-everywhere-opensource-src-4.8.4.tar.gz`.

Para compilar la biblioteca y generar binarios que se ejecuten en la arquitectura ARM se emplea el *GNU ARM toolchain*, también conocido como *Sourcery G++ Lite* disponible para la descarga en http://www.codesourcery.com/Resources seleccionando el archivo IA32 GNU/Linux TAR: `arm-2009q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2`. Para obtener más detalles del proceso de compilación de la biblioteca Qt para la MBT2440 consultar el Anexo I.

Para configurar el IDE Qt Creator para compilar para la tarjeta ARM:

- Ir a la barra de menú en *Tool -> Options... -> Build & Run*
- En el panel izquierdo seleccionar *Qt Versions* y en el botón *Add->* añadir la ruta al directorio donde se han dejado compilada las librerías en el paso anterior `/usr/local/mbti2440/qt-4.8.4/bin` (Figura 2.4).

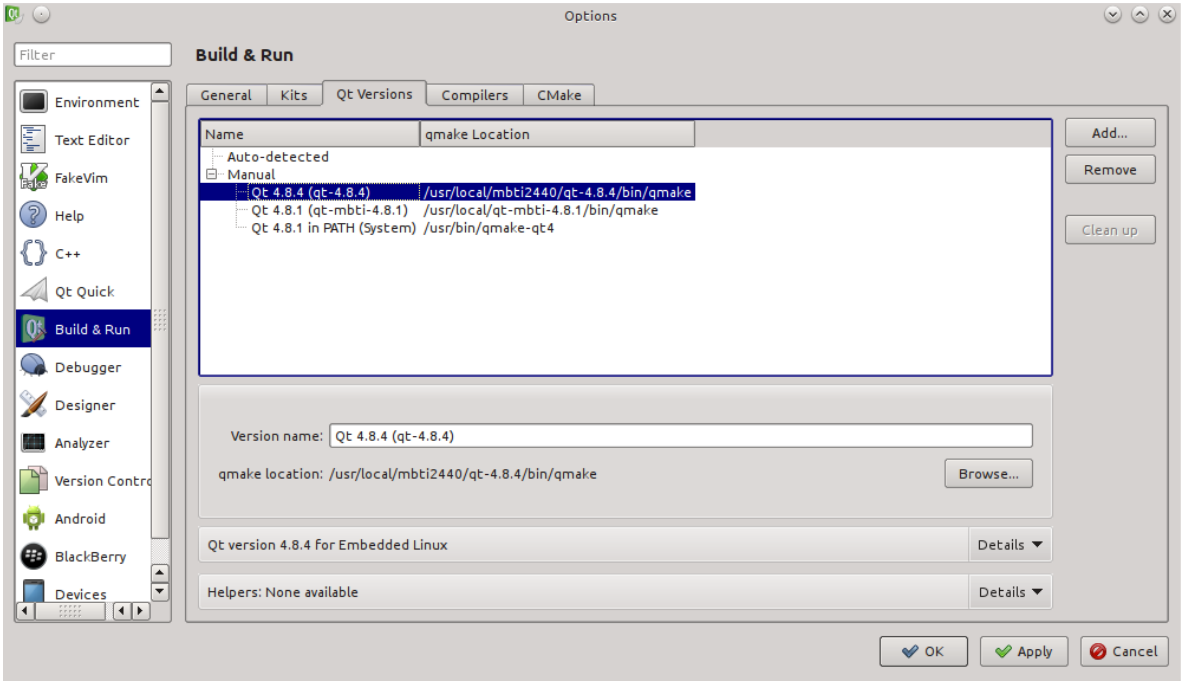


Figura 2.4. Configuración del compilador en Qt Creator

2.5 Descripción de los módulos

A la aplicación que se desarrolló se decidió darle el nombre de FríoCaribe y para que quedara bien implementada, el código del software se dividió en módulos o paquetes en función de las funcionalidades, de forma tal que existe una fuerte dependencia entre los elementos de código dentro de un módulo (alta cohesión) y a la vez poca interconexión de elementos de código entre módulos (bajo acoplamiento) (Rodríguez, 2013), obteniéndose para la versión inicial tres módulos principales como muestra la figura 2.5.

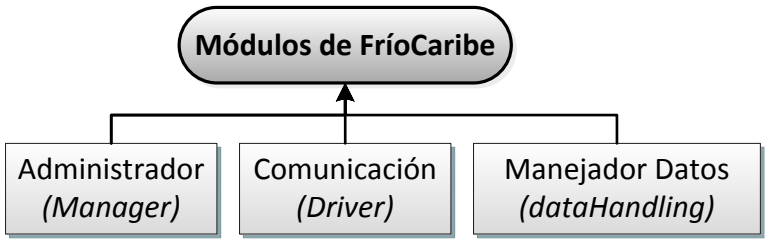


Figura 2.5. Módulos integrantes de FríoCaribe

Estos módulos contribuyen a dar forma a la solución final y a la vez sirven para dotar al sistema de una arquitectura de software robusta que permitirá ser usada como una representación de alto nivel del sistema y que pueda ser reutilizada en sistemas similares o en versiones posteriores del mismo, además de permitir un fácil mantenimiento del software y en el menor de los casos servirá como un excelente punto de partida documentado para nuevos diseños.

2.5.1 Módulo Manejador de Datos

La clase principal de este módulo es *DataHandler* y se encarga de recolectar los datos de las variables de cada *chiller* manejados por el sistema y almacenarlos en una base de datos analógica o digital, según el tipo de variable, para su organización y accesibilidad, para ello se crean dos bases de datos: Base de Datos de Configuración (CDB, *Configuration Database*) y Base de Datos Históricos (HDB, *Historical Database*)

La primera se encarga de mantener una imagen estática del estado del sistema en tiempo real, así como la información de configuración de la aplicación. Esta permite a los módulos intercambiar y mantener actualizada la información necesaria para llevar a cabo sus funciones.

La segunda, se encarga de almacenar cada un tiempo determinado, muestras del estado de las variables analógicas y digitales de los *chillers*, creando un arreglo de datos históricos de los mismos.

En la figura 2.6 se muestran las diferentes clases que componen el módulo manejador de datos y cómo se relacionan unas con otras. Nótese que este módulo también hace uso de una función del módulo *driver*.

