Universidad de las Ciencias Informáticas

MÓDULO BASE DE DATOS HISTÓRICOS PARA EL SISTEMA AREX

Tesis propuesta para obtener el título de Ingeniero en Ciencias Informáticas

Autor: Randy Mujica Díaz

Tutor: Dr.C Antonio Cedeño Pozo

La Habana, 2015

Declaro que soy el único autor de este trabajo y autorizo al Centro de Informática Industrial de la Universidad de las Ciencias Informáticas a que haga uso del mismo.

Para que así conste, firmo la presente a los _____ días del mes de _____ del año _____.

Randy Mujica Díaz

_____

Firma del autor

Dr.C Antonio Cedeño Pozo

_____

Firma del tutor

AGRADECIMIENTOS

Este trabajo, y más que este trabajo esta carrera no habría sido posible sin el apoyo y estímulo incondicional de mi abuelita querida Blanca de las Nieves Pardo que más que mi abuela ha sido mi madre desde que existo. Igualmente sin el apoyo incondicional de mi madre, mi padre y mi padrastro nada de esto hubiera sido realidad. Sin el apoyo de mi tutor y amigo, el Dr.C Antonio Cedeño Pozo, bajo cuya supervisión comencé y terminé esta tesis. Agradezco especialmente a mis amigos Álvaro Denis por ayudarme mayormente con temas de uso avanzado de Linux, compilación, patrones y formas de hacer las cosas además de inculcarme siempre el aprendizaje de nuevas tecnologías, también a mi amigo Rolando Morales por ayudarme con el documento y trabajar juntos en nuestras tesis. Gracias a la UCI y a la Revolución por la oportunidad que nos brinda a todos por igual, dado que un joven humilde como yo nunca hubiera podido aspirar a este privilegio.

DEDICATORIA

Dedico este trabajo especialmente a mi abuela Blanca de las Nieves Pardo. A mi madre, mi padre y mi padrastro. A toda mi familia y amigos.

SÍNTESIS

En el presente trabajo se desarrolló un módulo servidor de base de datos históricos. AREX por ser actualmente un sistema de medición no cuenta con dicho componente. Según la bibliografía consultada, no existe disponible un componente de versión libre que pueda ser integrado para realizar dicha tarea. Por lo cual surge la necesidad de desarrollar dicho módulo, que además realice la tarea necesaria de manera eficiente. La solución propuesta abarca la creación de dos submódulos bajo una arquitectura cliente/servidor y un protocolo del nivel de aplicación del modelo OSI para lograr la comunicación. El submódulo BDH-Server obtiene la información del componente Recolector mediante Ethernet, la almacena de manera local eficientemente, y la provee bajo el protocolo mencionado a todos los componentes del sistema que deseen consultarla como es el caso del submódulo BDH-Client, el cual puede obtener información tanto en tiempo real, como históricamente persistida, la cual es mostrada en gráficos de tendencia para poder ser analizada. A la solución desarrollada se le realizaron un conjunto de pruebas de aceptación y rendimiento; los resultados de dichas pruebas mostraron que el sistema realiza la tarea de manera eficiente y cumple con las expectativas del cliente de la solución.

Palabras claves: BDH, servidor, cliente, protocolo, SCADA, tendencia.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

A pesar de que la informática es una ciencia joven, en los últimos años ha alcanzado avances que han contribuido considerablemente al desarrollo económico de muchos países. Un sector en el cual la informática ha tenido una amplia repercusión es el industrial, llegando a existir en la actualidad aplicaciones de considerable envergadura que se encargan de monitorear y controlar procesos altamente complejos. Estas aplicaciones son conocidas como sistemas SCADA (en inglés, Supervisory Control And Data Acquisition)(Penin, 2007).

El Estado cubano está destinando una gran cantidad de recursos en temas de automatización de procesos. En ese sentido se realizan licitaciones de soluciones integrales en el extranjero para suplir las necesidades tecnológicas, incurriendo en considerables gastos. El SCADA desarrollado en la Universidad de las Ciencias Informáticas (UCI), específicamente en el Centro de Informática Industrial (CEDIN) y otros sistemas de este tipo de producción nacional, como el EROS, desarrollado por el grupo SERCONI de la provincia Holguín, están llamados a sustituir las importaciones de este tipo de tecnología. El despliegue de estos sistemas resulta costoso por los recursos de hardware necesarios para su implantación.

Existe un importante número de escenarios en los que se realiza una automatización básica, en la que intervienen un pequeño número de variables[1] y las actividades de control son poco complejas o inexistentes y solo se requiere el monitoreo de dichas variables. En consecuencia, la implantación de los sistemas mencionados anteriormente, implicaría un gasto innecesario de recursos debido a la subutilización de los mismos. Como ejemplo de estos escenarios se pueden mencionar algunos sectores del turismo, de la industria azucarera, así como un sinnúmero de pequeñas y medianas entidades que requieran algún tipo de automatización en procesos poco complejos, al igual que el área de la domótica referente a la automatización del hogar.

[1] Características a medir de determinado proceso.

Con el objetivo de dar una solución eficiente en los contextos mencionados anteriormente, en el departamento Desarrollo de Aplicaciones del CEDIN se desarrolla un sistema de medición de procesos (SM) llamado AREX, el cual corre sobre dispositivos en los que pueda ser embebido un sistema operativo basado en Linux[2] y la biblioteca Qt[3]. Hasta el momento dicho sistema solo permite el monitoreo de procesos de pequeña y mediana complejidad hasta un máximo de 100 variables. Se pretende que dicho sistema evolucione a un sistema de aquisición de datos (SAD) con lo cual se podrán almacenar los datos recibidos y/o generados en el núcleo de procesamiento en objetivo de proveerlos para futuros análisis. El componente que marca la evolución de un SM a un SAD es comúnmente llamado Base de datos históricos (BDH) el cual es un módulo predeterminado en este tipo de sistemas al igual que en sistemas integrales como los SCADA. Es necesario mencionar que existen en el CEDIN otros BDH pertenecientes a sistemas como el SCADA GALBA[4], pero los mismos están desarrollados para ejecutarse sobre hardware con amplias prestaciones, siendo poco probable que puedan ser integrados al sistema AREX pues este último está diseñado para utilizar hardware de bajas prestaciones. Además, en la bibliografía consultada no se encontró ningún módulo BDH que cumpla con los principios del software libre y al mismo tiempo con las especificaciones de software y hardware requeridas por el sistema AREX.

Ante la problemática enunciada se arriba al siguiente **problema de investigación**: ¿Cómo lograr el almacenamiento y acceso a los datos recibidos en el sistema AREX de manera eficiente?

Se define como **objeto de estudio**: Base de datos históricos para sistemas de adquicisión de datos.

Mientras que el **campo de acción** se enmarca en: Base de datos históricos para sistemas de adquicisión de datos sobre dispositivos con bajas prestaciones.

Para dar respuesta al problema de la investigación se define como **objetivo general**: Desarrollar un módulo BDH para el sistema AREX que permita realizar almacenamiento y acceso a los datos de manera eficiente.

Para dar cumplimiento al objetivo se derivan las siguientes tareas de investigación:

[2] Kernel de un sistema operativo desarrollado por Linus Torvalds.
[3] Biblioteca multiplataforma para el desarrollo de aplicaciones con interfaz gráfica.
[4] SCADA Guardián del ALBA desarrollado en el CEDIN.

1. Elaboración del marco teórico de la investigación referente a BDH para sistemas de adquisición de datos (SAD).
2. Selección de las tecnologías y herramientas adecuadas para el desarrollo.
    - Sistema operativo.
    - Lenguaje de programación.
    - Framework de desarrollo.
    - Entorno de desarrollo integrado.
    - Lenguaje de modelado.
    - Sistema gestor de base de datos.
    - Mecanismo de comunicación entre procesos.
    - Framework de serialización.
    - Herramienta de modelado.
    - Metodología de desarrollo de software.
3. Definición de la propuesta de solución.
4. Implementación de un prototipo no funcional del BDH.
5. Diseño e implementación del BDH .
6. Diseño de pruebas de software.
    - Pruebas de aceptación.
    - Pruebas de rendimiento.
7. Evaluación de los resultados de las pruebas de software.

Los métodos de investigación presentes en este trabajo son los siguientes:

- **Método de observación.** Con el objetivo de observar cómo se comportan las salidas y el rendimiento del sistema ante la ejecución del servidor BDH y la aplicación cliente.
- **Método de análisis histórico y lógico.** Con el objetivo de hacer un recorrido por la evolución de los sistemas desde los SM hasta los SAD, haciendo especial hincapié en los rasgos que marcan su evolución.
- **Método de análisis y síntesis.** Para realizar el análisis de la literatura consultada y lograr obtener de forma sintetizada información referente a los módulos BDH existentes.

- **Método de la modelación.** Con el objetivo de representar conceptos de la vida real a través de la abstracción, para modelar los elementos del dominio y arquitectura del módulo BDH a desarrollar.

Como **posible resultado** del trabajo de diploma se obtendrá un módulo BDH que pueda ser integrado al sistema AREX y se pueda ejecutar de manera eficiente en cualquier dispositivo con con Linux y Qt embebidos.El mismo se encargará de almacenar la información recibida en el núcleo de procesamiento y proveer un servicio TCP/IP[5] mediante su propio protocolo de comunicación, dicho protocolo de comunicación será otro resultado de la tesis el cual permitirá la comunicación de otras aplicaciones con el servidor BDH mediante Ethernet[6].

La estructura del trabajo es la siguiente:

- **Capítulo 1:** Fundamentación teórica. En este capítulo se realiza una comparación entre sistemas de medición (SM) y sistemas de adquisición de datos (SAD) con enfoque en el sistema AREX y cómo este puede evolucionar, así como la evaluación de las tecnologías y herramientas a utilizar.

- **Capítulo 2:** Diseño e implementación. En este capítulo se brinda una panorámica de la propuesta de solución, así como la modelación e implementación de la misma.

- **Capítulo 3:** Diseño y realización de pruebas. En este capítulo quedan registrados los casos de prueba realizados al sistema y los resultados de los mismos como validación de la solución.

[5]Familia de protocolos de Internet creados por DARPA en 1970.
[6]Estándar de la IEEE 802.3 para la trasmisión de datos en redes de área local.

Capítulo 1

FUNDAMENTACIÓN TEÓRICA

Entre los temas abordados en este capítulo están la profundización en sistemas de medición (SM) y sistemas de adquisición de datos (SAD), teniendo en cuenta que estos últimos solo cuentan con un subconjunto de los componentes de un SCADA. Se describen sus principales componentes tanto de software como de hardware, ventajas y desventajas y se mencionan diferentes sistemas de este tipo tanto de producción nacional como internacional. Además se realiza un enfoque al sistema AREX como objetivo fundamental del trabajo de diploma. Por último, se recogen las tecnologías y las herramientas a utilizar para desarrollar la solución.

1.1. Sistemas de medición (SM)

De manera general un Sistema de Medición(SM) es el grupo de instrumentos, calibres, estándares, operaciones, métodos, dispositivos, software, personal, medio ambiente y supuestos utilizados para cuantificar una unidad de medida (Reyes Aguilar, 2007). Es el proceso completo utilizado para obtener mediciones. De estas definiciones se puede decir que un proceso de medición puede ser visto como un proceso de manufactura que produce números (datos) para sus producciones (Reyes Aguilar, 2007).

En el área de la automatización de procesos estos sistemas amplían las capacidades de interacción con el medio de los usuarios, ya sea este: una planta industrial, un laboratorio, una tormenta tropical, un terremoto, el espacio exterior, el medio ambiente, etc. Entonces se puede afirmar que: se han ampliado la capacidad de los sentidos humanos gracias a los sistemas de medición. Un requerimiento básico de estos es su calibración, ya que con el tiempo sus propiedades se van modificando y es preciso evaluarlas y emprender acciones que permitan mantenerlas en el rango admisible de operación(Fernández Sánchez, 2009b).

El objetivo de un SM es indicar el valor de una o varias variables. Para ello toma del medio medido la información y la transforma en una cantidad obtenida a través de un sensor (Fernández Sánchez, 2009b).

Se puede relacionar entonces los componentes de estos sistemas de la forma siguiente:

- Sensor[1].
- Transmisor.
- Medio de transmisión.
- Indicador[2].

El sensor convierte el valor de la magnitud de interés a un tipo de señal que luego es trasmitida por el transmisor a través del medio de trasmisión para ser traducida por los indicadores a un formato que puede ser apreciado por los sentidos humanos.

Un ejemplo de SM es:

Monitoreo remoto de aerogeneradores:

T&M Solutions[3] y ECN Wind Energy[4] eligieron LabVIEW[5] y NI(en inglés, National Instruments)[6] con el hardware de CompactRIO[7] para reemplazar los sistemas de medición instalados en aerogeneradores remotos que no podrían ser reemplazados o reparados fácilmente. Utilizando las herramientas de NI, pudieron desarrollar sistemas fiables con la configuración del módulo de E/S[8] y una configuración del software de manera flexible. Los sistemas RIO adquieren las mediciones analógicas, las convierten a señales digitales, aplican una marca de tiempo, almacenan temporalmente los datos para evitar pérdidas, y transmiten datos a un servidor de base de datos, (Ver Figura 1.1).

[1]Dispositivo que obtiene el valor de una magnitud del ambiente.
[2]Dispositivo para mostrar la información al usuario.
[3]Empresa especializada en el desarrollo de plataformas de software integrado para sistemas de control de fabricación aditiva.
[4]Empresa especializada en innovación energética.
[5]Herramienta para el diseño de sistemas con el lenguaje de programación visual de National Instruments.
[6]Empresa que desarrolla sistemas de adquisición.
[7]Producto de NI.
[8]Módulo de entrada/salida.

Figura 1.1: Monitoreo remoto de aerogeneradores.

1.2. Sistemas de adquisición de datos (SAD)

De manera general un Sistema de Aquisición de Datos (SAD) es un conjunto de recursos de hardware y software diseñado para calcular la representación interna y ofrecer al usuario la representación externa(Merriam-Webster, 2000).

Por otro lado, un SAD es un conjunto de recursos de hardware y de software que proporciona los medios para obtener conocimiento de un sistema, proporciona los medios para acceder a los datos del sistema, convierte los datos del sistema a información del sistema más útil y distribuye esa información al usuario(Nieva y Wegmann, 2000).

De otra manera, en el contexto de la automatización de procesos, un SAD es la evolución de un SM; que además de sus prestaciones ahora tendrá la posibilidad de almacenar la información que captura además de mostrarla al usuario, el componente que realiza dicha tarea es llamado **Registrador**. Se puede mencionar entonces que los elementos que componen a un SAD son los siguientes(Fernández Sánchez, 2009a):

- Sensor.
- Transmisor.
- Medio de transmisión.
- Indicador.
- Registrador.

Un **Registrador** es un componente de hardware o software que ofrece al sistema la capacidad de almacenar la información de manera histórica en algún medio de

almacenamiento para su posterior análisis(Fernández Sánchez, 2009a).

Categorías de Registradores (Fernández Sánchez, 2009a):

- Basado en hardware
  - Con papel.
  - Sin papel.
  - Multicanal.
  - Con conexión a OPC.
- Basado en software (Base de datos históricos).

1.2.1. Etapas en un SAD

Según (A.H. y M.M, 2013) las etapas de un SAD se pueden describir de la manera siguiente:

- **Medición:** se logra con diferentes tipos de sensores, los cuales miden aspectos distintos de parámetros tales como temperatura, presión, aceleración, movimiento lineal, frecuencia e interruptor ON/OFF[9](A.H. y M.M, 2013).
- **Almacenamiento:** se logra convirtiendo las mediciones del sensor en una señal digital mediante un convertidor A/D[10]. Se definen tiempos de muestreo, los cuales dan una medida del período con que se obtendrán las mediciones de los sensores(A.H. y M.M, 2013).
- **Acceso:** el acceso a los canales monitorizados y la frecuencia de muestreo determinan cuánto se tardará en almacenar los datos en la memoria así como la cantidad de datos a almacenar. Una vez que se han almacenado los datos entonces debe accederse para realizar análisis con estos en objetivo de entender comportamientos y tendencias(A.H. y M.M, 2013).
- **Análisis:** un programa de aplicación para el análisis usa los datos del proceso almacenados, y se los muestra al usuario desde diferentes perspectivas, que le permite sacar conclusiones, generalidades y conocimiento de esos datos(A.H. y M.M, 2013).

[9]Control de encendido/apagado.
[10]Convertidor de valores analógicos a digitales y viceversa.

A continuación se presenta un ejemplo de SAD:
MIDAS: MIDAS (en inglés, Maximum Integrated Data Acquisition System) ha sido desarrollado como un sistema de propósito general para los experimentos de pequeña y mediana escala, desarrollado por Stefan Ritt en 1993 y seguido por Pierre-André Amaudruz en 1996. Está escrito en C y publicado bajo la licencia GPL(Ritt, 2015).

El complejo experimento abarca desde sistemas de prueba, donde un solo PC está conectado a CAMAC(en inglés, Computer Automated Measurement And Control) través de una interfaz de PC-CAMAC hasta varias PC en el nivel de front-end con varios nodos de análisis. El sistema se ejecuta actualmente en Linux, MS Windows, varias versiones de UNIX, VMS, VxWorks y MS-DOS y puede ser portado fácilmente a prácticamente cualquier sistema operativo que soporta sockets TCP/IP(Ritt, 2015), (Ver Figura 1.2).

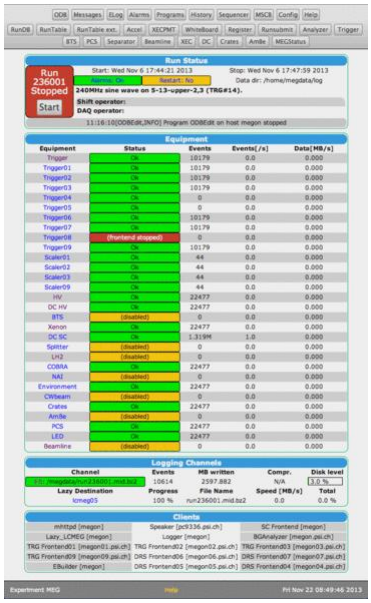

Figura 1.2: Sistema de adquisición de datos MIDAS.

1.2.2. Base de datos históricos (BDH)

Un módulo BDH(Base de Datos Históricos) es un tipo de base de datos diseñada para archivar datos de la automatización de determinado proceso. Los módulos BDH están diseñados para almacenar datos de alta frecuencia y datos sobre una base regular. Además, estos se utilizan para solucionar problemas de procesos, optimizar la fabricación, almacenar datos para cumplimiento de normas, etc y no

están diseñados para almacenar datos transaccionales o relacionales, éste es el papel de una base de datos relacional(Subnet, 2015).

