Universidad de las Ciencias Informáticas

Facultad 5

Título: Manejador para el intercambio de información con dispositivos que se comuniquen a través del protocolo DNP3.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TITULO DE INGENIERO EN CIENCIAS INFORMATICAS

Autor

Amides Rodríguez Rodríguez.

Tutor

Ing. Rubén Gómez Johnson.

Ciudad de la Habana

Junio 2010

Declaración de Autoría.

Declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los __ días del mes de _____ de 2010.

_____

Firma del Autor

(Amides Rodríguez Rodríguez)

_____

Firma del tutor

(Ing. Rubén Gómez Johnson)

Datos de contacto

Ing. Rubén Gómez Johnson

Graduado con el título de Ingeniero en Ciencias Informáticas en el 2008 en la Universidad de las Ciencias Informáticas.

Email: rjohnson@uci.cu

A mi preciosa madre, Juana Rodríguez Molinet

A mis abuelos, Estelvina e Idael

A mis hermanos queridos y al resto de mi familia

Agradecimientos

Quiero agradecer a mi mamá por haber sacrificado tantas cosas en su vida para que este momento llegara a la mía, por ser tan paciente y comprensiva con mis decisiones, por haberme inculcado la necesidad de estudiar y dar lo mejor de mí. Deseo agradecer de manera especial a mi abuelita, por haber confiado en mí cuando todos dudaban, por darme fuerzas y aconsejarme antes de cada prueba, por enseñarme los principios que hoy me rigen, por enseñarme a amar lo que ahora amo, por ser ejemplo de integridad y dedicación.

Quiero dar gracias a mi familia en general, por permitirme crecer rodeado de amor y bondad, por apoyarme, por corregir mis faltas y por el respeto que demuestran día a día hacia mi persona y hacia lo que hago. Gracias a mis amigos de aquí y los del lugar donde nací por ser fieles y tolerantes.

Deseo agradecer al proyecto SCADA, especialmente a la línea de drivers por formarme en su seno, y por hacerme parte de esta gran idea que seguimos construyendo. Gracias a mi tutor, el ingeniero Rubén Gómez Johnson por guiarme a lo largo del desarrollo de esta tesis, a Yanet Nieto y a Maikel Pérez por su apoyo incondicional y decisivo que nunca olvidaré.

Resumen

En los procesos a ser supervisados por un SCADA intervienen dispositivos (PLC) cuya comunicación es llevada a cabo por medio de estándares que establecen un conjunto de reglas a seguir y una forma de organizar la información que se transfiere, o sea, a través de protocolos de comunicación.

En varias de las plantas en las que se pretende desplegar el SCADA Guardián del ALBA, existe una determinada cantidad de PLC que usan para su comunicación el protocolo DNP3. Como dicho sistema no cuenta con una solución para acceder a la información almacenada en dispositivos con esta característica, se ha realizado un estudio al mismo y específicamente a su variante serial con el objetivo de brindar al Guardián del ALBA un manejador, mediante el cual, la supervisión de los procesos antes descritos sea posible.

Los manejadores en el Guardián del ALBA deben emitir un evento al recibir y enviar mensajes para que este sistema pueda tener la posibilidad de detectar y corregir errores en la comunicación con los dispositivos que supervisa. Con el objetivo de posibilitar que el Sniffer (herramienta encargada del monitoreo de la comunicación) sea capaz de analizar tramas con formato DNP3, se ha desarrollado una extensión al mismo con vistas a proporcionarle dicha funcionalidad.

La extensión DNP3 para el Sniffer facilita además que el manejador resultante de esta tesis sea ajustado de forma mucho más eficiente en su etapa de implementación y prueba.

Palabras clave: DNP3, Guardián del ALBA, Manejador, PLC, SCADA, Sniffer.

Índice

Introducción.

Con la continua evolución del Control Automático surgen los mecanismos de control. Inicialmente fueron sistemas de telemetría que proporcionaban reportes periódicos de los procesos que estaban siendo controlados. En la actualidad las computadoras asumen un importante papel en la recolección de datos. Trajeron consigo la utilización de los comandos de control y la representación de los datos en pantallas de video, los cuales hacen mucho más eficaces a estos sistemas y permitieron implementarle funcionalidades más complejas.

Los sistemas de Supervisión y Control de Adquisición de Datos (SCADA) surgen como consecuencia directa de estos avances. Fueron concebidos para evitar, prevenir y alertar situaciones inesperadas en la industria.

Cuba posee experiencias en el desarrollo sistemas SCADA. El EROS, por ejemplo, ha sido utilizado en varias ramas de la economía de nuestro país. Una de las instituciones que lo emplea es la Unión Nacional Eléctrica, permitiendo a los despachos provinciales, al despacho nacional, así como a cualquier usuario autorizado, la supervisión de las principales variables de los equipos de generación distribuida. Gracias a sus capacidades se ha logrado que zonas con gran cantidad de emplazamientos instalados (Holguín y Pinar del Río) y donde son mayores los problemas en la última milla (soporte de comunicaciones telefónicas) se haya podido supervisar en tiempo real las operaciones de los grupos electrógenos. También se cuenta con el sistema SCADA Sispro, producido por CEDAI Villa Clara.

Gracias a las buenas relaciones bilaterales entre Cuba y Venezuela, y dada la necesidad de este último de conseguir soberanía tecnológica, surge en la Universidad de las Ciencias Informáticas (UCI) el proyecto SCADA Nacional. Actualmente se le conoce como el Guardián del ALBA, nombre que tributa a la idea de su futuro empleo en los demás países que forman parte de esta organización. Entre sus múltiples líneas de desarrollo se encuentra la de manejadores (Drivers). Este equipo tiene la responsabilidad de que los datos obtenidos en el campo sean accedidos desde el sistema. Por tanto se encarga del manejo de las comunicaciones con los dispositivos que intervienen en los procesos que están siendo supervisados.

Los dispositivos antes mencionados se comunican a través de estándares que establecen un conjunto de reglas a seguir y una forma de organizar la información que

se transfiere, o sea, a través de protocolos de comunicación. Actualmente, la línea emplea varios en el acceso a diversos dispositivos, entre ellos se encuentran determinadas variantes del Modbus (ASCII, RTU, TCP/IP), BSAP, Ethernet-IP, ABEthernet entre otros

En varias de las plantas en las que se quiere desplegar el Guardián del ALBA, se cuenta con un número determinado de autómatas que se comunican a través de DNP3. El Guardián del ALBA no cuenta con una solución para el intercambio de información con dispositivos de este tipo, por lo que actualmente no es posible supervisar en toda su extensión a las plantas que poseen estas características.

De la situación problémica anterior se analizó el siguiente problema científico**:**

¿Cómo lograr la comunicación entre el sistema de Supervisión y Control de Adquisición de Datos, Guardián del ALBA, y dispositivos a través del protocolo de comunicación DNP3?

Para darle solución a dicho problema se abordó como **objeto de estudio**, manejadores para sistemas de Supervisión y Control de Adquisición de Datos. Como **objetivo general:** desarrollar un manejador que permita la comunicación entre el Guardián del ALBA y dispositivos a través del protocolo DNP3. Como **campo de acción**: mecanismos y funcionalidades para el intercambio con dispositivos que se comuniquen a través de DNP3.

Para dar cumplimiento al objetivo planteado se llevaron a cabo las siguientes tareas de investigación:

- Estudio de tecnologías para el acceso a datos a través del puerto serie más utilizadas en la comunicación con dispositivos externos, para seleccionar la más adecuada a emplear en el desarrollo del manejador.
- Determinación y definición de estructuras compatibles con la mayoría de los sistemas operativos más utilizados que puedan ser aplicadas en la solución.
- Estudio de la bibliografía referente a la especificación del protocolo DNP3 para identificar las características a tener en cuenta en su posterior implementación.

- Análisis y Diseño de la capa de protocolo del manejador.
- Implementación de la capa de protocolo del manejador.
- Análisis y Diseño de la capa de Driver del manejador.
- Implementación de la capa de Driver del manejador.
- Desarrollo de una extensión para el Sniffer, con vistas al monitoreo de la comunicación entre los dispositivos que se comunican a través de DNP3 y el sistema.
- Realización de pruebas al manejador y a su extensión para el Sniffer.

Se utilizaron varios métodos científicos de investigación como: el **Analítico-Sintético,** que permitió conocer los principales fundamentos y teorías relacionadas con el protocolo de comunicación DNP3. También fue utilizado el de **Modelación,** ya que se realizaron diagramas y modelos que han permitido estructurar teóricamente el sistema, y el **Histórico-Lógico**, con el objetivo de conocer la evolución de las teorías y tendencias del intercambio de mensajes DNP3 sobre comunicación serial.

El presente documento ha sido dividido en tres capítulos, cada uno compuesto por epígrafes y sub-epígrafes temáticos en los cuales se abordan temas particulares. En el primer capítulo fueron definidos los conceptos fundamentales, las tecnologías que más se utilizan en la solución de problemas similares al planteado y se describen las herramientas y metodologías vinculadas con el desarrollo del software. En el segundo capítulo se determina la tecnología de acceso al medio físico que luego fue empleada en la solución, se describe la Interfaz Genérica, cuyo cumplimiento obligatorio es una de las restricciones más importantes que se tuvieron en cuenta en el desarrollo, y se exponen todas las actividades relacionadas con el diseño de la solución.

Por otro lado, el capítulo tres está dedicado a la implementación y pruebas de la biblioteca resultante.

Capítulo 1: Fundamentación teórica

En el siguiente capítulo se hace referencia a las teorías relacionadas con el intercambio de mensajes entre un sistema SCADA y dispositivos que se comunican a través del protocolo DNP3 en su variante serial.

1.1 Sistemas SCADA

Los sistemas SCADA son aplicaciones de software, diseñadas con la finalidad de controlar y supervisar procesos a distancia. Se basan en la adquisición de datos de los procesos remotos. (1)

Como plantea el profesor de la Universidad Nacional de Rosario (Argentina), Juan Pablo Ferrari en su monografía “Sistemas de Control Distribuido”, los sistemas SCADA están diseñados para funcionar sobre computadoras en el control y supervisión de la producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, etc.) y controlando el proceso de forma automática.

Sus componentes incluyen varios subsistemas. Un ejemplo de esto puede observarse en los módulos de adquisición de datos, ya que la misma puede estar a cargo de un PLC (Controlador Lógico Programable), quien por medio de un protocolo determinado envía señales a las estaciones remotas y así logra la supervisión de la misma, y también podría estar a cargo de una computadora que realice la recolección a través de un hardware especializado y luego esta información se transmita hacia un equipo de radio vía su puerto serial para ser emitida la información, modulada en ondas radiales.

En estos sistemas el operador puede visualizar en las pantallas de cada una de las estaciones remotas los estados de la misma, tomar acciones físicas sobre algún equipo lejano, entre muchas otras operaciones.

La comunicación se realiza mediante buses especiales o redes LAN y los procesos de supervisión y control generalmente son realizados en tiempo real, aunque esto depende de las necesidades para las que haya sido concebido el SCADA.

1.2 Modelo de referencia OSI

El modelo de referencia OSI (Open Systems Interconnect) fue creado por la ISO en 1977 con el propósito de abrir la comunicación entre diferentes sistemas sin recurrir a

cambios a la lógica y fundamentos del hardware y software. No es un protocolo, es un modelo para diseñar una arquitectura de red que sea flexible, robusta e interoperable.

El modelo OSI está construido en 7 capas:

Ilustración 1 :Capas del Modelo OSI

- **Capa Física**: Se ocupa de la definición de las características de un canal de comunicación así como la transmisión de bits a través del mismo.
- **Capa de Enlace de Datos**: Ensambla los bits de la capa física en grupos de tramas (protocolos de red) y asegura su correcto envío. Además se ocupa de la verificación y corrección de errores de la capa física.
- **Capa de Red:** Es responsable de que los paquetes enviados lleguen a su destino. Sus responsabilidades específicas son el direccionamiento lógico y el enrutamiento.
- **Capa de Transporte:** Se encarga del manejo de los mensajes completos entre la fuente y el destino. Reconoce relaciones de dependencias entre diferentes mensajes simples por lo que asegura que el mensaje completo arribe intacto y en orden, supervisando el control de flujo y control de error al nivel de la fuente-destino.
- **Capa de Sesión:** Es controladora de diálogos en la red. Establece, mantiene y sincroniza la interacción de los sistemas.
- **Capa de presentación:** Se encarga de la sintaxis y la semántica de la

información intercambiada entre dos sistemas.

- **Capa de aplicación:** Provee de las interfaces de usuario y soporte para servicios tales como correo electrónico, trasferencia de archivos, administración de bases de datos compartidas y otros tipos de servicios distribuidos.