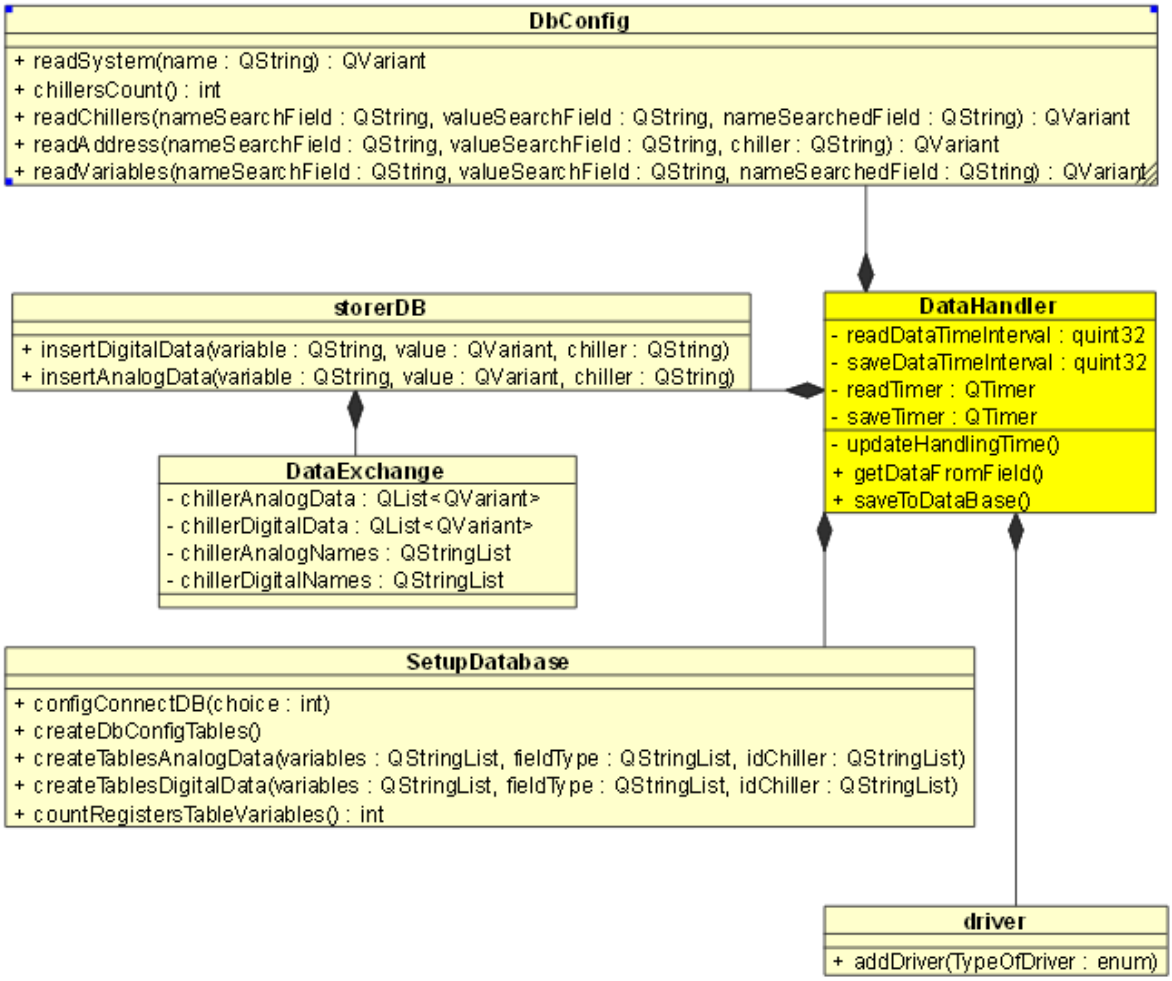


Figura 2.6. Diagrama de clases del módulo manejador de datos

Para crear las tablas necesarias en la CDB (configDB.sqlite) y en la HDB (dataDB.sqlite) mediante sentencias SQL, primero hay que establecer la conexión con la BD correspondiente, creando un objeto *QSqlDatabase* y a la función *addDatabase()* de dicha clase de Qt pasarle como primer argumento el tipo de *driver* que se usó para acceder a la

BD, en este caso "QSQLITE", después se pone la dirección de la BD extraída en la función *configConnectDB(int choice)*, la cual busca en un fichero de configuración llamado *system.conf* (figura 2.7) la dirección del archivo que contiene las bases de datos CDB y HDT, si el argumento es 0 devuelve una variable tipo lista con la ruta de la HDB y si es 1 devuelve la ruta a la CDB y finalmente se abre la conexión. Una vez que la conexión está establecida se emplea la clase *QSqlQuery* para ejecutar las sentencias SQL que permiten crear las tablas en las bases de datos anteriormente mencionadas.

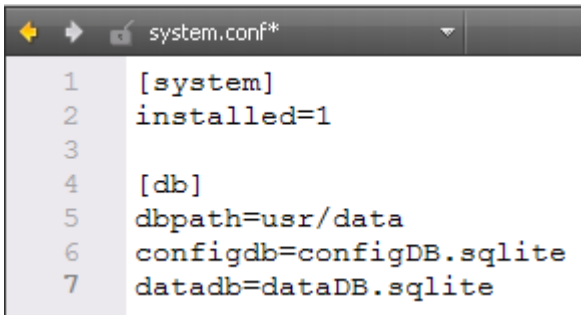


Figura 2.7. Archivo de configuración

En la CDB se crean las siguientes tablas:

1. Tabla de *chillers* (2.1):

Contiene la identificación para la aplicación de los *chillers* conectados al sistema (Id), el nombre, sus estados (activos o inactivos) y direcciones series.

Tabla 2.1. Tabla de *chillers*

Id	Nombre	Activo	Dirección
1	Enfriadora1	1	1
2	Enfriadora2	1	2
3	Enfriadora3	1	3
4	Enfriadora4	1	4

2. Tabla de configuración que lleva por nombre *system* (2.2).

Incluye variables de información y configuración de la aplicación, tales como los módulos habilitados y los deshabilitados (*httpService, dataHandler, localIface, automaticControl*),

el tipo de *driver* (*driver='0'* corresponde al simulado; *driver='1'*, Modbus RTU), los parámetros de comunicación entre el supervisor Modbus y el *Gateway* (*address, baud rate, parity, dataBits y stopBits*) y los tiempos para la lectura de los datos y salvarlos en la HDB.

Tabla 2.2. Tabla de configuración

Id	Nombre	Valor
1	versión	0.01
2	name	FríoCaribe
3	company	CEDAI Villa Clara
4	installed	1
5	checkTimeInterval	10000
6	httpService	1
7	dataHandler	1
8	localIface	0
9	automaticControl	0
10	driver	0
11	address	COM1
12	baud	9600
13	parity	N
14	dataBits	8
15	stopBits	1
16	path	../usr/www
17	port	8000
18	getDataTimeInterval	30000
19	saveDataTimeInterval	60000

3. Tabla de variables (Anexo II):

Muestra el identificador (Id), nombre, una descripción, la unidad de medida y el tipo de las variables (1-Analógica/0-Digital) a recolectar en cada *chiller*.

4. Tabla de base de datos Modbus a partir del Gateway Carel

Es necesario construir una tabla de direcciones de las variables para cada *chiller*, por lo que se crean tantas tablas como *chillers* existentes, las cuales contienen el identificador (Id) de cada variable en correspondencia a lo establecido en la tabla "*variables"* y la dirección Modbus de las mismas, siguiendo las especificaciones que ofrece el *Gateway* Carel.

La construcción de la BD que se implementa para administrar un *chiller* Carel con dirección 1, se hace de la siguiente forma:

- Las variables digitales Carel son transferidas con sus direcciones a la BD Modbus y son leídas usando el código de función Modbus 0x01HEX (Leer estado de salidas digitales –bobinas–).
- Las variables analógicas Carel son transferidas con sus direcciones a la BD Modbus y son leídas usando el código de función Modbus 0x03HEX (Leer registros).
- El resto de los *chillers* se sitúan como el primero, comenzando desde el *offset* que se muestra en la tabla 2.3.

Tabla 2.3. *Offset* para determinar las direcciones Modbus de las variables.

***Chiller* Carel**	***Offset* variable digital**	***Offset* variable analógica**
Chiller 1	0	0
Chiller 2	200	256
...	...	...
Chiller n	(n-1)×200	(n-1)×256

La dirección Modbus se calcula adicionándole a la dirección Carel de la variable el *offset* mostrado en la tabla anterior, basado en la dirección serie del *chiller* al que las variables pertenecen. Regla general:

- Variables digitales:

$$\text{dirección Modbus} = \text{dirección Carel} + 200 \times (\text{dirección serie chiller} - 1)$$

- Variables analógicas:

$$\text{dirección Modbus} = \text{dirección Carel} + 256 \times (\text{dirección serie chiller} - 1)$$

Por ejemplo:

Para confeccionar la BD de direcciones Modbus para el *chiller* 1, ver tabla 2.4 y para el *chiller* 2, ver tabla 2.5.

Tabla 2.4. Direcciones Modbus para un *chiller* con dirección serie 1.

Tipo de variable Carel	**Dirección Carel**	**Dirección Modbus***
Digital	1	1
Digital	…	…
Digital	182	182
Analógica	1	1
Analógica	…	…
Analógica	92	92

* Se refiere a la dirección recibida en el paquete mandado por el supervisor.

Tabla 2.5. Direcciones Modbus para un *chiller* con dirección serie 2.