En el enfoque distribuido de los SAD, la recepción de información desde los niveles de campo hasta los niveles superiores, se perfila como uno de los servicios más utilizados, resaltándose la captura y visualización en tiempo real en las consolas de operación. Sin embargo en los sistemas donde se requiera un análisis de la información histórica capturada por los dispositivos de campo, así como de la sucesión de alarmas[11] y eventos generados, se necesita un mecanismo que permita almacenar esos datos, partiendo de una configuración previa realizada por parte de los administradores del sistema(De la Horra Diaz, 2011).

La información histórica almacenada es utilizada por una serie de aplicaciones entre las cuales se destacan los servicios gerenciales como la gestión de producción, mantenimiento y control utilizando algoritmos inteligentes predictivos y adaptativos. La utilidad más inmediata es la generación de reportes por parte de los operadores y usuarios, además del monitoreo en tiempo real de las tendencias y análisis de las mismas(De la Horra Diaz, 2011).

Tipos de históricos

Existen varios tipos de históricos en la literatura, los más difundidos son los mencionados a continuación(De la Horra Diaz, 2011):

- Servicio de históricos de variables (puntos).
- Servicio de histórico de alarmas.
- Servicio de histórico de eventos.
- Servicio de bitácora.

Los servicios de históricos de variables. Denominados por algunos autores Data Loggers, se especializan en el manejo de los históricos de las variables del sistema, sean estas variables de campo, memoria o calculadas. Comúnmente este tipo de históricos se configuran para enviar al medio persistente el estado de una variable periódicamente o por excepción. El término por excepción indica que la variable se envía a los históricos cuando se cumple alguna condición que permite decidir en qué instante debe almacenarse. Por otro lado este servicio provee a los clientes la información de las variables en forma de series temporales, opcionalmente

[11] Advertencia de un comportamiento inadecuado del sistema.

sumarizadas por intervalos de tiempo(De la Horra Diaz, 2011).

Los servicios de históricos de alarmas, se especializan en el manejo de los históricos de eventos anormales conocidos como alarmas. Las alarmas identifican eventos que indican mal funcionamiento del sistema, los cuales podrían conllevar a catástrofes. Las alarmas más comunes son asociadas a las variables del sistema, sean estas variables de campo o calculadas, dentro de estos tipos de alarmas se pueden mencionar, alarmas de nivel, tasa de cambio, entre otras. Por otro lado pueden existir alarmas asociadas a los dispositivos de campo del sistema (por ejemplo, la falla de un PLC, que contiene la información de variables del sistema) y alarmas de sistema (como podría ser la caída de una red o el fallo catastrófico de recursos de hardware)(De la Horra Diaz, 2011).

Los históricos de alarmas pueden ser divididos en tres tipos fundamentales: los históricos de ocurrencia de alarmas, los históricos relativos al tratamiento de las alarmas por parte de los operadores (reconocimiento, etc) y los históricos de ayuda para la corrección de alarmas(De la Horra Diaz, 2011).

Los servicios de históricos de eventos, se especializan en el manejo de los eventos que no son considerados catastróficos, pero que tienen impacto en el funcionamiento del sistema y que podrían tornarlo inestable en el futuro. Dentro de estos se pueden mencionar autenticaciones, fallas en la ejecución de un comando, etc. Estos permiten las auditorias de los diferentes servicios del sistema.

Los servicios de bitácora, están orientados principalmente al almacenamiento de la información considerada relevante por los operadores, relativa a eventos que sucedan en su turno de trabajo(De la Horra Diaz, 2011). Estos servicios permiten a los operarios documentar situaciones de operación que podría ser interesante consultar por él o por los demás operadores en un futuro. Ejemplo de ello son las soluciones a problemas que se le presentaron en determinadas circunstancias de operación del sistema(De la Horra Diaz, 2011).

Flujo de información en los históricos

En la Figura 1.3 se muestra el esquema de comunicación entre los diferentes módulos que intervienen en el proceso de recolección y archivo de históricos. Muestra un esquema que relaciona los diferentes módulos que intervienen en el flujo de un módulo BDH. En ella se marcan cinco niveles, especializados en funcionalidades que van desde la captura de la información hasta la persistencia de los datos(De la

Horra Diaz, 2011).

- **Nivel de Campo**. Está compuesto por dispositivos de campo como controladores lógico-programables(PLC)., unidad de terminal remota(RTU), sensores, etc) que contienen la información de proceso que requiere ser registrada para un análisis en tiempo real o histórico(De la Horra Diaz, 2011).
- **Nivel de Recolección**. Se encarga de captar los datos del nivel de campo, respetando una lógica temporal o por eventos que es asociada a cada información requerida(De la Horra Diaz, 2011).
- **Nivel de Base de Datos de Tiempo Real(BDTR)**. Se encarga de recolectar los datos desde el nivel de recolección, procesarlos y servirlos a las aplicaciones de tiempo real, como por ejemplo las tendencias. Además entrega los datos al nivel de manejo de históricos para garantizar la persistencia de los mismos. Este nivel comúnmente es redundante para garantizar la tolerancia ante fallos(De la Horra Diaz, 2011).
- **Nivel de Historiador**. Se encuentra toda la lógica de recepción y envío de datos hacia los medios persistentes, ya sea por tareas periódicas o por excepción. Además es el encargado de entregar información histórica a aplicaciones como reportes, tendencias, etc. Este nivel también puede considerar la redundancia para garantizar un servicio estable libre de fallos(De la Horra Diaz, 2011).
- **El último nivel** es el encargado de recibir la información del módulo BDH, y almacenarla de manera segura, utilizando, opcionalmente, servicios de replicación. En este nivel se acostumbra a utilizar servidores de bases de datos que permitan servicios de réplica, respaldo, etc(De la Horra Diaz, 2011).

1.3. Sistema de medición AREX

AREX es un sistema para medir procesos de baja y mediana complejidad (procesos que incluyen hasta 100 variables), a partir de la adquisición y procesamiento de datos en tiempo real asociados a las variables incidentes dentro del proceso. Está compuesto por: interfaz de edición o diseño, interfaz de tiempo de ejecución y recolección de datos que incluye el manejo de protocolos de comunicación y transferencia de datos por las principales interfaces de red. AREX está diseñado para ser ejecutado en dispositivos con recursos de hardware limitados, por tal motivo no incluye de forma directa la posibilidad de almacenar los datos por largos períodos

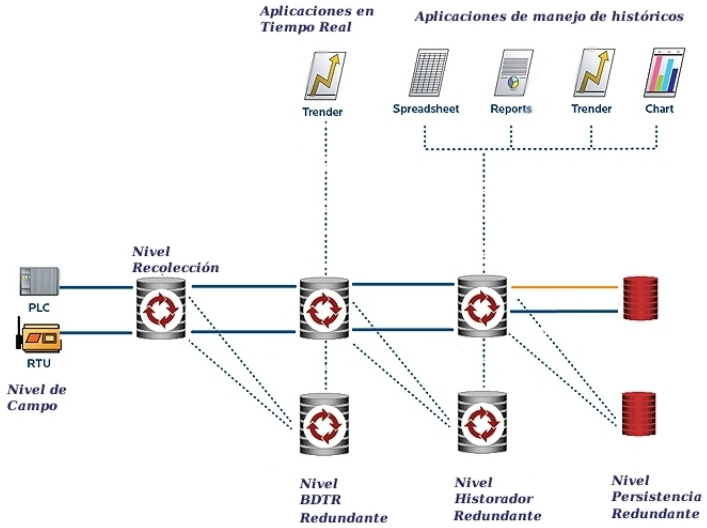

Figura 1.3: Flujo de información en históricos.

de tiempo. Sin embargo brinda las interfaces necesarias para incorporar este servicio en caso de que se cuente con el soporte físico para este fin. Está enfocado a servicios de la domótica, automatización de la industria azucarera, instituto de meteorología, en el sector corporativo y particular, para detectar estado de las luces y humedad de determinado lugar. El sistema permite comunicarse con los dispositivos de campo para medir el proceso de forma automática desde la pantalla del ordenador, que puede ser configurada y modificada por el usuario con facilidad(Delgado Alón y Jiménez López, 2012).

1.3.1. Hardware de AREX

AREX está formado por diferentes componentes de hardware entre los que se encuentran:

- Unidad de Terminal Maestra o MTU (por sus siglas en inglés).
- Unidad de Terminal Remota o RTU (por sus siglas en inglés).
- PC de edición de despliegues.
- Red de comunicación.
- Instrumentación de campo.

Unidad de Terminal Maestra (MTU)

El término MTU se refiere a los servidores y el software responsable para comunicarse con el equipo del campo (RTU), en estos se encuentra el software HMI-Ejecución. El

MTU es el *centro neurálgico* del sistema, y es el componente del cual el personal de operaciones se valdrá para monitorear el proceso. Una MTU a veces se llama HMI, (Interfaz Humano-Máquina)(CHACÓN y CASTRILLO, 2001). El MTU de AREX es una tarjeta que pueda empotrar un sistema operativo basado en Linux junto con la biblioteca Qt, actualmente se utiliza una Intel Atom, pero también se pretende utilizar una Rasberry-Pi o una CID-300/9. Las **funciones principales** de la MTU de AREX son:

- Recolección de datos de los RTU.
- Gráficos del equipamiento actualizado para reflejar datos del campo.
- Control ON/OFF.

Unidad de Terminal Remota (RTU)

Las unidades de terminales remotas(RTU, por sus siglas en inglés) consisten en una pequeña y robusta computadora que almacena datos y los transmite a la terminal maestra para que esta controle los instrumentos. Es una unidad independiente(stand-alone) de adquisición de datos. Su función es controlar el equipamiento de proceso en el sitio remoto, adquirir datos del mismo y transferirlos a la MTU. Uno de los tipos básicos de RTU son las de un solo módulo(single board), que contienen todas las entradas de datos en una sola tarjeta. Una RTU single board tiene normalmente E/S fijas y los modulares que tienen un módulo CPU separado y pueden tener otros módulos agregados, normalmente enchufándolos en una placa común (similar a una PC con una placa madre donde se montan procesador y periféricos)(MONTERO et al., 2004).

La RTU se conecta al equipo físicamente y lee los datos de estado, cómo los estados abierto/cerrado desde una válvula o un interruptor, lee las medidas como presión, flujo, voltaje o corriente. Por el equipo RTU se puede enviar señales de lectura/escritura. La RTU puede leer el estado de los datos digitales o medidas de datos analógicas y enviar comandos digitales de salida o puntos de ajuste analógicos, (Ver Figura 1.4). En AREX, el componente RTU es una tarjeta Arduino con microcontrolador ATMega.

Red de comunicación

La red de comunicación es el nivel que gestiona la información que los instrumentos de campo envían a la red de computadores del sistema. El tipo de bus utilizado en

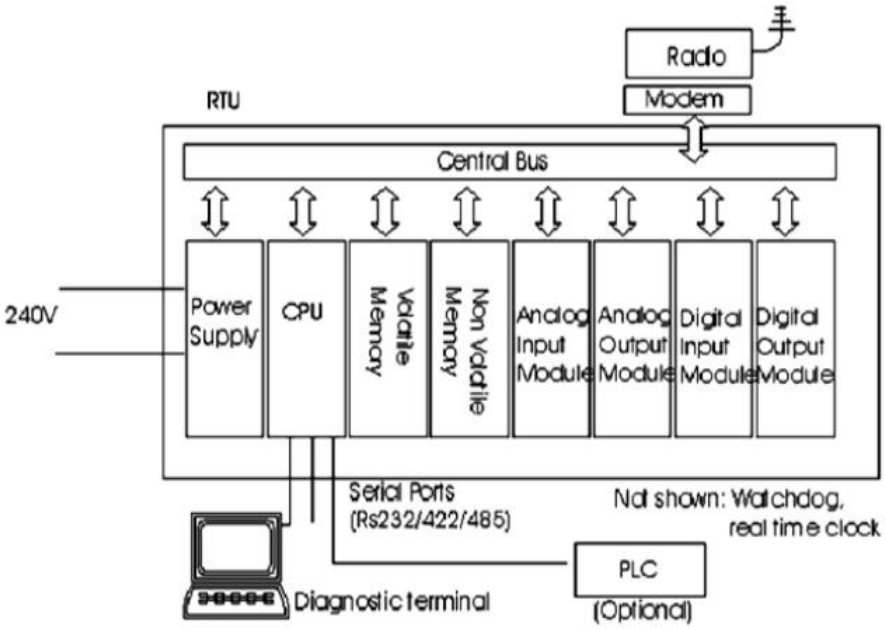

Figura 1.4: Diagrama de una RTU.

las comunicaciones puede ser muy variado según las necesidades del sistema y del software escogido para implementar el sistema. A partir del tipo de bus, el sistema provee interfaces de comunicación (Meza, 2007). En AREX, el tipo de bus utilizado implementa MODBUS[12] sobre los estándares RS-232[13] y Ethernet.

Instrumentación de campo

Los instrumentos de campo permiten realizar la captación de información del sistema (sensores). Los componentes son diseñados por distintos proveedores, sin coordinación entre sí. Así, se tienen diferentes proveedores para las RTU, sensores, módems, radios y minicomputadores. Actualmente en AREX se cuenta con sensores de luminosidad, temperatura, presencia, entre otros.

1.3.2. Software de AREX

Los módulos o bloques de software que permiten las actividades del sistema son (Delgado Alón y Jiménez López, 2012):

- **Interfaz gráfico del operador (HMI):** Proporciona al operador las funciones de monitoreo del proceso. El proceso se representa mediante sinópticos gráficos.

[12]Protocolo de comunicaciones del nivel siete del modelo OSI basado en una arquitectura maestro/esclavo.

[13]Estándar de comunicación por el puerto serie.

- **HMI-Edición:** Aplicación de escritorio con interfaz visual que puede ser ejecutada en una PC. La misma permite crear los despliegues del negocio y generarlos mediante un archivo XML[14].
- **HMI-Ejecución:** Aplicación que se ejecuta en la MTU, carga la configuración generada por el HMI-Edición (archivo XML) y muestra los correspondientes despliegues en la pantalla del MTU.

- **Driver:** Son los distintos manejadores que se ejecutan en la MTU y las RTU con el objetivo de establecer la comunicación entre estas. En AREX el driver implementa el protocolo industrial MODBUS sobre la comunicación por el puerto serie y Ethernet.

- **Recolector:** Aplicación que se ejecuta en la MTU u otro nodo especialmente dedicado para ella y se encarga de adquirir los datos del proceso generados en cada dispositivo RTU y proveerlos para los distintos módulos que los necesiten, por ejemplo los módulos HMI y BDH. Actualmente el Recolector de AREX implementa un planificador de tareas, dichas tareas tienen una prioridad y las mismas pueden ser tanto de escritura como de lectura, las tareas de escritura tienen prioridad sobre las de lectura. Además el Recolector expone la información a través de un servidor TCP.

1.4. Tecnologías y herramientas

A continuación se describen las tecnologías y herramientas a utilizar. Esta elección responde a múltiples razones, entre las que se encuentra el hecho de que las mismas son exponentes del movimiento de Software Libre, lo que contribuye al desarrollo del país y a la generalización de los principios y metas que propone este movimiento. Estas tecnologías y herramientas, además de poseer características que se ajustan a las necesidades de la presente investigación, son ampliamente utilizadas en Departamento de Desarrollo de Aplicaciones del CEDIN.

1.4.1. Sistema operativo

El sistema operativo Debian es una distribución de GNU/Linux creada por una comunidad de desarrolladores de igual nombre que sin ánimo de lucro lo provee libremente. Actualmente usan el kernel de Linux o el de FreeBSD (Debian, 2014). Debian es un sistema operativo bastante estable y configurable, pero sobre todo resalta por su bajo consumo de recursos. Gran parte de las herramientas básicas

[14]Lenguage de marcas extensible creado por W3C.

que completan el sistema operativo vienen del proyecto GNU; de ahí el nombre GNU/Linux(Debian, 2014). Teniendo en cuenta sobre todo el bajo consumo de recursos de Debian, fue el sistema seleccionado tanto para el desarrollo como para el despliegue de la solución propuesta.

1.4.2. Lenguaje de programación

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma (Stroustrup, 1994).

Es un lenguaje potente para el desarrollo de aplicaciones críticas o complejas, por la facilidad que brinda la manipulación de la memoria del ordenador a través de los punteros. Como lenguaje orientado a objetos brinda ventajas propias de esta filosofía de programación, como son: la abstracción, el encapsulamiento, la herencia y el polimorfismo. Lo que nos permite entre otras funcionalidades, poder representar un conjunto de características y comportamientos similares de objetos de la vida real y agruparlos en una unidad básica, brindando una interfaz para permitir la interacción con la misma. Las primeras aplicaciones tendieron a tener un fuerte componente de programación de sistemas. Por ejemplo, varios de los sistemas operativos más importantes se han escrito en C++ y muchos más poseen módulos claves escritos en este lenguaje (Stroustrup, 1994).

A pesar de que es un lenguaje orientado a objetos, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para la programación a bajo nivel. Esto nos permite utilizarlo para escribir controladores de dispositivos y otros software que se basen en la manipulación directa de hardware, bajo restricciones de tiempo real. C++ fue diseñado para que cada característica del lenguaje se pudiera utilizar en virtud de limitaciones severas de tiempo y espacio (Stroustrup, 1994).

Por el amplio conjunto de características y bondades que posee dicho lenguaje en el desarrollo de sistemas embebidos, se seleccionó de manera global en el proyecto AREX el uso del mismo.

1.4.3. Framework de desarrollo

Qt es un framework multiplataforma ampliamente usado para desarrollar aplicaciones que necesiten una interfaz gráfica de usuario amigable, así como programas de otra naturaleza. Se encuentra licenciado bajo los términos de la GNU Lesser General Public License (LGPL) versión 2.1. La API de Qt está implementada en C++, y ofrece funciones adicionales para facilitar el desarrollo multiplataforma. Existen adaptaciones o binding (en inglés) para otros lenguajes de programación como Java, C-Sharp, Python, Ada, Pascal, Perl, PHP, Ruby, entre otros. Qt va más allá de C++ en áreas como la comunicación entre objetos y la flexibilidad para el desarrollo avanzado de interfaces gráficas de usuario, añadiéndole las siguientes características a C++(Thelin, 2007):

- Poderoso mecanismo para la comunicación entre objetos conocido como: signals y slots.
- Facilidades para consultar y establecer las propiedades de un objeto.
- Poderoso filtro de eventos.
- Cadena de traducción contextual para la internacionalización.
- Punteros guardados que se igualan automáticamente a cero cuando el objeto se destruye, a diferencia de los punteros normales en C++ que se convierten en punteros colgantes (Thelin, 2007).

Se seleccionó el framework Qt para el desarrollo de la solución, porque, además de poseer las características mensionadas anteriormente, constituye al igual que el lenguaje C++, una tecnología seleccionada de manera global en el proyecto AREX.

1.4.4. Entorno integrado de desarrollo

Qt Creator es un entorno completo de desarrollo integrado (IDE) para la creación de aplicaciones con el framework Qt (Qt-Creator, 2014).

