Universidad de las Ciencias Informáticas

Facultad 5

Centro de Informática Industrial

Título: Manejador para la comunicación con dispositivos UP1-LCD

Trabajo de diploma para optar por el título de:
Ingeniero en Ciencias Informáticas

Autora: Lisandra Villarino Artiaga

Tutor: Ing. Adrián Carlos Moreno Borges
Co-Tutor: Ing. Pedro Alberto Uriarte Rodríguez

Ciudad de La Habana

"Año del 53 Aniversario del Triunfo de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro ser autora 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 _____ del año _____.

_____	_____	_____
Firma de la Autora (Lisandra Villarino Artiaga)	Firma del Tutor (Ing. Adrián Carlos Moreno Borges)	Firma del Co-Tutor (Ing. Pedro Alberto Uriarte Rodríguez)

DATOS DE CONTACTO

Ing. Adrián Carlos Moreno Borges
Email: acmoreno@uci.cu

Profesor Instructor, Graduado de la Universidad de las Ciencias Informáticas en el 2008. Tiene 6 años de experiencia en el desarrollo de software, ha participado en la realización de los drivers GE FANUC, DNP3 - IP y BSAP - IP.

Ing. Pedro Alberto Uriarte Rodríguez
Email: pauriarte@uci.cu

Graduado de la Universidad de las Ciencias Informáticas en el 2010. Posee 4 años de experiencia en el desarrollo de software.

AGRADECIMIENTOS

A mis padres Antonio y Meivis, por todo su apoyo, cariño y afán en convertirme en la persona que soy hoy y seré siempre. Papi gracias por tu amor, eres el mejor papá del mundo. Mami gracias por todo y me gustó mucho el traje, te quiero. Estoy orgullosa de ser su hija.

A mi segundo papá Ángel por su cariño y por ver en mi otra hija. A mi hermanita Laura por su amor y ternura. Eres especial, niña mía.

A mi amado novio por su paciencia, ayuda, comprensión e infinito amor. Gracias Yasma por amarme, soy feliz de tenerte a mi lado. Te amo.

A mi familia, porque de una forma u otra se han preocupado por mí en algún momento de mi vida, al igual que a mi suegra.

A todos mis maestros que desde chica me enseñaron el amor hacia los estudios y a querer superarme en la vida, y junto a los profesores de la carrera ayudaron en mi formación intelectual y profesional.

A mis amigos y compañeros por brindarme su apoyo cuando lo he necesitado.

A mis tutores por su ayuda, consejos y apoyo durante todo el proceso de elaboración de la tesis.

A la Revolución por darme la oportunidad de estudiar en esta universidad maravillosa, y por hacer parte de mis sueños realidad.

DEDICATORIA

A mis padres.

A mi hermana Laura.

A mí amado novio Yasmany.

A mis abuelitos y a los que ya no están.

RESUMEN

En el presente trabajo se describe el desarrollo de un manejador para la comunicación con dispositivos UP1-LCD. La necesidad del mismo surge porque el SCADA GECTEL, sistema de gestión en desarrollo para la empresa cubana ETECSA, no cuenta con un mecanismo para comunicarse con los dispositivos antes mencionados. Los UP1-LCD son controladores acoplados a grupos electrógenos, con los que cuenta esta empresa, para que en caso de fallas eléctricas sea posible mantener los servicios brindados a la población y al país de forma general.

A través de los UP1-LCD se conoce toda la información referente a los grupos electrógenos, por ejemplo el nivel de carga que tiene, con cuanto combustible cuenta para proporcionar energía, tiempo en horas de duración de la carga entre otras. Es de gran importancia el conocimiento de este tipo de información. De esta forma se contribuye a la correcta toma de decisiones en la empresa ante situaciones excepcionales o de máxima prioridad.

Palabras clave: manejador, SCADA GECTEL, UP1-LCD

ABSTRACT

The present work describes the development of a driver for the UP1-LCD devices. The need for it arises because the SCADA GECTEL, management system for the Cuban company ETECSA, has no mechanism for communicating with the aforementioned devices. The UP1-LCD controllers are coupled to generators, which the company has, so that in case of power failure is possible to maintain the services provided to the population and the country itself.

Using the UP1-LCD devices is possible to know all the information concerning the generators, for example the level of charge, how much fuel they have to provide energy, time in hours of load among others. It is very important to know this information because it will contribute to a good decision-making in the company in front of exceptional situations or high priority.

Keywords: *driver, SCADA GECTEL, UP1-LCD*

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

Las nuevas tecnologías están creando profundos cambios económicos y tecnológicos en el ámbito mundial, con visibles consecuencias sociales, políticas y laborales. Desde la antigüedad los cambios tecnológicos han tendido a facilitar el trabajo humano, reemplazando la fuerza física por la capacidad mental y la inteligencia de los hombres. En la actualidad el desarrollo alcanzado por los productos informáticos tiende a reemplazar también la parte más rutinaria y mecánica de la actividad mental humana por el trabajo de las computadoras. (1)

La automatización es entonces el sistema de fabricación diseñado con el fin de usar la capacidad de las máquinas para hacer determinadas tareas anteriormente efectuadas por seres humanos, y para controlar la secuencia de las operaciones sin intervención humana**.** Los sistemas SCADA (*Supervisory Control and Data Acquisition* por sus siglas en inglés o en español Supervisión y Control de Adquisición de Datos) surgen debido a la necesidad de automatizar los diferentes procesos que pueden existir en las industrias o en otros entornos. A través de estos sistemas se pueden supervisar y controlar importantes indicadores que permiten una realimentación del proceso en cuestión.

Actualmente los SCADA son empleados en diferentes ramas de la industria. En Cuba, existen instituciones dedicadas a la producción de software, como la Universidad de las Ciencias Informáticas (UCI). En estos momentos, entre los proyectos que enfrenta el CEDIN (*Centro de Informática Industrial*), de la Facultad 5, se encuentra el SCADA-ETECSA. Para el mismo, se quiere desarrollar un sistema que manejará la supervisión y control de los equipos ubicados en las instalaciones de Energía y Climatización pertenecientes a ETECSA (*Empresa de Telecomunicaciones de Cuba*) a lo largo de todo el país, que hasta ahora son controlados de forma manual, lo cual retrasa el trabajo e influye en la calidad del mismo.

La empresa cubana ETECSA tiene una alta responsabilidad en el desarrollo socio-económico del país, y en especial en la informatización de la sociedad, al garantizar la infraestructura tecnológica para una efectiva conectividad. (2)

Centra su actividad en la calidad de su capital humano, sobre la base de una gestión integral para la prestación de modernos servicios de beneficio popular, tanto en las ciudades como en las zonas rurales de difícil acceso. En los procesos inversionistas respeta el medio ambiente. (2)

Con la puesta en marcha de este sistema contribuirá al emprendimiento de acciones satisfactorias a favor de la protección y buen funcionamiento de los dispositivos tecnológicos de la empresa. También permitirá brindar soluciones parciales a cortos plazos, hasta completar la solución integral. Este producto recibe el nombre de SCADA GECTEL (*Sistema de Gestión de Energía y Climatización de la Empresa de Telecomunicaciones de Cuba*)

En las áreas mencionadas se cuentan con varios grupos electrógenos. Estos entran en funcionamiento una vez que se corta el abastecimiento de corriente eléctrica de la red nacional. Permitiendo así mantener activos los servicios que brinda ETECSA a la población y al país en general. Una demora en la puesta en marcha de los grupos electrógenos influiría negativamente en muchos sectores de gran importancia de nuestro país. Por lo que se hace necesario que los operadores de estos generadores conozcan datos de máximo interés en el menor tiempo posible, como: nivel de energía, cantidad de combustible restante dígase petróleo o gasolina, tiempo aproximado en horas de duración de la carga entre otros.

Con la instalación de este sistema se podrá disponer de información para la toma de decisiones rápidas y eficaces que contribuyan a un mayor funcionamiento de la empresa. La información que proporcionan los grupos electrógenos antes mencionadas se adquiere a través de dos tipos de controladores, siendo uno de ellos el UP1-LCD. Pero el sistema GECTEL, no cuenta con un mecanismo para comunicarse con este tipo de dispositivos.

Dada esta situación polémica, surge el siguiente **problema científico**: ¿Cómo proporcionarle al sistema GECTEL acceso a la información brindada por los dispositivos UP1-LCD?

Para resolver este problema se planteó como **objeto de estudio**: El proceso de adquisición de datos en los sistemas de Supervisión y Control.

Como **objetivo general**: Desarrollar un manejador que permita la comunicación entre el dispositivo UP1-LCD y el sistema GECTEL.

El **campo de acción** lo constituyen los mecanismos de acceso a la información de los dispositivos UP1-LCD.

Para dar cumplimiento al objetivo planteado se tuvieron en cuentas las siguientes **tareas de investigación:**

1. Estudio de la documentación referente a manejadores, para el establecimiento de los conceptos básicos asociados a su implementación.
2. Estudio del framework DriverCore para el diseño e implementación del manejador UP1- LCD.
3. Estudio de bibliografía referente a la especificación del protocolo que permite la comunicación con el UP1-LCD, para la implementación de la Capa de Protocolo del manejador.
4. Diseño e implementación de la Capa de Protocolo del Manejador UP1-LCD.
5. Diseño e implementación de la Capa de Driver del Manejador UP1-LCD.
6. Diseño y ejecución de casos de prueba para detectar posibles errores en el manejador.

Por lo que se plantea la siguiente **idea a defender**: Con el desarrollo del manejador para la comunicación con los dispositivos UP1-LCD, se espera lograr un intercambio de información entre el SCADA GECTEL y estos. Esto permitiría disponer de los datos que brindan los UP1-LCD, necesarios para la toma de decisiones rápidas y eficaces ante situaciones excepcionales en la empresa.

Hasta el momento el proceso de supervisión y control de las áreas antes mencionadas transcurre muy lento. Los operadores tienen que trasladarse al patio donde se encuentran los grupos electrógenos e informar la situación para ese momento. Con un sistema que automatice esto, se ganaría en tiempo y en recursos, pero si no se cuenta con el software necesario la inversión no daría sus frutos. El desarrollo de este manejador para el intercambio de datos con los controladores UP1-LCD proporciona al sistema GECTEL de información rápida y precisa acerca de estos generadores. Permitiendo así la toma de decisiones acertadas en situaciones de máxima prioridad.

Lo novedoso de este trabajo se centra en la obtención de un producto multiplataforma, desarrollado sobre plataforma Linux. Pues anterior a este no se contaba con ningún otro de características similares.

Constituirá un módulo independiente del sistema al cual se acople, facilitando su reutilización en otros. Así como la agregación de funcionalidades nuevas según lo requerido.

En la presente investigación se utilizarán diferentes métodos de investigación científica, como por ejemplo:

- ✓ **Analítico-Sintético**: utilizado en el análisis de las teorías principales referentes al desarrollo de manejadores y los elementos más importantes relacionados con el UP1-LCD.
- ✓ **Histórico-Lógico:** permite realizar un estudio del fenómeno en cuestión y su evolución histórica, permitiendo así profundizar los conocimientos sobre el tema.
- ✓ **Modelación**: haciendo uso del mismo se realizarán diagramas o modelos, que permitirán estructurar teóricamente el sistema, facilitando su comprensión.
- ✓ **Observación**: permitió valorar la necesidad de hacer un manejador para el intercambio de información con los dispositivos UP1-LCD sobre plataforma libre.

La presente investigación está estructurada por 4 Capítulos. El Capítulo 1 se concibe para reflejar un estado del arte sobre los principales conceptos, herramientas y metodologías empleadas a lo largo de la investigación. El Capítulo 2 está relacionado con las características del manejador a desarrollar, sus requisitos funcionales y no funcionales. La interacción con las bibliotecas DriverCore y TransportProvider. Las características más generales del protocolo en cuestión, entre otros aspectos. Seguidamente el Capítulo 3, estará dedicado al diseño e implementación del manejador. Se podrá observar la estructura del diseño que tendrá el manejador, facilitando así su comprensión. Además las características más generales del desarrollo del mismo. Por último, el Capítulo 4 estará enfocado a las pruebas realizadas al manejador una vez terminado. En el mismo se ejemplificarán los casos de Uso de Pruebas que se diseñen y los resultados de la aplicación de los mismos.