Tipo de variable Carel	**Dirección Carel**	**Dirección Modbus**
Digital	1	201(1+200)
Digital	…	…
Digital	182	382(182+200)
Analógica	1	257(1+256)
Analógica	…	…
Analógica	92	258(2+256)

En la HDB, por otra parte, se crea una tabla de las variables analógicas y una tabla para las digitales para cada uno de los *chillers* conectados al sistema e incluye un registro en cada una de sus tablas cada un tiempo determinado en la variable *saveDataTimeInterval* de la tabla *system* de la CDB.

Las funciones principales de la clase manejadora de datos son las de leer los datos desde Modbus (o el *driver* simulado en el marco de pruebas) y guardarlos en la HDB.

La función para recolectar los datos crea un arreglo temporal de datos analógicos y digitales de tipo *QList* (lista) y de nombres de las variables analógicas y digitales, guardando en una misma posición de memoria el valor recolectado y el nombre de la variable en cada uno de los arreglos correspondientes. El número de arreglos que se generarán depende de la variable entera con el número de *chillers* que hay en la tabla *chillers* de la CDB que se le pasa como argumento a dichos arreglos. A continuación se crea un objeto *driver* con el tipo de *driver* establecido en la tabla *system*; si es Modbus se obtiene de la tabla *chillers* la dirección del primer esclavo/*chiller* y comienza un ciclo para obtener el tipo de variable (A/D), el nombre y la dirección de la misma, si la variable es digital (*type=0*), al arreglo

temporal de variables digitales (*tempChillerDigitalData*[i]) se le asigna la función para leer datos desde Modbus, pasándole como argumentos los determinados anteriormente y conjuntamente el arreglo temporal de nombres digitales (*tempChillerDigitalNames*[i]) recibe el nombre de la variable y si la variable es analógica se va llenando el arreglo de lista temporal de datos y nombres analógicos (*tempChillerAnalogData*[i] y *tempChillerAnalogNames*[i])

Si el *driver* es simulado y la variable digital, al arreglo temporal de datos digitales se le asigna la función que genera datos *boolenos* implementada en la clase *simulated* del módulo *driver*. Estos arreglos se van llenando simultáneamente, es decir, el nombre de la variable y su valor ocupan la misma posición dentro del arreglo. Si la variable es analógica, el arreglo se va llenando con la función que genera datos enteros.

Este proceso se repite para cada uno de los *chillers* conectados al sistema y cuando se ejecuta el *slot* de salvar los datos, los datos analógicos se van guardando en su columna correspondiente en la tabla de datos analógicos, de igual forma ocurre con los digitales pero guardándose en la tabla de datos digitales. Este ciclo se repite tantas veces como número de *chillers* exista.

2.5.2 Módulo Administrador (Manager)

El módulo principal del sistema es el *Manager* debido a que se encarga de administrar el resto de los servicios. Chequea los módulos cada cierto tiempo determinado en una variable –*checkTimeInterval*– que se encuentra en la tabla *system* de la CDB y determina cuáles deben estar en funcionamiento según lo establecido en las variables booleanas *httpService* y *dataHandler* de esta misma tabla, si son iguales a '1' significa que estos módulos deben estar corriendo y si son '0' deben estar deshabilitados como muestra la figura 2.8.

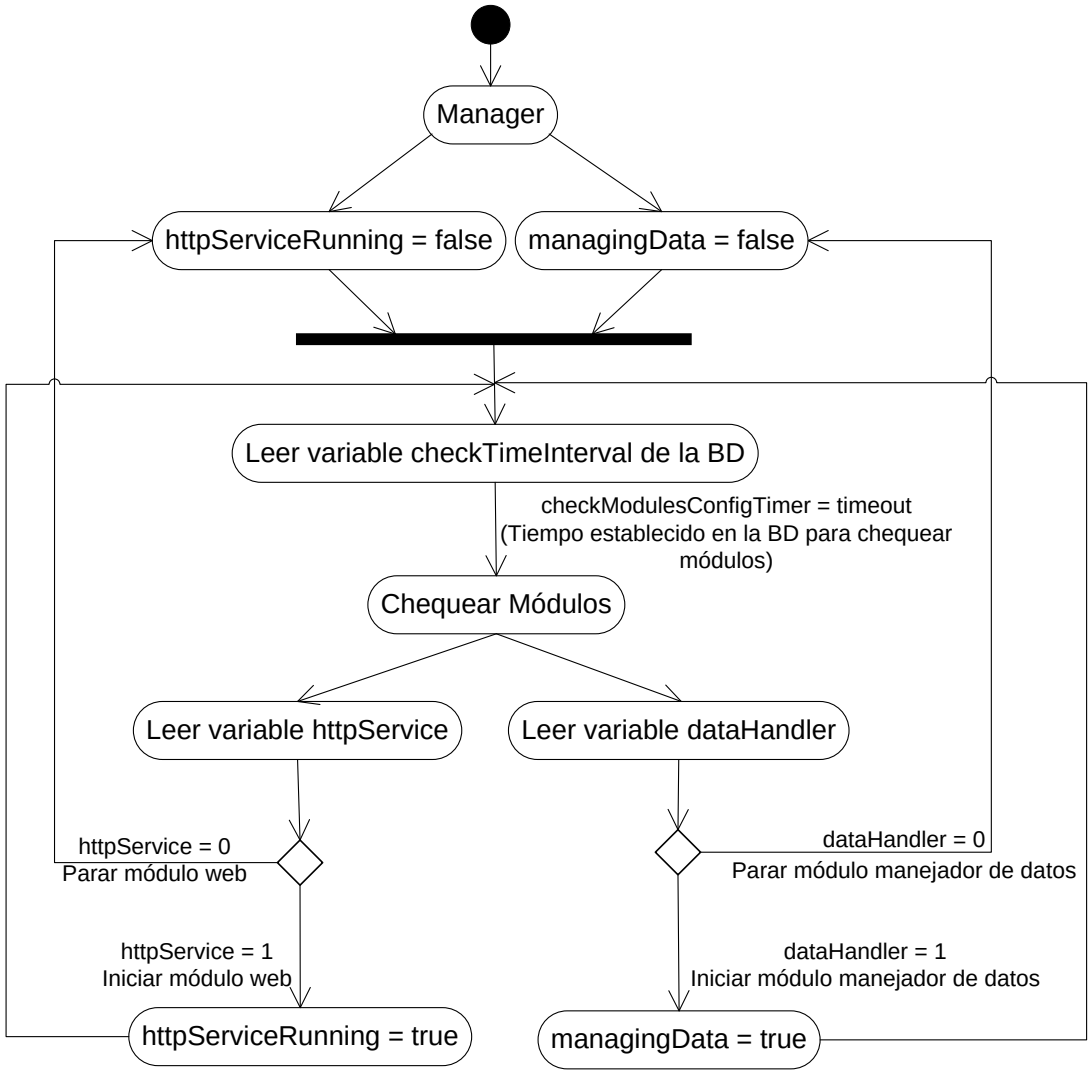


Figura 2.8. Diagrama de actividades para administrar los módulos

2.5.3 Módulo de comunicación (*Driver*)

El módulo *driver* se implementó para establecer la comunicación, es decir, intercambiar mensajes de datos entre el *Gateway* Carel y la aplicación FríoCaribe en la tarjeta procesadora MBTI2440 a través del protocolo Modbus RTU sobre un bus serie (norma RS232).

En este paquete de software se crea una interfaz genérica para la comunicación con otros dispositivos, entre ellos el *Gateway* Carel, debido a que es capaz de encapsular varios protocolos, realizando la tarea de seleccionar el requerido por la aplicación.

Para implementar el protocolo Modbus RTU se utiliza libmodbus, una biblioteca de Modbus hecha en C y se encapsulan la construcción e interpretación de los mensajes necesarios para la comunicación.