Una de las mayores ventajas de Qt Creator es que permite que un equipo de desarrolladores comparta un proyecto a través de diferentes plataformas de desarrollo (Microsoft Windows, Mac OS X y Linux) con una herramienta común para desarrollo y depurado (Qt-Creator, 2014).

El objetivo principal de Qt Creator es conocer las necesidades de desarrollo de los programadores Qt que buscan la simplicidad, facilidad de uso, productividad,

extensibilidad y apertura, mientras se baja la barrera de entrada para los recién llegados a Qt. Por ser el IDE con más potencialidades para desarrollar sobre el framework Qt, se selecciona dicho IDE, como norma del proyecto AREX.

1.4.5. Sistema Gestor de Base de Datos (SGBD)

SQLite es un sistema de gestión de bases de datos relacional compatible con ACID, contenida en una relativamente pequeña(275 KB) biblioteca escrita en C. SQLite es un proyecto de dominio público creado por D. Richard Hipp(Owens, 2006).

A diferencia de los sistema de gestión de bases de datos cliente/servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite 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. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción. En su versión 3, SQLite permite bases de datos de hasta dos terabytes de tamaño, y también permite la inclusión de campos tipo BLOB(Owens, 2006). Las características anteriormente mencionadas justifican la selección de SQLite para almacenamiento local, lo cual forma parte de los objetivos de la solución a desarrollar.

Seguidamente se describen algunas de las características atractivas de SQLite que conllevaron a que se eligiera el mismo como SGBD para realizar el almacenamiento de manera local requerido por la solución:

La biblioteca implementa la mayor parte del estándar SQL-92, incluyendo transacciones de base de datos atómicas, consistencia de base de datos, aislamiento, y durabilidad (ACID), triggers y la mayor parte de las consultas complejas(Owens, 2006), dicha característica provee a la solución un considerable nivel de escalabilidad.

Varios procesos o hilos pueden acceder a la misma base de datos sin problemas. Varios accesos de lectura pueden ser servidos en paralelo. Un acceso de escritura sólo puede ser servido si no se está sirviendo ningún otro acceso concurrentemente. En caso contrario, el acceso de escritura falla devolviendo un código de error (o puede automáticamente reintentarse hasta que expira un tiempo de expiración

configurable). Esta situación de acceso concurrente podría cambiar cuando se está trabajando con tablas temporales. Sin embargo, podría producirse un interbloqueo debido al multihilo. Este punto fue tratado en la versión 3.3.4, desarrollada el 11 de febrero de 2006(Owens, 2006), característica necesaria puesto que la solución propuesta está prevista en un entorno concurrente donde varios componentes deberán poder accesar a la información y un único componente podrá modificarla.

1.4.6. Mecanismo de comunicación entre procesos

Aunque se presenta TCP como parte del grupo de protocolos Internet TCP/IP, es un protocolo independiente de propósitos generales que se puede adaptar para utilizarlo con otros sistemas de entrega. Por ejemplo, debido a que TCP asume muy poco sobre la red subyacente, es posible utilizarlo en una sola red como Ethernet, así como en una red de redes compleja. De hecho, TCP es tan popular, que es uno de los protocolos para sistemas abiertos de la Organización Internacional para la Estandarización (ISO, por sus siglas en inglés) (Comer, 1996).
Cuando se implementan con el protocolo TCP, los sockets tienen las siguientes propiedades(Comer, 1996):

- Son orientados a la conexión.
- Se garantiza la transmisión de todos los octetos sin errores ni omisiones.
- Se garantiza que todo octeto llegará a su destino en el mismo orden en que se ha transmitido.

Estas propiedades son importantes para garantizar la corrección de los programas que tratan la información y la garantía de que la información llegará sin errores, lo cual constituye un requerimiento de la solución propuesta.

1.4.7. Framework de serialización

MessagePack es un formato de intercambio de datos de la computadora. Es un formato binario para representar estructuras de datos simples, como los arreglos y matrices asociativas. MessagePack pretende ser lo más compacto y simple como sea posible. El framework está disponible en una variedad de lenguajes como C, C ++, CSharp, D, Erlang, Go, Haskell, Java, JavaScript, Lua, OCaml, Perl, PHP, Python, Ruby, Scala y Smalltalk(MsgPack, 2015):
Características fundamentales de MessagePack:

- Biblioteca de serialización eficiente.

- Estructuras de datos enriquecidos, compatibles con JSON.
- Tipado dinámico.

Características que hacen a MessagePack tener una serialización eficiente:

- **Compacto:** Es compacto porque presenta un formato basado en codificación binaria e integra información de tipo (Ver Figura 1.5).
- **Rápido:** Es rápido porque usa serialización Zero-Copy y deserialización corriente (Ver Figura 1.6).

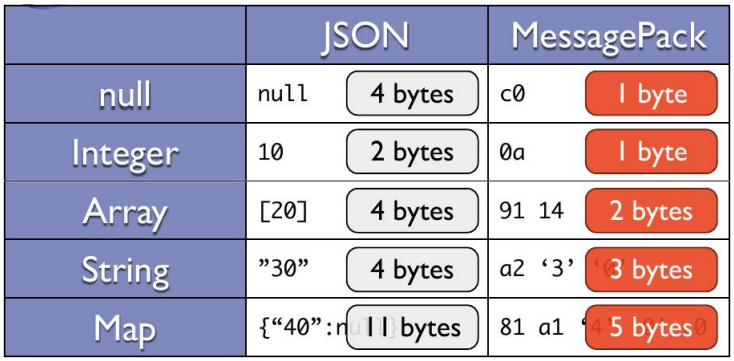

Figura 1.5: Codificación de MsgPack.

Se midió el tiempo transcurrido para serializar y deserializar 200.000 objetos. Cada objeto está compuesto por tres enteros y una cadena de caracteres de 512 bytes. Y se obtuvo la gráfica de la Figura 1.6, en la cual se aprecia que MessagePack supera a otros framework de serialización de referencia como son JSON y Protocol Buffers, por lo que fue selecionado para utilizarse en este trabajo.

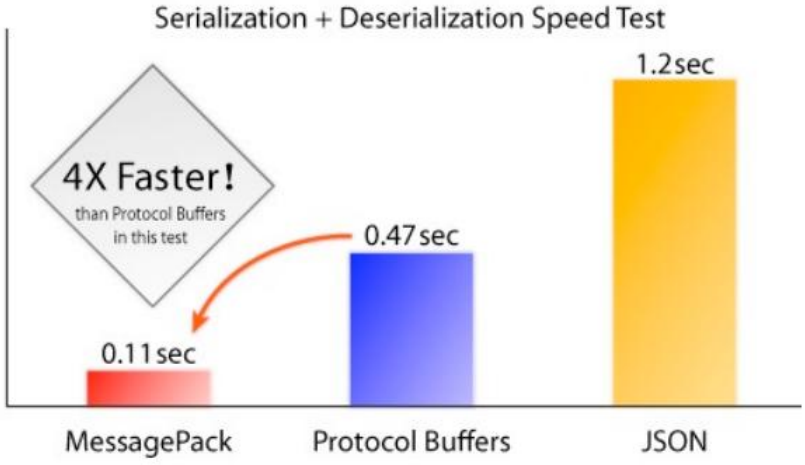

Figura 1.6: Rapidez de MsgPack respecto a otros serializadores.

1.4.8. Lenguaje de modelado

Para un mejor entendimiento del negocio y para el diseño de la solución, en este trabajo se utiliza el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés). UML ayuda a especificar, visualizar y documentar modelos de sistemas de software, incluyendo su estructura y diseño, de forma tal que cumpla con todos los requisitos descritos. Permite modelar casi cualquier tipo de aplicación, que se ejecute en cualquier tipo de combinación de hardware, sistema operativo, lenguaje de programación o infraestructura de red. Está diseñado para ser usado sobre los fundamentos de la orientación a objetos, convirtiéndolo en un marco ideal para los lenguajes orientados a objetos como C++, Java y el reciente C-Sharp, teniendo uso limitado para otros paradigmas de programación. El proceso de recopilación y análisis de los requisitos de una aplicación para incorporarlos a un diseño, es una tarea compleja y la industria actualmente es compatible con muchas metodologías que definen procedimientos formales que especifican como llevar a cabo esto. Una de las características de UML (de hecho la que permite que sea ampliamente utilizado en la industria de software) es que es independiente de la metodología que se seleccione. Independientemente de la metodología que se utilice para llevar a cabo el análisis y el diseño, se puede utilizar UML para expresar los resultados(OMG, 2014).

1.4.9. Herramienta de modelado

Hoy día, muchas empresas se han extendido a la adquisición de herramientas CASE (Computer Aided Software Engineering, Ingeniería Asistida por Computadora), con el fin de automatizar los aspectos claves de todo el proceso de desarrollo de un sistema y por ende, destinadas a aumentar la productividad de este proceso, reduciendo costes y tiempo(VisualParadigm, 2014).

Visual Paradigm es una herramienta para la creación de diagramas UML que es de gran ayuda en el proceso de desarrollo de software. Especialmente en las fases de análisis y diseño de este proceso, esta aplicación ayuda a conseguir un producto de alta calidad(VisualParadigm, 2014).

Este sistema puede manejar gran parte de los diagramas definidos en el lenguaje UML, ya sea creándolos manualmente o importándolos a partir de código en C++, Java, Python, Pascal, Delphi, entre otros, lo cual ofrece posibilidades de ingeniería inversa. Permite crear un modelo y generar el código automáticamente en varios lenguajes para ayudar a comenzar la implementación del proyecto.

El código generado incluye definiciones de clases con sus métodos y atributos proporcionando rapidez en el proceso inicial de desarrollo y haciéndolo menos propenso a errores(VisualParadigm, 2014). Por estas y otras características, que lo clasifican dentro de las mejores herramientas CASE, se elige Visual Paradigm como tal.

1.4.10. Metodología de desarrollo de software

El Proceso Unificado Ágil de Scott Ambler o Agile Unified Process (AUP) en inglés, es una versión simplificada del Proceso Unificado de Rational (RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles incluyendo Desarrollo Dirigido por Pruebas (Test Driven Development(TDD)), Modelado Agil, Gestión de Cambios Agil, y Refactorización de Base de Datos para mejorar la productividad(Cordero, 2014).

AUP se preocupa especialmente de la gestión de riesgos. Propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Para ello, se crean y mantienen listas identificando los riesgos desde etapas iniciales del proyecto. Especialmente relevante en este sentido es el desarrollo de prototipos ejecutables durante la base de elaboración del producto, donde se demuestre la validez de la arquitectura para los requisitos clave del producto y que determinan los riesgos técnicos(Cordero, 2014). El proceso AUP establece un Modelo más simple que el que aparece en RUP por lo que reúne en una única disciplina las disciplinas de Modelado de Negocio, Requisitos y Análisis y Diseño. El resto de disciplinas (Implementación, Pruebas, Despliegue, Gestión de Configuración, Gestión y Entorno) coinciden con las restantes de RUP(Cordero, 2014). Por recomendación de la dirección del CEDIN, en este trabajo se utiliza la metodología AUP variante UCI.

Conclusiones parciales

Aterrizando en el contexto de aplicación de las TIC[15] en la industria y la domótica, los SAD se diferencian fundamentalmente de los SM por un módulo o componente fundamental llamado BDH. El mismo, provee al sistema el almacenamiento y acceso

[15]Tegnologías de la Informática y las Comunicaciones.

a los datos generados a lo largo del tiempo. Particularmente el sistema AREX, que actualmente es un SM, no cuenta con BDH; por lo que se decide la incorporación de un BDH al mismo. De las funcionalidades que provee un BDH se seleccionaron las siguientes: histórico de variables y alarmas, almacenamiento y acceso eficiente a los datos de forma local y remota respectivamente. Respecto a la selección de tecnologías y herramientas se destaca la selección de C++ como lenguaje de programación, Qt como framework de desarrollo, MsgPack como framework de serialización, socket TCP como mecanismo de IPC[16] , SQLite como SGBD, AUP como metodología de desarrollo por sus potencialidades en proyectos de este tipo, y por último, para modelar la solución UML junto a Visual Paradigm como herramienta de modelado.

[16]Comunicación entre procesos.

Capítulo 2

DISEÑO E IMPLEMENTACIÓN

En el ámbito de la ingeniería de software, un buen sistema es aquel que cumple con las necesidades del cliente y posee facilidades de soporte luego de ser liberado. Con el objetivo de lograr un software con estas características, en el presente capítulo se exponen varios elementos que forman parte del proceso de desarrollo, como son: la propuesta del sistema, el modelo arquitectónico propuesto, una visión general del diseño y la implementación de la solución.

2.1. Propuesta de solución

La solución que se propone abarca la creación de un programa de aplicación sin interfaz gráfica que hará función de servidor BDH para el sistema AREX, y un protocolo de comunicación del nivel de aplicación del modelo OSI[1], que permitirá el entendimiento entre aplicaciones clientes y el servidor BDH. El servidor BDH implementará dos funcionalidades fundamentales pertenecientes a los BDH comunes: histórico de variables e histórico de alarmas, los cuales permitirán el conocimiento en el tiempo de los valores de las variables y las ocurrencias de alarmas. Dicha información se proveerá mediante el mecanismo de IPC Socket-TCP[2], por el cual se enviará serializada usando el framework[3] MsgPack, haciendo eficiente el envío de información a través de la red. Cabe señalar que dicho servidor podrá ofrecer la información que recibe el recolector de AREX a aplicaciones clientes, tanto de forma histórica como de tiempo real de las variables configuradas en el sistema. La información de configuración se obtiene de un archivo XML generado previamente en el submódulo HMI-Edición. El servidor BDH se debe ejecutar sobre cualquier

[1]Modelo de red descriptivo creado en el año 1980 por la Organización Internacional de Normalización(ISO).

[2]Punto final de la comunicación entre dos procesos mediante el protocolo TCP.

[3]Marco de trabajo para el desarrollo de software.

dispositivo que tenga como plataforma un sistema operativo basado en Linux (Ver Figura 2.1).

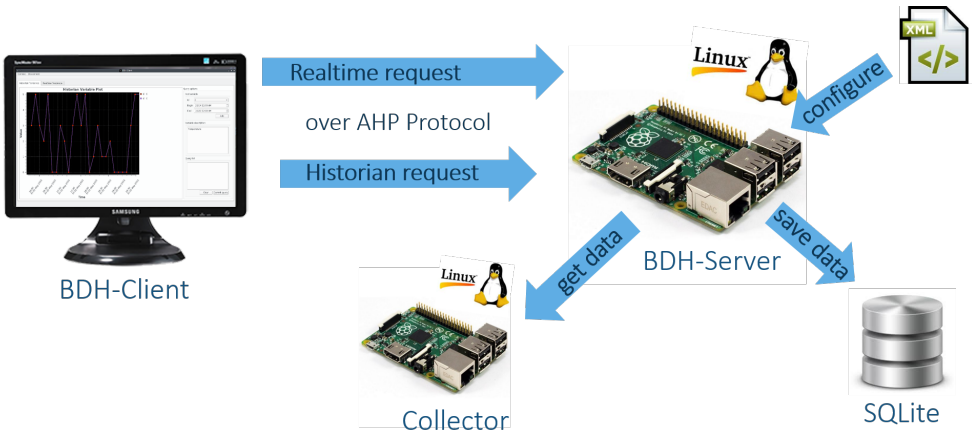

Figura 2.1: Propuesta de sistema: cliente y servidor BDH y protocolo AHP.

2.2. Modelo arquitectónico

La arquitectura del software alude a la estructura global del software y a las formas en que la estructura proporciona la integridad conceptual de un sistema. En su forma más simple, la arquitectura es la estructura jerárquica de los componentes del programa (módulos), la manera en que los componentes interactúan y la estructura de datos que van a utilizar los componentes. Sin embargo, en un sentido más amplio, los componentes se pueden generalizar para representar los elementos principales del sistema y sus interacciones(Pressman, 2010).

En la construcción de software existen diferentes estilos arquitectónicos. Los mismos proporcionan una plantilla de trabajo, que unifica la manera en que todos los miembros del equipo ven el sistema y además, impone una transformación al diseño del mismo.

Un patrón arquitectónico, al igual que un estilo, impone una transformación en el diseño de una arquitectura. Sin embargo, un patrón difiere de un estilo en varios elementos fundamentales: 1) el alcance de un patrón es menor, ya que se concentra en un aspecto en lugar de hacerlo en toda la arquitectura; 2) un patrón impone una regla sobre la arquitectura, pues describe la manera en que el software manejará algún aspecto de su funcionalidad al nivel de la infraestructura (por ejemplo, concurrencia); 3) los patrones arquitectónicos tienden a abarcar aspectos específicos del comportamiento dentro del contexto de la arquitectura (por ejemplo,

cómo maneja una aplicación en tiempo real la sincronización o las interrupciones). Los patrones se usan junto con un estilo arquitectónico para determinar la forma de la estructura general de un sistema(Pressman, 2010).

Con el objetivo de establecer una estructura para todos los componentes del sistema se escoge el estilo arquitectónico Orientado a Objetos(OO). El mismo plantea que los componentes de un sistema encapsulan los datos y las operaciones que deben aplicarse para manipular estos datos. La comunicación y coordinación entre los componentes se consigue mediante el paso de mensajes(Pressman, 2010).

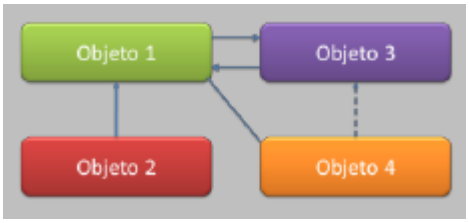

Figura 2.2: Estilo arquitectónico Orientado a Objetos.

Por otro lado, como patrón arquitectónico se elige el patrón N-Capas, el cual se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas, y las responsabilidades, la funcionalidad que implementan(Llorente, 2011).

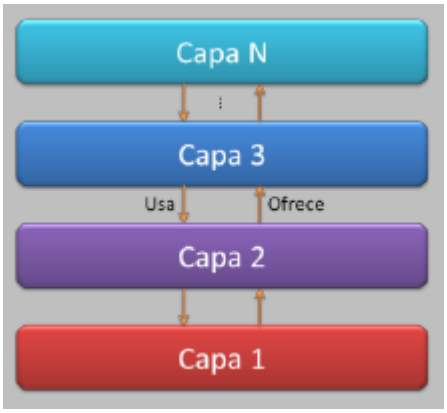

Figura 2.3: Patrón arquitectónico N-Capas.

Específicamente según la vista lógica de la arquitectura del sistema, se han definido las capas lógicas según el modelo N-Capas (Ver Figura 2.4).