Capítulo 1: Fundamentación Teórica

1.1. Sistemas SCADA

Un SCADA representa una solución informática para la automatización de los procesos industriales de una fábrica y/o planta industrial, como las telecomunicaciones, el agua, el control de residuos, energía, petróleo y gas y transporte. Estos sistemas permiten la transferencia de información entre un ordenador central y los dispositivos de campo, como los controladores lógicos programables *(Programmable Logic Controller* por sus siglas en inglés*)*. Un SCADA lo podemos encontrar tanto en lo simple como en lo complejo, es decir, supervisando las condiciones climáticas de una oficina o monitoreando toda la actividad en una planta de energía nuclear o refinería de petróleo. Por lo que de forma general se consideran como funciones básicas de un sistema SCADA las presentadas a continuación:

1. Supervisión Remota de Instalaciones
2. Control Remoto de Instalaciones
3. Procesamiento de Información
4. Presentación de Gráficos Dinámicos
5. Generación de Reportes
6. Presentación de Alarmas
7. Almacenamiento de Información Histórica
8. Presentación de Gráficos de Tendencias
9. Programación de Eventos

1.1.1. Requisitos

Un SCADA debe cumplir varios objetivos para que su instalación sea perfectamente aprovechada:

- Deben ser sistemas de arquitectura abierta, capaces de crecer o adaptarse según las necesidades cambiantes de la empresa
- Deben comunicarse con total facilidad y de forma transparente al usuario con el equipo de planta y con el resto de la empresa (redes locales y de gestión).

- Deben ser programas sencillos de instalar, sin excesivas de hardware, y fáciles de utilizar, con interfaces amigables con el usuario. (3)

1.1.2. Fabricantes

En la actualidad la producción de software ha marcado el desarrollo futuro. Los países más desarrollados han expandido su poderío en este sector del mercado, reduciendo cada vez más las posibilidades de los países que incursionan en este espacio. Nuestro país tiene una avanzada modesta en este aspecto, pues a pesar de existir trabas, como el Bloqueo Económico impuesto por los Estados Unidos y la brecha tecnológica con respecto a las potencias del primer mundo, el camino al desarrollo se está trazando poco a poco.

En la siguiente imagen se ejemplifican algunas de las principales empresas productoras de software, específicamente sistemas SCADA, en el ámbito internacional, como el nacional.

Figura: 1 Productores de SCADA

Fabricantes Internacionales

Siemens: Empresa alemana fundada por Werner von Siemens y Johann Halske en 1847 en Berlín. En la actualidad se dedica a las telecomunicaciones, transporte, medicina, energía y otras áreas de la ingeniería.

- **Producto**: Con el ***SCADA SIMATIC WinCC***, esta empresa ofrece un sistema innovador, escalable y visualización de procesos, con numerosas funciones de alto rendimiento para el monitoreo de procesos automatizados. Ya sea en un sistema de usuario único o un sistema distribuido multiusuario con servidores redundantes, el sistema ofrece una funcionalidad completa para todas las industrias y funciones apertura óptima

Rockwell Automation: Empresa norteamericana dedicada a brindar soluciones informáticas para la automatización de industrias. Entre sus marcas más reconocidas se encuentran Allen-Bradley y Rockwell Software.

- **Producto**: ***FactoryTalk View*** proporciona una suite de productos de software HMI diseñados con una apariencia común, para ayudar a acelerar el desarrollo de aplicaciones HMI y el tiempo de preparación. Soporta la arquitectura integrada de Rockwell Automation. FactoryTalk View incluye las herramientas: FactoryTalk View Studio, FactoryTalk View Machine Edition y FactoryTalk View Site Edition, todas estas basadas en la arquitectura PC. Dentro de los beneficios que proporciona se encuentran: maximizar la disponibilidad del sistema con una función de detección de fallos y recuperación. Además la prestación de servicios comunes, tales como la seguridad, alarmas, y diagnóstico con otros productos FactoryTalk habilitados.

Wonderware: Marca de software para la automatización industrial, propiedad de Invensys Operations Management. Las aplicaciones Wonderware son utilizados en sectores como minería, petróleo, gas, alimentos y bebidas, productos farmacéuticos, transporte, entre otros.

- **Producto**: ***Wonderware InTouch™ HMI***: es un potente sistema SCADA para la automatización industrial, supervisión y control de procesos. InTouch permite a los usuarios visualizar y controlar los procesos mientras que proporciona a los ingenieros un entorno de desarrollo fácil de usar y una amplia funcionalidad para crear, probar y desplegar potentes aplicaciones de automatización que se conectan y entregan información en tiempo real. InTouch es un software abierto y extensible al operador, que permite flexibilidad en el diseño de aplicaciones específicas con conectividad con un amplio conjunto de dispositivos de automatización en la industria.

Fabricantes Nacionales

CEDAI: fue fundada en Febrero de 1978, con el objetivo de brindar soluciones integrales en materia de automática, es una entidad cubana que se dedica al diseño ingenieril, montaje y posventa de proyectos integrales de automatización.

- **Producto**: uno de sus primeros productos fue un sistema automatizado de inmersión temporal (con supervisión remota incluida) llamado *Biosys*. Creado a petición del Instituto de Biotecnología de las Plantas de Villa Clara (IBP). Contenía un diseño sencillo de todos los niveles de automatización de un sistema de inmersión temporal sin embargo su diseño cerrado y poco escalable lo limita ante las tendencias actuales y lo imposibilita de una generalización que pueda a niveles de exportación. (4)

SerConi: Empresa de Servicios de Computación, Comunicaciones y Electrónica del Níquel "Rafael Fausto Orejón Forment", ubicada en Nicaro, municipio Mayarí de la provincia Holguín. No está categorizada como productora de software, pero vale la pena mencionarla por sus aportes en este campo.

- **Producto**: cuenta con el SCADA EROS, apoyado en autómatas que registran y le envían la información desde diferentes lugares donde esté instalado. Tuvo sus inicios en 1994, probándose por primera vez en la planta de Lixiviación y Lavado de la Fábrica de Níquel "Ernesto Che Guevara" de Moa. Actualmente el sistema se encuentra en la versión 5.0 y por la cantidad de copias instaladas es el más difundido en el país.

UCI: fundada en el 2002, a pesar de ser una institución dedica a formar profesionales universitarios, posee una infraestructura productiva. Cuenta con varios centros productivos de software, entre ellos, el CEDIN. En el mismo se ofrecen soluciones y servicios asociados a la informática industrial.

- **Producto**: entre los productos de este centro se encuentra el SCADA "Guardián del ALBA (GALBA)"; permite la solución de aplicaciones de supervisión y control de procesos, utilizando para ello una arquitectura distribuida de módulos que permiten escalar a aplicaciones de gran envergadura. Actualmente se encuentra en funcionamiento en la empresa petrolera PDVSA.

1.2. Manejador de Dispositivo

Se entiende como manejador (*drivers* en inglés*)* a un software, o en algunos casos componentes hardware, encargado del control de un dispositivo o una familia de ellos. Permite la comunicación entre el Sistema Operativo y el hardware. El manejador oculta al sistema operativo las particularidades del hardware, permitiendo que éste pueda acceder al dispositivo a través de una interfaz estandarizada. Después de cada operación realizada por el mismo se haya completado, este debe verificar si hubo errores.

Los manejadores de dispositivos constituyen piezas esenciales de los sistemas computarizados, pues sin ellos no se podría usar el hardware. Se puede afirmar que existen tantos tipos de controladores como tipos de periféricos, y con mucha frecuencia se puede encontrar más de un manejador para un mismo dispositivo, donde cada uno ofrece un nivel distinto de funcionalidades. (5)

En el caso de los sistemas SCADA, utilizan los manejadores para comunicarse con los dispositivos de campo e intercambiar información con ellos, de esta manera el sistema puede manejar la información que proporcionan los equipos.

1.3. Controladores Lógicos Programables (PLC)

Se podría definir como un dispositivo operado digitalmente, que usa memoria para el almacenamiento interno de instrucciones con el fin de implementar funciones específicas, tales como lógica , secuenciación , registro y control de tiempos, conteo y operaciones aritméticas, para controlar a través de las entradas/salidas digitales o analógicas varios tipos de máquinas o procesos.

Los PLC fueron inventados como respuesta a las necesidades de la industria automotriz, inicialmente fueron empleados por empresas para sustituir la lógica cableada. En la actualidad son muy utilizados en la automatización industrial

Los elementos que contiene son:

- Unidad Central de proceso
- Módulos de Entrada
- Módulos de Salida

- Fuente de Alimentación
- Dispositivos periféricos
- Interfaces

La unidad central es el “cerebro” del PLC, este toma decisiones relacionadas con el control de la máquina o proceso. Los módulos de entrada/salida son la sección del plc en donde sensores y actuadores son conectados y a través de los cuales monitorean y controla el proceso.

Un PLC trabaja en base a la información recibida por los captadores y el programa lógico interno, actuando sobre los accionadores de la instalación. Esta secuencia de acciones se ejercerá sobre las salidas del autómata a partir del estado de sus señales de entrada. Un autómata programable industrial, representa a la unidad de control dentro de un sistema de control. Estos dispositivos están diseñados para trabajar en tiempo real y en procesos industriales secuenciales (6)

1.4. Controlador UP1-LCD

Los PLC UP1-LCD son desarrollados por la empresa italiana Tecnocontrolli S.A.S, dedicada desde 1980 a la construcción de cuadros eléctricos por grupos electrógenos y moto-bombas, entre sus características generales se encuentran:

- Visualización de acontecimientos históricos con fecha y hora
- Envío de SMS a números programados sobre la guía telefónica
- Menú y mensajes operativos en italiano, español, francés e inglés
- Perfectamente intercambiable con UP1-G, ya sea eléctricamente como mecánicamente

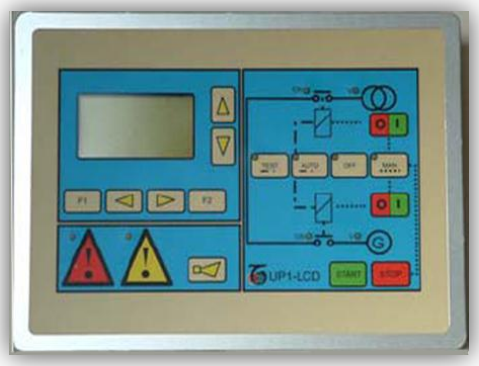

Figura: 2 Dispositivo UP1-LCD

La empresa Tecnocontrolli ofrece otros dos tipos de PLC muy similares al UP1-LCD, estos son el UP1-G y el UP15 como expansión del UP1-LCD.

1.4.1. Controlador UP15 como expansión del UP1-LCD

La lógica electrónica de los UP1-LCD no permite la transmisión y recepción de datos por la vía GSM-GPRS. Este sistema viene de uso frecuente, donde en la instalación del generador, la línea telefónica no es posible.

A través del módulo de expansión UP15, el UP1-LCD, puede enviar, en caso de anomalía del generador, de mensajes SMS de alarma desde 1 hasta 14 direcciones contemporáneamente.

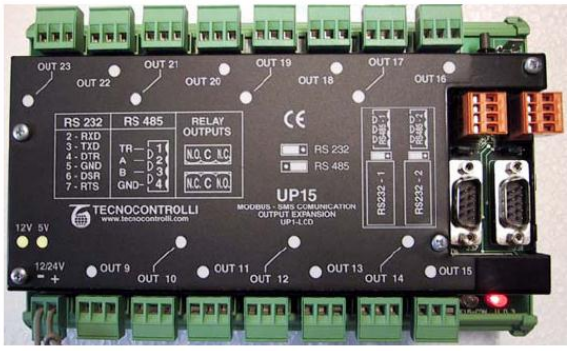

Figura: 3 UP15

En la figura 4, se observa el uso del UP15 en conexión con el UP1-LCD, vale aclarar que esta no es la única vía. Se puede también sustituir el conector RS-232 por el RS-485. De la misma manera en la figura 5, vemos como en vez de conectarse directo a una computadora se puede conectar con un modem.