2.5.3.1 Biblioteca libmodbus

Libmodbus es una biblioteca de software libre para enviar/recibir datos de un dispositivo de acuerdo con el protocolo Modbus. Tiene como principales características (libmodbus Manual Page, 2013):

- Escrita en C.
- Soporta comunicaciones RTU (serie) y TCP (Ethernet).
- Biblioteca portable con soporte para Linux, Mac OS X, QNX y Win32.
- Proporciona una abstracción de las capas más bajas de comunicación y ofrece la misma API en todas las plataformas soportadas.

Como el protocolo Modbus tiene versiones para líneas serie (Modbus RTU) y Ethernet (Modbus TCP), para facilitar la ejecución de una variante, la biblioteca fue diseñada para utilizar un contexto -*modbus_t*- para cada variante. Cada contexto ofrece una función específica para crear una conexión, es decir, una estructura que contiene toda la información necesaria para establecer una conexión con otros dispositivos Modbus de acuerdo con la variante seleccionada.

El contexto RTU se utiliza en la comunicación serie y hace uso de una representación binaria compacta de los datos del protocolo. El formato RTU finaliza la trama con una suma de control de redundancia cíclica (CRC), como un mecanismo de comprobación de errores para garantizar la fiabilidad de los datos.

La trama de Modbus RTU llama un esclavo (un dispositivo o servicio que manejan peticiones Modbus) y un maestro, un cliente que envía solicitudes. La comunicación siempre es iniciada por el maestro.

El contexto TCP implementa una variante de Modbus utilizada para las comunicaciones estableciendo la transmisión mediante paquetes TCP/IPv4.

Para utilizar libmodbus hay que encapsular su código en al adaptador Modbus (*ModbusAdapter*) que se usa desde la interfaz *driver*, pues está hecha en C y la aplicación se está haciendo en C++. Esta biblioteca está bajo GLP (licencia pública de GNU).

2.5.3.2 Funcionalidades típicas

El módulo *driver* está compuesto por una clase de igual nombre y las clases *simulated* y *modbusAdapter* y emplea una función de la clase *DBConfig* del módulo de manejo de datos que devuelve el valor del nombre de la variable que se pasa como argumento de la tabla *system*. (Ver figura 2.9)

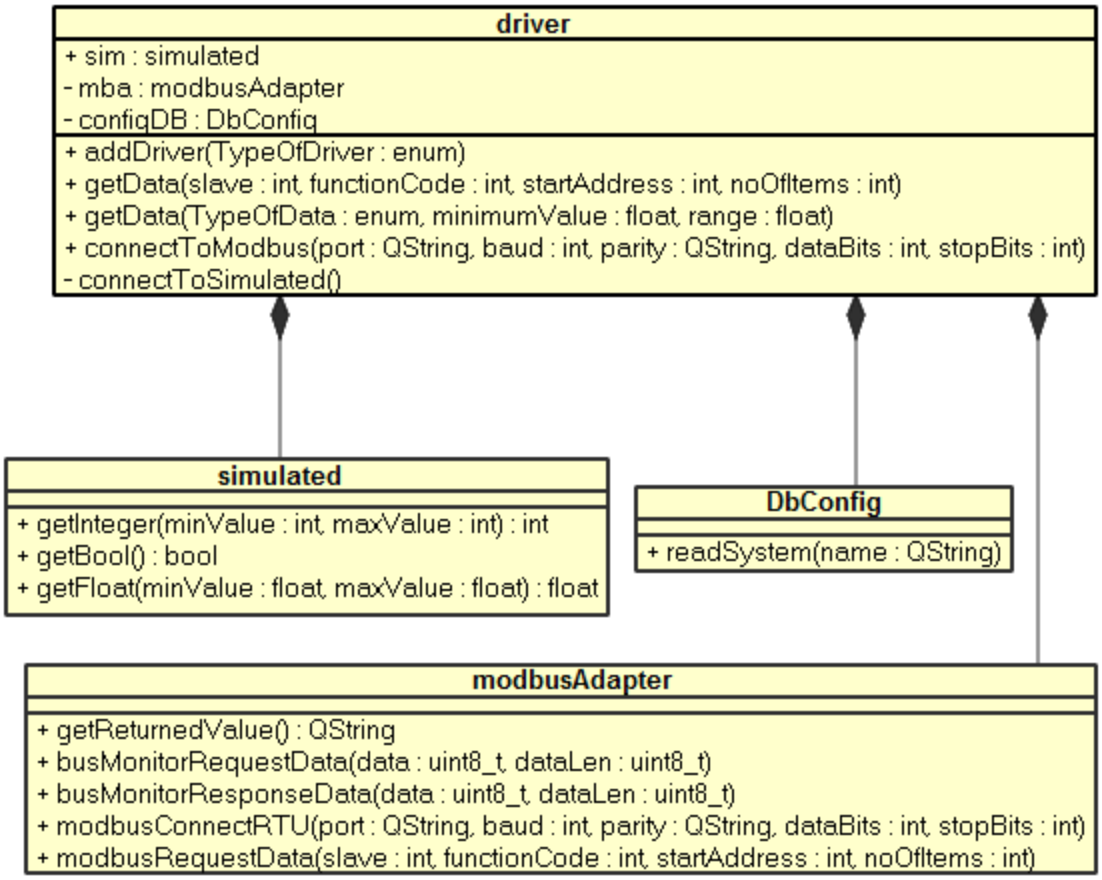


Figura 2.9. Diagrama de clases del módulo *driver*

En la clase *driver* se crea una variable "enum" o enumerado llamada *TypeOfDriver* en la que se declaran los diferentes *drivers* implementados en la aplicación como muestra la siguiente línea de código: *enum TypeOfDriver {simulated, modbusRTU};* de esta forma si

en la base de datos de configuración la variable *driver* tiene valor 0 corresponde al *driver* simulado y 1 a Modbus RTU.

Para añadir el tipo de *driver* deseado a la aplicación se declara la función *addDriver* en la que se lee de la tabla *system* el valor de la variable *driver* y se le pasa a la función, si es 1 (Modbus RTU) a la función *ConnectToModbus* se le asignan los valores de configuración de la conexión tales como; puerto serie de comunicación (COM1), *baud rate* (9600), paridad (*none*), bits de datos (8) y bits de parada (1); estos valores se pueden modificar en la tabla *system*.

La función *getData* lee datos booleanos, enteros entre 1 y 100 y de punto flotante desde el *driver* simulado generados por las funciones *getBool(), getInteger() y getFloat()* respectivamente de la clase *simulated* con fines de prueba de la aplicación en el proceso de diseño.

2.6 Consideraciones finales del capítulo.

En el capítulo que finaliza con esta sección concluye la implementación de los módulos que se decidió diseñar en correspondencia a los requisitos funcionales determinados, obteniéndose los módulos de administración, comunicación y adquisición de datos que junto a otros paquetes de software forman el sistema de control y supervisión para enfriadoras de agua sobre el dispositivo empotrado MBTI2440. Como esta tarjeta tiene recursos de hardware limitados se utilizó una máquina auxiliar para compilar el código y así crear un archivo binario ejecutable para la arquitectura ARM, lo que se conoce como proceso de compilación cruzada. Además se seleccionaron los lenguajes y herramientas de software libre para el desarrollo de los módulos, escogiendo Umbrello para realizar los diagramas UML de clases y actividad y el IDE Qt Creator para escribir y compilar el código debido a que está basado en la biblioteca Qt. En el próximo capítulo se realizarán pruebas para comprobar la funcionalidad de los módulos para recolectar datos a través del protocolo Modbus y almacenarlos en una base de datos.

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

En el capítulo anterior se describieron los requerimientos y el diseño e implementación de los módulos de administración, comunicación y adquisición de datos para el sistema de supervisión y control de los *chillers* sobre la computadora en una tarjeta MBTI2440. En este capítulo se realizarán pruebas para comprobar el funcionamiento de los módulos mediante el uso de un simulador de esclavos Modbus sobre RS232 para verificar que los datos leídos se almacenen en la HDB. Finalmente se desarrolla un análisis económico, para demostrar la viabilidad e importancia del trabajo realizado.