1.3 Comunicación a través del puerto serie, estándar RS-232.

El puerto serial es un dispositivo muy extendido. Debido a que el estándar se mantiene desde hace muchos años, la institución de normalización americana (EIA) ha escrito la norma RS-232-C que regula el protocolo de transmisión de datos, el cableado, las señales eléctricas y los conectores en los que debe basarse una conexión RS-232.
El estándar RS-232 consiste en un conector de tipo DB-25 de 25 pines, aunque es normal encontrar la versión de 9 pines DB-9, más barato e incluso más extendido para ciertos tipos de periféricos. En cualquier caso las PC no utilizan más de 9 pines en el conector DB-25. Las señales con las que trabaja son digitales, +12V (0 lógico)
y -12V (1 lógico). Dependiendo de la velocidad de transmisión empleada, es posible tener cables de hasta 15 metros. Cada pin puede ser de entrada o salida, teniendo una función específica cada uno de ellos. (3)

Un bit se transmite durante cada período, por tanto la velocidad en baudios (baudrate) de un puerto serie de la PC es igual al número de bits por segundo que se transmiten o reciben. Para enviar información codificada de esta manera, el transmisor y receptor deben estar a la misma frecuencia y sincronizados. Cada carácter se envía en una armazón, que consiste en un bit “0” llamado un bit de inicio, seguido por el carácter mismo, y luego, opcionalmente, un bit de paridad, y después un bit “1” llamado bit de paro. La lógica del bit bajo de inicio le dice al receptor que está empezando una armazón, y la lógica del bit alto de paro denota el final de la armazón. (3)

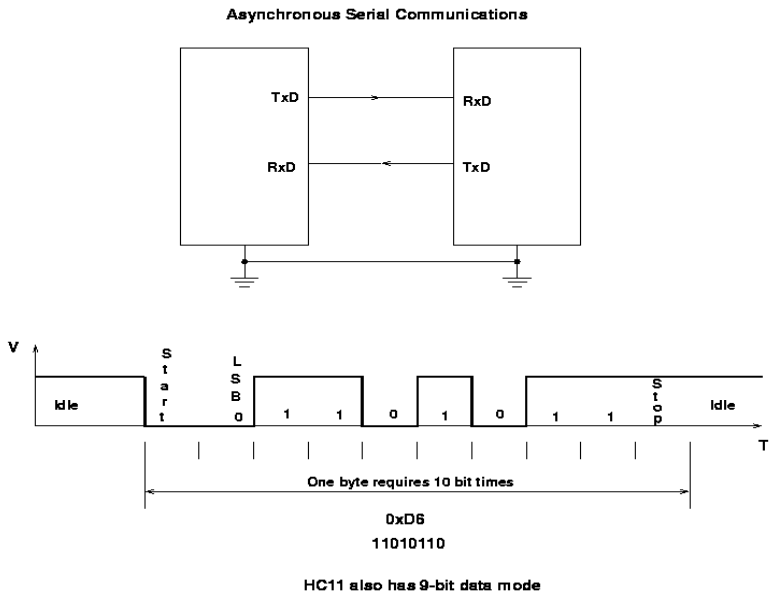

Ilustración 2: Comunicación serial asíncrona

1.4 Comunicación a través del puerto serie, estándar RS-485.

RS-485 es uno de los estándares más versátiles de comunicación por el puerto serie. Actualmente es muy utilizado en la adquisición de datos y control de aplicaciones en las que múltiples nodos se comunican entre sí.

Con el rs-485 se logra la comunicación a más largas distancias y con mayores tasas de bits. A través de él se puede transmitir información de manera fiable a 1200 m. Alcanza una velocidad de transmisión mucho mayor que otros estándares, incluyendo el estándar RS-232. Actualmente se producen conductores RS-485 que pueden llegar a velocidades de 35 Mbps.

1.5 Controladores lógicos programables

Un autómata programable industrial (API) o Programable Logic Controller (PLC), es un equipo electrónico, programable en lenguaje no informático, diseñado para controlar en tiempo real y en ambiente de tipo industrial, procesos secuenciales.

Un PLC trabaja en base a la información recibida por los captadores y el programa lógico interno, actuando sobre los accionadores de la instalación. (5)

Ilustración 3: Controladores Lógicos Programables

1.5.1 Campos de aplicación

Los PLC tienen un amplio campo de acción. Generalmente se utilizan en instalaciones donde son necesarios los procesos de control, automatización etc. Por tanto, su aplicación abarca desde procesos de fabricación industriales de cualquier tipo a transformaciones industriales, control de instalaciones, entre otros.

Sus reducidas dimensiones, la extremada facilidad de su montaje, la posibilidad de almacenar los programas para su posterior y rápida utilización, hace que su eficacia se aprecie fundamentalmente en procesos en que se producen necesidades tales como:

- ❖ Espacio reducido.
- ❖ Procesos de producción cambiantes.
- ❖ Procesos secuenciales.
- ❖ Instalaciones de procesos complejos.

1.6 Protocolo de comunicación DNP3

El Protocolo DNP 3.0 está basado en las normas del Comité 57, Grupo de Trabajo 03 de la IEC para el desarrollo de un protocolo para aplicaciones en telecontrol, SCADAs y sistemas de automatización distribuidos. Este protocolo fue desarrollado por la GE Harris en 1990, y en 1993 fue cedido al Grupo de Usuarios DNP, que es una

organización sin fines de lucro formada por compañías de servicio público y vendedores. Es un protocolo abierto que fue diseñado para lograr la interoperabilidad entre RTU, IED (Dispositivo Electrónico Inteligente en inglés: Intelligent Electronic Device) y estaciones maestras. Ha sido adoptado por la IEEE como práctica recomendada para la interconexión IED-RTU. (4)

El DNP 3.0 es un protocolo abierto, robusto y eficiente, con el cual se puede:

- Solicitar y responder múltiples tipos de datos en mensajes sencillos.
- Segmentar mensajes en múltiples tramas para asegurar una mejor detección y recuperación de errores.
- Incluir solamente nuevos datos en los mensajes de respuesta.
- Asignar prioridades a ciertas clases de datos y solicitar esos datos periódicamente de acuerdo con la prioridad establecida.
- Permitir respuestas no solicitadas.
- Soportar sincronización de temporización con un formato estándar de tiempo.
- Permitir múltiples maestros y operaciones de igual a igual (peer to peer)
- Permitir el uso de objetos definibles por el usuario incluyendo la transferencia de archivos.

1.6.1 Características.

- Control por Conteo de Caracteres.
- Carácter Básico de 8 dígitos de información, 1 bit de arranque y 1 de parada.
- Topologías punto a punto. Soporta múltiples MTUs y RTUs.
- Modos de Operación Normal y Balanceada. Permite transmisiones no solicitadas.
- Interfaces: RS-232C, UIT-T V.24/V.28 y RS-485. Modulación FSK
- Medios de transmisión: par trenzado, fibras ópticas y radio
- La velocidad de transmisión depende del medio utilizado

En la siguiente figura se muestran las diferentes topologías de redes en las que es efectivo.

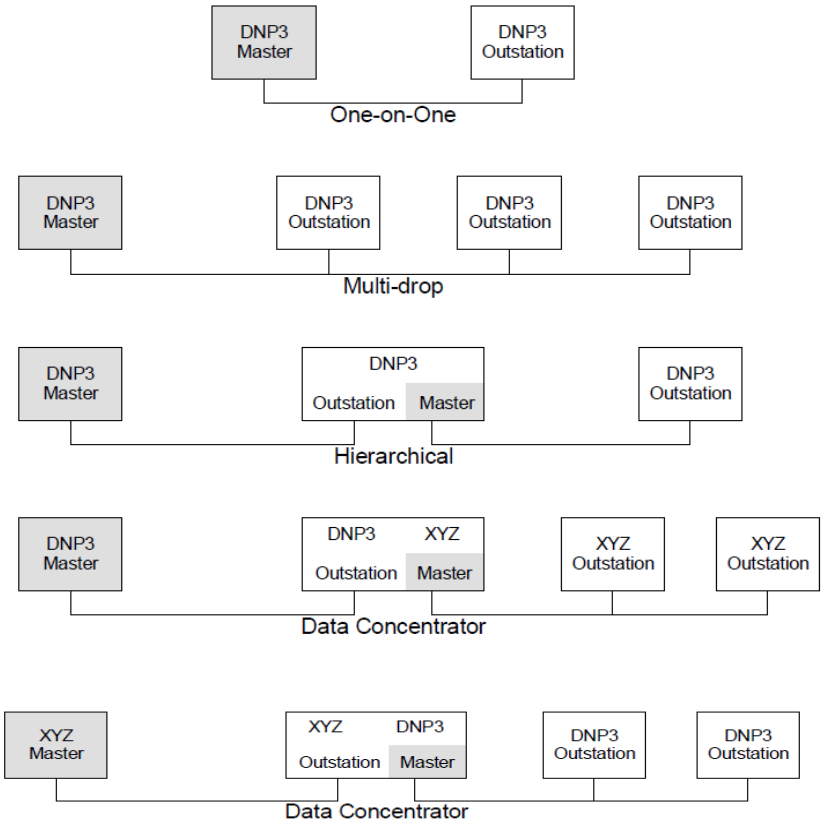

Ilustración 4: Arquitecturas soportadas

1.6.2 Formatos de los mensajes.

1.6.2.1 Secuencias al nivel de aplicación

Como se muestra en la siguiente figura, la estación maestra envía una petición a la remota la cual contesta con un mensaje de respuesta. La remota puede decidir espontáneamente transmitir datos mediante un mensaje de Respuesta no Solicitada. Para la maestra, la transacción petición/respuesta con una remota en particular debe completarse antes de que una nueva petición sea enviada a esa remota. Una maestra puede aceptar respuestas no solicitadas mientras una respuesta está en progreso. Sin embargo, para la remota una transacción petición/respuesta debe completarse antes de

que cualquiera otra petición o respuesta no solicitada sean enviadas. Las respuestas no solicitadas solamente deben ser enviadas antes o después de una transacción petición/respuesta.

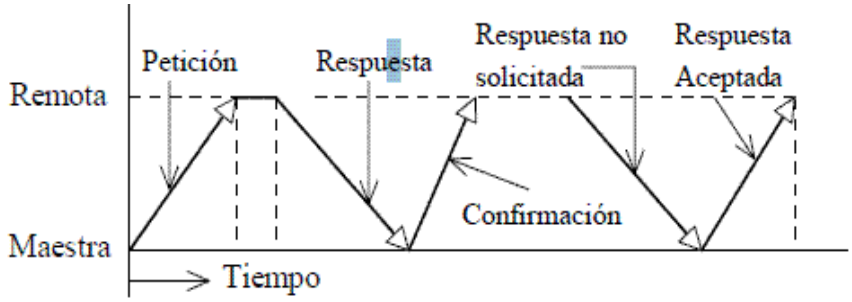

Ilustración 5: Secuencia de interrogación/respuesta a nivel de aplicación (4).

1.6.2.2 Formato de la Capa de aplicación

El formato de los mensajes en esta capa se divide en formatos de petición y formatos de respuesta. El campo Control de Aplicación proporciona la construcción de mansajes de múltiples fragmentos. El tamaño recomendado para cada fragmento es de 2048 bytes. Cada uno tiene su propio encabezado de manera que puede ser procesado como un mensaje individual y luego descartado para guardar espacio para el próximo fragmento.
El formato de los mensajes de solicitud y respuesta es similar en esta capa, aunque presenta como diferencia que el encabezado de la respuesta tiene después del campo Código de Función un campo adicional IIN (en inglés: Internal Indications) cuyos dígitos indican el estatus de la respuesta.

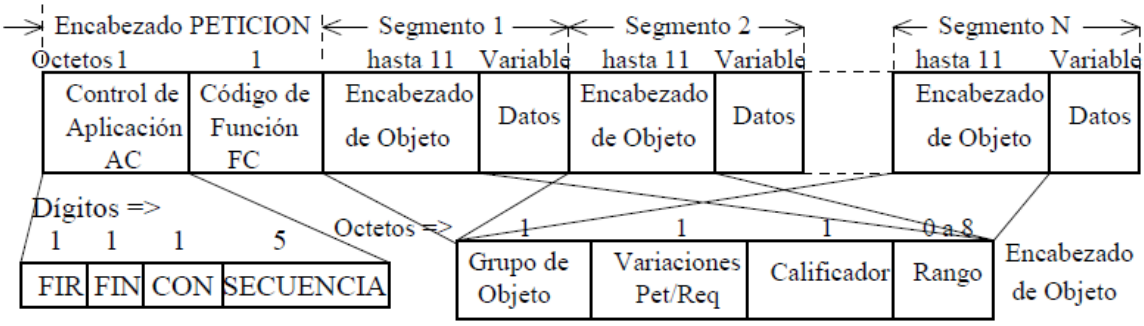

Ilustración 6: Formato de la Capa Aplicación DNP 3.0 (4)

Control de Aplicación (AC). Proporciona la información necesaria para la construcción de mensajes de múltiples fragmentos. Está compuesto por los siguientes elementos:

FIR. Cuando se establece en UNO, indica que el fragmento es el primer fragmento de un mensaje de aplicación.
FIN. Cuando se establece en UNO, indica que el fragmento es el último fragmento de un mensaje de aplicación.
CON. Cuando se establece en UNO, indica que la aplicación iniciadora queda en espera de una confirmación de la aplicación iniciada. El Código de Función 1 se utiliza en el mensaje de confirmación.
SECUENCIA. Indica el número del fragmento. Los números 0 a 15 están reservados para peticiones de la Estación Maestra. Los números 16 a 31 están reservados para las Respuestas no Solicitadas desde las Esclavas.