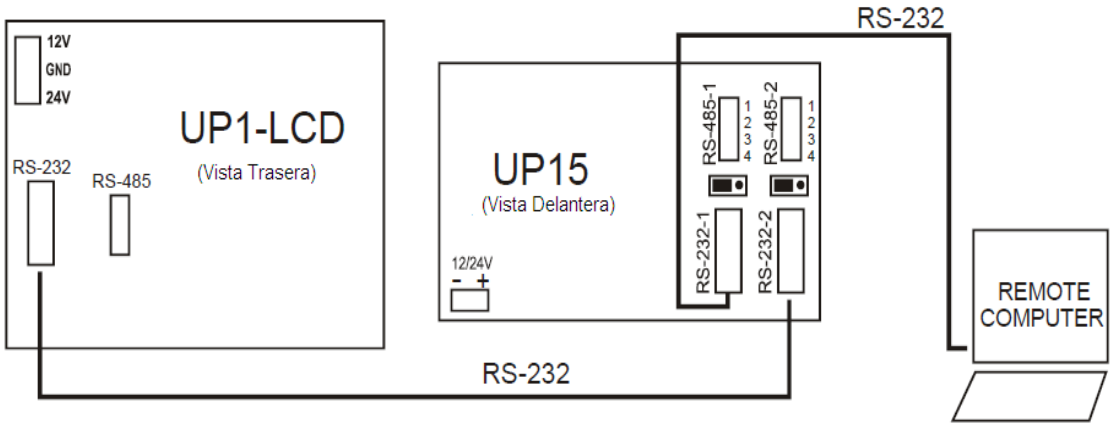

Figura: 4 Conexión del UP1-LCD con el UP15 directo al computador

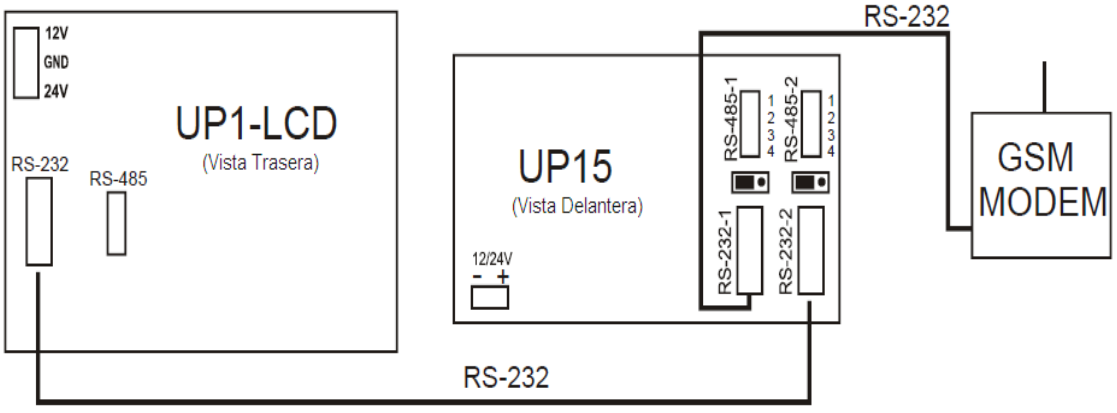

Figura: 5 Conexión del UP1-LCD con UP15 a un Modem GSM

El protocolo de comunicación que emplea este dispositivo sigue la convención MODBUS RTU, con los siguientes códigos de función:

Tabla 1: Códigos de Función empleando convención MODBUS RTU

Código	Función
04	Lectura de Input Registers
10	Escritura de Múltiples Registers
05	Escritura de Simple Coil

1.5. Protocolo de Comunicación

Un protocolo de comunicación se puede definir como el conjunto de reglas que gobiernan el intercambio de datos entre dos entidades (entiéndase por entidad a cualquier objeto capaz de enviar y recibir información). Los puntos clave que definen o caracterizan a un protocolo son:

- **La sintaxis**: incluye aspectos tales como el formato de los datos y los niveles de señal.
- **La semántica**: incluye información de control para la coordinación y el manejo de errores.
- **La temporización**: incluye la sintonización de velocidades y secuenciación. (7)

1.6. Interfaz de Comunicación

El manejador tema de este trabajo utiliza el puerto serie como interfaz de comunicación porque mediante este se comunica el dispositivo UP1-LCD. Un puerto serie es una interfaz de comunicación entre ordenadores y periféricos, mediante el cual los bytes de información son transmitidos bit a bit de forma secuencial. Se transmite un bit de comienzo (start bit), seguidamente los bits de información y por último un bit de parada (stop bit).

Típicamente, la comunicación serie se utiliza para transmitir datos en formato ASCII. Para realizar la comunicación se utilizan 3 líneas de transmisión: Tierra (o referencia), Transmitir, Recibir. Debido a que la transmisión es asincrónica, es posible enviar datos por una línea mientras se reciben datos por otra. (8)

Actualmente la mayoría de los periféricos que utilizan el puerto serie, han sido reemplazados por el puerto USB. Dentro de sus principales ventajas hallamos: brinda mayor velocidad de transmisión, y permite la conexión “en caliente”, es decir se pueden conectar y desconectar los periféricos sin necesidad de reiniciar la computadora. Sin embargo, los puertos serie todavía se encuentran en sistemas de automatización industrial y algunos productos industriales o de consumo. Para el caso específico del UP1-LCD soporta las variantes RS-232 y RS-485.

El UP1-LCD cuenta con interfaz RS-232 para las siguientes funciones:

- Conexión directa de ordenador para control y monitorización
- Conexión de Modem GSM o Analógicos
- Conexión con el módulo de expansión por repetición de alarmas/MODBUS (refiérase al UP15).

El UP1-LCD cuenta con interfaz RS-485 para las siguientes funciones:

- Conexión directa de ordenador para control y monitorización
- Conexión con el módulo de expansión por repetición de alarmas/MODBUS (refiérase al UP15).

A continuación se expondrán características generales de estos dos estándares.

1.6.1. Estándar RS-232

El puerto serial es muy utilizado en muchos de los instrumentos industriales y como componente de algunas computadoras. Debido a que el estándar se mantiene desde hace muchos años, la institución de normalización americana EIA (*Electronic Industries Association*) 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. 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. (9) El RS-232 está limitado a comunicaciones de punto a punto entre los dispositivos y el puerto serial de la computadora

Figura: 6 Puerto Serie, Norma RS-232 DB-9

1.6.2. Estándar RS-485

El estándar RS-485 es una mejora sobre RS-422 ya que incrementa el número de dispositivos que se pueden conectar (de 10 a 32) y define las características necesarias para asegurar los valores adecuados de voltaje cuando se tiene la carga máxima. El hardware de RS-485 se puede utilizar en comunicaciones seriales de distancias de hasta 4000 pies de cable.

1.7. Aplicación software para comunicarse con el UP1-LCD

El UP1-LCD no es uno de los PLC más difundidos en el mercado internacional, por tal motivo al buscar información referente al mismo es escasa la que se encuentra. Los datos obtenidos giran siempre entorno a las características técnicas del hardware y su fabricante. No se encontró fuente bibliográfica acerca de su protocolo de comunicación, por lo que no se conoce su nombre.

Sin embargo junto con el hardware, el fabricante proporciona una aplicación que permite establecer comunicación con el dispositivo. El inconveniente de este punto es que dicho software solo funciona en sistema operativo Windows. Con el uso del mismo es posible la conexión de cualquier PC remota al Modem GSM que se conecta al UP1-LCD, y la interacción con el mismo.
En las siguientes figuras se observan las principales funcionalidades del mismo:

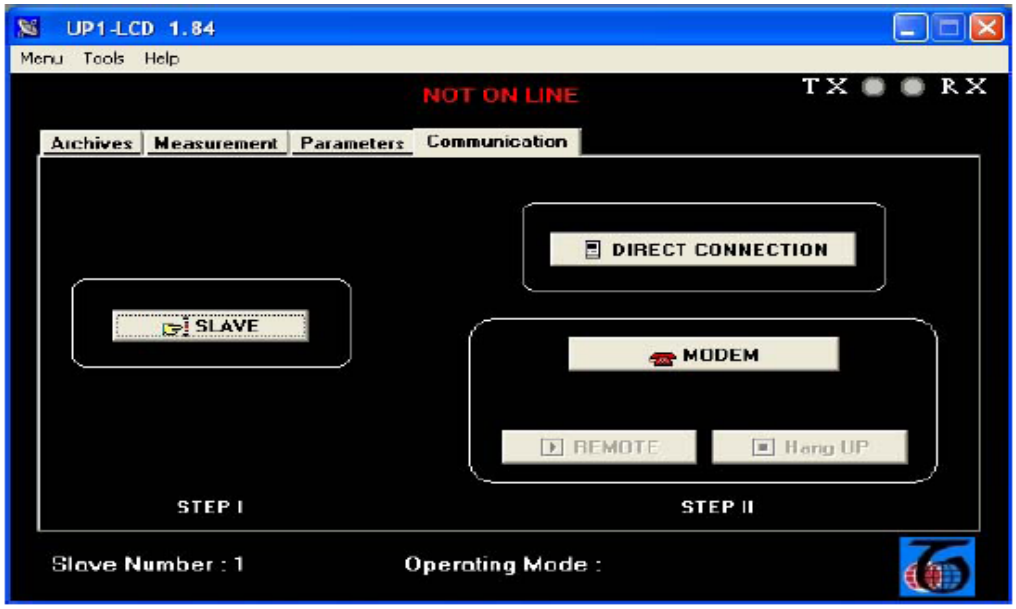

Figura: 7 Aplicación en Windows para el UP1-LCD

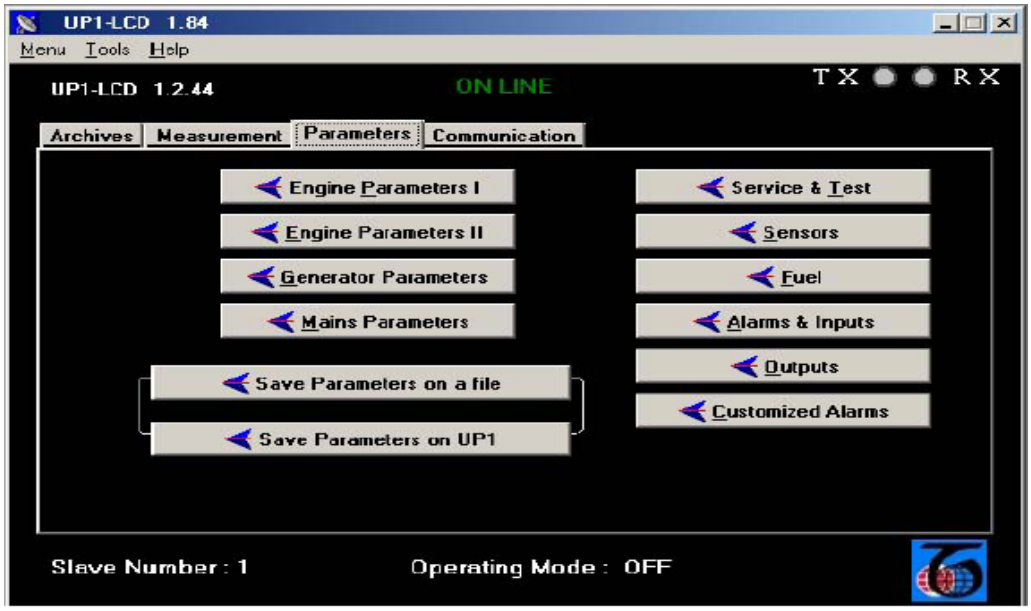

Figura: 8 Control y Cambio de todos los parámetros

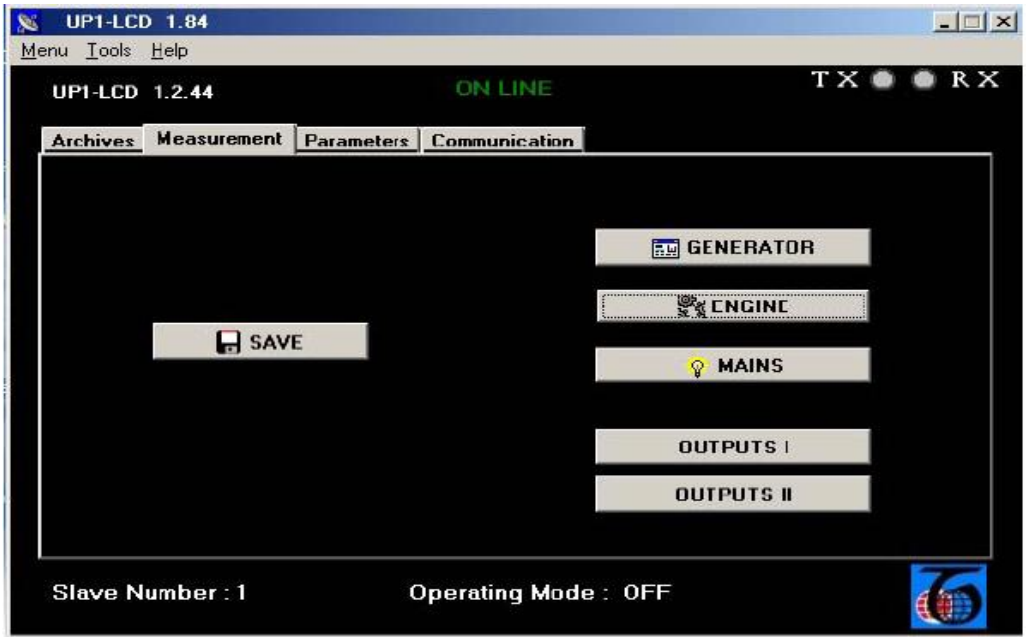

Figura: 9 Visualización y configuración de los parámetros del grupo electrógeno

1.8. Interfaz Genérica de Drivers (IGD)

Los datos recolectados en un SCADA, provienen de diferentes equipos, como por ejemplo autómatas, PLC, sensores inteligentes, reguladores, controladores, entre otros.