Es importante distinguir los conceptos de capas(layers) y niveles(Tiers), pues es bastante común que se confundan y/o se denominen de forma incorrecta. Las capas se ocupan de la división lógica de componentes y funcionalidad y no tienen en cuenta

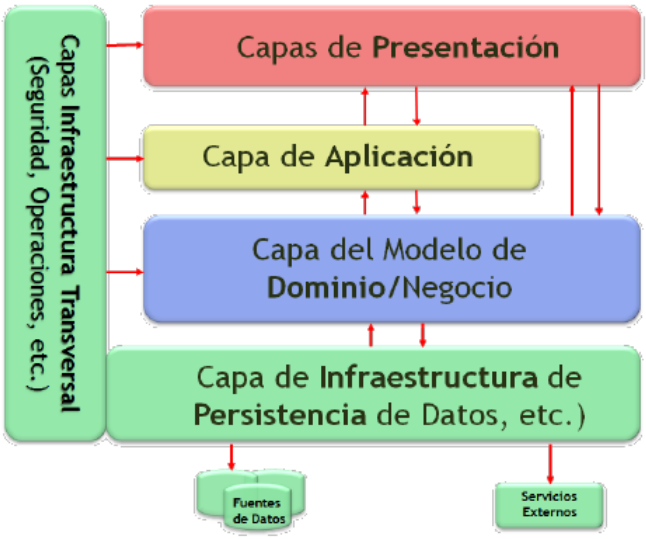

Figura 2.4: Capas según el modelo N-Capas.

la localización física de componentes en diferentes servidores o en diferentes lugares. Por el contrario, los niveles se ocupan de la distribución física de componentes y funcionalidad en servidores separados, teniendo en cuenta topología de redes y localizaciones remotas. Aunque tanto las Capas como los niveles usan conjuntos similares de nombres (presentación, servicios, negocio y datos), es importante no confundirlos y recordar que solo los niveles implican una separación física. Se suele utilizar el término tier refiriéndonos a patrones de distribución física como 2-tier, 3-tier y n-tier(Llorente, 2011) (Ver Figura 2.5).

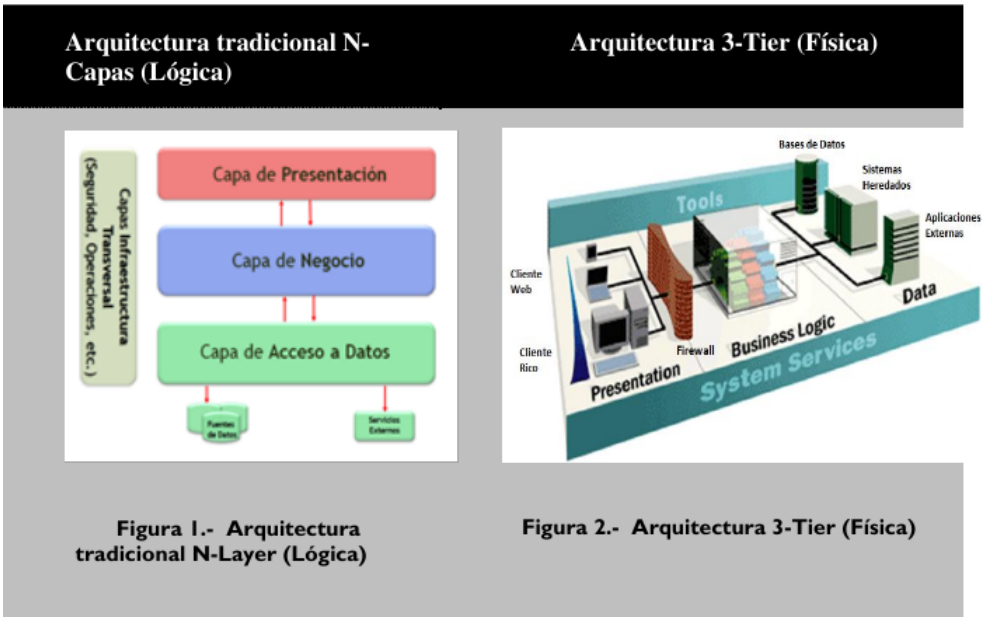

Figura 2.5: Diferencia entre capas y niveles.

De esta manera se ha elegido como principio de división en niveles el: 2-niveles, el cual divide el sistema en dos nodos físicos diferentes, los cuales serán específicamente cliente y seridor. El cliente implementará la capa lógica presentación mientras que el servidor implementará las capas lógicas restantes(Ver Figura 2.6).

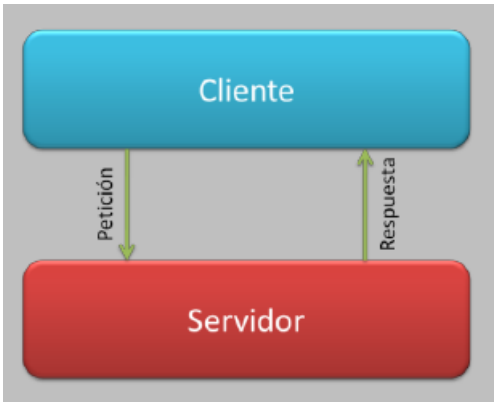

Figura 2.6: División en 2-niveles: Cliente/Servidor.

2.3. Historias de usuario

Entre los artefactos que define la metodología seleccionada se encuentran las historias de usuario, las cuales representan una breve descripción del comportamiento del sistema, emplean terminología del cliente sin lenguaje técnico, se realiza una por cada característica principal del sistema, se emplean para hacer estimaciones de tiempo, reemplazan un gran documento de requisitos y presiden la creación de las pruebas de aceptación.

Se redactaron un conjunto de historias de usuario en las cuales se recogen las principales funcionalidades con las que debe contar el sistema (Ver Anexo 3.2.2). En las Tablas 2.1 y 2.2 se muestran dos de las historias de usuario elaboradas (Ver las demás historias de usuario en el Anexo 3.2.2).

2.4. Diseño

2.4.1. Modelo de almacenamiento en bloques

Para almacenar las muestras de variables se van creando bloques de datos de igual tamaño de manera incremental, dicho tamaño depende del tamaño total configurado para el almacenamiento. Por otro lado se crea una tabla catálogo que contiene la información de los bloques creados (id, marca de tiempo de la primera muestra

Historia de usuario	
Numero: 1	**Usuario:** Línea Sistemas Embebidos
Nombre: Gestión de la configuración	
Prioridad en el negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 5	**Iteración asignada:** 1
Programador Responsable: Randy Mujica Díaz	
Descripción: Permitir cargar los parámetros de configuración del módulo BDH como son: las variables configuradas, alarmas configuradas y espacio disponible para el almacenamiento además del puerto donde escuchará el servidor.	
Observaciones:	

Tabla 2.1: Historia de usuario 1.

Historia de usuario	
Numero: 2	**Usuario:** Línea Sistemas Embebidos
Nombre: Almacenamiento local de muestras de variables y alarmas	
Prioridad en el negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 5	**Iteración asignada:** 1
Programador Responsable: Randy Mujica Díaz	
Descripción: Lograr almacenar de manera local(o sea en la misma estación de trabajo donde se ejecuta la aplicación) las muestras de variables y ocurrencias de alarmas, además este almacenamiento debe realizarse de forma que se haga uso eficiente del espacio de disco utilizado y se minimice el tiempo necesario para consultar la información almacenada.	
Observaciones:	

Tabla 2.2: Historia de usuario 2.

que se almacena y marca de tiempo de la última muestra que se almacena). Esto garantiza mayor rendimiento a la hora de realizar consultas a la información, pues la estructura de bloques creada permite discernir en cuál o cuáles bloques se encuentra la información que se solicita, por tanto la consulta SQL no se realiza sobre toda la información almacenada y de esta manera se hace evidente una mejora en el rendimiento (Ver Figura 2.7).

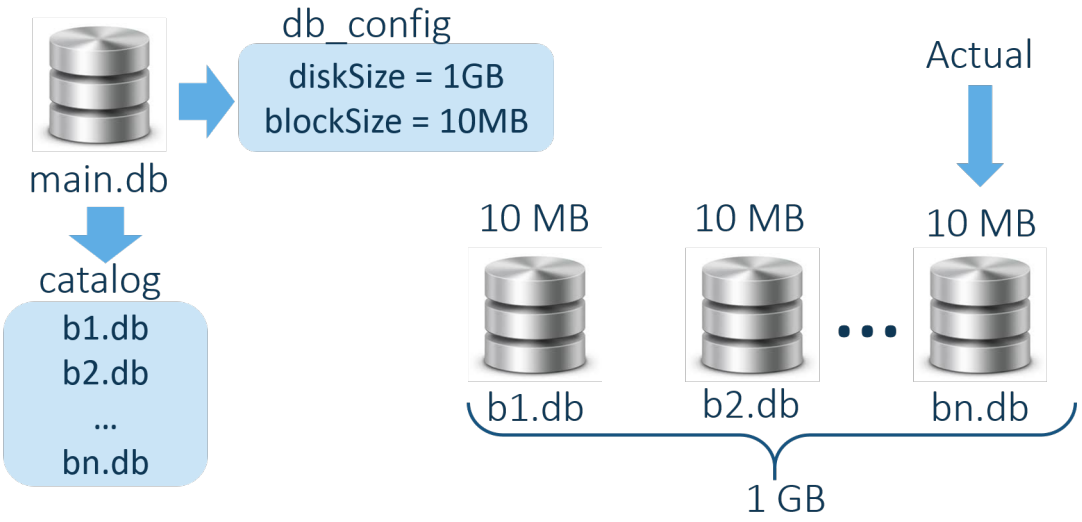

Figura 2.7: Modelo de almacenamiento en bloques.

2.4.2. Modelo Entidad-Relación

Un modelo entidad-relación (MER o ER, por sus siglas en inglés) es una herramienta para el modelado de datos que permite representar las entidades relevantes de un sistema de información así como sus interrelaciones y propiedades(Chávez Moreno, 2007).

En el negocio de la presente investigación existen diferentes entidades como son: *Variable*, *VariablePoint*, *Alarm*, *AlarmPoint* entre otras y las mismas están relacionadas. En la Figura 2.8 se observa el MER asociado al sistema.

2.4.3. Protocolo Arex Historian Protocol

El protocolo Arex Historian Protocol (AHP, por sus siglas en inglés), es un protocolo del nivel de aplicación del modelo OSI, el mismo fue creado sobre el protocolo TCP del nivel de transporte del modelo OSI. AHP es un protocolo sencillo basado en comandos, dichos comandos son enviados por el cliente al servidor con el objetivo de que dicho servidor le proporcione la información que solicita. AHP es síncrono,

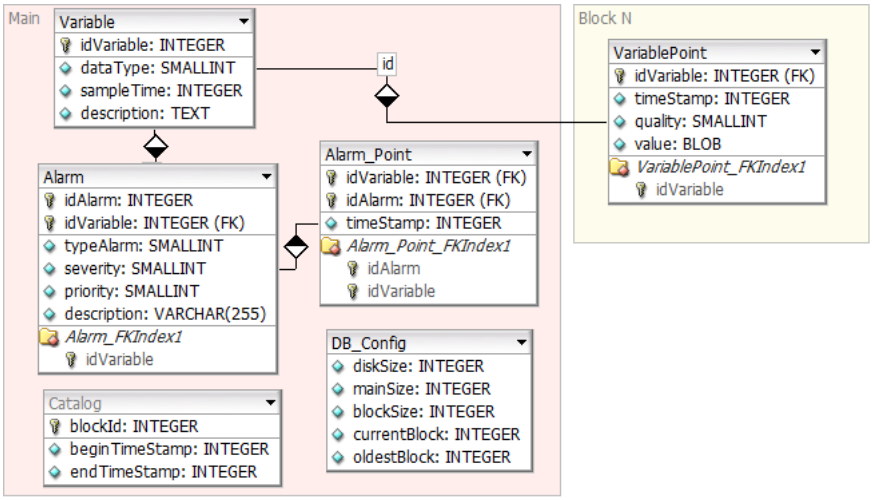

Figura 2.8: Modelo Entidad-Relación.

cada comando enviado por el cliente es respondido por el servidor. El cliente puede solicitar información de dos tipos: histórica y en tiempo real, teniendo en cuenta que solo puede solicitar un tipo de información a la vez, o sea si esta solicitando información histórica se encuentra en modo histórico y no puede solicitar información de tiempo real y viceversa. El formato de la trama AHP está compuesto por cabecera y datos (Ver Figura 2.9), en el campo command de la cabecera es donde se envía el comando a realizar (Ver Figura 2.10).

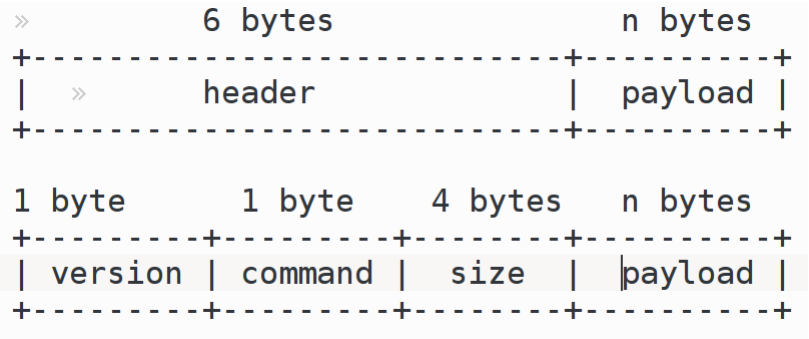

Figura 2.9: Formato de la trama de AHP.

2.4.4. Diagramas de secuencia del protocolo AHP

El diagrama de secuencia es un tipo de diagrama usado para modelar interacción entre objetos en un sistema según UML.

Secuencia de mensajes de tiempo real: Las secuencias de mensajes de tiempo real en este contexto se refieren a los mensajes que necesita mandar el cliente al servidor y las respuestas del servidor al cliente, para que el cliente pueda solicitar

```
COMMANDS

Code  » Name

01»     HELLO
02»     ACCEPT
03»     REJECT
04»     SET REALTIME MODE
05»     SET HISTORIAN MODE
06»     RESET MODE
07»     STOP REALTIME SENDING
08      GET REALTIME VARIABLES
09»     REALTIME VARIABLES RESPONSE
10 »    GET HISTORIAN VARIABLES
11»     HISTORIAN VARIABLES RESPONSE
12»     DISCONNECT
13»     MODE SUCCESSFUL
```

Figura 2.10: Comandos de AHP.

información de tiempo real de manera correcta. En la Figura 2.11 se observa un diagrama de ilustra la secuencia de mensajes para establecer el modo tiempo real.

Secuencia de mensajes de histórico: Se entiende por secuencias de mensajes de histórico el intercambio de comandos necesarios para que el cliente pueda solicitar información históricamente persistida en la BDH de manera correcta (Ver Figura 2.12).

2.4.5. Patrones de diseño

Existen una serie de situaciones comunes en las que los programadores caen una y otra vez, incluso llegando a codificar de forma diferente cada vez. Los patrones de diseño vienen a mitigar un poco este dilema. Se puede decir que son soluciones a diferentes clases de problemas conocidos que pueden ser aplicadas a diferentes códigos, por lo que no hay necesidad de reinventar la rueda. De manera más simple, un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos(Larman, 2003).

Un patrón es una descripción de un problema y la solución, a la que se da un nombre, y que se puede aplicar a nuevos contextos; idealmente, proporciona consejos sobre el modo de aplicarlo en varias circunstancias, y considera los puntos fuertes y compromisos. Muchos patrones proporcionan guías sobre el modo en que deberían asignarse las responsabilidades a los objetos, dada una categoría específica del

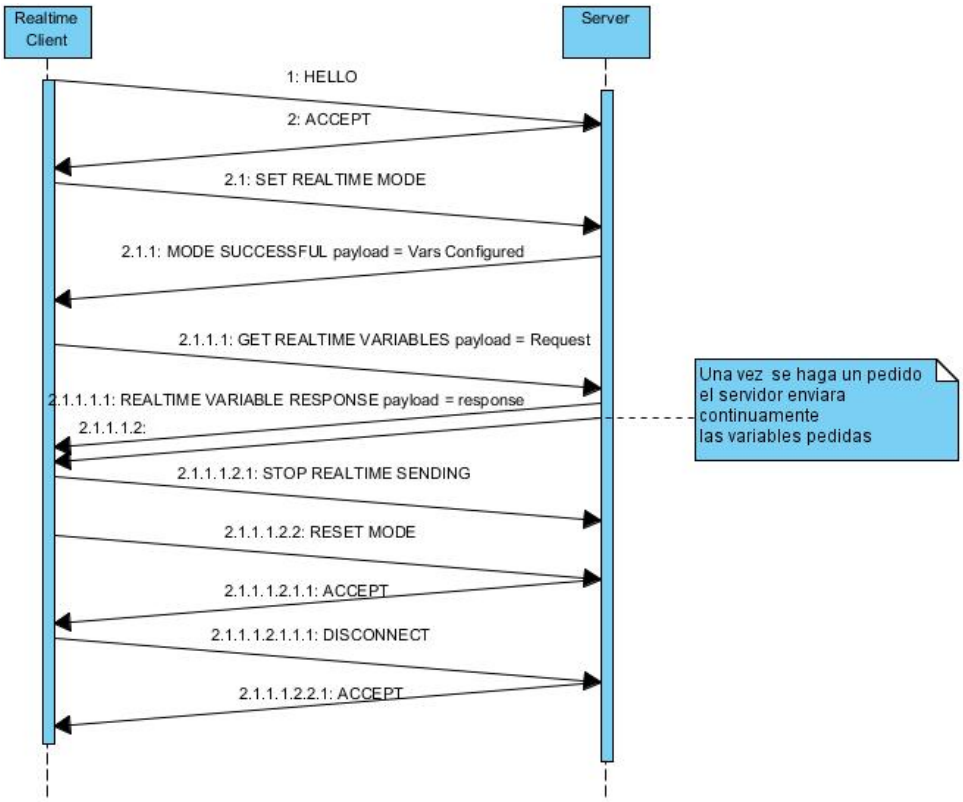

Figura 2.11: Secuencia de mensajes de tiempo real.

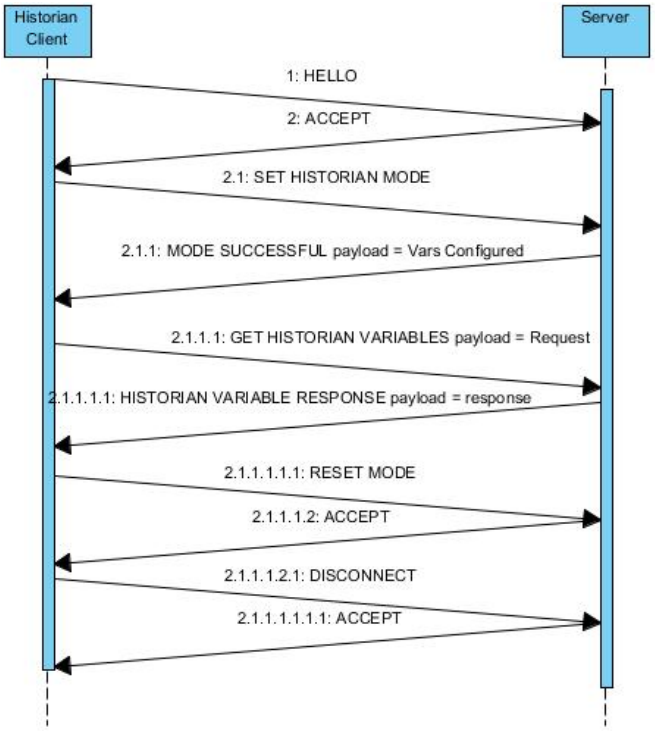

Figura 2.12: Secuencia de mensajes de histórico.

problema(Larman, 2003).