Código de Función (FC). Identifica el propósito del mensaje. Hay dos grupos de Código de Función: uno para peticiones y otro para respuestas. En la siguiente tabla se muestran los códigos de función más utilizados.

Tabla 1: Principales funciones de la Capa de aplicación del DNP3

Código	Función	Descripción
Códigos de función para solicitudes.		
0	Confirmación	Es usado tanto para mensajes de solicitud como de respuesta. No es necesaria una respuesta al ser estos recibidos. Su objetivo es avisar la buena recepción de un mensaje anterior enviado.
1	Leer	Solicitar objetos específicos de la estación esclava. Responde con el conjunto de objetos solicitados que estén disponibles.
2	Escribir	Almacena objetos específicos en la subestación. Esta responde con un mensaje portador del estado de la operación.
Códigos de función para respuestas.		

0	Confirmación	Es usado tanto para mensajes de solicitud como de respuesta. No es necesaria una respuesta al ser estos recibidos. Su objetivo es avisar la buena recepción de un mensaje anterior enviado.
129	Respuesta	Respuesta a una solicitud anteriormente realizada.
130	Respuestas no solicitadas	Respuestas que no son antecedidas por una solicitud.

Encabezado de Objeto. Especifica el tipo de datos que están contenidos en el mensaje o que van a ser utilizados para responder a ese mensaje. El encabezado de objeto es igual tanto en Solicitud como en Respuesta, pero la interpretación depende de si es una solicitud o una respuesta y del código de función que lo acompaña. El Encabezado de Objeto está formado por los siguientes campos:

GRUPO y VARIACIÓN. Estos dos campos especifican el grupo de datos y las variaciones de dichos grupos, lo que permite identificar el tipo, clase de datos u objeto.
CALIFICADOR. Especifica el significado del campo Rango, es decir, cómo debe interpretarse.
RANGO. Indica la cantidad de objetos, los índices de partida y final o identificadores de los objetos en cuestión.

1.6.2.3 Formato de la capa de transporte.

Cuando una aplicación solicita la transmisión de un mensaje grande, este mensaje se segmenta en fragmentos lo suficientemente pequeños para que quepan en una trama, la cual contiene, como máximo, 260 octetos. El número máximo de octetos del formato Seudocapa de Transporte es de 250, de los cuales uno es el Encabezado y los otros 249 son Datos de Usuario.
La siguiente figura muestra la disposición del mensaje en esta capa.

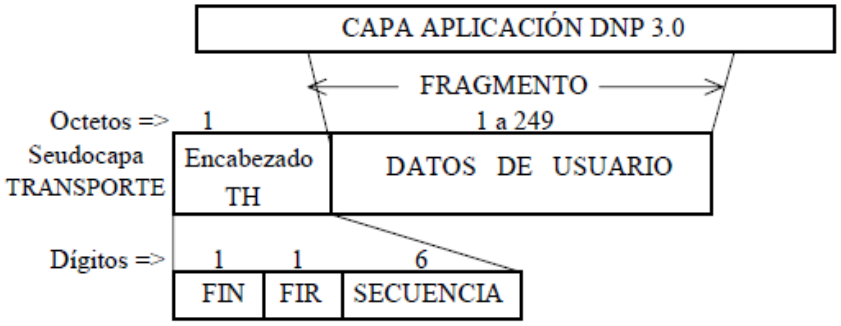

Ilustración 7: Formato de la Capa de transporte (4)

1.6.2.4 Formato de la Capa de Enlace

El formato de Capa Enlace DNP contiene un encabezado de 10 octetos seguido de una secuencia opcional de bloques de datos. Cada bloque contiene 16 octetos a los cuales se les agrega un CRC de dos octetos. La longitud máxima de la trama es de 260 octetos. En figura se muestra la trama de enlace DNP.

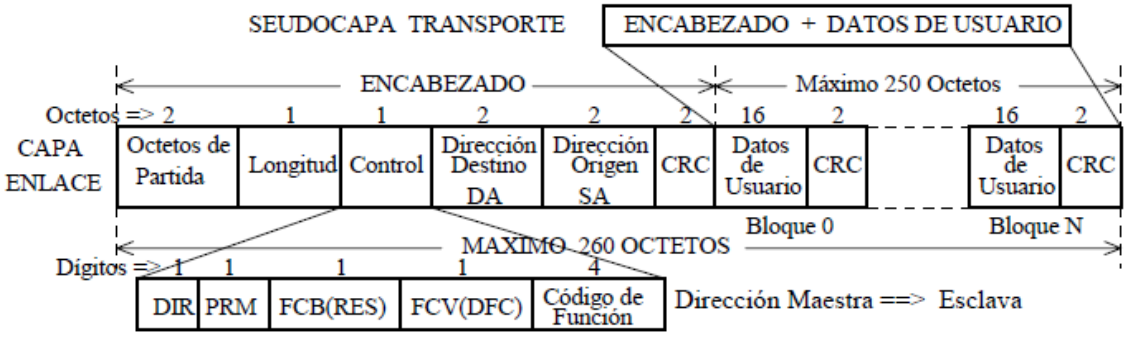

Ilustración 8: Formato de la Capa de Enlace (4)

La trama Enlace DNP contiene los siguientes campos:

Octetos de Partida. El primer octeto contiene el número 05H y el segundo 64H. Ellos son una especie de bandera y permiten la sincronización de la trama.

Longitud. Especifica la longitud, en octetos, de los octetos de usuario en la trama, incluyendo los campos Control, Dirección Destino y Dirección Fuente.

Control. Este campo contiene la dirección de la trama, el tipo de trama e información de control de flujo. Para la dirección Esclava-Maestra, los dígitos FCB y FCV se reemplazan por los dígitos RES y DFC, respectivamente. Las funciones de estos subcampos son:

DIR. Indica el sentido de transmisión: Maestra => Esclava, DIR = 1, Esclava => Maestra, DIR = 0.

PRM. Mensaje primario; PRM = 1, trama desde la estación primaria (estación llamante); **PRM** = 0, trama desde la estación secundaria (estación llamada).

FCB. Dígito de conteo de tramas. Se utiliza para suprimir pérdidas o duplicación de tramas hacia la estación secundaria.

RES. Reservado

FCV. Dígito de validación que permite el funcionamiento del dígito FCB. En este caso: **FCV** = 0, indica ignorar el estado del dígito FCB; FCV = 1, le indica a una estación secundaria que el estado del dígito FCB debe ser verificado en relación con el estado del dígito FCB de la última trama enviada cuyo dígito FCV era 1.

DFC. Dígito de control del flujo de datos. Se utiliza para prevenir el desbordamiento de los buffers de la estación secundaria.

Código de Función. Identifica el tipo de trama. La definición de los valores colocados en este campo es diferente en las estaciones primarias y secundarias.

Dirección de Destino (DA). Especifica la dirección de la estación hacia la cual se envía la trama. El primer octeto es el octeto de menor orden y el segundo es el de mayor orden. Cuando la estación primaria coloca la dirección FFFFH (broadcast), todas las estaciones secundarias escuchan y aceptan las tramas que vienen de la estación primaria.

Dirección de Origen (SA). Identifica la dirección de la estación de donde viene la trama. El primer octeto es el octeto de menor orden y el segundo es el de mayor orden.

Datos de Usuario. Los bloques contienen de 1 a 16 octetos de datos de usuario. Si el mensaje consta de más de 16 octetos, se llenarán bloques con 16 octetos excepto el último que puede contener de 0 a 16 octetos.

CRC. Campos para verificación de error; código CRC. (4)

1.6.2.5 Formato de la Capa Física

La capa física que se recomienda para el enlace de datos en DNP 3.0 es una capa asincrónica con caracteres de 8 dígitos de información, 1 dígito de arranque, 1 de pare y

no paridad. Las interfaces físicas recomendadas son la RS-232C, la UIT-T V.24/V.28 y la RS-485, con Módems FSK.

1.7 Tendencias y desarrollo del protocolo DNP3.

El grupo DNP3 ha sido el encargado desde 1993 de mantener y desarrollar el protocolo de comunicación DNP3. Gracias a su trabajo este se hace cada vez más popular. En todos estos años han desarrollado varias aplicaciones para probar tanto estaciones maestras como esclavas, además de nuevas variantes como DNP3s la cual incluye opciones de seguridad a la versión anterior. Proveen las especificaciones de forma gratuita además de las actualizaciones que van desarrollando. El 11 de enero del 2010 la IEEE ha anunciado al DNP3 como su nuevo estándar. (6)

Es ampliamente utilizado en sistemas eléctricos, donde las estampas y sincronizaciones de tiempo, como el hecho de que un esclavo transmita información sin ser solicitada, son fundamentales al momento de analizar fallas y sincronizar el accionamiento de todos los dispositivos.

Entre los sistemas SCADA que lo incluyen se encuentra el TUÚXTIK, que tiene la particularidad de haber sido desarrollado sobre GNU/Linux al igual que el Guardián del ALBA. Es un sistema embebido dentro de la máquina sobre plataformas de servidores, desde 1 a 4 procesadores trabajando en línea.
Su diseño permite controlar la temporización de peticiones, controles y comandos en tiempo real. Posee módulos independientes por protocolo de comunicación, que permiten obtener una mayor robustez y modularidad.
Los módulos DNP3 trabajan a nivel 3 de implantación. Presenta las variantes de DNP3 Serial y TCP, con configuración para modo maestro y esclavo. (7)

Una aplicación novedosa de DNP3 es su integración con la tecnología CDPD (Cellular Digital Packed Data). CDPD es una tecnología que permite usar ciertos canales de voz del sistema de telefonía celular para transmitir datos digitales. La amplitud de las especificaciones de CDPD permite su compatibilidad a los sistemas de interconexión abierta (OSI) aportándole la ventaja de futuras expansiones. El objetivo de esta idea es

utilizar la red CDPD como plataforma en un Sistema SCADA transportando información utilizándose el protocolo DNP 3.0 como protocolo de comunicaciones entre los dispositivos. La siguiente figura es una representación gráfica de esta aplicación. (8)

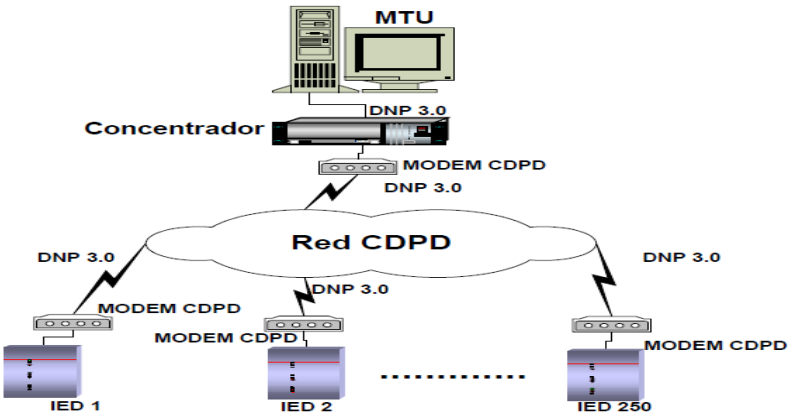

Ilustración 9: Integración entre DNP3 y CDPD

1.8 Tecnologías más utilizadas en la recolección por medio del puerto serie.

En este epígrafe se mencionarán algunas de las tecnologías más utilizadas actualmente a nivel internacional para la resolución de problemas similares al que se aborda. Se realizará un análisis de las mismas con fin de escoger aquellas que por sus características resulten más beneficiosas para lograr un resultado que satisfaga el objetivo planteado.

1.8.1 Asio C++ Library

Asio es una biblioteca multiplataforma que se utiliza para la implementación de aplicaciones de red, brinda un desarrollo consistente de un mecanismo asíncrono de flujos de entrada y salida. Da soporte para la resolución de direcciones, aceptación de nuevas conexiones, socket orientados a datagrama y funcionalidades de temporizador.
Fue insertada dentro de la Boost Library el 30 de diciembre del 2005. Ha sido desarrollada por Christopher M. Kohlhoff desde el 2003. Sus características fundamentales son las siguientes:

- Portabilidad: puede ser empleada en cualquiera de los sistemas operativos más

conocidos internacionalmente.

- Escalabilidad: La biblioteca permite, e incluso fomenta, el desarrollo de aplicaciones de red que escala a cientos o miles de conexiones simultáneas. La implementación de la misma para cada sistema operativo utiliza el mecanismo que mejor permite esta escalabilidad.
- Modelo de Berkeley sockets: es muy aplicado en la biblioteca.
- Bases para una mayor abstracción: la biblioteca permite el desarrollo de otras con un mayor nivel de abstracción.

Desde la versión 1.35 de Boost, la Asio se incluye dentro de su conjunto de bibliotecas.

Asio incluye clases para la manipulación del puerto serie de manera sencilla y portable. Una vez abierto el mismo, gracias a las funcionalidades de la biblioteca, puede ser usado como un flujo, lo cual facilita su comprensión y manejo.
La biblioteca también incluye clases para el establecimiento de parámetros de configuración como: el baud rate, el tipo de control de flujo (flow control), la paridad, el bit de parada etc.