Es difícil que un SCADA contemple, en su código, todos los protocolos posibles para esta gran variedad de dispositivos. Por el contrario lo común es crear un protocolo genérico (o unos pocos) y la tarea de traducir este protocolo genérico a los protocolos específicos se le encarga a los manejadores, que, como regla, se programan en módulos independientes al SCADA. (10)

El reto principal en el diseño de la Interfaz Genérica ha sido la creación de una arquitectura, para los manejadores de dispositivos, que proporcione una interfaz estándar de acceso a los mismos y que oculte al SCADA las diferencias entre los protocolos y características de Hardware de los dispositivos que con él se enlazan. Al propio tiempo la generalidad de la arquitectura no debe impactar negativamente en el rendimiento de los manejadores, tema, que en la supervisión y control de procesos es muy importante. (10)

Teniendo como base la Interfaz Genérica se han podido desarrollar varios manejadores para la línea Recolección, entre los que se encuentran ModBus con sus variantes RTU, TCP y ASCII. También se encuentran el Ethernet/IP, ABEthernet, BSAP Serial y BSAP IP, DF1, DNP3, Hitachi 912.

1.9. Biblioteca TransportProvider

La biblioteca de transporte TransportProvider fue desarrollada en la línea de manejadores del proyecto Guardián del ALBA, basada en la biblioteca Asio C++. Permite el intercambio de información a través del puerto serie y del protocolo TCP/IP, posibilitando que el intercambio sea asíncrono, dándole una mayor eficiencia en tiempo y recursos a los manejadores que la emplean. El TransportProvider es multiplataforma, por lo que puede ser utilizado para desarrollar aplicaciones que necesiten comunicarse con dispositivos de campo. Es una biblioteca robusta y es compatible con la mayoría de los compiladores de C++ más utilizados actualmente.

1.10. Herramientas y tecnologías para el Desarrollo

Para la selección de las herramientas y metodología de desarrollo, a las que se les hace mención en este epígrafe, se tomó en consideración las escogidas en el documento de “Arquitectura de Software” v2.0 del proyecto SCADA ETECSA (11), pues es para el mismo que se desarrolla el presente manejador.

1.10.1. Lenguaje de Programación

Se emplea como lenguaje de programación a C++. Esta caracterizado por ser imperativo y orientado a objetos; aunque en realidad es un superconjunto de C, que nació para añadirle cualidades y características de las que carecía.

El resultado es que como su ancestro, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. (12)

1.10.2. Entorno Integrado de Desarrollo

Un Entorno Integrado de Desarrollo (por sus siglas en inglés *Integreted Development Environment*) es un software compuesto por varias herramientas, con el objetivo de ayudar a los programadores a escribir código rápido y eficiente. Dentro de las características de los IDE es que pueden ser por si solo una aplicación o simplemente formar parte de ellas. Dentro de sus componentes se encuentran Editor de Texto, Compilador, Intérprete, Depurador, Interfaz Gráfica de Usuario.

Como IDE se utiliza al Eclipse, caracterizado por ser de código abierto multiplataforma. Emplea módulos (en inglés *plugin*) para proporcionar todas sus funcionalidades, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Fue concebido para lenguaje de programación Java pero adicionalmente se le ha permitido extenderse usando otros lenguajes de programación como son C/C++ y Python.

1.10.3. Herramienta de Modelado

Como herramienta de modelado se utiliza Visual Paradigm. Entre sus principales características mencionar que es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue.

El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

1.10.4. Metodología de Desarrollo

Para el desarrollo de software existen diferentes metodologías, en este caso se utiliza OpenUP (*Open Unified Process*, por sus siglas en inglés*).* Se caracteriza por centrarse en el desarrollo de rápido de sistemas e involucra un conjunto mínimo de prácticas que ayudan a los equipos de trabajo a ser más efectivos.

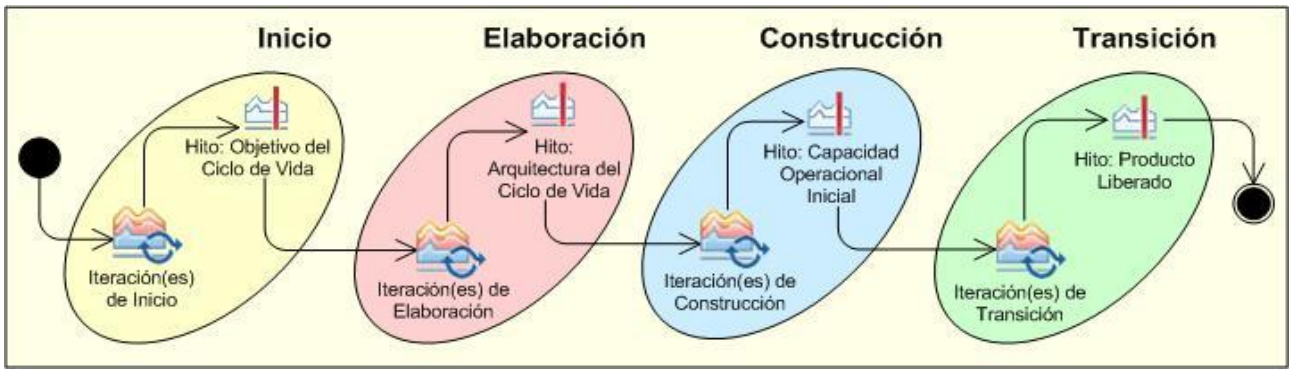

Figura: 10 Ciclo de Vida de OpenUP

Esta metodología integra una filosofía pragmática y ágil que se centra en la naturaleza colaborativa del desarrollo de software. Es un proceso antiburocrático y agnóstico en cuanto a herramientas (IDE, lenguajes, sistemas operativos, etc.) que puede ser usado tal como lo ha definido la fundación Eclipse (de donde procede el OpenUP), o que puede ser expandido y adaptado de acuerdo a las especificaciones de cada proyecto.

1.11. Conclusiones parciales del capítulo

En este capítulo se abordaron los conceptos fundamentales entorno al desarrollo de un manejador y sus principales características, lo que permitió profundizar los conocimientos en esta área y el papel que juega un manejador para un sistema SCADA.

También se investigó sobre la existencia de aplicaciones que hayan sido desarrolladas para la comunicación con dispositivos UP1-LCD por otras empresas y estudiar con profundidad las mismas. Por la poca información encontrada se considera que aparte del software que desarrolla el fabricante del equipo, no existe hasta el momento otro ejemplar. Por tal motivo solo se estudia el mencionado y se llega a la conclusión que al ser desarrollado para plataforma Windows, y siendo de código cerrado, pues el fabricante no proporciona el código fuente, es necesario desarrollar un manejador multiplataforma que permita al SCADA GECTEL obtener la información proporcionada por el controlador UP1-LCD, a la vez este manejador pueda ser usado en otros entornos en caso de necesitarse.

Capítulo 2: Características del Sistema

2.1. Características generales del protocolo para la comunicación con el UP1-LCD

2.1.1. Estructura de los mensajes

En el documento “Protocolo UP1-LCD” (13) v 3.0 aparece la especificación detallado de la estructura y tipos de mensajes para los UP1-LCD. Los mensajes enviados tienen dos tipos de estructura, pueden ser mensajes de lectura o mensajes de escritura. De manera genérica ambos se caracterizan por tener un carácter de inicio de trama SOH (0x01) y un carácter de fin de trama EOT (0x04). A continuación se especificará la estructura para ambos casos.

2.1.1.1. Formato de mensaje de Lectura

En el caso específico de la trama de lectura, el mensaje debe quedar conformado como lo muestra la figura 11. A continuación del carácter de inicio, le sigue el valor del id esclavo (solo se admite 1 ó 0), seguidamente se específica el comando que corresponde a la petición de lectura que se desee hacer, los posibles significados para cada comando se detallan en el epígrafe “Tipos de comandos”. A este campo le sigue la suma de chequeo, la fórmula para calcularlo se específica en el epígrafe “Cálculo del Checksum” y por último el carácter de fin de trama.

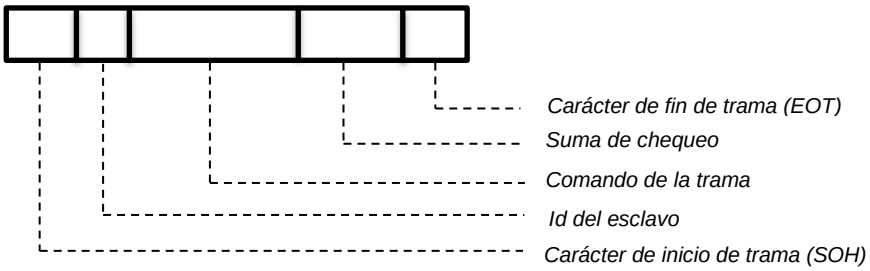

Figura: 11 Formato de la trama de lectura

Al pedido realizado por la unidad de control, el dispositivo le responde con un mensaje que el manejador debe interpretar. Esta trama de respuesta debe seguir el formato ejemplificado en la figura 12. Para el caso de las tramas de respuestas a los pedidos de lectura como de escritura, siguen también la estructura de las tramas que las invocan, es decir, tienen el

carácter de inicio de trama SOH (0x01) y el de fin de trama EOT (0x04). Después del carácter SOH le sigue un campo de datos, en el cual se envía la información requerida por el pedido. El tamaño de este campo varía según el comando de lectura especificado. A este campo le sigue la suma de chequeo correspondiente al mensaje y finalmente el carácter EOT.

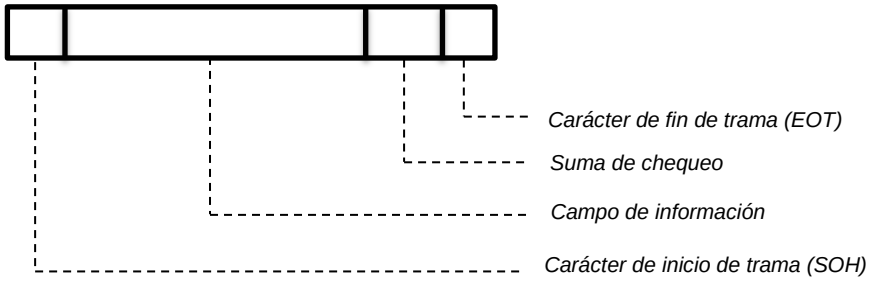

Figura: 12 Formato de la trama de respuesta al pedido de Lectura

En la siguiente figura se observa un ejemplo de un típico mensaje de lectura que soporta el protocolo y como sería la respuesta al mismo.