3.1 Desarrollo de las pruebas

El proceso de pruebas es el medio adecuado para medir el grado en que los módulos de software implementados cumplen los requerimientos. Las pruebas desarrolladas en este capítulo incluyen la compilación y ejecución de los módulos para la arquitectura huésped x86 y para la tarjeta MBTI2440 con arquitectura ARM para evaluar el consumo de memoria RAM y de CPU. Por último, se corre la aplicación sobre el dispositivo empotrado y se conecta a la PC sobre la cual se está emulando un simulador de esclavo Modbus.

3.1.1 Prueba 1

- Descripción del experimento: Compilación y ejecución de los módulos en diferentes arquitecturas procesadoras, sobre x86 con SO Windows XP y Ubuntu 13.04 y para ARM con Linux Empotrado 2.6.29.
- Parámetros a medir: Consumo de memoria RAM y de CPU.

Primera variante

- Características de hardware de la arquitectura huésped o *host*
  - PC de escritorio
  - Procesador Intel Dual Core a 2.5 GHz
  - 4 GB de memoria RAM

Para compilar la aplicación hay que configurar el compilador en el Qt Creator para cada uno de los sistemas operativos. En el caso de Windows se utiliza el compilador MinGW descargando el archivo `qt-win-opensource-4.8.4-mingw.exe` del enlace http://qt-project.org/downloads (Figura 3.1)

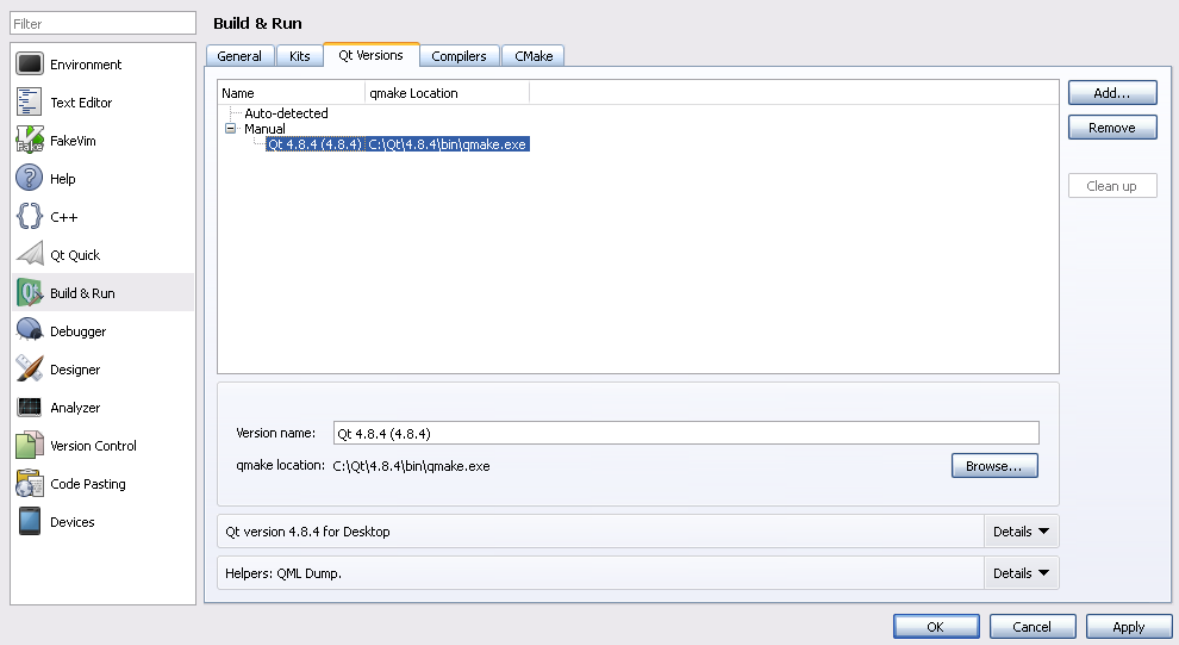


Figura 3.1. Configuración del compilador MinGW en Qt Creator

Para Ubuntu se instala la biblioteca `qt-everywhere-opensource-src-4.8.4.tar.gz` que en la sección 2.4 se compiló para ARM y el Qt Creator para Linux `qt-creator-linux-x86-opensource-2.7.1.bin`, todos descargados del enlace anterior y en el Qt Creator se añade la ruta al directorio de instalación de la biblioteca `/usr/bin/`

Segunda variante

- Características de hardware de la arquitectura objetivo o *target*
  - Computadora en una tarjeta MBTI2440 con SO Linux Empotrado 2.6.29
  - Procesador ARM920T
  - 32 MB de memoria RAM

En la tabla 3.1 se muestra un resumen de los resultados obtenidos. Nótese que la captura de top para la aplicación sobre la arquitectura ARM se toma en dos situaciones; la primera con el módulo manejador de datos recolectando datos del esclavo Modbus simulado por lo que hace un mayor uso de la CPU y la segunda con este módulo desactivado y con cinco usuarios accediendo a la aplicación web simultáneamente, lo cual disminuye el uso de la CPU pero aumenta notablemente el consumo de memoria RAM.

Tabla 3.1. Consumo de recursos de FríoCaribe en diferentes arquitecturas procesadoras.

Arquitectura	Sistema Operativo	Uso de memoria	CPU
x86	Windows XP	5.168 kB	32%
	Ubuntu 13.04	8.192 kB	2%
ARM	Linux Empotrado 2.6.29	4.224 kB (13.2%)	99%
	Linux Empotrado 2.6.29	28.416 kB (88.8%)	9.5%

3.1.2 Prueba 2

- Descripción: Ejecución de la aplicación sobre el dispositivo empotrado MBTI2440 que se conecta a la PC por RS232 donde se está emulando un esclavo Modbus con el programa *pyModSlaveQt-0.3.3* descargado del enlace: http://sourceforge.net/projects/pymodslave
- Objetivo: Comprobar el funcionamiento del *driver* Modbus RTU y las funciones de lectura y almacenamiento de los datos generados en la HDB

En el simulador de esclavo Modbus se seleccionó como tipo de codificación de los mensajes Modbus RTU, la dirección que debe identificar unívocamente al dispositivo esclavo de la red, que debe corresponderse con un número entre 1 y 247 (es el número máximo de nodos esclavos en conexiones multipunto aunque el Gateway Carel admite

como máximo 16 esclavos), como el programa solo puede simular un esclavo Modbus se escogió como dirección serie del esclavo (ID del esclavo) la 1 y generar datos digitales y analógicos cada 30seg debido a que la aplicación en este marco de prueba lee los datos cada medio minuto y los almacena en la HDB cada un minuto. Los parámetros de configuración de la conexión (puerto, *baud rate*, bits de datos, bits de parada y paridad respectivamente) se muestran en la figura 3.2 y en las figuras 3.3 y 3.4 aparecen los datos aleatorios generados en un ciclo para las funciones Modbus asociadas de leer estado de salidas digitales –bobinas– y leer registros analógicos de salida.