1.8.2 Lenguaje de programación gráfico para el diseño de sistemas de adquisición de datos, instrumentación y control, LabVIEW.

LabVIEW (en inglés: Laboratory Virtual Instrument Engineering Workbench), permite diseñar interfaces de usuario mediante una consola interactiva basada en software. (9)
Es compatible con con herramientas de desarrollo similares y puede trabajar con programas como por ejemplo MatLab, que pertenecen a otras áreas de aplicación. Permite una fácil integración con hardware, especialmente con tarjetas de medición, adquisición y procesamiento de datos.
LabVIEW es muy utilizado en sistemas de medición, como monitoreo de procesos y aplicaciones de control. Un ejemplo de esto pueden ser, sistemas de procesamiento digital de señales, procesamiento en tiempo real de aplicaciones biomédicas, automatización, entre otras.
Tiene soporte para el manejo del puerto serial, con el objetivo de hacer posible el control a una fuente programable mediante comandos por el puerto mencionado.
LabView no es una tecnología que funcione sobre sistemas operativos UNIX.

1.8.3 Biblioteca para comunicaciones serie: SERCOM

La biblioteca SERCOM fue creada por Juan González Gómez en agosto del 2002. Está regida por los términos de la licencia GPL.

Esta biblioteca ofrece una API basada en GLIB para acceder al puerto serie. Las operaciones para el manejo del mismo son realizadas con la utilización de los IOChannels de la GLIB-2.0. Para la configuración del puerto se utilizan funciones de POSIX, y sin embargo para acceder a detalles de más bajo nivel se utilizan la “ioctls” y funciones específicas que dependen del sistema operativo.

SERCOM nació con la idea de crear una biblioteca serie, multiplataforma y de fácil integración con sistemas gráficos. (10)

1.9 Biblioteca de transporte TransportProvider.

La biblioteca de transporte TransportProvider fue desarrollada en el seno de la línea de manejadores del proyecto Guardián del ALBA. Está basada en la Asio C++ Library. Permite el intercambio de información a través del puerto serie y del protocolo TCP/IP.

Posibilita que el intercambio sea asíncrono, dándole una mayor eficiencia en tiempo y recursos a los manejadores que la emplean. Su interfaz es intuitiva y fácil de utilizar.

El TransportProvider se adapta a variados entornos de acción y ejecución por lo que puede ser utilizado por cualquier sistema que necesite comunicarse con dispositivos de campo.

Es un sistema que posee una alta usabilidad y confiabilidad. Es multiplataforma, ya que puede ser utilizada en cualquiera de los sistemas operativos más conocidos y compilada con cualquiera de los compiladores de C++ más utilizados actualmente.

1.10 Herramientas de desarrollo empleadas.

Entre las herramientas necesarias para el desarrollo del manejador están los entornos de desarrollo integrados (IDE). Estos deben cumplir con un conjunto de cualidades que influirán en el resultado de la aplicación a desarrollar. El nivel de desarrollo y estabilidad alcanzada, la documentación que esté disponible, la flexibilidad, la personalidad y los

términos de la licencia bajo la cual fue desarrollado, son algunas de las más importantes.

El C++ fue escogido como lenguaje de programación para el desarrollo. La decisión se debe a que fue utilizado no solo en los restantes entregables de la línea sino en los entregables de todos los demás componentes del Guardián del ALBA.

Este lenguaje surge en 1980 gracias a Bjarne Stroustrup. Fue concebido con el objetivo de brindarle nuevas funcionalidades al lenguaje ANSI C, como es el caso de las clases y el polimorfismo, los tipos de datos genéricos, la posibilidad de declarar variables en cualquier lugar del programa y además un motor de objetos con herencia múltiple que permite combinar la programación imperativa de C con la programación orientada a objetos.

El entorno de desarrollo integrado seleccionado es el eclipse. Este IDE fue desarrollado en un principio por IBM, en la actualidad lo mantiene la Fundación Eclipse, organización independiente que fomenta otros productos de código abierto. Es multiplataforma, emplea módulos o plug-ins para la extensión de sus funcionalidades, gracias a los cuales es posible programar en C/C++, Python, PHP y Java, da soporte para sistemas de gestión de bases de datos, control de versiones CVS y Subversion y además para la realización de pruebas de unidad (Junit, CxxTest).

1.11 Visual Paradigm

La herramienta CASE escogida para el modelado de la solución fue el Visual Paradigm. La decisión está basada en que es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software. Permite representar todos los tipos de diagramas de clases, generar código desde diagramas, y generar documentación.

Es un software multiplataforma, por lo que puede ser utilizado en los sistemas operativos más populares de la actualidad, aunque no está regido por las licencias del software libre, Cuba cuenta con la autorización correspondiente para su utilización.

En las nuevas versiones al Visual Paradigm se incluye el modelado colaborativo con CVS y Subversion.

1.12 Herramienta de prueba: Recolector Gráfico.

Con el Recolector Gráfico son llevadas a cabo algunas de las pruebas que se le realizan a los manejadores del Guardián del ALBA. Esta herramienta fue desarrollada en el marco de la línea de desarrollo de manejadores, por lo que está adecuada a las

características propias de los módulos que implementan la Interfaz Genérica. Posee varias vistas entre las que se encuentra una vista de configuración, donde es posible crear y configurar instancias tanto de manejadores como de dispositivos; también es incluida una vista para la creación de las variables y los bloques resultantes a este proceso; y por último incluye una vista en la que pueden verse los distintos estados de la comunicación por los que transitan los manejadores analizados y el desglose de las tramas gracias a la acción de las extensiones del Sniffer correspondientes a cada protocolo.

1.13 Metodología de desarrollo de software.

La metodología escogida para desarrollar la solución fue el Rational Unified Process (RUP). RUP se caracteriza por ser un proceso dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental. Define cuatro fases para el desarrollo del software: inicio o conceptualización, elaboración, construcción y transición. Tiene definidos 9 flujos de trabajo, 6 de ingeniería y 3 de apoyo. Como de ingeniería se tiene: Modelamiento del Negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas y Despliegue. Los restantes, Gestión de Proyecto, Gestión de Configuración y Cambio y Ambiente, son los de apoyo.

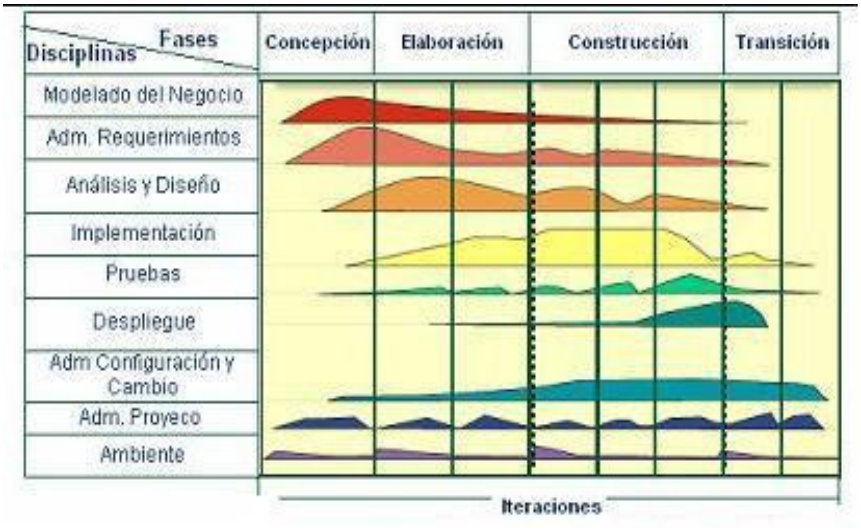

Ilustración 10: Fases y disciplinas de RUP

RUP define como lenguaje de modelado al UML (Unified Modeling Language) utilizado

para especificar, visualizar, construir y documentar artefactos de un sistema de software. Entre los diagramas que propone se encuentran:

Diagramas de estructura estática:

- Diagrama de clases: Conjunto de clases, interfaces y colaboraciones; así como sus colaboraciones.

- Diagrama de objetos: Conjunto de objetos y sus relaciones.

- Diagrama de casos de uso: Conjunto de casos de uso y actores y sus relaciones.

Diagramas de comportamiento:

- Diagramas de interacción (secuencia y colaboración): Objetos y sus relaciones, incluyendo los mensajes que pueden ser enviados entre ellos.

- Diagrama de estados: Muestra una máquina de estado que consta de estados, transiciones, eventos y actividades.

- Diagrama de actividad: Es un tipo especial de diagrama de estados que muestra el flujo de actividades dentro de un sistema.

Diagramas de implementación:

- Diagrama de componentes: Organización y las dependencias entre un conjunto de componentes.

- Diagrama de despliegue: Configuración de nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos. (11)

Capítulo 2: Diseño de la solución propuesta.

Este capítulo refleja la solución propuesta para resolver el problema científico identificado.

2.1 Interfaz Genérica de los manejadores.

En el mundo existe una considerable variedad de protocolos de comunicación, que a su vez, son usados por los más disímiles dispositivos. Un sistema SCADA debe ser capaz de adaptarse al cambio que implica la comunicación con un dispositivo cuyo protocolo aún no esté soportado por el sistema. Hay que tener en cuenta que un protocolo puede ser muy diferente a otro en cuanto a su configuración, sus restricciones de lectura y escritura, entre otros factores. Por esta razón es poco factible que en estas condiciones, sea necesario tener que realizar costosos cambios para adaptar el mismo a las nuevas exigencias.

Precisamente para resolver este problema fue diseñada la Interfaz Genérica de los manejadores, la cual es la especificación de un conjunto de funciones en lenguaje C, que tienen el objetivo de abstraer a las capas superiores del sistema de la mayoría de las diferencias entre un protocolo y otro.

La Interfaz Genérica de los manejadores no debe tener impactos negativos en el rendimiento del sistema por genérica que esta sea. Para ello se reúnen las características y funcionalidades comunes de todos los manejadores del SCADA Guardián del ALBA en un conjunto de clases denominadas DriversCore. Este framework nos proporciona una arquitectura que garantiza un desarrollo más eficiente en cuanto a tiempo de desarrollo y recursos del sistema operativo.

2.2 Tecnología de acceso al medio físico seleccionada para el desarrollo del manejador.

Un manejador para el sistema SCADA, Guardián del ALBA, es una biblioteca que permite el acceso a dispositivos, que a su vez se comunican gracias a la semántica brindada a sus mensajes por un protocolo de comunicación específico. Estos componentes necesitan acceder al medio físico para la transmisión de sus tramas.

Algunos, como el que se planea desarrollar, se basan en la comunicación serial para la transmisión de sus telegramas. Por lo tanto, se hizo necesario hacer un estudio de algunas tecnologías que podrían ser favorables para dar cumplimiento al objetivo

general.

El TransportProvider fue la opción escogida. El mismo contiene todas las cualidades que le brinda la Asio C++ Library (muy utilizada a nivel internacional), en la cual se basa, y además posee otras que favorecen específicamente a los manejadores desarrollados para el Guardián del ALBA, como la posibilidad de lograr comportamientos asíncronos de forma sencilla.

2.3 Planteamiento del modelo del dominio.

Luego de un profundo análisis y en ausencia de una estructura definida para los procesos del negocio, se hace necesario el planteamiento del modelo del dominio. El objetivo es lograr un entendimiento mínimo del contexto en que se empleará el manejador.

2.3.1 Diagrama de clases del modelo del dominio.

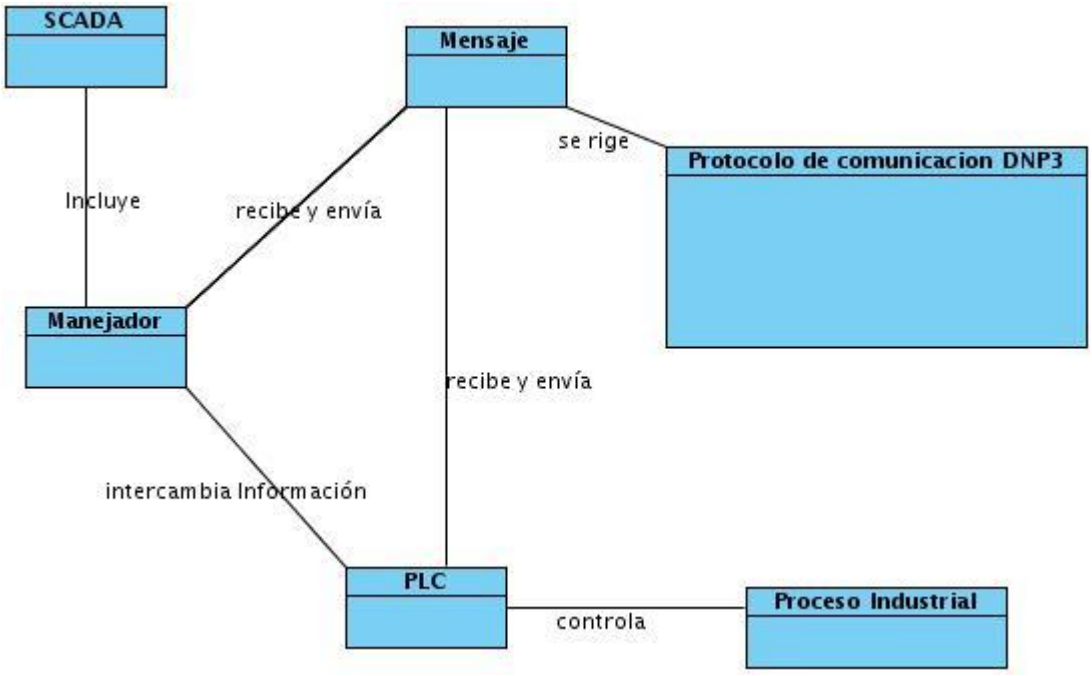

Ilustración 11: Diagrama de clases del dominio

2.3.2 Análisis de los conceptos del dominio.

Se define:

SCADA: Este concepto engloba a todos los componentes que conforman al Sistema de

Adquisición y Supervisión de Datos.