Read mains settings command :

<SOH> x "RDM" x x x <EOT>, chk = xxx, pt. Slave = 1

Answer from UP1 :

	<SOH>		0	
	"MAINS_SETTINGS"		1	14
0	x x x	Mains over voltage	15	17
1	x x x	Mains under voltage	18	20
2	x x x	Mains over frequency	21	23
3	x x x	Mains under frequency	24	26
4	x x	Mains breaker	27	28
5	x x	Mains failure	29	30
6	x x	Mains restore	31	32
7	x	Phases	33	
8	x	Input mains sim.	34	
	x x x	checksum	35	37
	<EOT>		38	

Figura: 13 Ejemplo de mensaje de lectura

2.1.1.2. Formato de mensaje de escritura

El caso de los mensajes de escritura, es muy similar a los de lectura. El formato de la trama, como se observa en la figura 14, está encabezado por el carácter de inicio SOH, seguido del id del esclavo (con valor 1 ó 0). Siguiéndole así el comando de escritura, el cual le específica al dispositivo exactamente que va a modificar, por lo que a continuación irá la lista de los nuevos parámetros, el tamaño de la misma depende del comando especificado, pues no todos los mensajes brindan la misma cantidad de información. Los dos campos restantes serían la suma de chequeo y el carácter de fin de trama EOT.

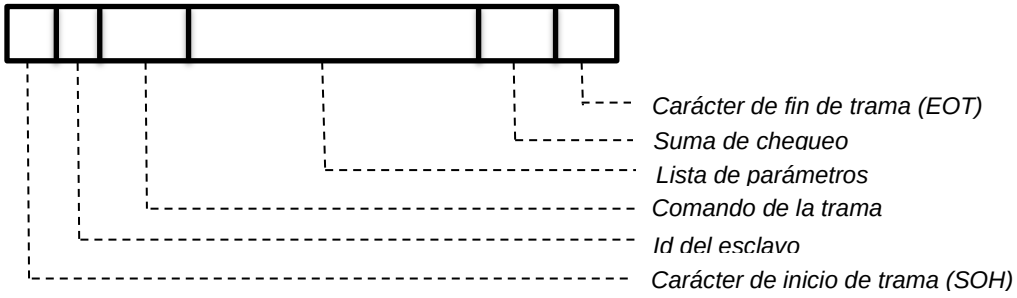

Figura: 14 Formato de la trama de Escritura

Una vez recibido el mensaje de escritura, el dispositivo responde con una trama sencilla, como se observa en la figura 15. El campo de respuesta varía en dependencia de la calidad de la acción realizada entre las que se encuentran “OK”, “CHKERR” (error de suma de chequeo), “ERROR” (error de escritura), “RANGE_ERROR” (error de fuera de rango).

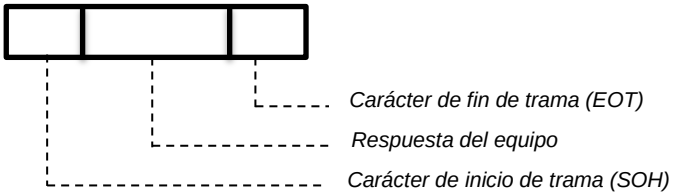

Figura: 15 Formato de la trama de respuesta al pedido de escritura

En la figura 16 se observa un típico ejemplo de una trama de escritura con su respectiva respuesta

Write mains settings

5. Write mains settings command :

<SOH> x "WRITE_MAINS_SETTINGS" (<lista of parameters>) x x x <EOT>

Answers from UP1 :

<SOH > "OK" <EOT>
<SOH> "CHKERR" <EOT> checksum error
<SOH > "ERROR" <EOT> write error
<SOH > "RANGE_ERROR" <EOT>
Check the user manual for correct range
one of the parameter is out of range.

Write mains settings command and list of parameters :

	<SOH>		0	
	x	Slave no.	1	
	"WRITE_MAINS_SETTINGS"		2	21
0	x x x	Mains over voltage	22	24
1	x x x	Mains under voltage	25	27
2	x x x	Mains over frequency	28	30
3	x x x	Mains under frequency	31	33
4	x x	Mains breaker	34	35
5	x x	Mains failure	36	37
6	x x	Mains restore	38	39
7	x	Phases	40	
8	x	Input mains sim.	41	
	x x x	checksum	42	44
	<EOT>		45	

Figura: 16 Ejemplo de Mensaje de Escritura

2.2. Tipos de comandos

Cada trama tiene un comando identificador. De esta manera el equipo conoce la respuesta que, en dependencia del pedido, debe responder. Las tramas pueden ser de lectura y escritura. Aunque es válido aclarar que no todas las tramas que implican lectura tienen por consecuencia tener su contraparte de escritura. En la siguiente tabla se observa todos los comandos aceptados por el protocolo.

Tabla 2: Comandos de Lectura

Comandos de Lectura	Descripción
Rdf	Lectura del Firmware del Dispositivo
RDD	Lectura de Datos
RDO	Lectura de Valores de Salida
RDA	Lectura de Alarmas
RDM	Lectura de la Configuración de Red
RDG	Lectura de la Configuración de Grupos Electrógenos
RDI	Lectura de Ajustes de Entrada
RDE	Lectura de la Configuración del Motor
RDv	Lectura de Servicios
RDF	Lectura de Nivel de Combustible
RDo	Lectura de Configuración de Salidas
RDS	Lectura de Sensores
READ_SN	Lectura del Número serial
READ_CLOCK	Lectura de Fecha y Hora
READ_EVENTS	Lectura de Eventos
Rdp	Lectura de la Agenda
Rdca	Lectura de las Alarmas Personalizadas(1-8)

Tabla 3: Comandos de Escritura

Comandos de Escritura	Descripción
WRITE_MAINS_SETTINGS	Escritura en la Configuración de la Red
WRITE_GENSET_SETTINGS	Escritura en la Configuración de Grupos Electrógenos
WRITE_INPUTS_SETTINGS	Escritura en los Ajustes de Entrada
WRITE_ENGINE_SETTINGS	Escritura en la Configuración del Motor
WRITE_SERVICE	Escritura en los Servicios
WRITE_FUEL	Escritura en el Nivel de Combustible
WRITE_OUTPUTS_SETTINGS	Escritura en la Configuración de Salidas
WRITE_SENSORS	Escritura en los Sensores
SET_HOURS	Escritura en el Contador de Horas
SET_SLAVE	Escritura de un nuevo Numero de esclavo
CLEAR_ALARMS	Borrar las Alarmas
ENGINE_START	Orden de Arranque del motor de prueba
ENGINE_STOP	Orden de Parada del motor de prueba
WRITE_SN	Escritura del número serial
WRITE_CLOCK	Escritura de Fecha y Hora
CLEAR_EVENTS	Borrado del Historial de Eventos
WRITE_PHONEBOOK	Escritura en la Agenda
WRITE_ALARMS1_8	Escritura de las Alarmas Personalizadas (1-8)

2.3. Cálculo de Checksum

La suma de verificación o checksum no es más que una forma de control de redundancia. Se emplea con el objetivo de proteger la integridad de los datos que se envían, verificando así que no hayan sido corruptos o haya habido algún error en el envío de la trama al receptor.

Para el caso del protocolo del UP1-LCD, es calculado del siguiente modo: se suman todos los valores ACII de los caracteres individuales, se desprecia el carácter de inicio de trama (SOH). El resultado se divide entre 256 y el resto de la división es el checksum. Este valor es expresado en 3 caracteres (3 bytes) al final del campo de datos. Los caracteres útiles por el cálculo del checksum son indicados con un asterisco, prácticamente todo aquellos siguientes a SOH, caracteres de los cordones comprendidos, hasta el byte antes del campo checksum. Por ejemplo, para el cálculo del checksum de la trama de datos sería de la siguiente manera:

<SOH> x “RDD” xxx <EOT>

Teniendo en cuenta que x = 1(esclavo), quedaría la sumatoria de los ACII de la siguiente manera:

x = 49; R = 82; D = 68

49 + 82 + 68 + 68 = 267 % 256 = 11

xxx = 011 (checksum)

2.4. Especificación de Requerimientos

Los manejadores que se desarrollan en el Proyecto Recolección del CEDIN deben cumplir con un conjunto de requisitos funcionales y no funcionales. Estos se definen en el documento “Especificación de Requerimientos de Software” versión 1.0 (14), el cual tiene como objetivo, a partir de los requerimientos generales de los manejadores, definir una línea base para los mismos que sirva como punto de apoyo para futuros desarrollos. Además se tomó en consideración los requeridos por el cliente. De forma general el manejador UP1-LCD debe cumplir con los siguientes:

Requisitos funcionales:

- Función de Lectura
- Función de Escritura
- Estampado de tiempo
- Parametrización del protocolo, de los dispositivos y redes.
- Validación de las direcciones admisibles.

En el caso de los requisitos no funcionales de este manejador, se tendrán en cuenta todos los definidos en el documento antes mencionado.

2.5. Modelo de Dominio

A continuación se presenta un modelo de dominio cuyo objetivo es lograr una mínima comprensión del contexto en el que se empleará el manejador a desarrollar.

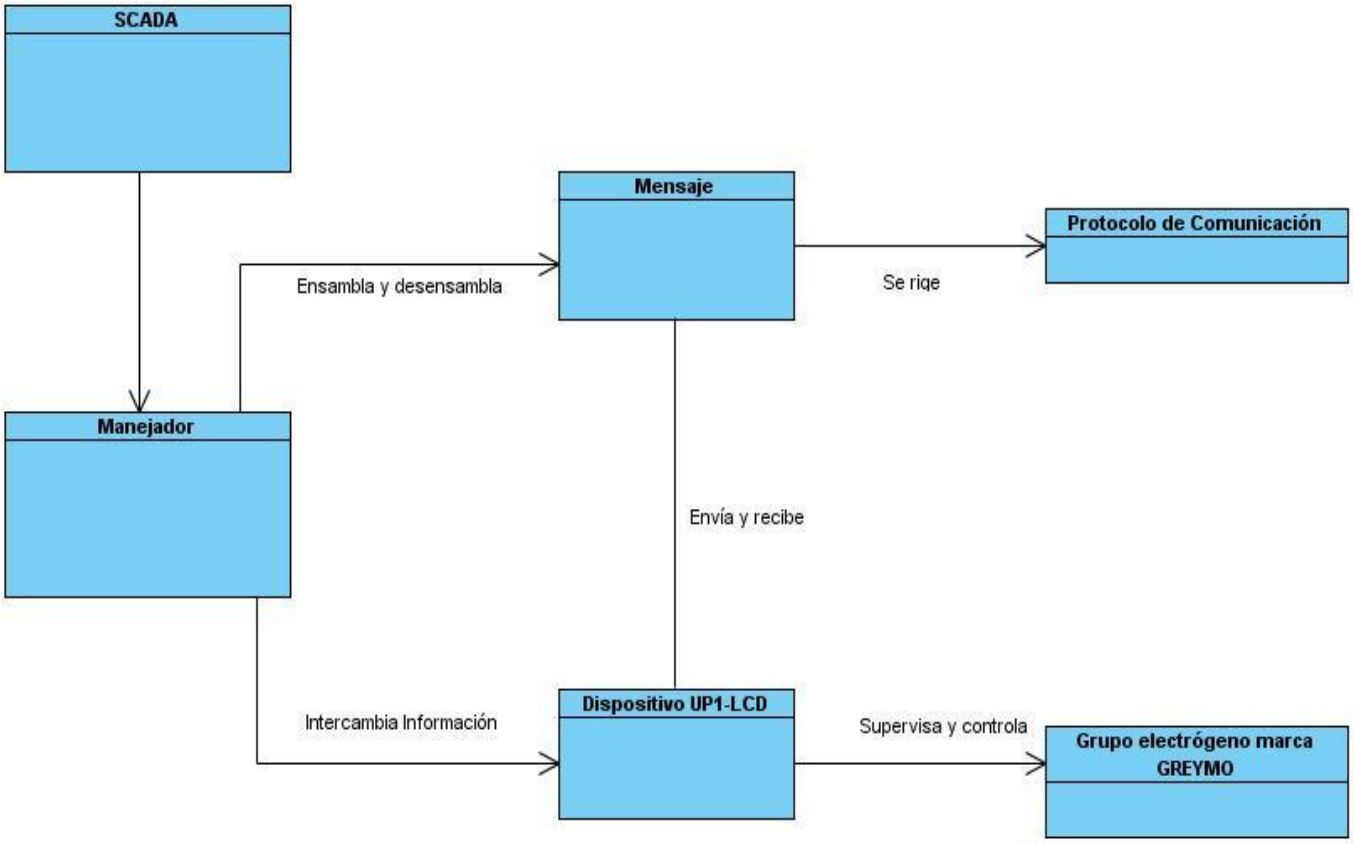

Figura: 17 Modelo de Dominio

Glosario de términos del dominio:

- **Dispositivo UP1-LCD:** Equipo electrónico, en este caso un PLC, que permite la adquisición de los datos generados por los Grupos Electrógenos
- **Grupo Electrógeno marca GREYMO:** Equipo electrónico encargado de generar energía eléctrica en casos extraordinarios, dígase apagón, ciclones, roturas, entre otros.