Patrones GRASP

Experto: la responsabilidad de realizar una labor es de la clase que tiene o puede tener la información necesaria (atributos), respondiendo al problema del cuál es un principio general para asignar responsabilidades a las clases(Larman, 2003). Este patrón se pone de manifiesto en la mayor parte de las clases del proyecto, por ejemplo, la clase *SQLiteDatabase* tiene la responsabilidad de almacenar la información del negocio en una base de datos SQLite porque la misma es la que cuenta con la información para realizar dicha tarea.

Creador: se asigna la responsabilidad de que una clase B cree un objeto de la clase A, si se cumple uno o más de los casos siguientes (respondiendo al problema de quién debería ser el responsable de la creación de una nueva instancia de alguna clase): B agrega objetos de A, B contiene objetos de A, B registra instancias de A. Dicho patrón mayormente se pone de manifiesto en las clases controladoras de la solución, por ejemplo, la clase *BDHServerController*, la cual contiene objetos de la clase *AHPServer*, *RealtimeCache*, *SQLiteDatabase*, entre otras.

Bajo acoplamiento: El acoplamiento es una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, confía en, otros elementos. Un elemento con bajo (o débil) acoplamiento no depende de demasiados otros elementos; "demasiados"depende del contexto. Estos elementos pueden ser clases, subsistemas, sistemas, etcétera(Larman, 2003). En el diseño lógico propuesto se puede apreciar el bajo acoplamiento entre las capas a través de las interacciones entre las clases contenidas en capas distintas.

Alta cohesión: En cuanto al diseño de objetos, la cohesión (o de manera más específica, la cohesión funcional) es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento. Un elemento con responsabilidades altamente relacionadas, y que no hace una gran cantidad de trabajo, tiene alta cohesión. Estos elementos pueden ser clases, subsistemas, etcétera. En la solución propuesta mayormente se puede ver dicho patrón en las capas Infraestructura y Dominio.

Patrones GoF y otros

Máquina de estados finitos (FSM): Una Máquina de Estados Finitos (Finite State Machine), llamada también Autómata Finito es una abstracción computacional que describe el comportamiento de un sistema reactivo mediante un número determinado de estados y un número determinado de transiciones entre dicho estados(Álvarez Torrico, 2015). Este patrón se usó para modelar y desarrollar el comportamiento de una conexión en el servidor (BDH-Server), y los distintos mensajes que puede mandar el cliente (BDH-Client) al servidor para lograr el objetivo propuesto, de esta manera se describen los distintos estados por los que pasa una conexión, y los comandos que puede enviar el cliente en cada estado. En la Figura 2.13 se observa el diagrama correspondiente a dicha máquina de estados finitos.

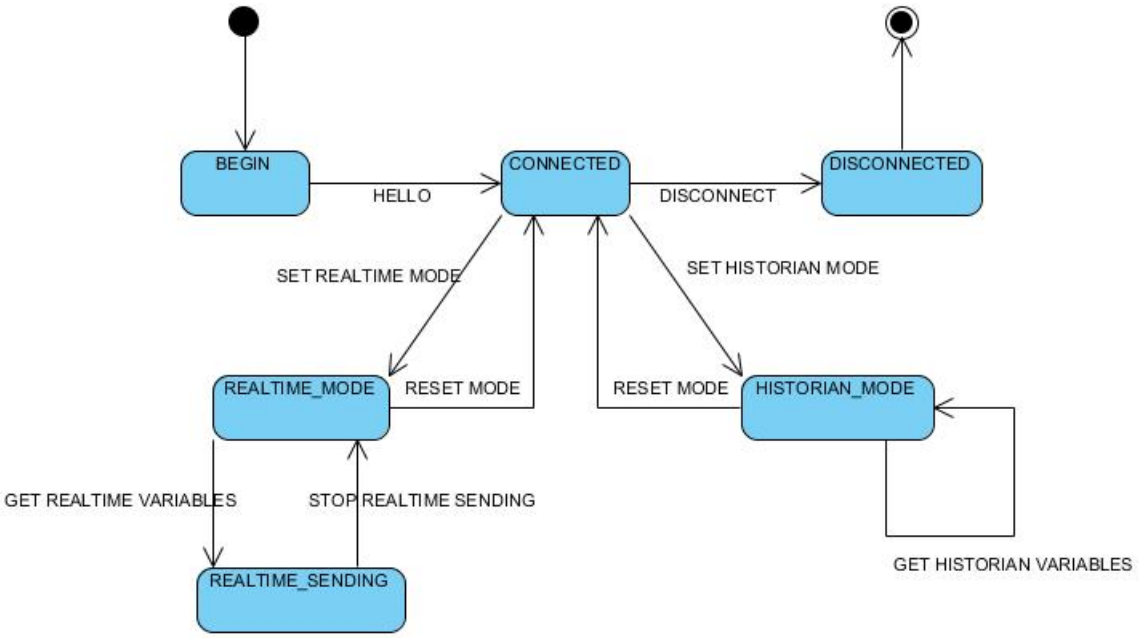

Figura 2.13: Patrón Máquina de estados finitos.

Singlenton: El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella(Gamma, 1994). Este patrón se empleó en la clase *AHPFrameFactory* para garantizar que exista una única instancia de dicha clase para todo el proyecto (Ver Figura 2.14).

Programación paralela (paradigma): La programación paralela es la simultaneidad en la ejecución de múltiples tareas interactivas. Estas tareas

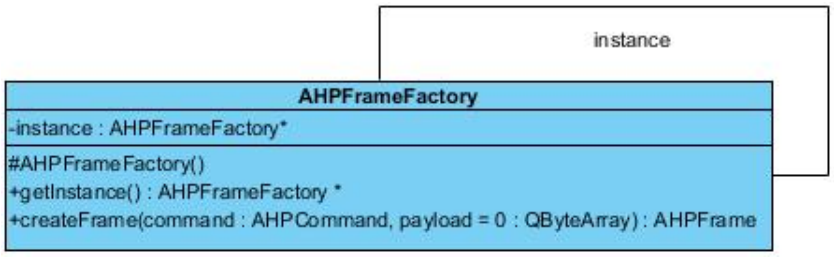

Figura 2.14: Patrón Singlenton.

pueden ser un conjunto de procesos o hilos[4] de ejecución creados por un único programa(Burns y Davies, 1993). Este paradigma se empleó en el BDH-Server para la atención de múltiples conexiones de manera simultánea, lo cual mejora el rendimiento del servidor. En la Figura 2.15 un diagrama que muestra el uso del paradigma mencionado.

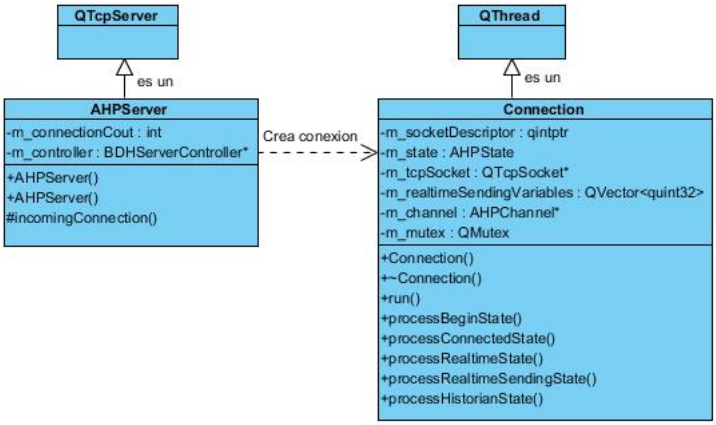

Figura 2.15: Paradigma de programación concurrente.

2.4.6. Diagramas de clases

Los diagramas de clase son el pilar básico del modelado con UML, siendo utilizados tanto para mostrar lo que el sistema puede hacer (análisis), como para mostrar cómo puede ser construido(SlideShare, 2013). A continuación se muestran separadas por capas, las principales clases que componen al sistema.

Capa Presentación

Esta capa es responsable de mostrar información al usuario e interpretar sus acciones. Los componentes de las capas de presentación implementan por lo tanto las funcionalidades requerida para que los usuarios interactúen con la aplicación (Llorente, 2011) (Ver Figura 2.16).

[4]Unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

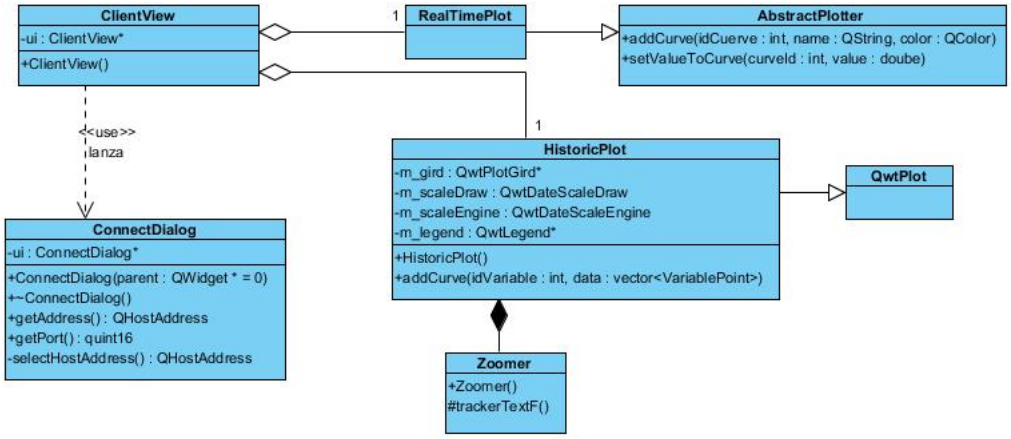

Figura 2.16: Diagrama de clases de la capa Presentación.

Capa Aplicación

Esta capa forma parte de la propuesta de arquitecturas orientadas a la capa infraestructura. La misma debe ser delgada y no contener reglas del dominio o conocimiento de la lógica de negocio, debe delegar este trabajo a los objetos de la siguiente capa (Llorente, 2011). En la arquitectura propuesta, esta capa se refleja con las clases BDH-Client y BDH-Server (Ver Figura 2.17).

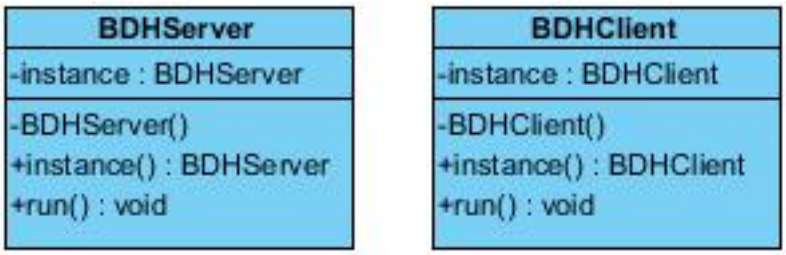

Figura 2.17: Diagrama de clases de la capa Aplicación.

Capa Infraestructura

Esta capa proporciona la capacidad de persistir datos y cómo acceder a ellos. Pueden ser datos propios del sistema o incluso acceder a datos expuestos por sistemas externos (servicios web externos, etc.) (Llorente, 2011). Específicamente para esta propuesta de solución, la capa infraestructura además de persistir datos, también es la encargada de todos los detalles técnicos y la lógica general del sistema, con lo cual se puede afirmar que el presente diseño arquitectónico está enfocado a esta capa (Ver Figura 2.18).

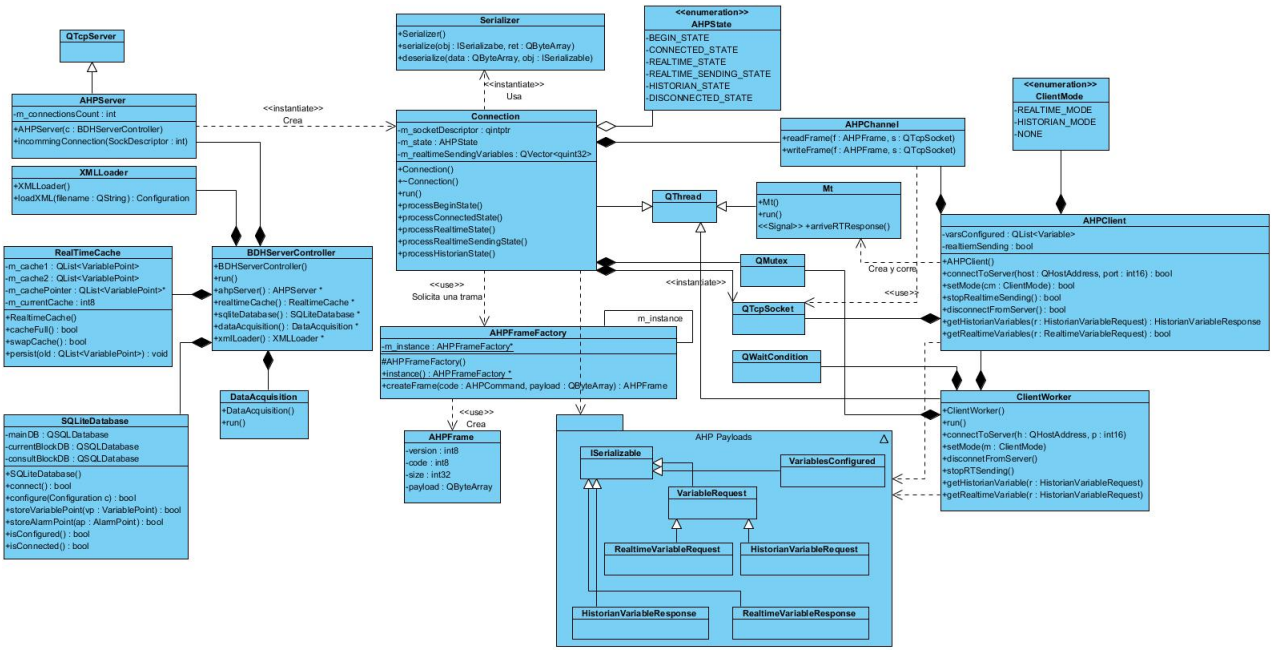

Figura 2.18: Diagrama de clases de la capa Infraestructura.

Capa Entidades del dominio

Esta capa contiene las entidades del negocio que se utilizan para obtener y transferir datos de entidades entre las diferentes capas (Ver Figura 2.19). Estos datos representan entidades de negocio del mundo real, como productos o pedidos. (Llorente, 2011).

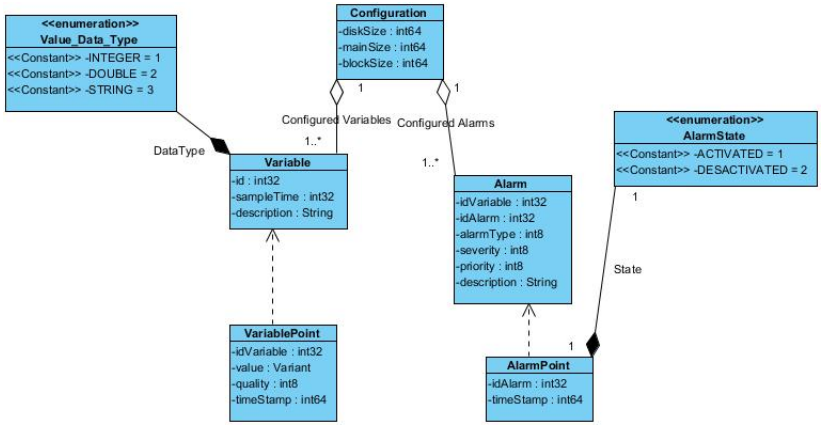

Figura 2.19: Diagrama de clases de la capa Dominio.

2.4.7. Diagrama de paquetes

Los paquetes se pueden utilizar para mostrar agrupaciones lógicas de objetos. Cada paquete agrupa un conjunto cohesivo de responsabilidades. Esta es la práctica básica

de aplicar la modularidad para dar soporte a la separación de intereses (Larman, 2003). En la Figura 2.20 se observa una vista del diagrama de paquetes de la solución.

2.4.8. Diagrama de despliegue

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos(SPARX, 2007).

En la configuración propuesta se cuenta con una tarjeta que pueda empotrar un sistema operativo basado en Linux, por ejemplo, la CID-300/9, Rasberry Pi, entre otras. En un dispositivo de este tipo se ejecutará el BDH-Server. Por otra parte el BDH-Client se debe ejecutar en una PC convencional. La comunicación entre el BDH-Client y el BDH-Server se realiza mediante el protocolo AHP. La configuración del módulo BDH se obtiene a partir de un archivo XML generado en el HMI-Editor (Ver Figura 2.21).

2.5. Diagramas de flujo

El servidor BDH consta de varios componentes con distintas responsabilidades para lograr su funcionamiento. Entre estos están: *XMLLoader* con la responsabilidad de cargar la configuración a partir del archivo XML generado en el HMI-Editor, *SQLiteDatabase* que se encarga de persistir la información en bases de datos SQLite, *RealtimeCache* mantiene en memoria la información antes de ser persistida, *TCPProxy* permite la conexión con el Recolector y obtiene la información que este envía, *AHPServer* escucha conexiones TCP y las atiende en un hilo diferente para cada una de ellas, por último el *BDHServerController* tiene la responsabilidad de controlar todo el funcionamiento y la comunicación entre los componentes mencionados anteriormente.

La Figura 2.22 muestra el funcionamiento del servidor BDH desde que la aplicación BDH-Server es ejecutada. Dicho funcionamiento es implementado en el método *run()* de la clase *BDHServerController*.