Manejador: Módulo del SCADA encargado del intercambio de información con dispositivos que se comunican por medio de un protocolo de comunicación específico.

Protocolo de comunicación DNP3: Conjunto de reglas o leyes a tener en cuenta para la comunicación entre dispositivos que se comuniquen con DNP3.

Mensaje: Representa cualquier tipo de datos que se pueda enviar o recibir entre una PC y un PLC.

PLC: Dispositivo de Adquisición de Datos el cual mediante mensajes se comunica con los programas que residen en una computadora.

Proceso Industrial: Aquel en el cual se producen transformaciones físicas o químicas.

2.4 Especificación de los requerimientos de software.

Los manejadores para el Guardián del ALBA poseen requerimientos funcionales y no funcionales comunes. Cada uno de ellos está especificado en el documento oficial denominado “Especificación de Requerimientos de software del Módulo de Drivers”, en su versión 1.3. (13). Se cuenta además con una explicación de los mismos, lo cual hace más sencillo su entendimiento.

Los requerimientos especificados en el mencionado documento fueron tenidos en cuenta para el diseño de la Interfaz Genérica de los manejadores. Por tanto, todo manejador desarrollado con el objetivo de ser integrado en el SCADA Guardián del ALBA, al tener que cumplir con lo especificado en la interfaz, cumple también con la mayoría de estos requerimientos.

La solución propuesta debe cumplir con los requisitos antes mencionados, pero con la particularidad de que los mismos serán aplicados a las condiciones específicas del DNP3.

2.5 Decisiones de diseño.

Para el desarrollo del manejador serán usadas solo aquellas funcionalidades descritas en el protocolo que aportan valor al intercambio de mensajes entre el Guardián del ALBA y dispositivos que se comunican a través del DNP3. Básicamente, para el cumplimiento de los requisitos especificados, es suficiente el uso de las funciones Leer (código: 1), Escribir (código: 2), Confirmar (código 0) y Respuesta (código 129).

Se decidió no incluir soporte para las respuestas no solicitadas, esto se debe a que el

uso de la Interfaz Genérica de los manejadores indica que las variables sean creadas, agrupadas en bloques de variables y solicitadas por indicación del SCADA. Si se empleara esta funcionalidad no habría variables en qué ubicar sus datos. La comunicación eficiente con redes industriales que funcionen bajo el modelo productor consumidor, aunque es un requisito planteado aún no está soportada en la interfaz genérica.

Otra decisión es la relacionada con el modo en que son solicitados los datos. Dada la necesidad en el SCADA de que las operaciones sean atómicas, o sea, que todas las variables que forman el bloque puedan recuperar sus valores en un solo mensaje de respuesta desde el dispositivo. Esto trae consigo que se conformen solicitudes que no provoquen mensajes de múltiples fragmentos.

Mucho tiene que ver la forma de organizar los bloques de variables en esta última decisión. Los mismos son construidos con variables que formen parte del mismo grupo y variación, solo se pueden diferenciar en el atributo que esté solicitando y el índice que identifica al objeto correspondiente dentro de su grupo. Por tanto en las tramas de solicitud solo aparecerá un ObjectHeader, que es el campo que indica los parámetros de un conjunto de objetos.

Los tipos de lectura son variados, en dependencia de si se desea leer en un rango determinado o especificando el índice requerido. El uso eficiente de cada tipo de lectura está relacionado con la cantidad de objetos no deseados que están ubicados en medio de los requeridos para un bloque. Si los objetos solicitados son consecutivos es mejor leer con el modo para lecturas por rango, en el que se recuperan todas aquellas variables cuyos índices estén en medio de dos solicitados. Para el caso que estén muy separadas dichas variables existe el método directo, en el que se solicita solo los índices especificados. No es usada siempre esta variante porque demandaría más recursos en cuanto tiempo y memoria por parte del manejador.

2.6 Diseño del Manejador DNP3 para el Guardián del ALBA.

2.6.1 Estilo arquitectónico empleado.

Para el desarrollo del manejador se ha utilizado una arquitectura en capas. Esta arquitectura consiste en la definición de un conjunto de capas que van disminuyendo el nivel de abstracción a medida que son más internas. Se enfoca en la distribución de roles y responsabilidades de forma jerárquica proveyendo una forma efectiva de

separación de responsabilidades.

Existen variaciones en cuanto a las comunicaciones permitidas entre elementos de capas diferentes. Por esta razón existen las variantes de:

- Arquitectura *top-down* de capas
- Arquitectura *bottom-up* de capas
- Arquitectura bidireccional de capas

La arquitectura bidireccional de capas es la más conveniente a utilizar en la solución ya que permite que las relaciones entre las capas sean en los dos sentidos. Esta es una cualidad fundamental para que las peticiones puedan hacerse eco desde la Interfaz Genérica de los manejadores hasta la capa de transporte y que al mismo tiempo las capas superiores puedan ser notificadas de manera asíncrona del estado de las operaciones en las capas inferiores.

En la siguiente figura se muestra la disposición de las capas en la solución propuesta.

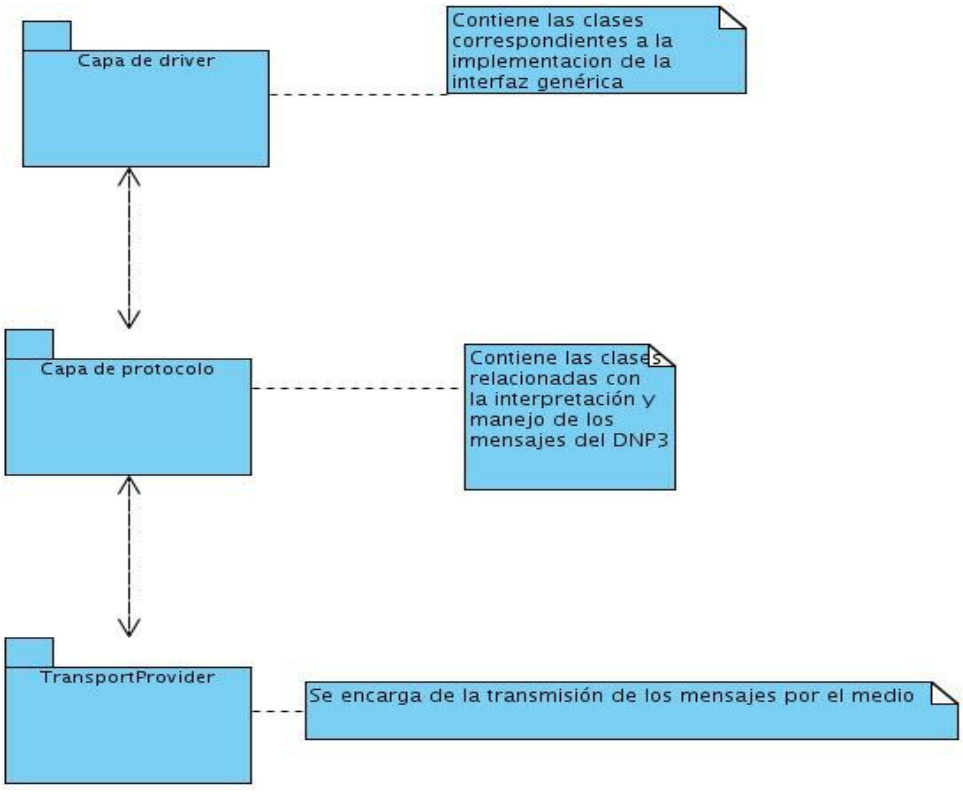

Ilustración 12: Arquitectura de los manejadores para el guardián del ALBA.

Figura 13: Arquitectura de un manejador para el Guardián del ALBA.

En la capa de driver se encuentran aquellas clases que se relacionan directamente con el DriverCore. Ellas representan las características particulares del sistema propuesto.

Posibilitan que la información recolectada cumpla con las restricciones impuestas por la Interfaz Genérica de los manejadores y por las particularidades del protocolo en cuestión.

La capa de protocolo se encarga de la comunicación con el dispositivo que se supervisa. En ella se sitúan las clases que interpretan el sentido y organización de los mensajes recibidos, además de aquellas que realizan las operaciones necesarias para solicitar y escribir datos en el dispositivo, entre otras operaciones.

En la capa de transporte se encuentran las clases que conforman la biblioteca TransportProvider. La misma se encarga de la transmisión de los datos por el puerto serie.

2.6.2 Diagrama de clases del diseño.

En el siguiente epígrafe se mostrará el diagrama de clases del diseño de la solución propuesta.

Ilustración 13: Diagrama de clases del manejador DNP3

2.6.3 Especificación de las clases del diseño.

Tabla 2: Clase del diseño DNP3Block

Nombre: DNP3Block	
Tipo de clase:	
Atributo	Tipo
minPos	unsigned long
maxPos	unsigned long
masked	bool
isWriteMasked	bool
dateTime	TDateTime
devolutionObjects	vector<objects*>*
operationWrite	objectWriteOperation*
writeVectorObject	vector< objectWriteOperation *>*
arrayWriteObjects	vector<objects*>*
arrayIndexes	vector<unsigned long long>
writeFlag	bool
Para cada responsabilidad	
Nombre:	DNP3Block()
Descripción:	Constructor por defecto.
Nombre:	readInteger()
Descripción:	Lee un valor entero del bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	readFloat()
Descripción:	Lee un valor flotante del bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	writeInteger()
Descripción	Escribe un valor entero en el bloque a partir de un IProtocolAddress y de

	un tipo en el dispositivo.
Nombre:	writeFloat()
Descripción:	Escribe un valor flotante en el bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	writeMaskedInteger()
Descripción:	El método permite modificar bits específicos en el bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	readText()
Descripción:	Lee una cadena del bloque a partir de un IProtocolAddress
Nombre:	writeText()
Descripción	Escribe una cadena del bloque a partir de un IProtocolAddress
Nombre:	beginWrite()
Descripción:	El método beginWriteCompleted es llamado una vez que se finalizan las operaciones iniciadas en el beginWrite. La implementación actual del método escribe las variables en el buffer del bloque y despacha la escritura asíncrona.
Nombre:	asyncMakeReadRequest()

Tabla 3: Clase del diseño DNP3EndPoint

Nombre: DNP3EndPoint	
Tipo de clase:	
Atributo	Tipo
readBuffer	unsigned char*
writeBuffer	unsigned char*
msgSize	unsigned long
myHeaderSize	unsigned long
readHeader	bool
confirmationMessage	bool

message	DNP3Message*
responseSize	unsigned long
timeOut	unsigned long
connectionTimeOut	unsigned long
requestSize	unsigned long
myRetries	int
comState	int
timeStamp	TDateTime
endPointID	unsigned long
transport	ITransport*
timer	IAsyncTimer
destinationAddress	unsigned short
sourceAddress	unsigned short
snifferHandler	SnifferHandler
retriesInterval	unsigned long
myHandler	DNP3Handler*
readObjects	vector<objects*>*
error	unsigned long long
device	Device*

Para cada responsabilidad	
Nombre:	DNP3Endpoint()
Descripción:	Constructor por defecto.
Nombre:	readRegisters()
Descripción:	Función para leer variables del dispositivo.
Nombre:	writeRegisters()
Descripción:	Función para escribir variables del dispositivo
Nombre:	sentConfirmationRequest()
Descripción:	Función que ensambla todos los mensaje de confirmación
Nombre:	setMyHandle()

Descripción:	Función que establece nuevo handle
Nombre:	setDestinationAddress()
Descripción:	Función que establece la dirección del dispositivo.
Nombre:	getDestinationAddress()
Descripción:	Función para acceder a la dirección del dispositivo
Nombre:	snifferEvent()
Descripción:	Evento del sniffer que se lanza en las operaciones realizadas en la transacción
Nombre:	getCOMMSTATE()
Descripción:	Función para acceder al estado de la comunicación
Nombre:	setEnableSniffer()
Descripción:	Función para establecer el uso Sniffer
Nombre:	getDiagnosticInfo()
Descripción:	Función para acceder a los parámetros de diagnostico del dispositivo
Nombre:	setSerialTransport()
Descripción:	Función para establecer el transporte serial
Nombre:	connectHandler()
Descripción:	Handler de conexión del transporte lanzado luego de un intento de conexión
Nombre:	writeHandler()
Descripción:	Handler de escritura del transporte, se envía al intentar escribir.
Nombre:	disconnectHandler()
Descripción:	Handler de desconexión del transporte
Nombre:	expiresHandler()
Descripción:	Handler del timer asíncrono
Nombre:	readHandler()
Descripción:	Handler de lectura asíncrono
Nombre:	executeTransaction()
Descripción:	Función que inicia las transacciones
Nombre:	onTransaction()
Descripción:	Función que inicia la máquina de estado del endpoint
Nombre:	onConfirmTransaction()

Descripción:	Se ejecuta después de haber leído un mensaje de confirmación
Nombre:	onResponseTransaction()
Descripción:	Se ejecuta después de haber leído un mensaje de respuesta del dispositivo correspondiente a una solicitud de lectura.
Nombre:	endTransaction()
Descripción:	Función que pone fin a las transacciones

Tabla 4: Clase del diseño DNP3Handler

Nombre: DNP3Handler	
Tipo de clase:	
Para cada responsabilidad	
Nombre:	DNP3Handler()
Descripción:	Constructor por defecto.
Nombre:	confirmHandler()
Descripción:	Este handler es invocado al recibir un mensaje de confirmación a una transacción
Nombre:	responseHandler ()
Descripción:	Handler para las respuestas con datos del dispositivo. Es invocado cuando finaliza la ejecución de la lectura de registros del endpoint, ya sea porque se ejecutó de forma satisfactoria u ocurrió algún error. La función debe ser reimplementada por los que quieran leer variables del endpoint.