- **Manejador:** Módulo del SCADA encargado del intercambio de información con dispositivos que se comunican por medio de un protocolo de comunicación específico.
- **Mensaje:** Representa las tramas que se envían para el intercambio de información entre el SCADA y el dispositivo.
- **Protocolo de Comunicación:** Conjunto de reglas o leyes a tener en cuenta para la comunicación con los dispositivos UP1-LCD.
- **SCADA:** Conjunto de reglas o leyes a tener en cuenta para la comunicación con los dispositivos

2.6. Arquitectura Multicapa

Hasta el momento debe quedar claro que la función principal de los manejadores es conectar el SCADA con los diferentes dispositivos de campo. Esta interacción se realiza mediante el envío y recepción de mensajes a través de un medio físico determinado, en este caso sería el puerto serie.

Generalmente quienes definen la semántica y la estructura de los mensajes que se le envían a los dispositivos son los protocolos de comunicación, delegando la transmisión de los mismos a capas inferiores de transporte, como TCP/IP o Serie. El SCADA debe soportar una interfaz suficientemente general para que a ella puedan conectarse varios manejadores con características diferentes. Al mismo tiempo debe ser eficiente para aprovechar al máximo las posibilidades de cada protocolo y que el diseño de los manejadores sea simple. Basados en esta concepción general los manejadores deben desarrollarse como un sistema multicapa. (10)

En la figura siguiente se puede observar las tres capas que conforman la arquitectura de los manejadores que se desarrollan en la línea Recolección, y que por consiguiente es la que se seguirá para el desarrollo del UP1-LCD.

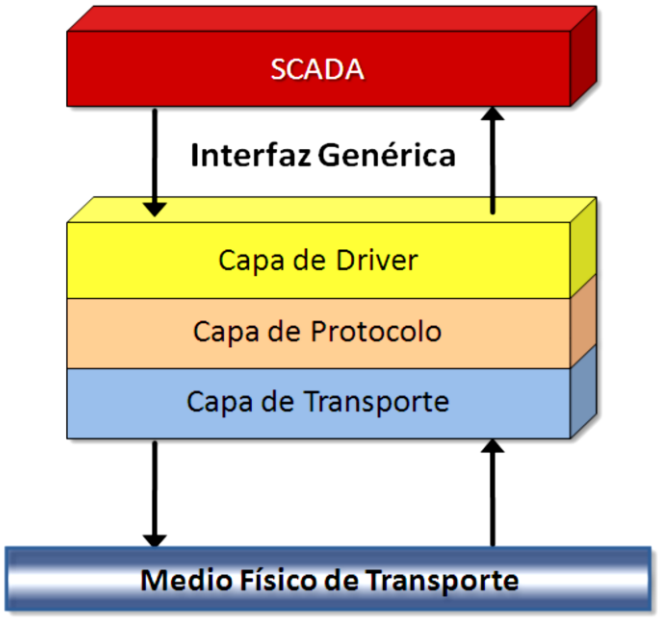

Figura: 18 Arquitectura Multicapa de los Manejadores

La capa de transporte es la encargada de manejar la conexión (si existiera) y la transmisión de un flujo de datos a través del medio físico. La capa de protocolo proporciona una serie de funciones (API) que encapsulan la construcción e interpretación de los mensajes necesarios para la comunicación. La capa de protocolo utiliza la capa de transporte de modo que no debe entrar en las especificidades de cómo se envían o reciben los mensajes. Por último la capa de Driver debe traducir en términos de llamadas a la capa de protocolo la interfaz genérica del SCADA. (10)

Esta arquitectura presenta las siguientes ventajas:

- Las dos primeras capas de los manejadores son reutilizables y tienen valor por sí mismos. En particular disponer de la capa de protocolo permite modificar de manera más simple la interfaz genérica en caso necesario.

- Cada capa tiene una funcionalidad lógica propia. La capa de transporte envía y recibe mensajes, la capa del protocolo construye e interpreta los mensajes y la capa de driver traduce la interfaz genérica en términos del protocolo. A partir de esta separación lógica es posible arrancar en paralelo los trabajos en las tres capas siempre y cuando estén claras las interfaces correspondientes. La puesta a punto de las dos primeras capas no requiere de la funcionalidad del resto de los módulos del SCADA**.** (10)

2.7. Biblioteca DriverCore

El desarrollo de un manejador es tan complejo como sea su protocolo de comunicación o las características de hardware del dispositivo con el que pretendamos comunicarnos. Por tal motivo en nuestra línea de trabajo podemos optar dos formas a la hora de implementar uno. La primera consiste en hacerlo todo desde cero e implementar directamente la Interfaz Genérica, esto trae como consecuencia que el trabajo sea más largo y engorroso. La segunda vía consiste en hacer uso de las clases que brinda la biblioteca DriverCore, para el desarrollo del manejador UP1-LCD se opta por la segunda vía.

La biblioteca dinámica DriverCore es una implementación de la Interfaz Genérica (IG) en C++. En esta se definen un conjunto de clases e interfaces, que encapsulan los conceptos fundamentales en el proceso de recolección que existen en la IG y brinda un conjunto de funcionalidades que deben facilitar el desarrollo de los manejadores en la mayoría de los casos. La clase DriversLibrary define las características comunes de una biblioteca que puede contener el código de varios manejadores. Responde por la implementación de las funciones de la interfaz genérica a partir de las demás clases y por mantener la correlación entre objetos creados y los identificadores numéricos (handles). Responde además por mantener una lista interna con la metainformación correspondiente a cada manejador (15)

Los objetivos principales que persigue esta biblioteca son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones de diseño. Además haciendo uso de la misma obtenemos ventajas como: la introspección de manejadores y dispositivos, el manejo de las direcciones, clasificación de las variables en bloques de lectura o

escritura y la entrada/salida de modo asíncrono. Para lograr una mejor comprensión, se explican a continuación las ventajas antes mencionadas.

La introspección se refiere a la capacidad de los objetos de mostrar las propiedades que expone, de obtener el valor de esas propiedades a partir del nombre de la misma y modificar ese valor igualmente a partir del nombre. A los efectos de la IG se añaden a esa definición la capacidad de mostrar los parámetros de diagnóstico y obtener los valores de esos parámetros a partir del nombre del mismo. Estas funcionalidades responden a las funciones de la IG: listDrivers, getParameterValue, setParameteraValue y getDiagnosticParameterValue. Este modelo permite un máximo de 24 manejadores cargados de forma simultánea en el DriverCore, no más de 32 propiedades de configuración por objeto, ya sea driver o dispositivo y no más de 32 parámetros de diagnóstico por objeto (driver o dispositivo). El manejo de las direcciones permite definir dos tipos de direcciones fundamentales, las direcciones simples y las compuestas. El manejo de estas direcciones se realiza en las clases, Address y SimpleAddress. Las direcciones son agrupadas en el DriverCore por bloques, estos están formados por un conjunto de variables que tienen un mismo período de encuesta y que además pueden ser recuperadas o escritas en el dispositivo de campo en una operación atómica del protocolo. La entrada y salida asíncrona permite realizar funciones de lectura y escritura en los dispositivos de campo de forma asíncrona, retornando de forma inmediata el hilo principal de ejecución y encolando las tareas de lectura y escritura en hilos secundarios de ejecución. (15)

2.8. Decisiones de Diseño para el manejador

Para la implementación del manejador se tomaron decisiones con el propósito de satisfacer las necesidades del cliente, además de, proporcionar una buena comunicación entre el SCADA GECTEL y el manejador; obteniendo así un producto de calidad, con posibilidades de adicionarle en un futuro nuevas funcionalidades en caso de necesitarse.

2.7.1. Bloques de Direcciones

Para la creación de un bloque de direcciones se tomaron en cuenta los datos pertenecientes a una misma trama, debido a que el protocolo soporta la lectura y escritura de mensajes. La estructura de las direcciones se identifica como índice: subíndice. Representando el primero a una trama en

específico, y el segundo a una variable en especial de esa trama, por lo que el total de subíndices varían para cada caso en particular.

El número de índices no sobrepasan los 23, a pesar de que el protocolo cuenta con 35 comandos, esto se debe a que existen tramas de lectura/escritura. En estos casos no se crea un bloque específico, pues por restricciones del protocolo no es permisible modificar una variable en especial dado un comando. Para la escritura se enviar la trama completa que es específica, con todos sus respectivos valores. Por tal motivo se toma la decisión de ante cada petición de escritura se realiza una lectura previa del comando en cuestión. De los datos leídos solo se modifican los requeridos, y se procede a enviar el mensaje de escritura correctamente conformado.

A continuación se muestran ejemplos de direcciones:

- El conjunto de direcciones desde 2:1 hasta 2:48 (correcto, porque el bloque 2 tiene hasta 48 variables)
- El conjunto de direcciones desde 5:1 hasta 5:60 (incorrecto porque el valor de subíndice más alto que se pueda esperar es 49)

2.7.2. Estampado de tiempo

En el caso de la estampa de tiempo de cada mensaje, no viene implícita en las tramas de respuesta del UP1-LCD cuando se encuesta. Por lo que se decide obtener este dato por medio del método ***boostEpochTime***, que devuelve la cantidad de milisegundos transcurridos desde el comienzo de la época Unix (1/01/1970) hasta el instante de tiempo (hora Universal) en que se invoca al método.

2.7.3. Implementación de las tramas

Se ha mencionado anteriormente que el protocolo del UP1LCD soporta las tramas de tipo lectura/escritura. Para su implementación, se decide crear una clase genérica, llamada “UP1LCDCommand”, que englobe los datos comunes de ambos tipos de tramas y a su vez heredarán de esta, clases hijas, que contendrán las particularidades de cada trama dado un comando.

Así cuando en el mensaje se lleve a cabo el proceso de ensamblar o desensamblar una trama, se auxiliará, de la UP1LCDCommand y sus hijas para conocer los detalles de cada trama en particular, y poder concluir el proceso en cuestión.

2.9. Conclusiones parciales del capítulo

Este capítulo permite abordar temas específicos sobre las características del protocolo, con el objetivo de establecer una mejor comprensión del mismo. Esto representa gran ayuda para la implementación del mismo en el manejador. Se definieron además los requerimientos del manejador y las decisiones de diseño que se llevarán a cabo en la implementación. Todo esto con el objetivo de lograr un producto de calidad que cumpla con las expectativas del cliente y además logre su cometido con el sistema GECTEL.

Capítulo 3: Diseño e Implementación

3.1. Arquitectura de Software

La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones. (16)

A raíz de la diversidad de criterios con respecto a este tema, la organización IEEE establece en uno de sus documentos como definición oficial la siguiente: “La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.”

3.1.1. Arquitectura en Capas del Manejador

Para el diseño del manejador se utilizó una arquitectura en capas, la misma fue explicada desde el punto de vista funcional en el capítulo anterior. Este diseño permite probar los componentes por separado, las capas creadas son: Capa de Driver, Capa de Protocolo y Capa de Transporte. A continuación se describe cada una de ellas.

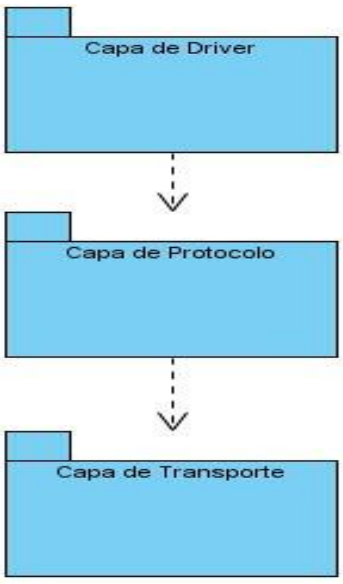

Figura: 19 Arquitectura en Tres Capas del Manejador UP1-LCD

Capa de Driver:

En esta capa se definen e implementan las clases que heredarán de las interfaces que provee el DriverCore, para obtener un comportamiento acorde a las características del protocolo del manejador en cuestión. Para que este tenga la capacidad de introspección, es necesario crear las siguientes cuatro nuevas clases: DriverMetaClass, DeviceMetaClass, Driver y Device, descendentes de la biblioteca antes mencionada. Las dos primeras clases tienen como ancestro común a la clase BaseMetaClass y sirven de base a las metaclases de los descendientes de la clase Driver y Device respectivamente. De la misma manera las dos últimas permiten modelar las características comunes de los manejadores y los dispositivos respectivamente.