En la Figura 2.22 la actividad Attend Connection significa la creación de una instancia de la clase *Connection* la cual representa un hilo que atenderá la conexión que ha arribado al servidor. En el método *run()* de dicha clase se implementa la

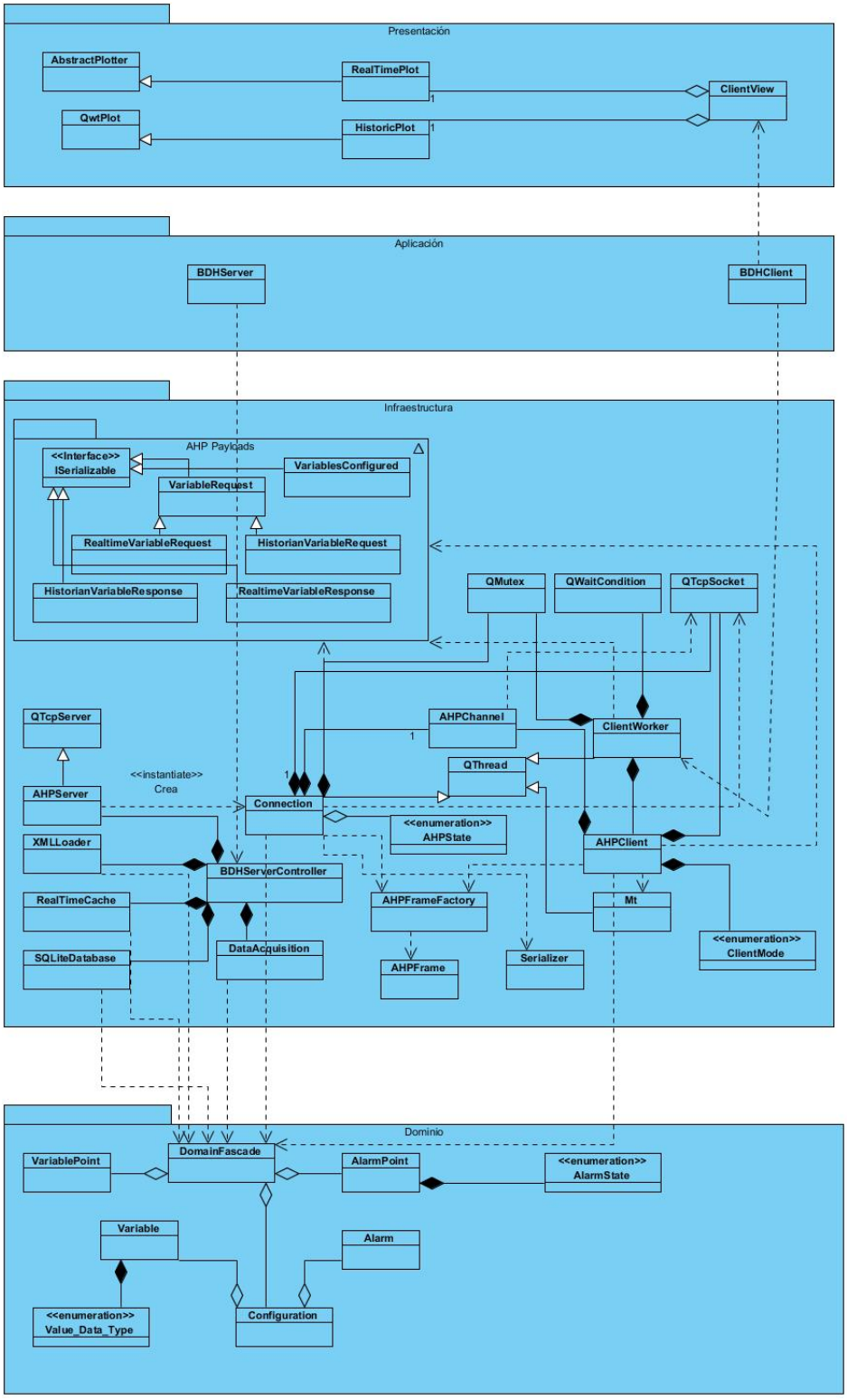

Figura 2.20: Diagrama de paquetes.

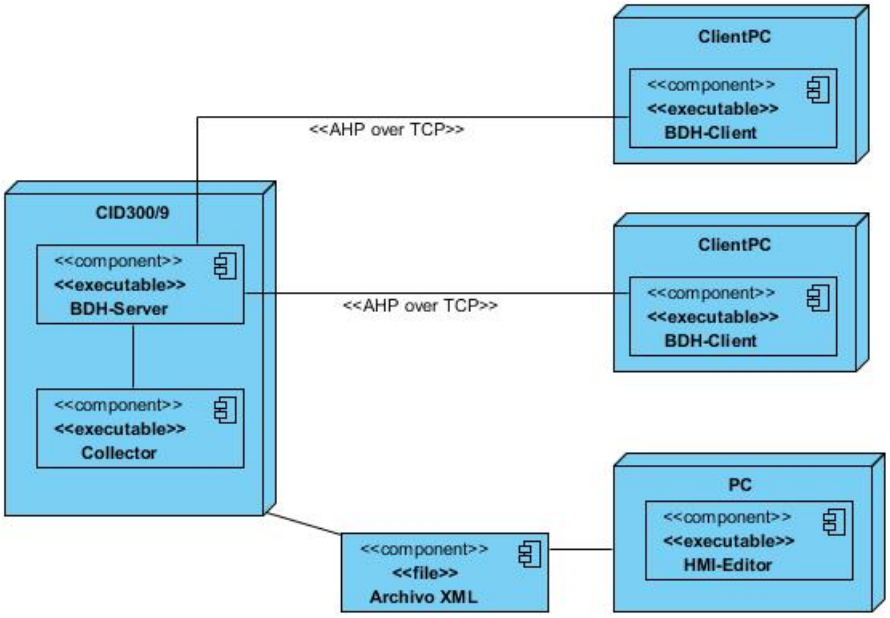

Figura 2.21: Diagrama de despliegue.

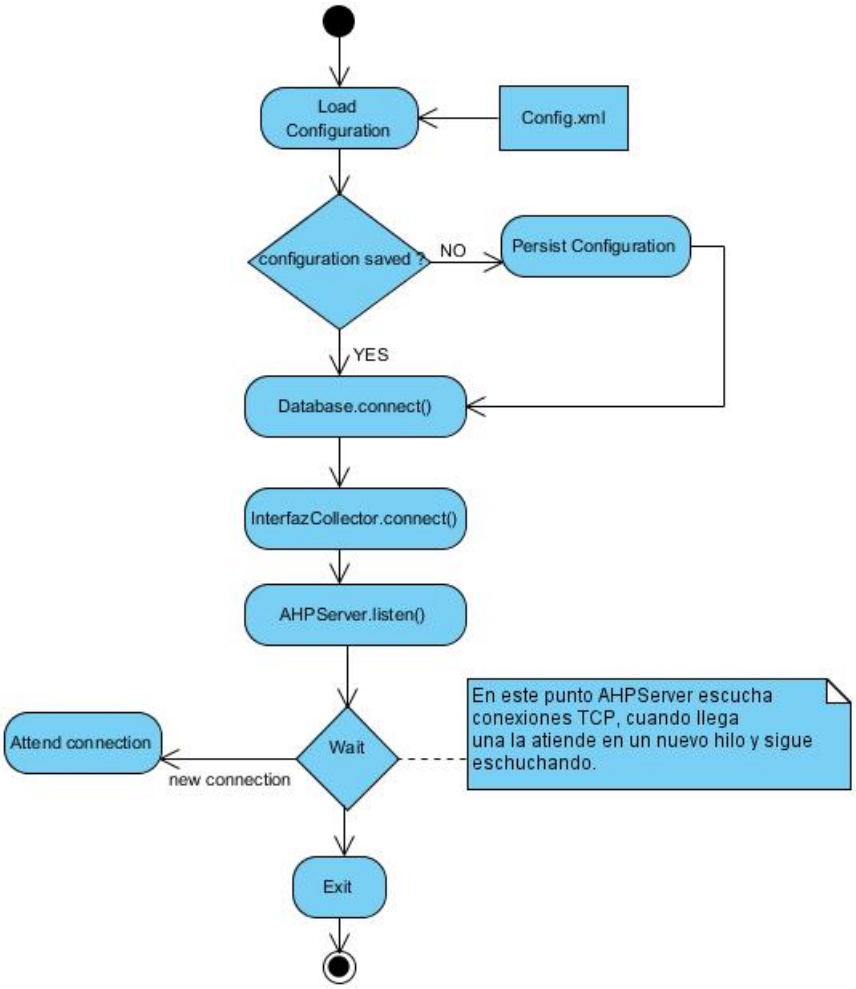

Figura 2.22: Funcionamiento del BDH-Server.

lógica de la misma la cual se muestra en la Figura 2.23. Para implementar la lógica se usó una máquina de estados finita, con lo cual cada vez que arribe un pedido(request), este será validado y luego procesado en dependencia del estado en que se encuentre la conexión. Los estados por los que puede pasar una conexión son: BEGIN, CONNECTED, HISTORIAN, REALTIME, REALTIME_SENDING y DISCONNECTED, cada uno de ellos tiene su respectivo método para procesar un pedido.

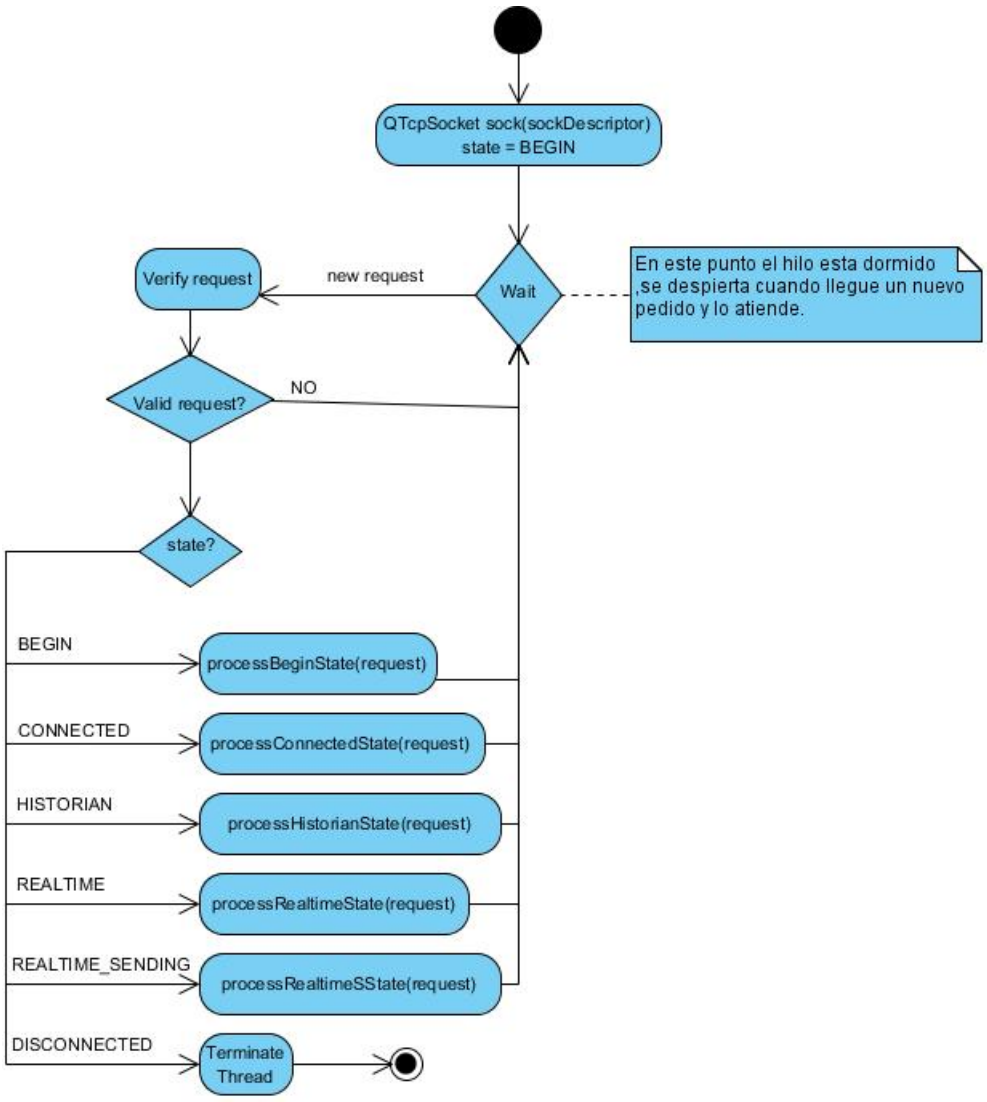

Figura 2.23: Lógica de una conexión al servidor.

El método *processBeginState* es el encargado de procesar un pedido en el estado BEGIN, asimismo, los métodos *processConnectedState*, *processHistorianState*, *processRealtimeState*, *processRealtimeSState* son los encargados de procesar un pedido en los estados CONNECTED, HISTORIAN, REALTIME, REALTIME_SENDING respectivamente. Las Figuras 2.24, 2.25, 2.26, 2.27, 2.28 muestran la lógica de implementación de los métodos descritos anteriormente.

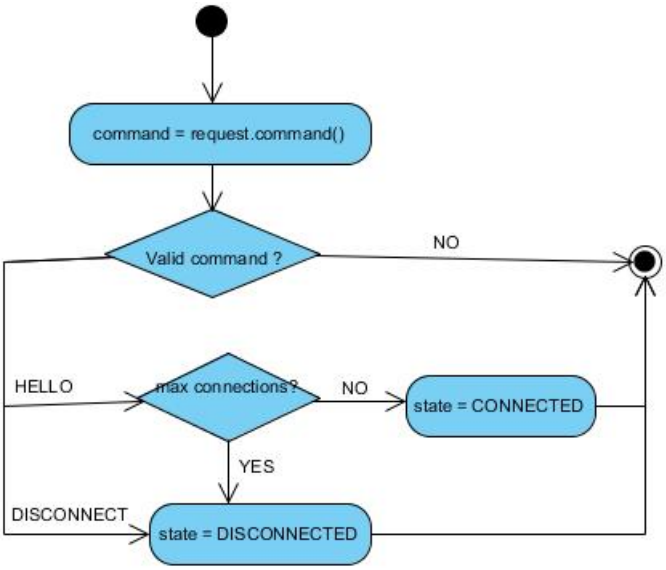

Figura 2.24: Lógica de *processBeginState*.

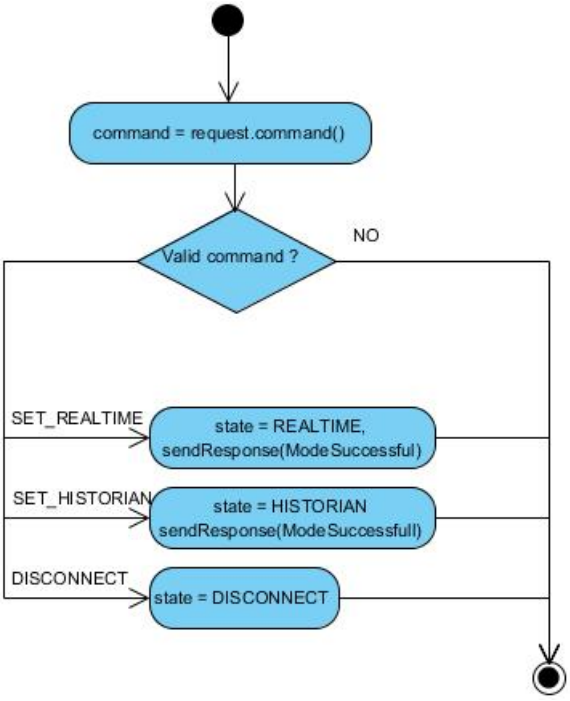

Figura 2.25: Lógica de *processConnectedState*.

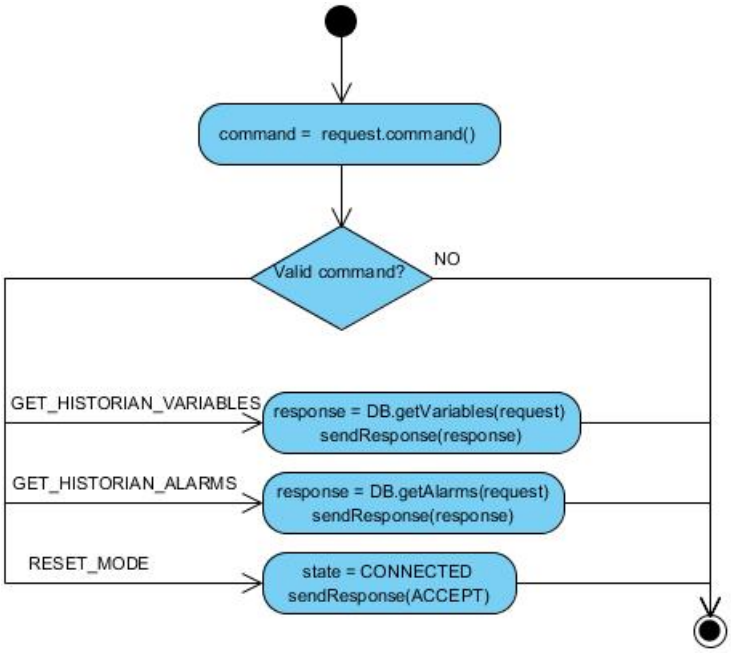

Figura 2.26: Lógica de *processHistorianState*.

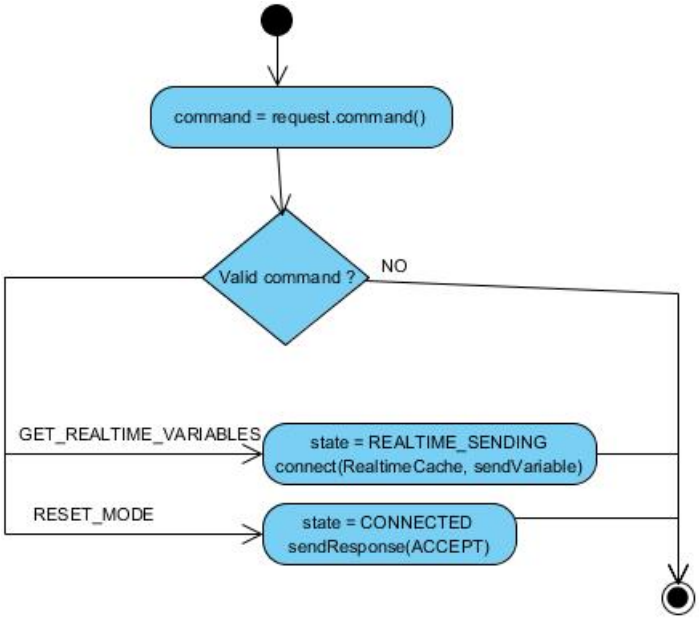

Figura 2.27: Lógica de *processRealtimeState*.

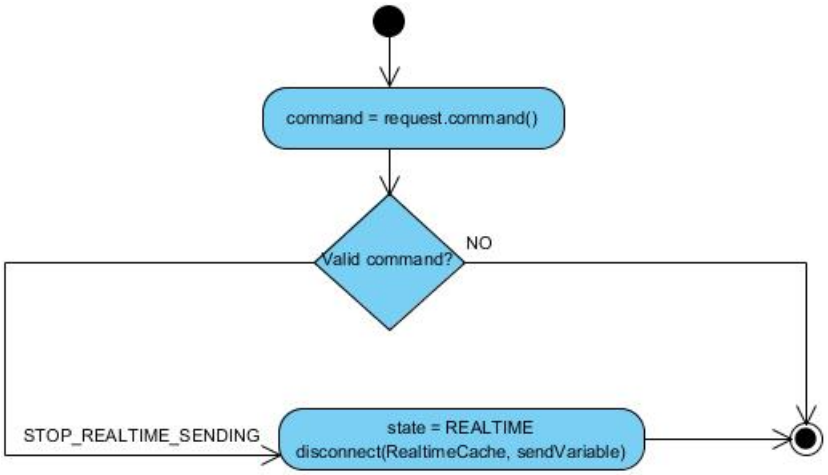

Figura 2.28: Lógica de *processRealtimeSendingState*.