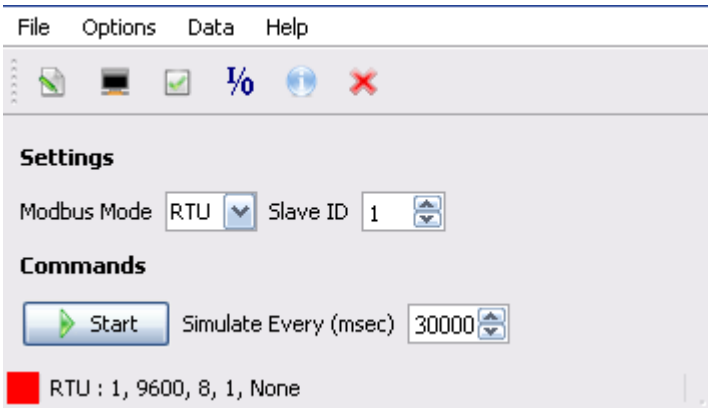


Figura 3.2. Simulador de esclavo Modbus *pyModSlaveQt*

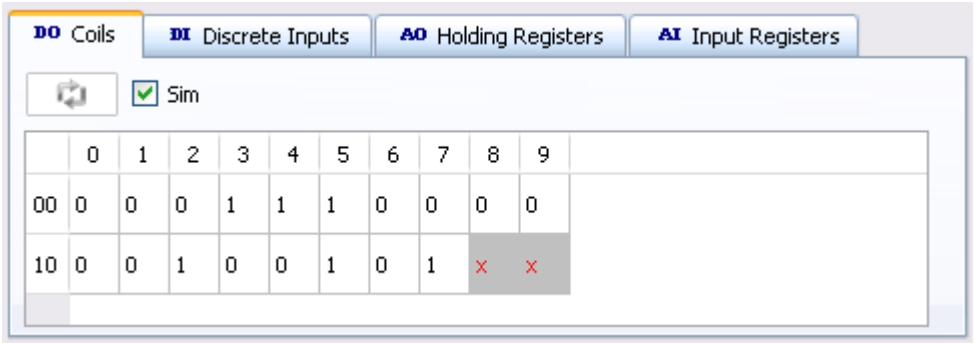
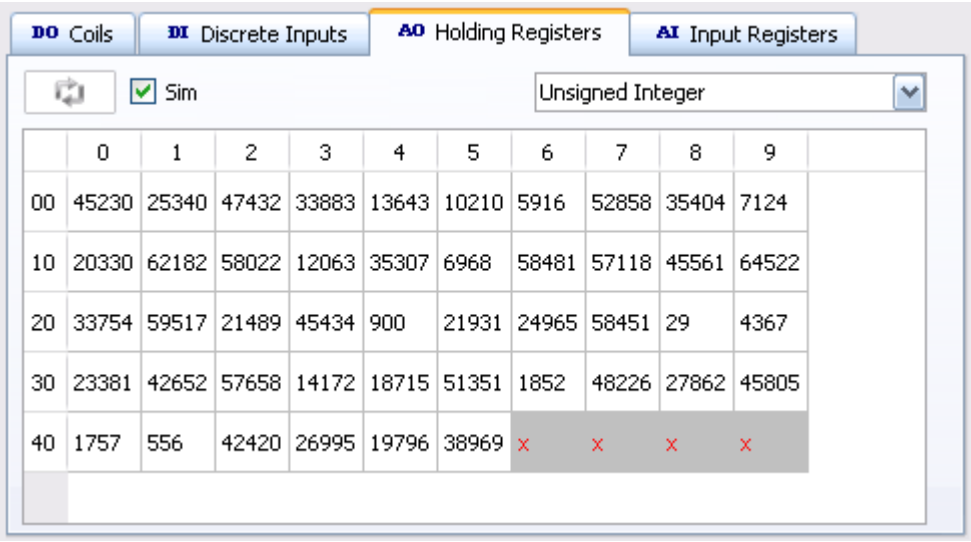


Figura 3.3. Salidas digitales generadas por el esclavo Modbus



	0	1	2	3	4	5	6	7	8	9
00	45230	25340	47432	33883	13643	10210	5916	52858	35404	7124
10	20330	62182	58022	12063	35307	6968	58481	57118	45561	64522
20	33754	59517	21489	45434	900	21931	24965	58451	29	4367
30	23381	42652	57658	14172	18715	51351	1852	48226	27862	45805
40	1757	556	42420	26995	19796	38969	x	x	x	x

Figura 3.4. Valores analógicos generados por el esclavo Modbus

Como se explicó anteriormente en la sección 2.5.1, los datos digitales generados aleatoriamente por el simulador para el esclavo/*chiller* son guardados en una tabla de variables digitales – *digitalDatal*– y los analógicos en una tabla de variables analógicas -*analogData1*-. En las figuras 3.5 y 3.6 respectivamente se visualizan estas tablas que se encuentran en la HDB con el empleo de la herramienta SQLite Manager. Las direcciones de las salidas digitales y los registros analógicos en el simulador coinciden con las direcciones Modbus de las variables que se encuentran en la "tabla de BD Modbus a partir del *Gateway* Carel" porque como es el primer esclavo no hay que sumarle el *offset*, es decir, la dirección Modbus es igual a la dirección Carel (en este caso serían las direcciones del simulador), pero sin tener en cuenta el registro 00, pues las direcciones de las variables Modbus comienzan a partir de la 01.

dDataId	chillerState	generalAlarm	stateOfComp1	alarmOfComp1	maintainOfComp1	stateOfComp2	alarmOfComp2	maintainOfComp2
1	**false**	**false**	**true**	**true**	**true**	**false**	**false**	**false**
2	false	false	true	false	true	false	false	true
3	true	false	true	false	true	true	false	true
4	true	false	false	true	true	true	false	false
5	true	false	false	true	true	true	false	false
6	true	true	true	false	false	false	false	false
7	false	false	true	true	true	false	false	true
8	true	false	false	true	false	true	false	true
9	false	true	false	true	false	true	false	true
10	true	true	false	true	false	true	false	true
11	true	true	false	false	true	false	false	true
12	true	false	false	false	true	false	false	true
13	true	true	false	true	true	false	false	false
14	false	true	false	false	true	true	false	true
15	true	false	true	false	false	true	false	false
16	false	true	false	false	false	true	false	true
17	false	false	false	true	true	false	true	false
18	false	false	false	false	false	false	true	true
19	false	true	true	false	true	false	true	false
20	false	true	false	true	false	true	true	false
21	true	true	false	false	false	false	false	false
22	true	false	false	true	false	true	false	true
23	true	true	false	true	false	true	false	true
24	false	true	false	true	false	true	false	true
25	false	true	true	false	true	true	false	false
26	false	true	true	false	true	true	false	false
27	false	true	true	false	true	false	true	false

Figura 3.5. Tabla de datos digitales del *chiller* con dirección serie 1

aDataId	plcWatch	waterTempIn	waterTempOut	envirommentTemp	recoveryTemp	compStage1	compStage2	waterTempSP	downPressureComp1
1	**25340**	**47432**	**33883**	**13643**	**10210**	**5916**	**52858**	**35404**	**7124**
2	16158	17079	18046	15750	16671	17638	15342	18199	15903
3	16210	17180	18251	16005	16975	17944	15649	15802	17027
4	15802	16566	17282	18046	15495	16210	16926	15903	16671
5	15802	16566	17282	18046	15495	16210	16926	15903	16671
6	17180	18046	15597	16413	17383	18098	15597	17843	15342
7	16158	16926	17690	15495	16720	17487	18199	15293	16109
8	16210	17180	18150	15855	16821	17742	15495	15087	16057
9	16210	17128	18098	15802	16772	17791	15495	15342	16364
10	16619	17690	15495	16671	17638	15597	16566	16210	17180
11	16158	16873	17589	18251	15802	16517	17229	16158	16873
12	16772	17742	15394	16263	17282	15038	16005	15802	16821
13	16263	17282	18251	16057	17027	18046	15802	15548	16566
14	16210	17180	18098	15802	16772	17742	15446	15038	15956
15	15855	16566	17282	18046	15495	16210	16926	15802	16619
16	16312	17282	18251	15956	16926	17896	15597	15188	16158
17	16158	17180	18098	15750	16720	17843	16263	17435	15139
18	16364	17487	15293	16057	16821	17487	15038	18098	15701
19	16210	17128	18046	15802	16772	17742	15394	18199	15903
20	16210	17180	18098	15750	16926	17944	15649	15446	16465
21	15903	16566	17282	15038	16158	16926	17742	17638	15293
22	16312	17334	15139	16210	17180	15139	16210	16057	17027
23	16312	17334	18251	15956	17027	17997	15701	15342	16312
24	16158	17128	18098	15750	16720	17690	15342	18199	15903
25	15855	16517	17282	17997	15394	16109	16821	15649	16364
26	15855	16517	17282	17997	15394	16109	16821	15649	16364
27	15597	16263	17027	17690	15139	15802	16517	15293	15956

Figura 3.6. Tabla de datos analógicos del *chiller* con dirección serie 1.