Para el manejo de las direcciones, están en el DriverCore las clases: Address, SimpleAddress e IProtocolAddress. Esta última encapsula el concepto de dirección de protocolo o de punto de medición en el campo. Las direcciones de las variables pueden contener una o varias direcciones de protocolo. En esta clase es donde se realiza la validación de las direcciones.

La clase Block implementa el concepto de conjunto "consecutivo" de direcciones que tienen el mismo período de medición y que puede ser accedido a través de un solo mensaje del protocolo. Los bloques se construyen dinámicamente por los manejadores cada vez que se asocian variables al dispositivo y cada vez que se efectúan operaciones de escritura. Los bloques pueden ser de variables de diagnóstico del dispositivo, de variables de diagnóstico del manejador o de variables de protocolo. Los descendientes de Block deben reimplementar los métodos para la lectura de enteros, flotantes, texto, vector de enteros y vector de flotantes, sucediendo lo mismo para la escritura.

Capa de Protocolo:

En esta capa se implementa la lógica de comunicación entre el manejador y dispositivos. Aquí se cumple la especificación del protocolo en cuanto a envío y recepción de mensajes. La clase EndPoint se controla todos los pasos que define el protocolo para lograr una comunicación

satisfactoria, segura y preparada para responder a los fallos que pudieran surgir, haciendo uso de una máquina de estado que facilita el diseño del comportamiento requerido por la especificación del protocolo. De la misma manera se crea una clase Message que especifica las características de las tramas que conforman el protocolo del manejador en cuestión, abstrayendo de esta forma al EndPoint de la necesidad de conocer cómo se crean o cuáles son los elementos que constituyen una trama.

Capa de Transporte:

En la capa de transporte es utilizada la biblioteca dinámica TransportProvider. Esta biblioteca brinda una clase fábrica denominada TransportProvider para la creación de transportes asíncronos TCP, UDP y Serie, la cual hereda de una clase interfaz ITransportProvider. Para cada uno de los transportes que brinda esta biblioteca, existen dos clases: una clase interfaz que hereda de ITransport, adquiriendo sus características y una clase en la que se implementan las funcionalidades de su interfaz, correspondiente con las características específicas del transporte a que pertenece. En el caso del transporte Serie están definidas las clases ISerialTransport y SerialTransport. Es válido aclarar que la interfaz ITransport encapsula las funcionalidades asíncronas de los diferentes tipos de transporte que brinda la biblioteca. Las funcionalidades asíncronas tienen la particularidad que retornan inmediatamente al ser invocadas, reciben como parámetro un handler, que no es más que una instancia de la clase que reimplementa la súper clase ITransportHandler.

3.1.2. Patrones de Diseño empleados en el Manejador

El desarrollo del manejador UP1-LCD está guiado por la línea base definida para el desarrollo de los manejadores en el Proyecto Recolección de nuestro centro. La misma ha sido probada en el desarrollo de manejadores anteriores e incluye patrones arquitectónicos que ayudaron en su diseño. Por tanto en este epígrafe solo se hará referencia a los específicos del UP1-LCD y que están excluidos, como es lógico, de esta línea base, por ser características particulares del mismo.

En el desarrollo del manejador UP1-LCD, se utiliza el patrón denominado Constructor (*Builder* en inglés). Para, como su nombre lo indica, construir las diferentes tramas que soporta el protocolo. En la capa de Protocolo se crea una clase “UP1LCDCommand” que engloba las características

comunes de las tramas de escritura y lectura. Esta clase contiene dos funcionalidades genéricas capaces de interpretar el cuerpo de las tramas independientemente, refiérase a *readFrameBody* (para las tramas de lectura) y *writeFrameBody* (para las tramas de escrituras). Los mismos son implementados en cada clase hija correspondiente, que herede de la "UP1LCDCommand".

De aquí entonces el empleo del patrón y la vigencia de su principal ventaja "… favorece el encapsulamiento y el control, puesto que cada constructor concreto contiene todo el código necesario para crear y ensamblar todas las partes de un producto y además el constructor crea el producto paso a paso (parte a parte) bajo la supervisión del director, que al final recibe el producto completo, de manos del constructor". (17) Actuando como director la clase "UP1LCDMessage" que manda a ensamblar la trama y en dependencia del comando, es delegado para la clase hija correspondiente y la cual devuelve el cuerpo de la trama completo al Mensaje, para concluir el ensamblaje.