Conclusiones parciales

En este capítulo quedaron definidas las características para el desarrollo de la solución, por ejemplo: la adopción del patrón arquitectónico n-capas del cual se obtuvo la separación en cuatro capas lógicas. Se definieron los principales requisitos funcionales y no funcionales con que cuenta el sistema a partir de las historias de usuario. Además se describieron los patrones de diseño utilizados para lograr una correcta estructuración del sistema y se especificaron los detalles del protocolo diseñado(AHP). Se utilizaron diagramas UML para modelar la solución evidenciándose el bajo acoplamiento entre las capas lógicas y la alta cohesión. Por último se mostraron detalles lógicos esenciales en la implementación del servidor BDH en forma de diagramas de flujo.

Capítulo 3

DISEÑO Y REALIZACIÓN DE PRUEBAS

Entre los elementos más importantes en el proceso de desarrollo de software se encuentran las pruebas. A continuación se muestra un compendio de las pruebas realizadas al sistema obtenido como resultado de este trabajo.

3.1. Pruebas realizadas

En la metodología de desarrollo seleccionada, las historias de usuario son la principal fuente de información a la hora de construir las pruebas de aceptación. A una historia de usuario se le puede definir más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento, y no se considera completa hasta que no superan estas pruebas. El objetivo de las mismas es verificar el cumplimiento de los requisitos, y es responsabilidad del cliente comprobar su ejecución para tomar decisiones al respecto.

Por otra parte, las pruebas de sistema tienen como objetivo ejercitar profundamente el producto, comprobándolo de forma global. Esto posibilita alcanzar una visión similar a su comportamiento en el entorno de producción. Uno de los tipos de pruebas de sistema son las pruebas de rendimiento, las cuales consisten en determinar que los tiempos de respuesta estén dentro de los intervalos establecidos en las especificaciones del sistema.

3.1.1. Ambiente de pruebas

Las pruebas del BDH-Server fueron realizadas en un ambiente en el cual se cuenta con una tarjeta Intel Atom que posee un microprocesador Intel Atom a 1100 MHz,

memoria RAM de 1GB, y almacenamiento FLASH[1] de 2GB, de igual forma el BDH-Client fue probado en una PC con microprocesador Intel Core i3 con 4 núcleos a una frecuencia de 2.13 GHz, una memoria RAM de 4 GB y 500 GB de disco duro, ambos sobre la distribución GNU/Linux Debian 7.5 con versión del kernel 3.16.0.

3.1.2. Pruebas de aceptación

Una prueba de aceptación es un escenario de utilización del sistema y el comportamiento que de él se espera, visto desde la perspectiva del cliente, usuario o sistema externo que interactúa con el programa(Delgado Alón y Jiménez López, 2012). A continuación se provee una definición más completa.

Una prueba de aceptación (PA) tiene como propósito demostrar al cliente el cumplimiento de un requisito del software. Las características de una PA son: describe un escenario (secuencia de pasos) de ejecución o uso del sistema desde la perspectiva del cliente, puede estar asociada a un requisito funcional o no funcional, un requisito tiene una o más PA asociadas, las PA cubren desde escenarios típicos/frecuentes hasta los más excepcionales, una PA puede tener infinitas instanciaciones (ejecuciones con valores concretos). El diseño de las instanciaciones y su aplicación es trabajo del probador(Muñoz Pérez, 2010).

En las Tablas 3.1 y 3.2 se pueden detallar algunas de las PA realizadas al sistema, para más detalles ver Anexo 3.2.2.

Caso de prueba: 2
Numero de historia: 2
Nombre: Almacenamiento de muestras de variables
Descripción: Esta prueba se realiza con el objetivo de verificar que una vez generada una muestra de variable esta se almacena satisfactoriamente en la base de datos.
Condiciones de ejecución: Debe haberse gestionado con anterioridad la configuración, y el ID de dicha muestra de variable debe corresponder con el ID de alguna de las variables configuradas, al igual que debe estar corriendo el Recolector y muestreando alguna variable.
Entradas/Pasos de ejecución: Se corre el BDH-Server.
Resultado esperado: Los datos de las muestras de variables deben ser persistidos en la base de datos satisfactoriamente
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.1: Caso de prueba 2 / Historia de usuario 2.

[1]Memoria flash es un medio de almacenamiento no volátil que puede ser borrado y reprogramado eléctricamente.

Caso de prueba: 2
Numero de historia: 3
Nombre: Acceso a información histórica.
Descripción: Esta prueba se realiza con el objetivo de verificar que un cliente se conecta al servidor, establece el modo histórico, solicita muestras de variables en rangos de tiempo y dicha información es recibida correctamente en la aplicación cliente.
Condiciones de ejecución: El servidor debe estar corriendo y escuchando en el IP y puerto configurado, además de que ambos cliente y servidor deben ser accesibles mediante Ethernet.
Entradas/Pasos de ejecución: Se corre el BDH-Server y el BDH-Client y se realiza una petición de información histórica en el BDH-Client.
Resultado esperado: El BDH-Client recibe satisfactoriamente la información referente a la petición hecha.
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.2: Caso de prueba 2 / Historia de usuario 3.

3.1.3. Pruebas de rendimiento

En la ingeniería del software, las pruebas de rendimiento (PR) son las pruebas que se realizan, desde una perspectiva, para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo. También puede servir para validar y verificar otros atributos de la calidad del sistema, tales como la escalabilidad, fiabilidad y uso de los recursos. Las pruebas de rendimiento son un subconjunto de la ingeniería de pruebas, una práctica informática que se esfuerza por mejorar el rendimiento, englobándose en el diseño y la arquitectura de un sistema, antes incluso del esfuerzo inicial de la codificación(Molyneaux, 2009).

En las Tablas 3.3 y 3.4 se pueden detallar algunas de las PR realizadas al sistema, para más detalles ver el Anexo 3.2.2).

3.2. Análisis de los resultados

3.2.1. Pruebas de aceptación

Iteración 1

En la primera iteración se probaron todas las funcionalidades del sistema de las cuales se detectaron no conformidades en los siguientes casos de prueba: *Caso de prueba 1/ Historia de usuario 1* (Ver Tabla 3.14), *Caso de prueba 2/ Historia de usuario 1* (Ver Tabla 3.15), *Caso de prueba 1/ Historia de usuario 2* (Ver Tabla 3.16) y *Caso de prueba 2/ Historia de usuario 2* (Ver Tabla 3.17). La Figura 3.1

Caso de prueba: 1				
Número de historia: 9				
Nombre: Configurar el BDH-Server				
Descripción: Esta prueba se realiza con el objetivo de verificar el tiempo que demora configurándose el servidor BDH a partir del archivo XML.				
Iteraciones:				
No.	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Evaluación**
1	10 variables, 30 alarmas	10 segundos	3 segundos	Satisfactoria
2	50 variables, 150 alarmas	20 segundos	5.5 segundos	Satisfactoria
3	100 variables, 300 alarmas	30 segundos	7.4 segundos	Satisfactoria

Tabla 3.3: Caso de prueba 1 / Historia de usuario 9.

Caso de prueba: 3				
Número de historia: 9				
Nombre: Realizar consulta de datos históricos				
Descripción: Esta prueba se realiza con el objetivo de verificar el tiempo que demora realizar una consulta en el BDH-Client en función del tamaño en MB de la información persistida.				
Iteraciones:				
No.	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Evaluación**
1	1 variable, 10MB de historia	5 segundos	1.54 segundos	Satisfactoria
2	1 variable, 100MB de historia	50 segundos	3.5 segundos	Satisfactoria
3	1 variable, 1000MB de historia	500 segundos	12 segundos	Satisfactoria

Tabla 3.4: Caso de prueba 3 / Historia de usuario 9.

muestra los resultados generales.

Iteración 2

Se planificó una segunda iteración para intentar dar solución a las no conformidades detectadas en la iteración anterior. Luego de la fase de desarrollo se probaron las funcionalidades del sistema mediante los casos de prueba que detectaron no conformidades en la iteración anterior y se detectaron no conformidades en los siguientes casos de prueba: *Caso de prueba 1/ Historia de usuario 1* (Ver Tabla 3.14), *Caso de prueba 2/ Historia de usuario 1* (Ver Tabla 3.15). La Figura 3.1 muestra los resultados generales.

Iteración 3

Se planificó una tercera iteración para intentar dar solución a las no conformidades detectadas en la iteración anterior. Luego de la fase de desarrollo se probaron las funcionalidades del sistema mediante los casos de prueba que detectaron no conformidades en la iteración anterior y no se detectaron no conformidades con lo cual quedaron validadas todas las funcionalidades del sistema. La Figura 3.1 muestra los resultados generales.

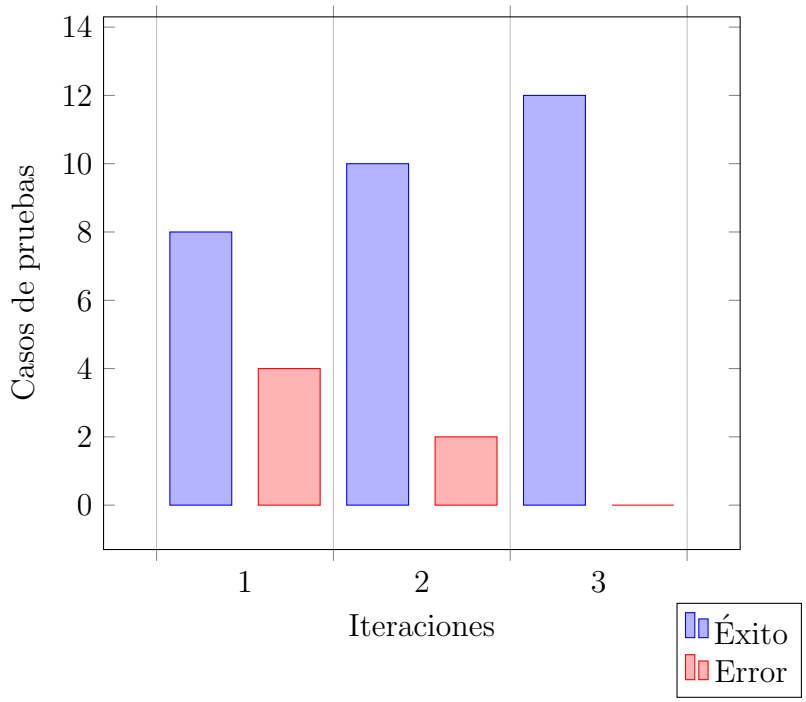

Figura 3.1: Resultado de las PA por cada iteración.

3.2.2. Pruebas de rendimiento

A partir del análisis de las pruebas de rendimiento realizadas se pudo concluir que a mayor cantidad de variables configuradas en el servidor BDH, mayor será el tiempo que tardará en conectarse el cliente BDH con el servidor (Ver Figura 3.2), y también se arribó a la conclusión de que mientras mayor sea el tamaño de la historia almacenada mayor será el tiempo de demora de una consulta en toda la historia y cuyo tiempo para un almacenamiento de 1GB oscila los 12 segundos. Ver Figura 3.3.

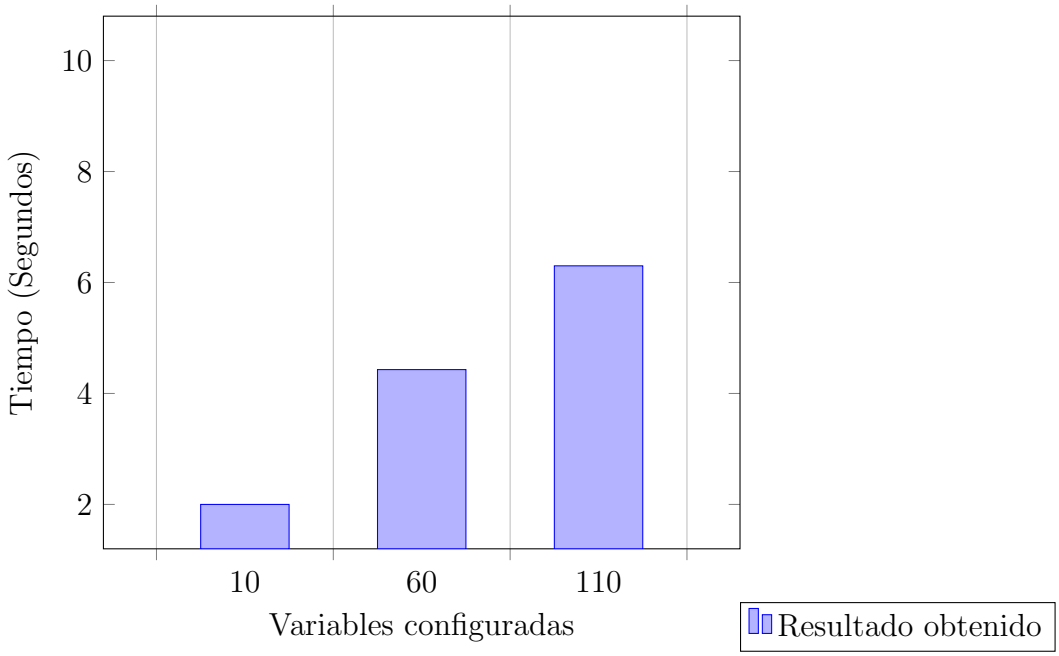

Figura 3.2: Tiempo que demora en conectarse el cliente al servidor.

Conclusiones parciales

A la solución se le aplicaron un total de 15 casos de prueba, 12 de aceptación y tres de rendimiento. Las pruebas de aceptación brindaron una medida de la correcta implementación de los requisitos funcionales y de su completitud. Mientras que las pruebas de rendimiento estuvieron encaminadas a verificar los requisitos de rendimiento del sistema. Se puede concluir que el módulo BDH desarrollado cumple con el objetivo general de la investigación y con las necesidades del cliente.

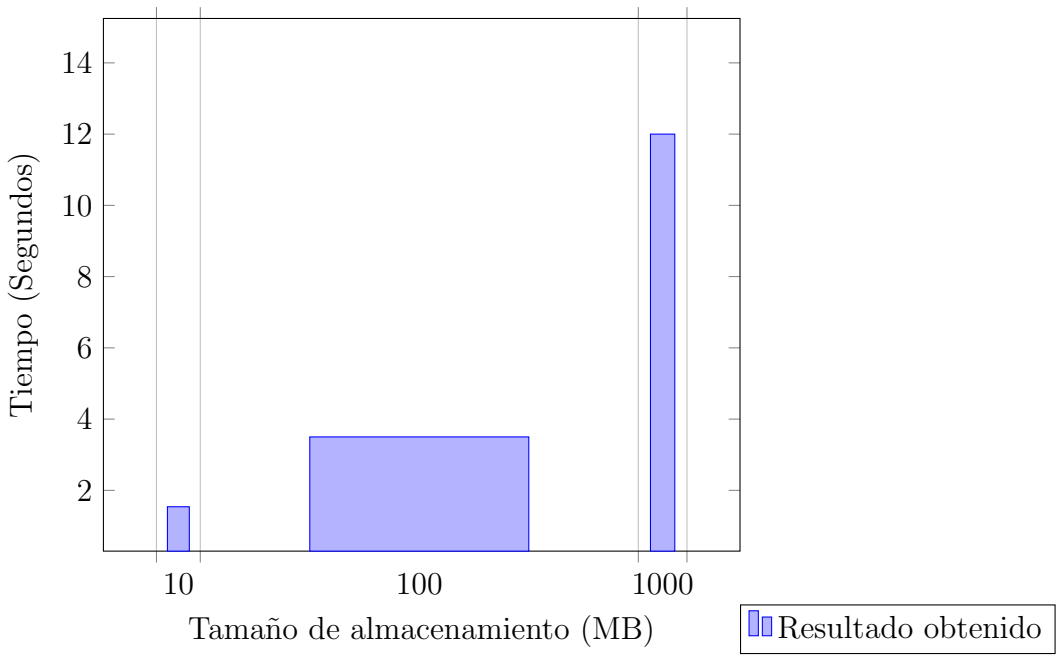

Figura 3.3: Tiempo que demora una consulta histórica.

CONCLUSIONES

Al término de la presente investigación se arriba a las siguientes conclusiones:

- El módulo BDH desarrollado provee el almacenamiento y acceso a los datos de manera eficiente.
- El acceso eficiente a la información se logra utilizando un conjunto de conceptos y tecnologías que se consideraron adecuados para el desarrollo de la solución, entre los que se destacan MsgPack, programación paralela, TCP Socket, entre otros.
- La aplicación cliente desarrollada permite visualizar la información con interfaz entendible por el usuario, la misma permite visualizar las tendencias de las variables muestreadas y consultar la historia de ocurrencias de alarmas.

RECOMENDACIONES

Al concluir la presente investigación se proponen, a manera de recomendaciones, una serie de tareas para ampliar y dar continuidad al trabajo realizado:

- Desarrollar e implementar un mecanismo de compresión para lograr mejorar la eficiencia en el almacenamiento.
- Estudiar algoritmos existentes para compresión de datos online en señales industriales.

REFERENCIAS BIBLIOGRÁFICAS

Osman A.H. y Massoud M.M. Setting up a data acquisition system for spark engines. *1st Annual International Interdisciplinary Conference, AIIC 2013, 24-26 April, Azores, Portugal*, 2013.

Raúl Álvarez Torrico. Máquina de estados finitos. 2015. URL http://tecbolivia.com/index.php/articulos-y-tutoriales-microcontroladores/13-introduccion-a-las-maquinas-de-estado-finito.

Alam Burns y Geoff Davies. *Concurrent programming.* Adison Wesley, 1993. URL https://books.google.com.cu/books?id=hJRQAAAAMAAJ&q=Concurrent+programming&dq=Concurrent+programming&hl=es&sa=X&ei=AY1iVZ2tC5HlsASIwICwCw&redir_esc=y.

O. CHACÓN, D.and DIJORT y J. CASTRILLO. Supervisión y control de procesos. 2001. URL http://upcommons.upc.edu/ocw/diposit/material/23654/23654.pdf.

Oswaldo Daniel Chávez Moreno. Modelo entidad-relación. 2007.

Douglas E. Comer. *Redes globales de información con Internet y TCP/IP.* PRENTICE-HALL, 1996.

Jorge Luis Cordero. *Metodologías ágiles. Proceso unificado ágil (AUP).* 2014.

Abdelaziz De la Horra Diaz. Desarrollo scada guardián del alba especificación de históricos. 2011.

Developers Debian. About debian. 2014. URL https://www.debian.org/intro/about#what.

K. Delgado Alón y A. Jiménez López. Módulo hmi para una tarjeta basada en microcontroladores. 2012.

Luis Muiguel Fernández Sánchez. *Sistemas de adquisición de datos*. ICIMAF, 2009a.

Luis Muiguel Fernández Sánchez. Sistemas de medición. 2009b.