Tabla 5: Clase del diseño DNP3Driver

Nombre: DNP3Driver	
Tipo de clase:	
Atributo	Tipo
library	DriversLibrary*

provider	ITransportProvider*
transport	ISerialTransport

Para cada responsabilidad	
Nombre:	DNP3Driver()
Descripción:	Constructor por defecto.
Nombre:	resetDiagnosticParameters()
Descripción:	Restablece los parámetros de diagnóstico del manejador.
Nombre:	getTransportProvider()
Descripción:	Devuelve el provider que utiliza el driver.
Nombre:	getErrorString()
Descripción:	El método getErrorString permite obtener información textual comprensible a partir de un código de error del manejador.
Nombre:	internalCreateDevice()
Descripción:	La función responde por la creación de una instancia de instancia de DNP3Device. Es llamada cada vez que se requiere la creación de un dispositivo DNP3.
Nombre:	setDeviceName()
Descripción:	Función estática utilizada por DNP3DriverMetaClass para el establecimiento de la propiedad relacionada con el nombre del dispositivo serial (ejemplo en Debian GNU/Linux /dev/ttyS0, en Windows COM1).
Nombre:	getDeviceName()
Descripción:	Función estática utilizada por DNP3DriverMetaClass para obtener el valor de la propiedad relacionada con el nombre del dispositivo serial (ejemplo en Debian GNU/Linux /dev/ttyS0, en Windows COM1).
Nombre:	setBaudRate()
Descripción:	Función estática utilizada por DNP3DriverMetaClass para el establecimiento de la propiedad relacionada con la velocidad en baudios del dispositivo serial

Nombre:	getBaudRate()
Descripción:	Función estática utilizada por DNP3DriverMetaClass para obtener el valor de la propiedad relacionada con la velocidad en baudios del dispositivo serial.

Las descripciones de las clases restantes pueden ser consultadas en los anexos.

2.7 Diseño de la extensión del DNP3 para el Sniffer.

La solución propuesta, como todos los manejadores del Guardián del ALBA, dispara un evento cada vez que se envía y se recibe un mensaje desde el dispositivo. El objetivo de dicha acción es que el SCADA pueda a través de Sniffer poner a punto tanto los manejadores como la programación de los dispositivos de campo sin necesidad de desconectarlos del sistema.

El Sniffer necesita para interpretar las tramas de un protocolo determinado una extensión que sea capaz de desglosar cada uno de sus mensajes como nodos de un árbol general cuya raíz sería el propio mensaje y sus hijos estarían dados en dependencia de la estructura interna de este. Por este motivo, se ha desarrollado de forma paralela a la solución una biblioteca capaz de cumplir con la funcionalidad descrita. La misma ha facilitado el trabajo y las pruebas realizadas al manejador.

2.7.1 Diagrama de clases del diseño de la extensión del Sniffer para el DNP3.

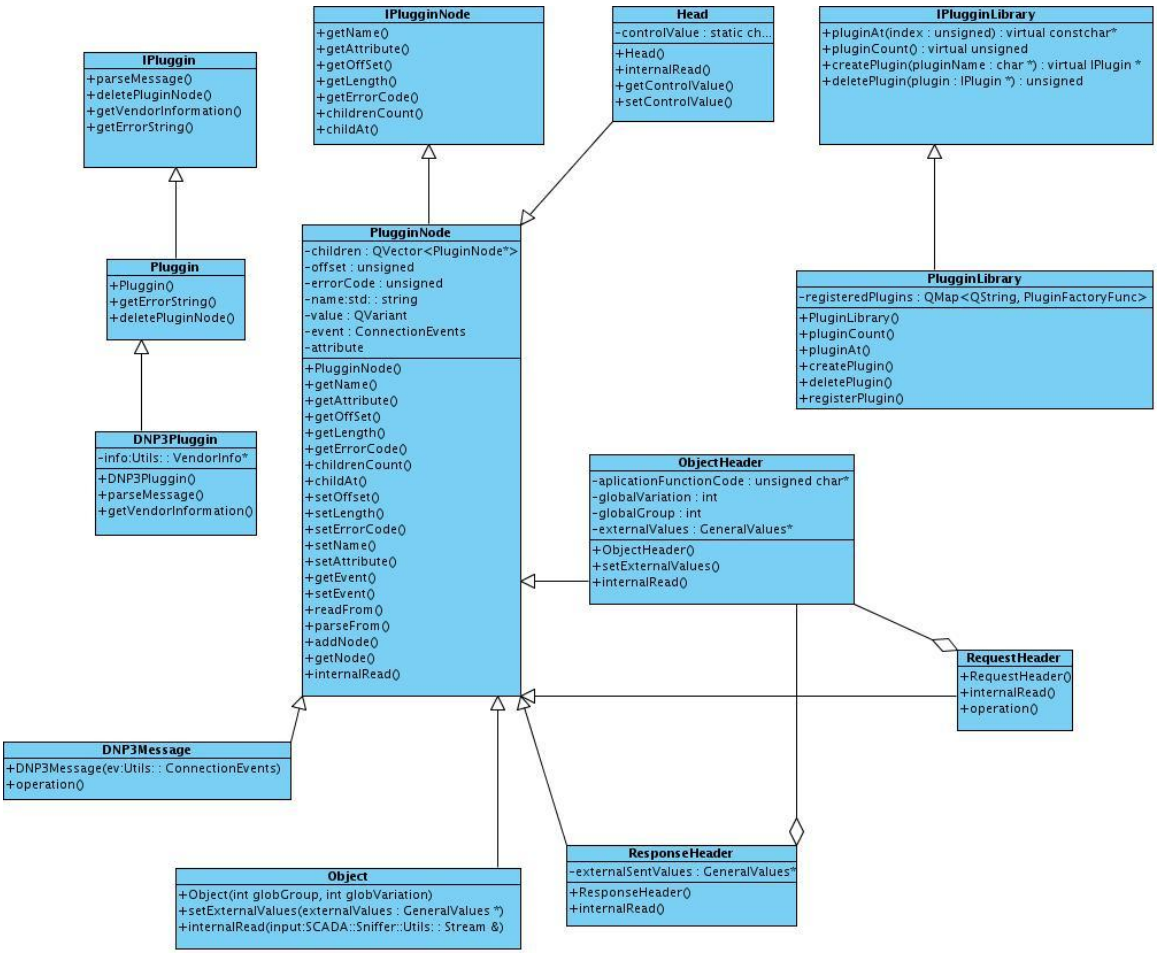

Ilustración 14: Diagrama de clases del diseño de la extensión del Sniffer

Capítulo 3: Implementación y pruebas.

En este capítulo se tratan temas que tributan a la implementación y prueba del manejador que da solución al problema planteado y de su extensión para el Sniffer.
Son expuestos además algunos de los artefactos producidos como resultado de las actividades de estos flujos de trabajo.

3.1 Estándar de codificación.

El código del Manejador DNP3 para el Guardián del ALBA sigue estándares propuestos para el módulo de la línea de manejadores del ya mencionado SCADA. Está programado en inglés debido a que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático.
El entendimiento de dichas normas contribuye a un mejor entendimiento del código fuente.
Nombre de los ficheros. Se nombraran los ficheros .h y .cpp de la siguiente manera. NameOfUnits.h NameOfUnit.cpp.
Clases. Los nombres de las clases deben tener la siguiente forma: MyClass. En un fichero .h solo debe ser definida una clase en caso de que sea necesario y a dicha definición le debe corresponder un fichero.cpp donde se implemente las funcionalidades de la clase definida.
Métodos o función miembro. Los métodos de clases deben seguir la siguiente estructura: myFunction a excepción de los constructores y destructores todos los métodos deben empezar con minúscula.
Condicionales y ciclos.
Todas las expresiones condicionales y los ciclos poseerán las llaves de inicio y cierre del campo de acción de la expresión, aunque la misma posea una sola línea de código. (6)

3.2 Diagrama de despliegue.

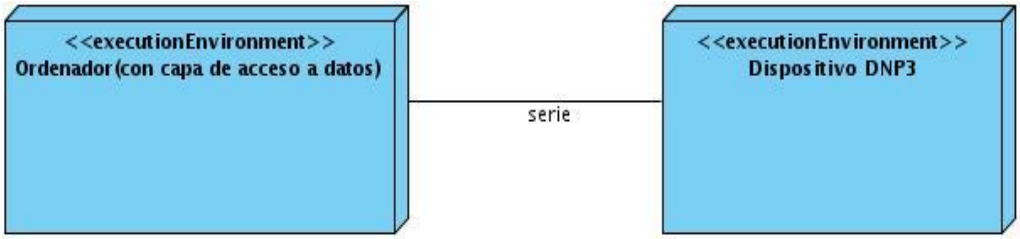

Ilustración 15: Diagrama de despliegue

3.3 Diagrama de componentes del manejador.

El siguiente diagrama muestra los ficheros .h más importantes y su relación con los ficheros específicos de las bibliotecas utilizadas para el desarrollo. Debe tenerse en cuenta que en cada fichero se ha declarado una clase cuya implementación se encuentra en el fichero .cpp correspondiente.

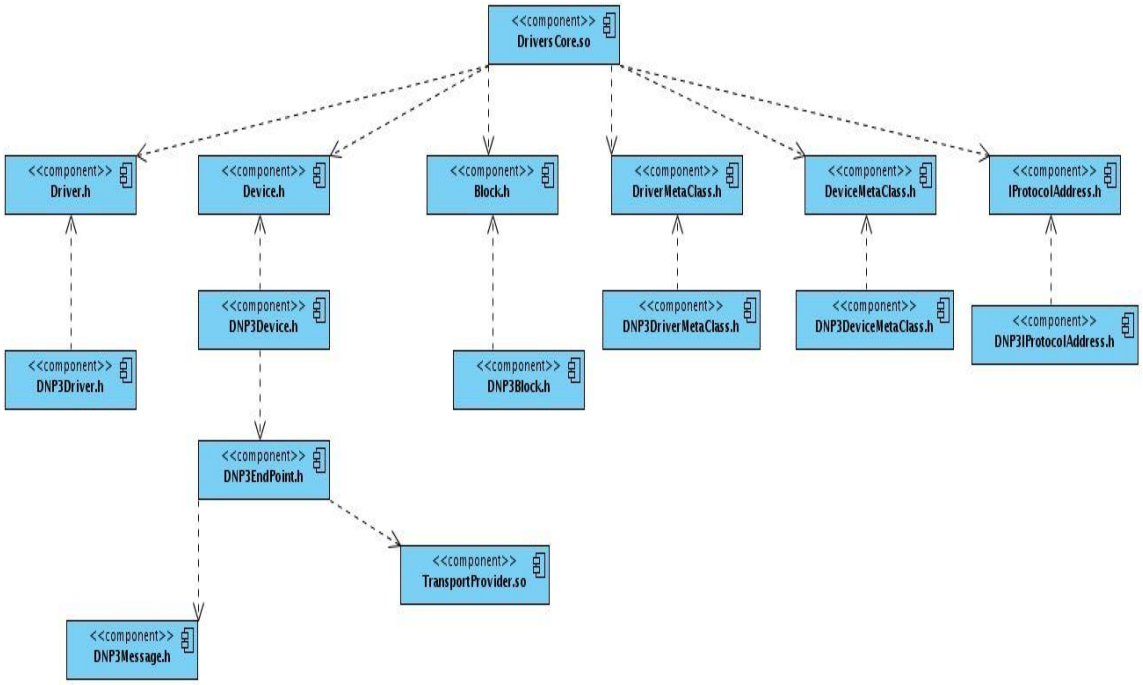

Ilustración 16: Diagrama de componentes

Como pudo ser apreciado en la figura anterior, existe una dependencia directa entre un conjunto de ficheros .h correspondientes a la biblioteca DriversCore y ficheros que

contienen las clases que forman parte de la capa de driver del Manejador DNP3. Cada una de las clases contenidas en estos ficheros tienen funcionalidades heredadas de sus padres las cuales el framework obliga a implementar para darle el comportamiento específico del protocolo a la biblioteca dinámica que resulta de la unión de sus partes.