3.2 Análisis económico

Para realizar una valoración económica del trabajo desarrollado se tomará en cuenta la aplicación en general, es decir, el sistema para supervisar y controlar los *chillers* sobre la tarjeta MBTI2440, debido a que los módulos diseñados tienen la finalidad de integrarse al mismo, para ello se hace un análisis comparativo del costo de un sistema similar al anterior desarrollado en Cuba y del costo estimado del sistema propuesto.

El "Sistema de control para la optimización de las máquinas enfriadoras en el Hotel Meliá Las Dunas" desarrollado por la empresa CEDAI Villa Clara tiene un costo de $12,939CUC, esta oferta incluye el dispositivo secuenciador de setpoints, supervisión y control de la batería de enfriadoras, el sistema de supervisión con posibilidades de almacenamiento de los datos y los recursos humanos.

Por otra parte, el proyecto de supervisión y control de enfriadores sobre dispositivo empotrado asciende a los $5836CUC, tomando en cuenta como recurso material principal el MBTI2440, cuyo precio está alrededor de 250CUC e incluyendo la contratación del personal, considerando un tiempo estimado para la realización del proyecto de dos meses más un mes de instalación y puesta en marcha y que solo se necesita contratar un profesional con experiencia (Tabla 3.1).

Tabla 3.2. Costo de la contratación de un profesional de la empresa CEDAI

Número de profesionales	Tarifa horaria	Cantidad de horas por mes	A cobrar en un mes	Tiempo de duración	Total (CUC)
1	9.80CUC	190	1862CUC	3 meses	5586

Para demostrar la factibilidad de este proyecto para la instalación hotelera, se tomó un estudio realizado por la empresa CEDAI basado en resultados anteriormente obtenidos por FRIOCLIMA, la empresa líder en el mercado de la climatización centralizada en Cuba, donde se evidencia que se han logrado ahorros de $5.00CUC/TR al mes (para cuatro enfriadoras de 170TR/u - toneladas de refrigeración cada unidad-). Si se toma una cifra conservadora de $2.00CUC/TR, la inversión inicial se recupera en nueve meses

aproximadamente. A partir de este período se ahorrarían $1360.00CUC mensuales por concepto de disminución de consumo eléctrico.

La diferencia de costos entre los dos proyectos es de $7353CUC, esta diferencia hace notar que el sistema desarrollado sobre la MBTI2440 es menos costoso en cuanto a la inversión inicial, por lo cual se puede concluir que el proyecto representa una ventaja competitiva económicamente contra otros sistemas, en este caso el "Sistema de control para la optimización de las máquinas enfriadoras para el Hotel Meliá Las Dunas"

3.3 Conclusiones del capítulo

En el capítulo que concluye con esta sección se comprobó la funcionalidad de los módulos diseñados en el segundo capítulo, en la prueba de mayor trascendencia que se realizó las bases de datos analógicas y digitales se llenaron de valores aleatorios generados por un esclavo Modbus, pero la prueba de total validez no se pudo realizar, instalar el sistema desarrollado en un hotel y recolectar datos reales de cada uno de los *chillers* activos para realizar ajustes en las direcciones Modbus de las variables, es decir, verificar que los datos que corresponden a una determinada variable se guarden en la columna que ocupa dicha variable y no en otra puesto que puede ocurrir que la dirección Modbus de una variable esté intercambiada con otra.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

El proyecto realizado como parte de este Trabajo de Diploma permitió arribar a las siguientes conclusiones:

1 Existe una tendencia al aumento de aplicaciones dedicadas sobre sistemas empotrados a nivel mundial.

2 El empleo de un dispositivo empotrado fabricado en Cuba y la implementación de los módulos con herramientas de software libre contribuyen a la independencia tecnológica.

3 La solución desarrollada amplía el número de aplicaciones existentes hasta el momento para la computadora en una tarjeta MBTI2440.

4 Las pruebas realizadas demuestran que los módulos cumplen con las funcionalidades requeridas por la aplicación, pero no se puede asegurar la completa integridad de la misma hasta que no se efectúen pruebas con datos reales de comportamiento de las variables de los *chillers*.

Recomendaciones

1 Implementar el módulo de control según reglas establecidas por los especialistas en sistemas de clima centralizado que permita la secuenciación, rotación y cambio de *setpoints* horarios para cada una de las enfriadoras conectadas al sistema.

2 Realizar pruebas a los módulos creados en una instalación hotelera, es decir, conectar la MBTI2440 al *Gateway* de los *chillers* y poner en funcionamiento la aplicación para recolectar datos reales.

3 Completar la documentación del software para facilitar el mantenimiento del mismo y para que sirva como un punto de partida en nuevos diseños.

4 Divulgar los resultados obtenidos para generar el interés de otras empresas en el proyecto.

REFERENCIAS BIBLIOGRÁFICAS

Digia Qt. (2013). "Página oficial de Qt". Consultado en http://qt.digia.com/Product/Qt-SDK/, accedido el 29 de mayo.

González, E. L., y F. G. Vázquez. (2011). Ambiente de desarrollo y virtualización de aplicaciones para Sistemas Linux Empotrados. CIE2011.

González, F., M. Herrera y R. González. (2011). *Procesamiento intensivo de datos en aplicaciones empotradas con restricciones de tiempo real*. Universidad Central Marta Abreu de Las Villas.

Hallinan, C. (2006). *Embedded Linux*. Prentice Hall.

Hensgen, P. (2003). *Manual de Umbrello UML Modeller*. Disponible en http://docs.kde.org/stable/es/kdesdk/umbrello/, accedido el 28 de mayo de 2013.

Jiménez, D. (2009). *Manual de Usuario de la computadora en una tarjeta MBTI2440*. ICID, La Habana.

Junyan, L., X. Shiguo y L. Yijie. (2009). "Application research of embedded database SQLite", *Information Technology and Applications, 2009. IFITA'09.* En International Forum, 2:539-543. IEEE.

libmodbus Manual Page. (2013). *libmodbus: A Modbus library for Linux, Mac OS X, FreeBSD, QNX and Win32*.Disponible en http://www.libmodbus.org/, accedido el 30 de mayo.

Montesino, M. (2005). *Gestión de la Energía en hoteles turísticos en Cuba*. Informe Técnico, Centro de Estudios de Energía y Medio Ambiente, Universidad de Cienfuegos, Cuba.

Owens, M. (2006). *The definitive guide to SQLite*. Apress.

Palacios, H. (2003). "Windows CE: el windows más pequeño que existe". *Revista española de electrónica* (581): 50-51.

Pérez, D. A. (2009). *Sistemas embebidos y sistemas operativos embebidos*. Universidad Central de Venezuela, Caracas.

Petazzoni, T., y M. Opdenacker. (2009). *Cross-compiling toolchains*. Free Electrons.

Pout, Christine, L. Perez-Lombard y J. Ortiz. (2008). "A review on buildings energy consumption information". *Energy and buildings* 40 (3): 394-398.

Puente, E., D. Hernández, M. Hurtado y J. E. Torres. (2012). "Nova como sistema operativo embebido para hardware cubano", *Revista Cubana de Ciencias Informáticas*. Consultado en http://rcci.uci.cu.

Raghavan, P., A. Lad y S. Neelakandan. (2006). *Embedded Linux system design and development*. Auerbach Pub.

Rodríguez, A. (2013). "Modularidad: Un factor vital para la calidad del software". *Revista Ciencias*. Consultado en http://www.revistaciencias.com/publicaciones/, accedido el 8 de mayo.

Sancho, I. (2012). *Unidad de hardware para un sistema de control supervisor y de adquisición de datos empotrado utilizando herramientas de software libre*. Instituto Tecnológico de Costa Rica.

Schmuller, J. (2000). *Aprendiendo UML en 24 horas*. Prentice Hall.