Erich Gamma. *Design Patterns*. Addison-Wesley, 1994.

Craig Larman. *UML y Patrones*. Pearson Education, 2003.

Cesar D.L.T. Llorente. *Guía de arquitectura N-Capas orientada al dominio con .NET 4.0*. Krasis PRESS, 2011.

Merriam-Webster. Wwwebster dictionary. 2000. URL http://www.m-w.com/.

L.E.C Meza. *SCADA Systems and Telemetry*. Atlantic Internatinal University, 2007.

Ian Molyneaux. *The Art of Application Performance Testing*. O'Reilly, 2009.

D. MONTERO, D. B. BARRANTES, y J. M. QUIRS. Introducción a los sistemas de control supervisor y de adquisición de datos. 2004. URL http://www.infoplc.net/Documentacion/Docu_SCADA/infpPLC_net_Introduccion_Sistemas_SCADA.pdf.

MsgPack. Msgpack: Serialization framework. 2015. URL www.msgpack.org.

Alvaro Muñoz Pérez. Gestión de requisitos dirigida por pruebas de aceptación. 2010.

T. Nieva y A. Wegmann. A conceptual model for remote data acquisition systems. *Proceedings of the 19th International Conference on Conceptual Modeling*, 2000.

OMG. Introduction to uml. 2014. URL http://www.omg.org/gettingstarted/what_is_uml.htm.

Michael Owens. *The definitive guide of SQLite*. Apress, 2006.

Aquilino Rodríguez Penin. *Sistemas de visualización industrial*. MARCOMBO, S.A., 2007.

Roger S Pressman. *Ingeniería del software. Un enfoque práctico*. McGraw-Hill Interamericana Editores S.A., 2010.

Qt-Creator. Introducing qt 5. 2014. URL http://qt-project.org/wiki/QtCreatorWhitepaper.

Primitivo Reyes Aguilar. *Análisis de los Sistemas de Medición (MSA)*. MSA, 2007.

Stefan Ritt. Maximum integration data acquisition system. 2015. URL https://midas.triumf.ca/MidasWiki/index.php/Main_Page.

SlideShare. Diagrama de clases. 2013. URL http://es.slideshare.net/nedowwhaw/diagrama-de-clases-16208245.

SPARX. Diagrama de despliegue. 2007. URL http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html.

Bjarne Stroustrup. *The Design and Evolution of C++*. 1994.

solution Inc. Subnet. Historian. 2015. URL http://www.subnet.com/resources/dictionary/Historian.aspx.

Johan Thelin. *Fundation of Qt Development*. Apress, 2007.

VisualParadigm. Visualparadigm for uml. 2014. URL http://www.visual-paradigm.com.

ANEXO A. HISTORIAS DE USUARIO

Historia de usuario	
Numero: 1	**Usuario:** Línea Sistemas Embebidos
Nombre: Gestión de la configuración	
Prioridad en el negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 5	**Iteración asignada:** 1
Programador Responsable: Randy Mujica Díaz	
Descripción: Permitir cargar los parámetros de configuración del módulo BDH como son: las variables configuradas, alarmas configuradas y espacio disponible para el almacenamiento además del puerto donde escuchará el servidor.	
Observaciones:	

Tabla 3.5: Historia de usuario 1.

Historia de usuario	
Numero: 2	**Usuario:** Línea Sistemas Embebidos
Nombre: Almacenamiento local de muestras de variables y alarmas	
Prioridad en el negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 5	**Iteración asignada:** 1
Programador Responsable: Randy Mujica Díaz	
Descripción: Lograr almacenar de manera local(o sea en la misma estación de trabajo donde se ejecuta la aplicación) las muestras de variables y ocurrencias de alarmas, además este almacenamiento debe realizarse de forma que se haga uso eficiente del espacio de disco utilizado y se minimice el tiempo necesario para consultar la información almacenada.	
Observaciones:	

Tabla 3.6: Historia de usuario 2.

Historia de usuario	
Numero: 3	**Usuario:** Línea Sistemas Embebidos
Nombre: Acceso a la información remotamente	
Prioridad en el negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 5	**Iteración asignada:** 1
Programador Responsable: Randy Mujica Díaz	
Descripción: Permitir el acceso a la información de tiempo real y a la históricamente persistida desde otra estación de trabajo distinta a donde se encuentra almacenada, para ello la comunicación debe realizarse mediante Ethernet.	
Observaciones:	

Tabla 3.7: Historia de usuario 3

Historia de usuario	
Numero: 4	**Usuario:** Línea Sistemas Embebidos
Nombre: Aplicación de prueba con interfaz entendible por el usuario	
Prioridad en el negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 5	**Iteración asignada:** 1
Programador Responsable: Randy Mujica Díaz	
Descripción: Realizar una aplicación que le permita a un usuario ubicado en una estación de trabajo distinta a donde se encuentra almacenada la información, acceder a la misma y visualizarla de una manera entendible por él para poder ser analizada.	
Observaciones:	

Tabla 3.8: Historia de usuario 4

Historia de usuario	
Numero: 5	**Usuario:** Línea Sistemas Embebidos
Nombre: Requisitos de software	
Descripción: Se debe poseer un sistema operativo Linux que consuma pocos recursos de hardware y además cuente con los siguientes módulos: Qt 5.* o superior , Xorg server 1.7.5, SQLite 3 y MessagePack.	
Observaciones:	

Tabla 3.9: Historia de usuario 5

Historia de usuario	
Numero: 6	**Usuario:** Línea Sistemas Embebidos
Nombre: Requisitos de hardware	
Descripción: Para la ejecución del BDH-Server se debe contar con una tarjeta basada en microcontroladores en la que se pueda empotrar un sistema GNU/Linux y posea las siguientes prestaciones (Requisitos mínimos): ▪ Frecuencia del CPU: 1100 MHz. ▪ Memoria RAM: 1 GB. ▪ Puertos: 1 Ethernet y 1 SD (2 GB). Por otra parte para la ejecucin del BDH-Client se debe disponer de un ordenador que tenga como mínimo las prestaciones siguientes: ▪ Procesador: Intel Core i3. ▪ Frecuencia del CPU: 2.13 GHz. ▪ Memoria estática: 500 GB. ▪ Memoria dinámica: 4 GB.	
Observaciones:	

Tabla 3.10: Historia de usuario 6

Historia de usuario	
Numero: 7	**Usuario:** Línea Sistemas Embebidos
Nombre: Requisitos en el diseño e implementación	
Descripción: Para el desarrollo de la solución se definen una serie de restricciones: ▪ Lenguaje de programación: C++. ▪ Framework: Qt 5.* ▪ Entorno integrado de desarrollo: Qt Creator. ▪ SGBD SQLite. ▪ Framework de serialización MessagePack. ▪ TCP-Socket como mecanismo de IPC.	
Observaciones:	

Tabla 3.11: Historia de usuario 7

Historia de usuario	
Numero: 8	**Usuario:** Línea Sistemas Embebidos
Nombre: Requisitos de desempeño	
Descripción: El servidor BDH (BDH-Server) debe poder atender múltiples conexiones de manera simultánea hasta un máximo de 10, las operaciones de adquisición tendrán máxima prioridad y las mismas no podrán ser interrumpidas, además se debe garantizar el uso justo y eficiente del espacio de disco configurado para ser usado en el almacenamiento.	
Observaciones:	

Tabla 3.12: Historia de usuario 8

Historia de usuario	
Numero: 9	**Usuario:** Línea Sistemas Embebidos
Nombre: Requisitos de rendimiento	
Descripción: Estos requisitos describen los valores que indican el adecuado comportamiento del sistema para las exigencias del cliente, las cuales consisten en el manejo como máximo de 100 variables y como máximo 10 alarmas por variable. Se define 30 segundos como máximo para que el BDH-Server se configure, 15 segundos como máximo tiempo que debe tardar en conectarse el BDH-Client al BDH-Server y 500 segundos como tiempo máximo que debe tardar una consulta histórica de las muestras de las variables y alarmas.	
Observaciones:	

Tabla 3.13: Historia de usuario 9

ANEXO B. CASOS DE PRUEBAS

Caso de prueba: 1
Numero de historia: 1
Nombre: Cargar la configuración
Descripción: Esta prueba se realiza con el objetivo de verificar que se carga en memoria correctamente los datos de configuración a partir del archivo XML generado por el HMI-Editor
Condiciones de ejecución: Debe existir un archivo XML con al menos una variable configurada y el espacio de disco a usar para el almacenamiento.
Entradas/Pasos de ejecución: Se copia manualmente el archivo XML y se corre el BDH-Server
Resultado esperado: Los datos de configuración deben ser cargados en memoria satisfactoriamente
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.14: Caso de prueba 1 / Historia de usuario 1

Caso de prueba: 2
Numero de historia: 1
Nombre: Persistir la configuración
Descripción: Esta prueba se realiza con el objetivo de verificar que luego de cargados en memoria correctamente los datos de configuración a partir del archivo XML generado por el HMI-Editor, los mismos sean persistidos en la base de datos correctamente.
Condiciones de ejecución: Debe existir un archivo XML con al menos una variable configurada y el espacio de disco a usar para el almacenamiento y el mismo debe haber sido cargado en memoria satisfactoriamente.
Entradas/Pasos de ejecución: Se copia manualmente el archivo XML y se corre el BDH-Server
Resultado esperado: Los datos de configuración deben ser persistidos en la base de datos satisfactoriamente
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.15: Caso de prueba 2 / Historia de usuario 1

Caso de prueba: 1
Numero de historia: 2
Nombre: Almacenamiento de ocurrencias de alarmas
Descripción: Esta prueba se realiza con el objetivo de verificar que una vez generada una ocurrencia de alarma esta se almacena satisfactoriamente en la base de datos.
Condiciones de ejecución: Debe haberse gestionado con anterioridad la configuración, y el ID de dicha ocurrencia de alarma debe corresponder con el ID de alguna de las alarmas configuradas, al igual que debe estar corriendo el Recolector y generando alarmas.
Entradas/Pasos de ejecución: Se corre el BDH-Server.
Resultado esperado: Los datos de las ocurrencias de alarmas deben ser persistidos en la base de datos satisfactoriamente
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.16: Caso de prueba 1 / Historia de usuario 2

Caso de prueba: 2
Numero de historia: 2
Nombre: Almacenamiento de muestras de variables
Descripción: Esta prueba se realiza con el objetivo de verificar que una vez generada una muestra de variable esta se almacena satisfactoriamente en la base de datos.
Condiciones de ejecución: Debe haberse gestionado con anterioridad la configuración, y el ID de dicha muestra de variable debe corresponder con el ID de alguna de las variables configuradas, al igual que debe estar corriendo el Recolector y muestreando alguna variable.
Entradas/Pasos de ejecución: Se corre el BDH-Server.
Resultado esperado: Los datos de las muestras de variables deben ser persistidos en la base de datos satisfactoriamente
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.17: Caso de prueba 2 / Historia de usuario 2

Caso de prueba: 1
Numero de historia: 3
Nombre: El servidor (BDH-Server) escucha peticiones TCP.
Descripción: Esta prueba se realiza con el objetivo de verificar que el servidor (BDH-Server) escucha peticiones TCP en el IP y puerto configurados.
Condiciones de ejecución: Debe correrse como root.
Entradas/Pasos de ejecución: Se corre el BDH-Server.
Resultado esperado: El servidor (BDH-Server) se encuentra escuchando conexiones TCP en el IP y puerto configurados.
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.18: Caso de prueba 1 / Historia de usuario 3

Caso de prueba: 2
Numero de historia: 3
Nombre: Acceso a información histórica.
Descripción: Esta prueba se realiza con el objetivo de verificar que un cliente se conecta al servidor, establece el modo histórico, solicita muestras de variables en rangos de tiempo y dicha información es recibida correctamente en la aplicación cliente.
Condiciones de ejecución: El servidor debe estar corriendo y escuchando en el IP y puerto configurado, además de que ambos cliente y servidor deben ser accesibles mediante Ethernet.
Entradas/Pasos de ejecución: Se corre el BDH-Server y el BDH-Client y se realiza una petición de información histórica en el BDH-Client.
Resultado esperado: El BDH-Client recibe satisfactoriamente la información referente a la petición hecha.
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.19: Caso de prueba 2 / Historia de usuario 3

Caso de prueba: 3
Numero de historia: 3
Nombre: Acceso a información de tiempo real.
Descripción: Esta prueba se realiza con el objetivo de verificar que un cliente se conecta al servidor, establece el modo tiempo real, solicita que se le empiece a enviar en tiempo real las muestras de algunas de las variables configuradas y ocurra el resultado esperado .
Condiciones de ejecución: El servidor debe estar corriendo y escuchando en el IP y puerto configurado, además de que ambos cliente y servidor deben ser accesibles mediante Ethernet.
Entradas/Pasos de ejecución: Se corre el BDH-Server y el BDH-Client y se realiza una petición de información de tiempo real en el BDH-Client.
Resultado esperado: El BDH-Client empieza a recibir información de tiempo real referente a las variables solicitadas.
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.20: Caso de prueba 3 / Historia de usuario 3

Caso de prueba: 4
Numero de historia: 3
Nombre: Detener el envío de muestras en tiempo real.
Descripción: Esta prueba se realiza con el objetivo de verificar que un cliente que se encuentra recibiendo muestras en tiempo real pida al servidor que deje de enviarle dichas muestras y ocurra el resultado esperado .
Condiciones de ejecución: El servidor debe estar corriendo y escuchando en el IP y puerto configurado, además de que ambos cliente y servidor deben ser accesibles mediante Ethernet.
Entradas/Pasos de ejecución: Se corre el BDH-Server y el BDH-Client y se solicita detener el envío de muestras en tiempo real en el BDH-Client.
Resultado esperado: El BDH-Client deja de recibir información de tiempo real referente a las variables solicitadas.
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.21: Caso de prueba 4 / Historia de usuario 3

Caso de prueba: 1
Numero de historia: 4
Nombre: Visualización de la información de manera entendible .
Descripción: Esta prueba se realiza con el objetivo de verificar que un usuario pueda hacer peticiones al BDH-Server mediante el BDH-Client y ocurra el resultado esperado.
Condiciones de ejecución: El servidor debe estar corriendo y escuchando en el IP y puerto configurado, además de que ambos cliente y servidor deben ser accesibles mediante Ethernet y el BDH-Client debe haberse conectado al BDH-Server y establecido un modo.
Entradas/Pasos de ejecución: Hacer peticiones históricas y de tiempo real en el BDH-Client.
Resultado esperado: Se visualice en el BDH-Client la información solicitada de una manera entendible por el usuario para poder ser analizada.
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.22: Caso de prueba 1 / Historia de usuario 4

Caso de prueba: 1
Numero de historia: 8
Nombre: Crear un nuevo bloque de datos
Descripción: Esta prueba se realiza con el objetivo de verificar que se crea un nuevo bloque de datos una vez se llene el bloque donde actualmente se está persistiendo las muestras de variables.
Condiciones de ejecución: Debe haberse agotado el espacio en el bloque actual de muestras de variables, al igual que debe estar corriendo el Recolector y muestreando alguna variable.
Entradas/Pasos de ejecución: Se corre el BDH-Server.
Resultado esperado: Se crea un nuevo bloque de información el cual no es más que un nuevo archivo de base de datos SQLite, y se agrega en la tabla *catalog* de la base de datos principal (main.db) la información del bloque anterior.
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.23: Caso de prueba 1 / Historia de usuario 8

Caso de prueba: 2
Numero de historia: 8
Nombre: Borrar el bloque de datos más antiguo
Descripción: Esta prueba se realiza con el objetivo de verificar que se borra la información del bloque de datos más antiguo una vez se llene el espacio de disco configurado para el almacenamiento, y una vez borrada, este bloque pasa a ser el bloque actual donde se persiste la información.
Condiciones de ejecución: Debe haberse agotado el espacio en el bloque actual y no debe quedar suficiente para crear un nuevo bloque de datos, al igual que debe estar corriendo el Recolector y muestreando alguna variable.
Entradas/Pasos de ejecución: Se corre el BDH-Server.
Resultado esperado: Se borra la información del bloque de datos más antiguo, pasa este a ser el bloque actual donde se persiste la información y se actualizan los rangos de tiempo del bloque anterior y del actual en la tabla *catalog* de la base de datos principal (main.db).
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.24: Caso de prueba 2 / Historia de usuario 8

Caso de prueba: 3
Numero de historia: 8
Nombre: Creación de un hilo por cada conexión en el servidor
Descripción: Esta prueba se realiza con el objetivo de verificar que una vez arribe una nueva conexión al servidor, esta sea atendida y procesada en un hilo de ejecución aparte al hilo principal.
Condiciones de ejecución: Debe haberse conectado al menos un cliente al servidor.
Entradas/Pasos de ejecución: Se corre el BDH-Server y BDH-Client y se conecta el BDH-Client con el BDH-Server.
Resultado esperado: Se levanta un nuevo hilo de ejecución hijo del proceso BDH-Server en el cual se atiende la nueva conexión.
Evaluación de la prueba: Prueba satisfactoria

Tabla 3.25: Caso de prueba 3 / Historia de usuario 8

Caso de prueba: 1				
Número de historia: 9				
Nombre: Configurar el BDH-Server				
Descripción: Esta prueba se realiza con el objetivo de verificar el tiempo que demora configurándose el servidor BDH a partir del archivo XML.				
Iteraciones:				
No.	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Evaluación**
1	10 variables, 30 alarmas	10 segundos	3 segundos	Satisfactoria
2	50 variables, 150 alarmas	20 segundos	5.5 segundos	Satisfactoria
3	100 variables, 300 alarmas	30 segundos	7.4 segundos	Satisfactoria

Tabla 3.26: Caso de Prueba 1 / Historia de usuario 9

Caso de prueba: 2				
Número de historia: 9				
Nombre: Conectar el BDH-Client al BDH-Server				
Descripción: Esta prueba se realiza con el objetivo de verificar el tiempo que demora en conectarse el cliente BDH al servidor en función de la cantidad de variables configuradas.				
Iteraciones:				
No.	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Evaluación**
1	10 variables	5 segundos	2 segundos	Satisfactoria
2	60 variables	10 segundos	4.43 segundos	Satisfactoria
3	110 variables	15 segundos	6.3 segundos	Satisfactoria

Tabla 3.27: Caso de Prueba 2 / Historia de usuario 9

Caso de prueba: 3				
Número de historia: 9				
Nombre: Realizar consulta de datos históricos				
Descripción: Esta prueba se realiza con el objetivo de verificar el tiempo que demora realizar una consulta en el BDH-Client en función del tamaño en MB de la información persistida.				
Iteraciones:				
No.	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Evaluación**
1	1 variable, 10MB de historia	5 segundos	1.54 segundos	Satisfactoria
2	1 variable, 100MB de historia	50 segundos	3.5 segundos	Satisfactoria
3	1 variable, 1000MB de historia	500 segundos	12 segundos	Satisfactoria

Tabla 3.28: Caso de Prueba 3 / Historia de usuario 9