Por ejemplo, la clase especificada en DNP3Device.h implementa métodos que dan estructura a los bloques de variables que luego serán usados en las operaciones del manejador. Muchas de las decisiones de diseño se materializan en el código fuente necesario para cumplir con estas funcionalidades, y precisamente eso hace que los manejadores sean diferentes unos de otros y al mismo tiempo se asemejen en la forma de ser usados.

La otra biblioteca relacionada (TransportProvider), es utilizada a través de las funcionalidades que brindan sus interfaces. Con ella se consigue el acceso al puerto serie sobre el cual se transmiten los mensajes deseados. En dependencia del tipo de conexión entre el SCADA y los dispositivos a supervisar es utilizada una interfaz diferente del TransportProvider. En este caso, la clase contenida en el .h denominado DNP3EndPoint se relaciona con el fichero del transporte ISerialTransport, que le brinda las funciones necesarias para la interacción con el dispositivo que da nombre a la interfaz.

Con la implementación de un pequeño conjunto de operaciones en la capa de protocolo es asegurado el comportamiento asíncrono necesario que exige la especificación de la Interfaz Genérica.

3.4 Tipos de pruebas de software.

Existen dos vertientes fundamentales entre los tipos de pruebas, las pruebas de tipo caja negra y las de tipo caja blanca. Las pruebas de caja negra son aquellas en que la aplicación es probada usando su interfaz externa. En las mismas no se tiene conocimiento de la estructura y funcionamiento interno del sistema. Solo se conoce las entradas específicas con sus respectivas posibles salidas. En las pruebas de caja blanca la aplicación es probada desde dentro. Se utilizan datos derivados de un análisis directo del código a ser probado por lo que es necesario tener conocimiento del código fuente.

Otras clasificaciones de pruebas incluyen las siguientes:

- Pruebas de Integración.

- ❖ Pruebas de Validación.
- ❖ Pruebas de Sistema.
    - Pruebas de Rendimiento.
    - Pruebas de Resistencia.
    - Pruebas de Robustez.
    - Pruebas de Usabilidad.
    - Pruebas de instalación.
    - Pruebas de estrés.
- ❖ Pruebas de aceptación. (14)

3.5 Pruebas realizadas al Manejador DNP3 para el SCADA Guardián del ALBA.

El manejador desarrollado con el objetivo de dar solución al problema científico planteado, actualmente está siendo probado en la República Bolivariana de Venezuela con vistas a ser incluido en las próximas versiones del Guardián del ALBA. El equipo de prueba está conformado por especialistas cubanos y venezolanos. Los mismos cuentan con la experiencia de los años acumulados en el proceso de desarrollo del SCADA, por lo que su trabajo ayuda a obtener altos índices de calidad en los productos generados en el marco del proyecto.

Las pruebas realizadas fueron planeadas con el objetivo de identificar posibles problemas en cuanto al cumplimiento, por parte de la solución, de los requisitos funcionales establecidos para todos los manejadores. De estas funcionalidades las más críticas son la lectura y escritura de variables en bloque. Para garantizar su cumplimiento, antes de pasar al equipo de pruebas fueron realizadas las siguientes evaluaciones:

Las pruebas a continuación fueron realizadas en una PC local, con procesador Intel Dual-Core a 1.6 de velocidad y 1GBb de memoria RAM. Los resultados de las pruebas pueden variar, en dependencia del contexto en que se realicen las mismas. Hay que señalar que las mismas siguen el paradigma de las pruebas de Tipo Caja Negra.

3.5.1 Caso 1: Leer variables en bloque.

Teniendo en cuenta que los manejadores en un SCADA estarán en ejecución largos periodos de tiempo, es necesario que además de medir la capacidad de recolectar los

bloques de variables especificados en la configuración, sea medido el consumo de recursos que esta acción requiere, si recursos como el uso de la memoria RAM se mantiene estable durante toda la ejecución, por cuanto tiempo se mantiene el manejador sin interrumpir su ejecución a pesar de problemas como la pérdida de conexión entre otras muchas eventualidades negativas que ocurren frecuentemente en los despliegues.

Los resultados esperados y los reales fueron coincidentes en un alto por ciento. La lectura de variables fue satisfactoria, en los momentos en que el sistema tenía acceso al medio de transmisión los valores siempre fueron recuperados y en el otro caso fueron reportados los errores correspondientes y devueltos los valores de calidad de las variables que se esperaban en este caso.

A continuación se muestra una vista del Recolector Gráfico en la que se observan los resultados de la lectura de un variado número de bloques.

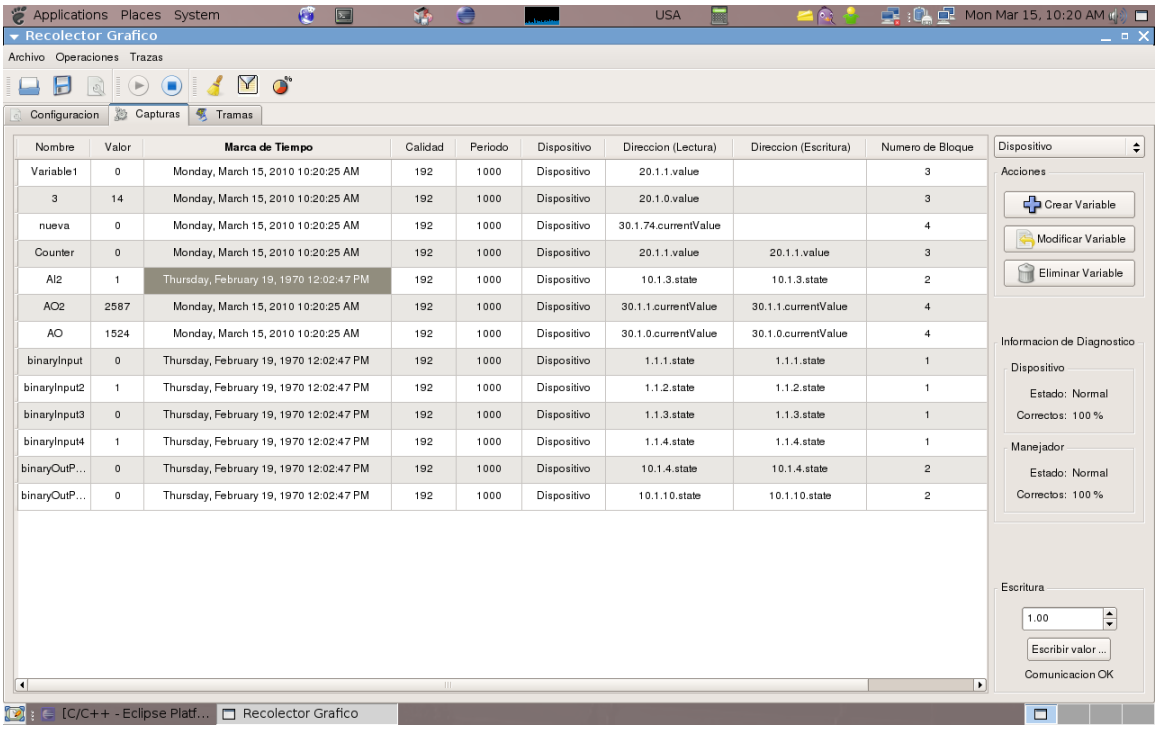

Ilustración 17: Vista de capturas del Recolector Gráfico

A continuación se listan un conjunto de resultados obtenidos en cuanto al rendimiento de la biblioteca en cuanto a rendimiento.

- Transacciones exitosas: Durante la ejecución de las pruebas todas las transacciones fueron exitosas excepto aquellas en que no había conexión. Los

errores fueron gestionados también de forma correcta.

- Tiempo medio: El tiempo medio tomado para la ejecución de una transacción tanto de lectura como de escritura es en milisegundos, y la diferencia depende de las características físicas del medio de transmisión.
- El consumo de memoria se mantuvo estable durante toda la prueba, los valores oscilaron entre 6.2 y 6.4 Mbyte.
- El consumo del procesador es mínimo.

3.5.2 Caso 2: Escribir variables en bloque.

Las operaciones de escritura son críticas en este tipo de aplicaciones, básicamente las mismas permiten modificar valores en la memoria de aquellos PLC que están siendo supervisados. Por lo tanto, son un elemento importante para el control de las plantas.
No todos los objetos especificados en DNP3 soportan esta operación. Generalmente la misma solo es usada para restablecer los valores de las indicaciones internas del dispositivo (IIN), para sincronizar el sistema con el tiempo del dispositivo y para descargarle ficheros de configuración a la subestación. Se aclara que esta última posibilidad no es soportada por lo descrito en la Interfaz Genérica.
Para evaluar esta funcionalidad se hizo énfasis en los dos grupos de objetos más utilizados en ella, estos son: el grupo 50, el cual contiene los atributos para el manejo del tiempo en el PLC, y el grupo 80 que se refiere al IIN. En el primer caso se ejecutó la escritura con un valor que coincidió con el tamaño requerido por el protocolo y luego al leer se pudo comprobar que hubo un cambio en la estampa de tiempo almacenada en el dispositivo. Para el segundo grupo el resultado fue analizado en la trama recibida producto de una lectura aleatoria (El cambio se refleja en el valor IIN de la trama de respuesta) con la ayuda de la extensión DNP3 para el Sniffer desarrollada también en el marco de esta tesis.

3.6 Pruebas a la extensión del Sniffer para el protocolo DNP3.

Las pruebas realizadas a la extensión del Sniffer para el DNP3 se basaron fundamentalmente en chequear su funcionamiento en el desglose de diversas tramas tanto de solicitudes como de respuesta. Los resultados fueron los esperados para cada uno de los tipos de tramas posibles. El análisis de los mensajes se produce de forma instantánea y con el nivel de detalle necesario para que un usuario del Sniffer pueda

detectar errores entre el sistema y dispositivos con protocolo DNP3.

En la parte derecha de la siguiente figura se puede observar el resultado de las acciones previstas para esta herramienta. Se puede observar que el mensaje está desfragmentado en un árbol cuyas raíz principal es todo el mensaje, y luego es separado de forma recursiva en hijos, que representan los distintos campos que conforman la trama.

En ese mismo segmento de la vista se puede visualizar el valor y su validez, lo cual indica si existe un error en el intercambio .

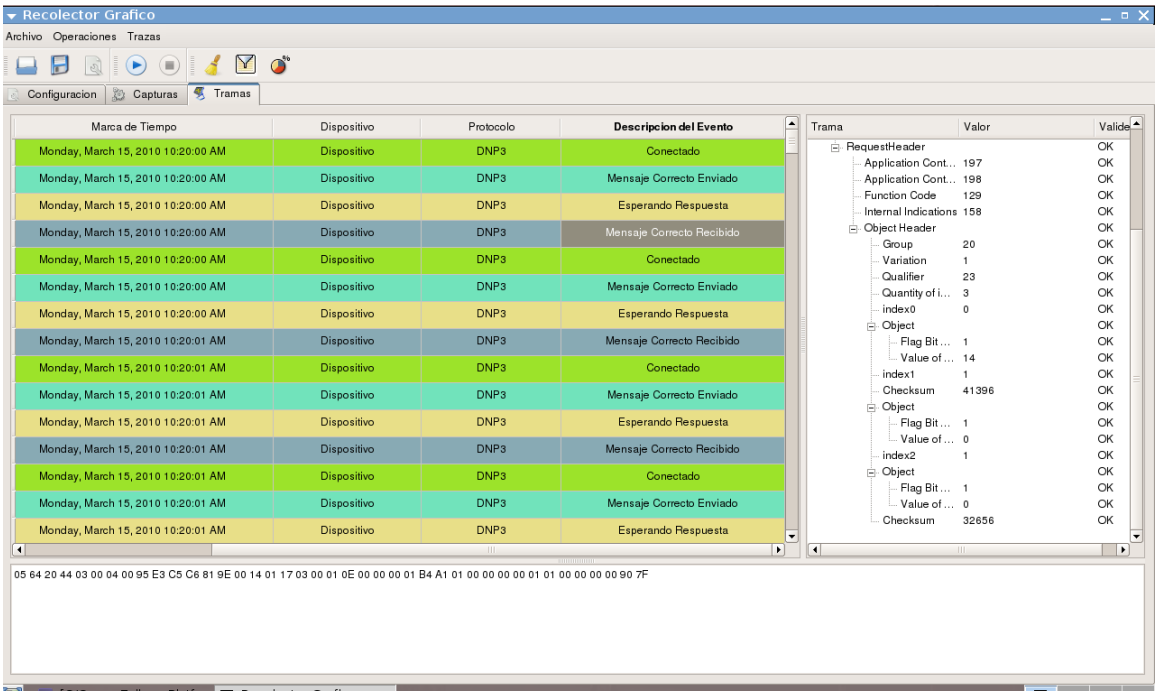

Ilustración 18: Vista de análisis de tramas

Conclusiones.