Torres, R. (2009). *Diseño de sistema empotrado para la optimización de máquinas enfriadoras en sistemas de clima centralizado*. Universidad Central Marta Abreu de Las Villas.

Vahid, F. y T. Givargis. (2002). *Embedded System Desing: A Unified Hardware/Software Introduction*. John Wiley & Son.

Vidal, M. y J. E. Manrique. (2012). *Ingeniería y gestión de software*. Ciencias Médicas, La Habana.

Yaghmour, K. (2003). *Building Embedded Linux Systems*. O´Reilly.

ANEXOS

Anexo I. Guía para compilar la biblioteca Qt para la MBTI2440.

1. Instalar el sdk de qt (apt-get install qtcreator)
2. Instalar los esenciales de desarrollo apt-get install build-essential)
3. Descargar biblioteca qt y descomprimirla

Entrar al directorio desde la consola y poner

$export PATH=/camino/a/tu/compilador/cruzado/bin/:$PATH en este caso es el arm-none-linux-gnueabi

4. Entrar a mkspecs/qws/linux-arm-gnueabi-g++/ y editar qmake.conf

include(../../common/linux.conf)

include(../../common/gcc-base-unix.conf)

include(../../common/g++-unix.conf)

include(../../common/qws.conf)

Modificaciones para g++.conf

QMAKE_CC = arm-none-linux-gnueabi-gcc -msoft-float -D_GCC_FLOAT_NOT_NEEDED -march=armv4t -mtune=arm920t -O0 -lts

QMAKE_CXX = arm-none-linux-gnueabi-g++ -msoft-float -D_GCC_FLOAT_NOT_NEEDED -march=armv4t -mtune=arm920t -O0 -lts

QMAKE_LINK = arm-none-linux-gnueabi-g++ -msoft-float -D_GCC_FLOAT_NOT_NEEDED -march=armv4t -mtune=arm920t -O0 -lts

```
QMAKE_LINK_SHLIB    =    arm-none-linux-gnueabi-g++    -msoft-float    -
D_GCC_FLOAT_NOT_NEEDED -march=armv4t -mtune=arm920t -O0 -lts
```

Modificaciones para linux.conf

```
QMAKE_AR                = arm-none-linux-gnueabi-ar cqs

QMAKE_OBJCOPY           = arm-none-linux-gnueabi-objcopy

QMAKE_STRIP             = arm-none-linux-gnueabi-strip

QMAKE_RANLIB            = arm-none-linux-gnueabi-ranlib
```

5. Configurar las bibliotecas de qt como sigue (esto depende de las necesidades, se pueden quitar bibliotecas y/o poner más)

```
./configure -prefix /usr/local/mbti2440/qt-4.8.4 -embedded arm -
opensource -confirm-license -release -shared -fast -plugin-gfx-linuxfb -
plugin-gfx-transformed -no-webkit -qt-kbd-tty -qt-kbd-linuxinput -plugin-
mouse-pc -plugin-mouse-linuxtp -qt-mouse-linuxinput -qt-mouse-tslib -
declarative -qt-sql-sqlite -qt-zlib -qt-libpng -xplatform qws/linux-arm-
gnueabi-g++ -no-glib -force-pkg-config -nomake demos -nomake examples -
no-freetype -no-qt3support -no-largefile -no-xshape -no-separate-debug-
info -no-qvfb -no-gfx-qvfb -no-kbd-qvfb -no-xfixes -no-xrandr -no-xrender
```

6. Compilar el qt con el comando $make
7. Instalar las bibliotecas de qt con $sudo make install
8. Instalar las bibliotecas y fuentes de qt en la MBTI2440 copiar en /usr/local/ en la sd card la carpeta /usr/local/qt-mbti-4.8.1 del *host* y copiar en /usr/local/ de la sd la carpeta /usr/local/tslib-arm/ del host

Anexo II. Tabla de variables a recolectar de los *chillers*

Id	Variable	Descripción	Comentario	Tipo
1	plcWatch	Reloj del PLC	Hora militar	1
2	waterTempIn	Temperatura de entrada del Agua	en grados celcius	1
3	waterTempOut	Temperatura de salida del Agua	en grados celcius	1
4	envirommentTemp	Temperatura ambiente	en grados celcius	1
5	recoveryTemp	Temperatura de recuperacion	en grados celcius	1
6	compStage1	Etapa de compresor 1	Capacidades comp. 1 - 50%, 2 - 75%	1
7	compStage2	Etapa de compresor 2	Capacidades comp. 1 - 50%, 2 - 75%	1
8	compStage3	Etapa de compresor 3	Capacidades comp. 1 - 50%, 2 - 75%	1
9	compStage4	Etapa de compresor 4	Capacidades comp. 1 - 50%, 2 - 75%	1
10	waterTempSP	SetPoint de la temperatura del agua	en grados celcius	1
11	downPressureComp1	Presion de descarga del compresor 1	en bar	1
12	upPressureComp1	Presion de succion del compresor 1	en bar	1
13	oilPressureComp1	Presion de aceite del compresor 1	en bar	1
14	downPressureComp2	Presion de descarga del compresor 2	en bar	1
15	upPressureComp2	Presion de succion del compresor 2	en bar	1
16	oilPressureComp2	Presion de aceite del compresor 2	en bar	1
17	downPressureComp3	Presion de descarga del compresor 3	en bar	1
18	upPressureComp3	Presion de succion del compresor 3	en bar	1
19	oilPressureComp3	Presion de aceite del compresor 3	en bar	1
20	downPressureComp4	Presion de descarga del compresor 4	en bar	1
21	upPressureComp4	Presion de succion del compresor 4	en bar	1
22	oilPressureComp4	Presion de aceite del compresor 4	en bar	1
23	startHour	Hora de inicio (lectura / escritura)	Para la programacion diaria (24 h)	1
24	startMinute	Minutos de inicio (lectura / escritura)	Para la programacion diaria	1
25	stopHour	Hora final (lectura / escritura)	Para la programacion diaria (24 h)	1
26	stopMinute	Minutos finales (lectura / escritura)	Para la programacion diaria	1
27	waterTempSP1	Set Point 1 de la temp del agua	en grados celcius	1
28	waterTempSP2	Set Point 2 de la temp del agua	en grados celcius	1
29	analogDateTime	Fecha y Hora	Hora militar	1
30	chillerState	Estado de la enfriadora	0 - Apagada / 1 - Encendida	0
31	generalAlarm	Alarma general	0 - no alarma / 1 - alarma	0
32	stateOfComp1	Estado del Compresor 1	0 - Apagado / 1 - Encendido	0
33	alarmOfComp1	Alarma del Compresor 1	0 - no alarma / 1 - alarma	0
34	maintainOfComp1	Mantenimiento del Compresor 1	0 - no mantenimiento / 1 - mant.	0
35	stateOfComp2	Estado del Compresor 2	0 - Apagado / 1 - Encendido	0
36	alarmOfComp2	Alarma del Compresor 2	0 - no alarma / 1 - alarma	0
37	maintainOfComp2	Mantenimiento del Compresor 2	0 - no mantenimiento / 1 - mant.	0
38	stateOfComp3	Estado del Compresor 3	0 - Apagado / 1 - Encendido	0
39	alarmOfComp3	Alarma del Compresor 3	0 - no alarma / 1 - alarma	0
40	maintainOfComp3	Mantenimiento del Compresor 3	0 - no mantenimiento / 1 - mant.	0
41	stateOfComp4	Estado del Compresor 4	0 - Apagado / 1 - Encendido	0
42	alarmOfComp4	Alarma del Compresor 4	0 - no alarma / 1 - alarma	0
43	maintainOfComp4	Mantenimiento del Compresor 4	0 - no mantenimiento / 1 - mant.	0
44	stopStart	Paro / Marcha	0 - Apagada / 1 - Encendida	0
45	spcSS	Cambio de setPoint/arranque y paro	0-cambio de SP/1-arranque y paro	0
46	digitalDateTime	Fecha y Hora	Hora militar	0