El manejador desarrollado es totalmente funcional y capaz de comunicarse a través del protocolo DNP3 en su variante serial. Cumple con todos los requerimientos especificados para los manejadores del SCADA Guardián del ALBA por lo que su integración con el mismo no debe tener muchos inconvenientes.

La extensión DNP3 desarrollada para el Sniffer permite monitorear el intercambio de mensajes entre las estaciones involucradas y al igual que el manejador es capaz de adaptarse a diferentes entornos de acción y ejecución.

Recomendaciones.

Aspectos fundamentales que se recomiendan del trabajo.

- Desarrollar un manejador para la variante TCP del DNP3.
- Desarrollar un manejador para la variante DNP3s, para de este modo aumentar la seguridad del intercambio de información entre el SCADA Guardián del ALBA y dispositivos que se comunican con el mencionado protocolo.

Referencias bibliográficas.

1. galeom.com. [En línea] [Citado el: 22 de 2 de 2010.] http://www.galeon.com/hamd/pdf/scada.pdf.

2. http://www.uclm.es. [En línea] [Citado el: 24 de 2 de 2010.] http://www.uclm.es/profesorado/rcarcelen_plc/Index.htm.

3. *OSI Reference Model-The ISO Model of Architecture for Open System Interconnections.* **Zimmerann, Hubert.** 4, 1980, Vols. com-28.

4. **Roncancio, Henry Antonio.** *Tutorial de labVIEW.* Universidad Distrital "Francisco Jose de Caldas" : s.n., 2001.

5. **Ramon Vicenc Yuste, Luis Guerrero Martinez.** Comunicaciones Industriales. *Comunicaciones Industriales.*

6. **Johnson, Ing.Rubén Gómez.** *Capa de acceso a datos síncrona y asíncrona para dispositivos Modbus TCP.* La Habana,Cuba : s.n., 2008.

7. **Alejandro Ramirez, Francisco Maldonado.** *Comunicacion DNP3.0 a traves de una red CDPD. Caracas,Venezuela.* Caracas,Venezuela : s.n., 2009.

8. www.sedpcsa.com. [En línea] [Citado el: 23 de 2 de 2010.] http://www.sedpcsa.com/pdf/scada.pdf.

9. www.iearobotics.com. [En línea] [Citado el: 22 de 2 de 2010.] http://www.iearobotics.com/personal/juan/proyectos/sercom/sercom.html.

10. smartmeters.com. [En línea] [Citado el: 23 de 2 de 2010.] http://www.smartmeters.com/the-news/763-ieee-announces-dnp3-protocol-as-standard-for-smart-grid.html.

11. http://www.upseros.com. [En línea] [Citado el: 24 de 2 de 2010.] http://www.upseros.com/fotocopiadora/ficheros/Comunicaciones%20I/practica%20del%20interfaz%20rs%20232.pdf.

12. **Booch, Ivar Jacobson y Grady.** *El proceso unificado de desarrollo de software.* Ciudad Habana : s.n., 2004.

13. **Trujillo, Dr. Rafael.** *Especificación de los Requerimientos de software para módulo de Drivers.* Merida,Venezuela : s.n., 2006.

14. *Casi todas las pruebas del software.* **Prado, Elena Raja.** 4, s.l. : Actas de Talleres de Ingeniería de Software y Bases de Datos, 2007, Vol. Vol1.

Anexos.

Anexo 1. Especificación de la clase DNP3Device

Tabla 6: Clase del diseño DNP3Device

Nombre: DNP3Device	
Tipo de clase:	
Atributo	Tipo
startBlockAddress	unsigned long
library	DriversLibrary*
provider	ITransportProvider*
transport	ITransport*
deviceTimer	IAsyncTimer*
snifferHandler	SnifferHandler
endPoint	DNP3EndPoint*
forRead	bool
kindRead	KindRead
Para cada responsabilidad	
Nombre:	DNP3Device()
Descripción:	Constructor por defecto.
Nombre:	getCOMMSTATE()
Descripción:	Función para obtener el estado de la comunicación
Nombre:	resetDiagnosticParameters()
Descripción:	Inicializa los contadores de la información de diagnostico del DNP3EndPoint relacionado con el DNP3Device
Nombre:	getEndPoint()
Descripción:	Devuelve el endpoint utilizado por la clase.
Nombre:	enableSniffer()

Descripción:	Habilita o deshabilita el uso del evento del Sniffer.
Nombre:	getDeviceTransport()
Descripción:	Devuelve la instancia del transporte creada
Nombre:	getDeviceTimer()
Descripción:	Devuelve la instancia del timer asíncrono creada
Nombre:	incrementFailure()
Descripción:	Incrementa el conteo de fallas detectadas.
Nombre:	incrementSucces()
Descripción:	Incrementa el conteo de acciones realizadas.
Nombre:	createBlock()
Descripción:	Responde por la creación de un Bloque de variables.
Nombre:	createIProtocolAddress()
Descripción:	Crea una instancia de IProtocolAddress a partir de la representación textual de la dirección dada por el parámetro address
Nombre:	startBlock()
Descripción:	Establece una dirección como comienzo de un bloque
Nombre:	fitInBlock()
Descripción:	Debe determinar si el rango de direcciones definido entre la dirección establecida por startBlock y la dirección que se define por los parámetros del método, puede alojarse en un bloque del protocolo.
Nombre:	fitInBlockSingle()
Descripción:	Debe determinar si una variable puede alojarse en un bloque. Esta determinación se hace a través de las direcciones address1 y address2 que representan las direcciones de protocolo de la menor y mayor de las direcciones simples de la variable respectivamente.
Nombre:	compare()
Descripción:	Compara dos direcciones de protocolo de acuerdo a las reglas del protocolo.
Nombre:	setNumberOfRetries()
Descripción:	Función estática utilizada por DNP3DeviceMetaClass para el establecimiento de la propiedad relacionada con la cantidad de reintentos en caso de error del dispositivo.

Nombre:	getNumberOfRetries()
Descripción:	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad relacionada con la cantidad de reintentos en caso de error del dispositivo.
Nombre:	setRetriesInterval()
	Función estática utilizada por DNP3DeviceMetaClass para el establecimiento de la propiedad relacionada con el tiempo de espera entre reintentos en caso de error.
Nombre:	getRetriesInterval()
	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad relacionada con el tiempo de espera entre reintentos en caso de error
Nombre:	setSlaveID()
	Función estática utilizada por DNP3DeviceMetaClass para establecer el valor de la propiedad relacionada con el identificador del dispositivo físico.
Nombre:	getSlaveID()
	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad relacionada con el identificador del dispositivo físico.
Nombre:	setMasterID()
	Función estática utilizada por DNP3DeviceMetaClass para establecer el valor de la propiedad relacionada con el identificador de la estación maestra.
Nombre:	getMasterID()
	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad relacionada con el identificador de la estación maestra.
Nombre:	setUseSingleWrite()
	Función estática utilizada por DNP3DeviceMetaClass para establecer el

	valor de la propiedad relacionada con el uso de las escrituras simples.
Nombre:	getUseSingleWrite()
	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad relacionada con el uso de las escrituras simples.
Nombre:	setResponseTimeOut()
	Función estática utilizada por DNP3DeviceMetaClass para establecer el valor de la propiedad relacionada con el tiempo que se debe esperar una respuesta.
Nombre:	getResponseTimeOut()
	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad relacionada con el tiempo que se debe esperar una respuesta.
Nombre:	setConnectionTimeOut()
	Función estática utilizada por DNP3DeviceMetaClass para establecer el valor de la propiedad relacionada con el tiempo que se debe esperar para una confirmación de error de conexión.
Nombre:	getConnectionTimeOut()
	Función estática utilizada por DNP3DeviceMetaClass para establecer el valor de la propiedad relacionada con el tiempo que se debe esperar para una confirmación de error de conexión.
Nombre:	getEndianness()
	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad relacionada con el ordenamiento de los bytes
Nombre:	setEndianness()
	Función estática utilizada por DNP3DeviceMetaClass para establecer el valor de la propiedad relacionada con el ordenamiento de los bytes
Nombre:	getErrConnectionTimeOut()
	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad de diagnostico para obtener el valor de la propiedad de diagnostico relacionada con la cantidad de errores de time out de conexión.
Nombre:	getErrConnectionRefused()

	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad de diagnostico relacionada con la cantidad de errores de conexión rehusada
Nombre	getErrMsgTimeOut()
	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad de diagnostico relacionada con la cantidad de errores time out de lectura
Nombre	getErrFrame()
	Función estática utilizada por DNP3DeviceMetaClass para obtener el valor de la propiedad de diagnostico relacionada con la con la cantidad de errores de trama

Anexo 2. Especificación de la clase DNP3DriverMetaclass

Tabla 7: Clase del diseño DNP3DriverMetaclass

Nombre: DNP3DriverMetaclass	
Tipo de clase:	
Para cada responsabilidad	
Nombre:	DNP3DriverMetaclass()
Descripción:	Constructor por defecto.
Nombre:	registerProperties()
Descripción:	El método registerProperties es el método donde se deben registrar las propiedades de configuración y los parámetros de diagnostico que almacena la metaclase.
Nombre:	internalCreateInstance()
Descripción:	Es un método virtual que responde por la creación de una instancia de la clase Driver

Anexo 3. Especificación de la clase DNP3DeviceMetaclass.

Tabla 8: Clase del diseño DNP3DeviceMetaclass

Nombre: DNP3DeviceMetaclass	
Tipo de clase:	
Para cada responsabilidad	
Nombre:	DNP3DeviceMetaclass()
Descripción:	Constructor por defecto.
Nombre:	registerProperties()
Descripción:	El método registerProperties es el método donde se deben registrar las propiedades de configuración y los parámetros de diagnostico que almacena la metaclase.

Anexo 4. Especificación de la clase DNP3IProtocolAddress.

Tabla 9: Clase del diseño DNP3IProtocolAddress

Nombre: DNP3IProtocolAddress	
Tipo de clase:	
Atributo	Tipo
valid	bool
size	unsigned int
atributte	char*
forRead	bool
Para cada responsabilidad	
Nombre:	DNP3IProtocolAddress ()
Descripción:	Constructor por defecto.
Nombre:	getValid()
Descripción:	Retorna un valor booleano que expresa si la dirección es válida o no de acuerdo a las reglas especificas del protocolo

Nombre:	checkAddress()
Descripción:	La función checkAddress() retorna un valor booleano que expresa si de la dirección entrada es válida o no lo es

Glosario de términos.

B:

Boost: Ofrece un conjunto de bibliotecas portables para código en C++.

Broadcast: (Difusión). Sistema de entrega que proporciona la copia de un paquete dado a todos los anfitriones conectados para la difusión del paquete.

Bus de campo: Es un sólo enlace de comunicaciones, al cual se conectan directamente todos los dispositivos. Existen dos formas de comunicación en esta topología, colisión y maestro/esclavo.

C:

CRC: El CRC es un código de detección de error, cuyo cálculo es una larga división de computación en el que se descarta el cociente y el resto se convierte en el resultado, con la importante diferencia de que la aritmética que es usada determina que el cálculo utilizado es el arrastre de un campo finito, en este caso los bits.

F:

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

H:

Handler: Manipulador. Rutina de software que realiza una determinada tarea. Por ejemplo, cuando se detecta un error, se llama a un manipulador de error para recuperarse de esa condición. En este contexto el handler es utilizado en forma de función y de estructura o clase.

I:

ISO: (International Organization for Standardization). Organización internacional que bosqueja, discute, propone y especifica estándares para los protocolos de red. ISO es mejor conocido por su modelo de referencia de 7 capas que describe la organización

conceptual de los protocolos

L:

Licencia GPL: (GNU General Public License). Las licencias de la mayoría del software están diseñadas para eliminar su libertad de compartir y modificar dicho software. Por contra, la GNU General Public License (GPL) está diseñada para garantizar su libertad de compartir y modificar el software. Software libre para garantizar la libertad de sus usuarios. Esta licencia GNU General Public License (GPL) se aplica en la mayoría de los programas realizado por la Free Software Foundation (FSF, Fundación del Software Libre) y en cualquier otro programa en los que los autores quieran aplicarla.

P:

Protocolos de comunicación: son como reglas de comunicación que permiten el flujo de información entre computadoras o dispositivos diferentes que manejan lenguajes distintos.

Productor-consumidor: Paradigma de programación de redes, los paquetes de datos se identifican de acuerdo con los contenidos de sus datos en vez de su ubicación física. Todos los nodos "escuchan" en la red y consumen aquellos paquetes de datos que tienen los identificadores apropiados.

R:

Router: (Ruteador). Computadora dedicada o dispositivo, de propósito especial que se conecta a dos o más redes y envía paquetes de una red a la otra. En particular un ruteador IP, envía datagramas IP entre las redes a las que está conectado. Utiliza las direcciones de destino de un datagrama para decidir el próximo salto al que enviará el datagrama.

T:

Trama: Es una unidad de envío de datos. Viene a ser sinónimo de paquete de datos.

Índice de ilustraciones y tablas.

Índice de Ilustraciones.

Índice de Tablas.