3.2. Diagrama de Clases del Manejador UP1LCD

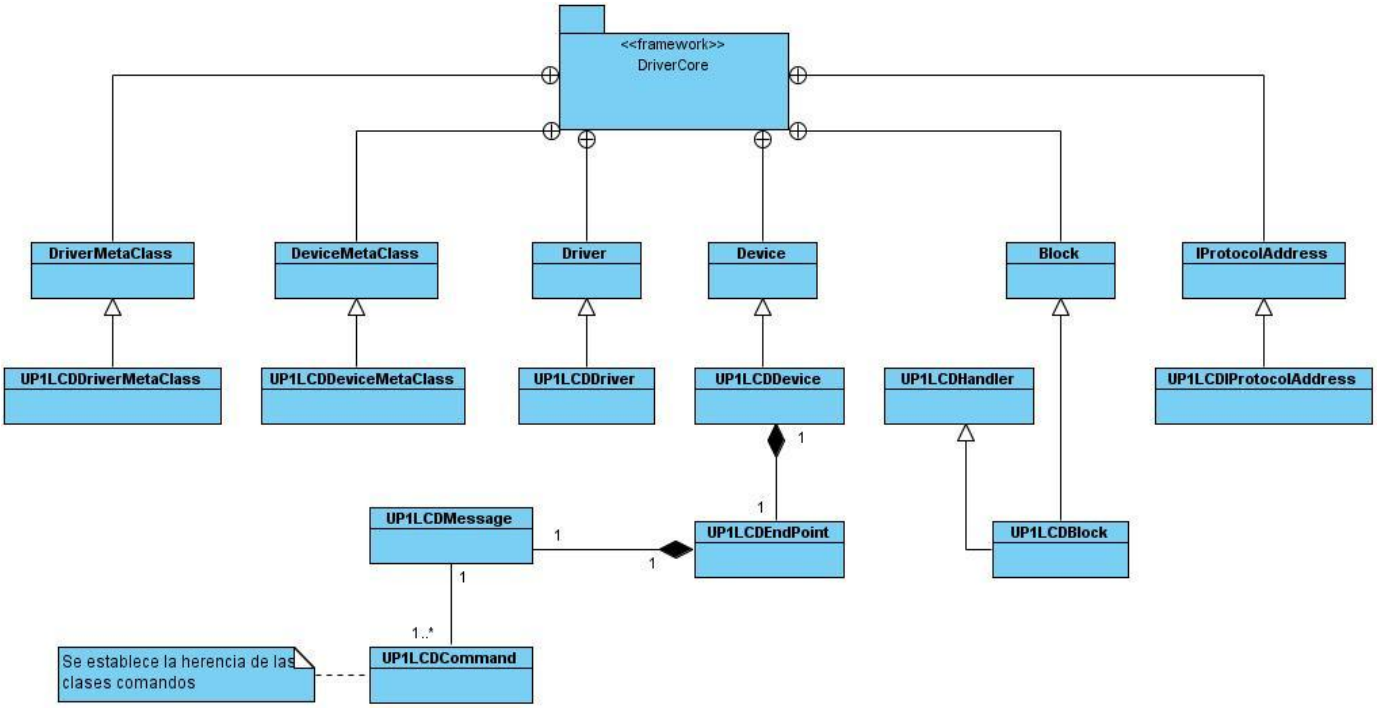

Figura: 20 Diagrama de clases del diseño General del Manejador

3.2.1. Diagrama de Clases de la Capa Protocolo

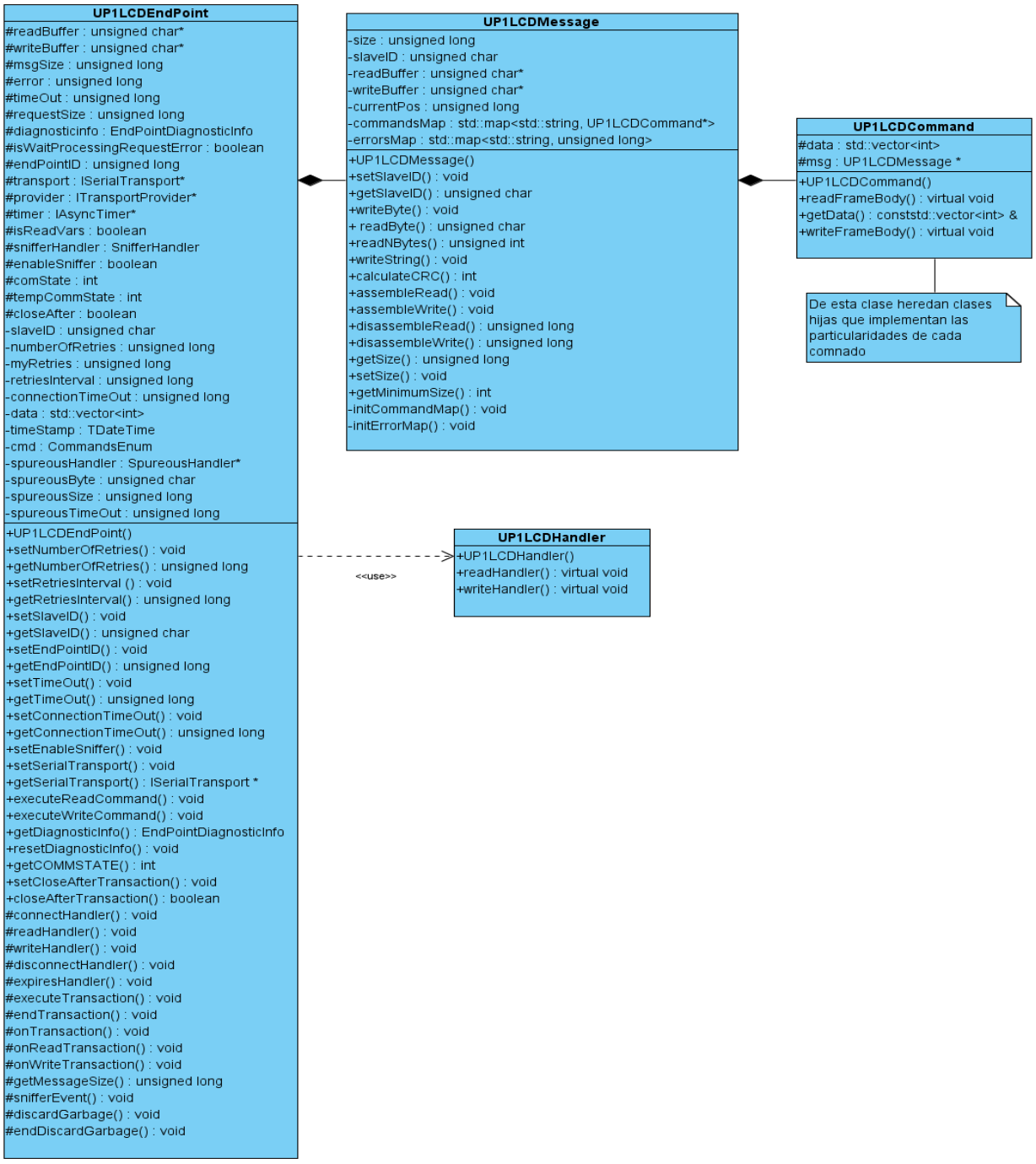

Figura: 21 Capa de Protocolo

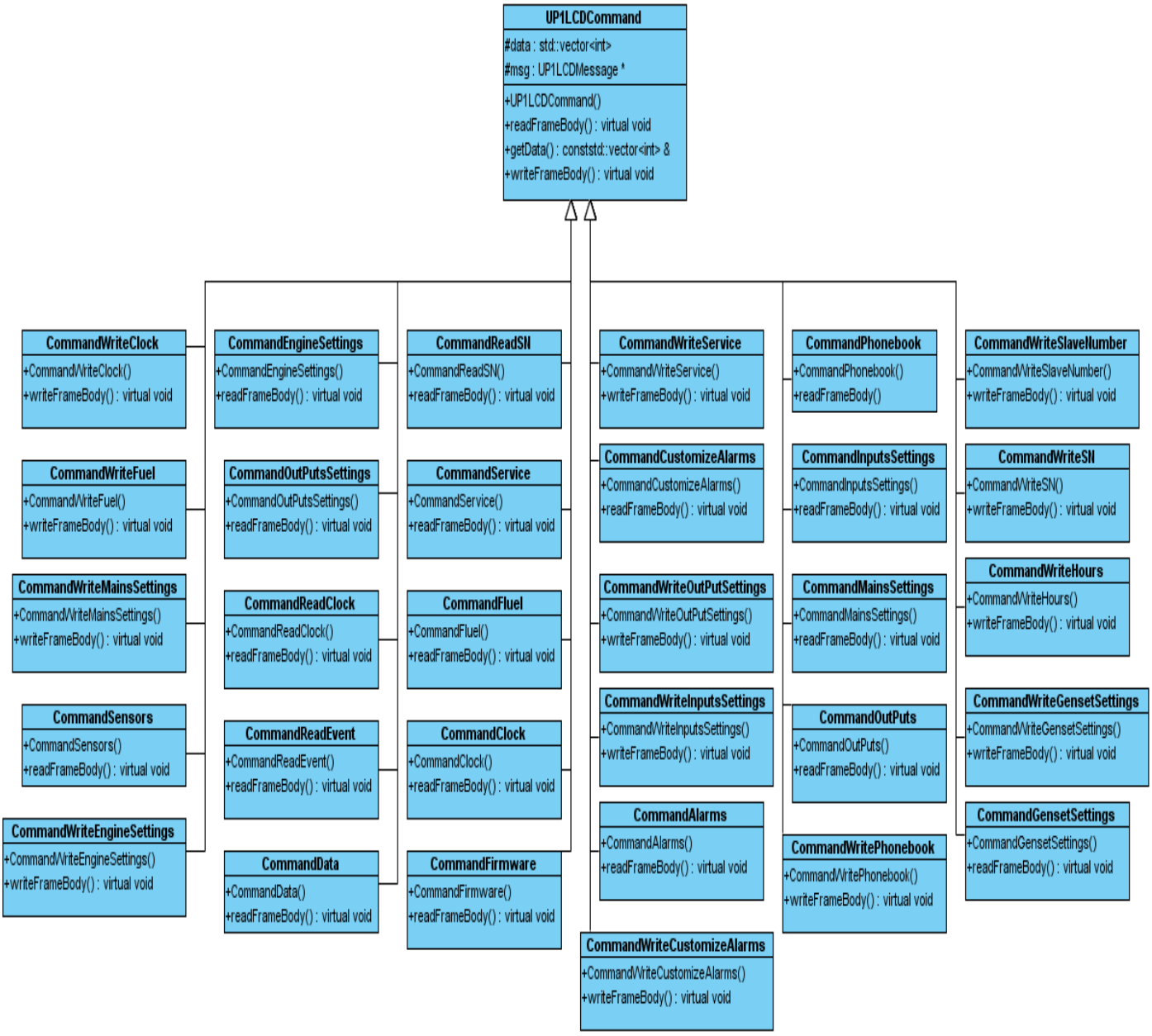

Figura: 22 Herencia de clases Comandos

3.2.2. Diagrama de Clases de la Capa Driver

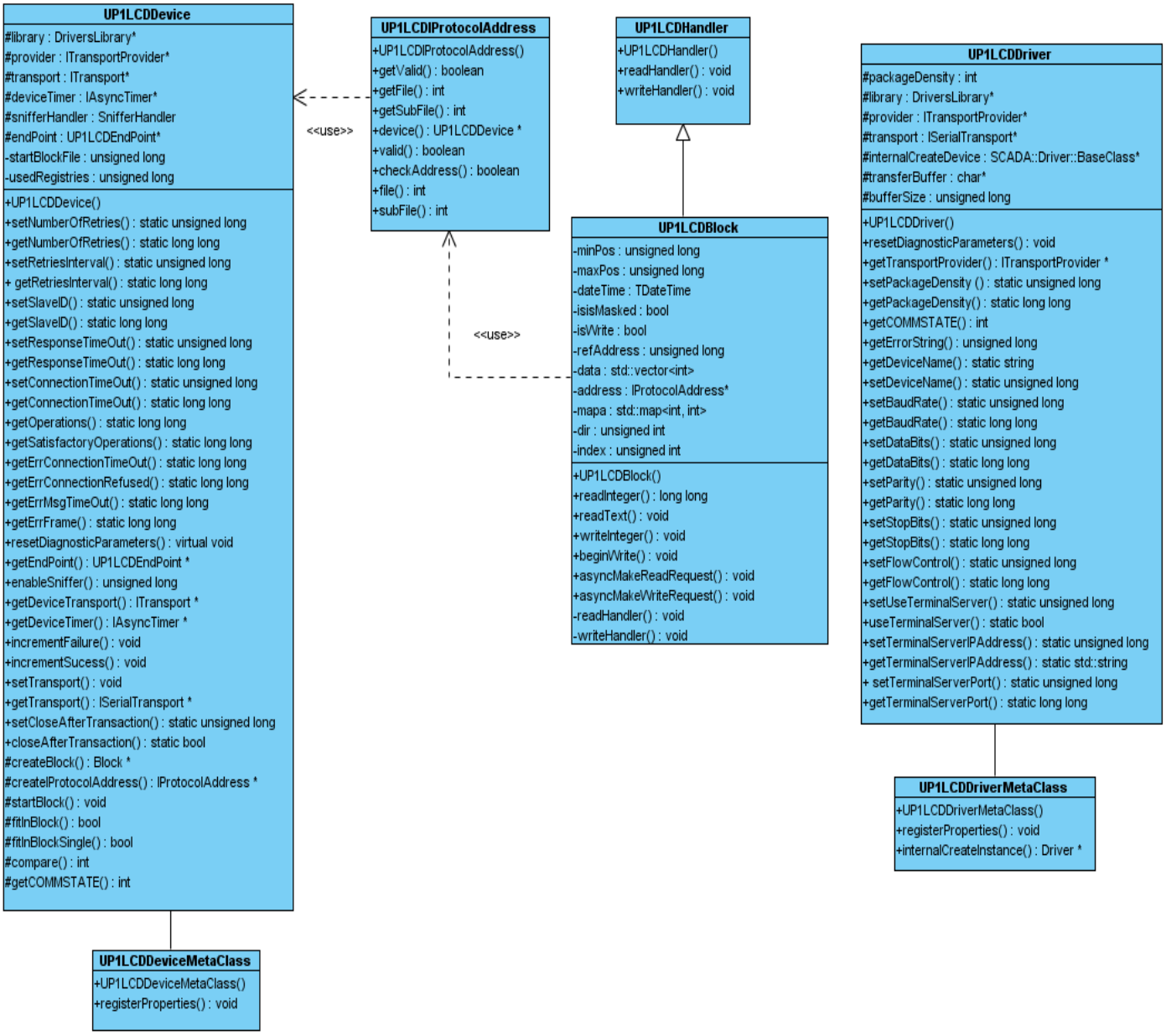

Figura: 23 Herencia de clases Comandos

3.3. Descripción de las principales clases implementadas

A continuación se describen las principales clases que conforman el Manejador UP1LCd, pues no es objetivo de este trabajo plasmar cada clase que lo componen.

3.3.1. Clase UP1LCDEndPoint

Esta clase encapsula la lógica de comunicación con el dispositivo, permitiendo obtener y configurar algunas propiedades de los mismos. Hereda de ITransportHandler e ITimerHandler, ambas clase viabilizadas por el DriverCore. Al establecer comunicación con el UP1LCD se especifica el tipo de trama (lectura o escritura), para así especificarle al Mensaje cual ensamblar.

Tabla 4: Descripción de la clase UP1LCDEndPoint

Nombre:UP1LCDEndPoint	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
slaveID	unsigned char
numberOfRetries	unsigned long
myRetries	unsigned long
retriesInterval	unsigned long
connectionTimeOut	unsigned long
myHandler	UP1LCDHandler*
Data	std::vector<int>
timeStamp	TDateTime
Cmd	CommandsEnum
readBuffer	unsigned char*

writeBuffer	unsigned char*
msgSize	unsigned long
message	UP1LCDMessage*
error	unsigned long
timeOut	unsigned long
requestSize	unsigned long
diagnosticinfo	EndPointDiagnosticInfo
isWaitProcessingRequestError	bool
endPointID	unsigned long
transport	ISerialTransport*
provider	ITransportProvider*
timer	IAsyncTimer*
isReadVars	bool
snifferHandler	SnifferHandler
enableSniffer	bool
comState	int
closeAfter	bool
Para cada responsabilidad	
Nombre:	UP1LCDEndPoint (SnifferHandler snifferHandler)
Descripción:	Constructor de la clase UP1LCDEndPoint
Nombre:	setNumberOfRetries (unsigned long newRetriesNumber)
Descripción:	Cambia la cantidad de reintentos en caso de error en las operaciones de lectura y escritura
Nombre:	getNumberOfRetries ()

Descripción:	Devuelve la cantidad de reintentos en caso de error.
Nombre:	setRetriesInterval (unsigned long newRetriesInterval)
Descripción:	Cambia el valor del intervalo entre reintentos en caso de error en las operaciones de lectura y escritura.
Nombre:	getRetriesInterval ()
Descripción:	Devuelve intervalo entre reintentos.
Nombre:	setSlaveID (unsigned char newSlaveID
Descripción:	Cambia el identificador del esclavo al cual se hará referencia.
Nombre:	getSlaveID ()
Descripción:	Devuelve el identificador del esclavo al cual se hace referencia.
Nombre:	setEndPointID (unsigned long newEndPointID)
Descripción:	Cambia el identificador del EndPoint.
Nombre:	getEndPointID ()
Descripción:	Devuelve el identificador del EndPoint.
Nombre:	setTimeOut (unsigned long timeOut)
Descripción:	Función para establecer el tiempo de espera para la recepción del mensaje (en milisegundos).
Nombre:	getTimeOut ()
Descripción:	Función que devuelve el tiempo de espera para recibir el mensaje
Nombre:	setConnectionTimeOut (unsigned long timeOut)
Descripción:	Función para establecer el tiempo de espera para establecer la conexión (en milisegundos)
Nombre:	getConnectionTimeOut ()
Descripción:	Función que devuelve el tiempo de espera para establecer la conexión
Nombre:	setEnableSniffer (bool enable)
Descripción:	Establece el uso o no del sniffer. En dependencia del uso del sniffer o no se ejecutan los callback de la función sniffer.

Nombre:	setSerialTransport (ISerialTransport* ntransport)
Descripción:	Establece el transporte al endpoint.
Nombre:	getSerialTransport ()
Descripción:	Devuelve el transporte serial usado por el endpoint.
Nombre:	executeReadCommand (CommandsEnum command,UP1LCDHandler* handler)
Descripción:	Manda a ejecutar el ensamblado de los comandos de lectura
Nombre:	executeWriteCommand(CommandsEnum command, const std::vector<int> &parameters, UP1LCDHandler* handler)
Descripción:	Manda a ejecutar el ensamblaje de los comandos de escritura
Nombre:	getDiagnosticInfo ()
Descripción:	Funcionalidad que retorna la información de un determinado parámetro de diagnostico
Nombre:	resetDiagnosticInfo ()
Descripción:	Funcionalidad que reinicia la información de todos los parámetros de diagnostico
Nombre:	getCOMMSTATE (TDateTime* dateTime)
Descripción:	Funcionalidad que retorna la variable es estado de la comunicación
Nombre:	setCloseAfterTransaction (bool value)
Descripción:	Establece el cerrar o no el dispositivo serial utilizado al finalizar cada transacción
Nombre:	closeAfterTransaction ()
Descripción:	Devuelve si se está cerrando o no el dispositivo serial al finalizar cada transacción
Nombre:	connectHandler (unsigned long error)
Descripción:	Handler de conexión del transporte reimplementado.
Nombre:	readHandler (unsigned char* data, unsigned long bytesTransfered, unsigned long error)
Descripción:	Handler de lectura del transporte reimplementado.

Nombre:	void writeHandler (unsigned long bytesTransfered, unsigned long error)
Descripción:	Handler de escritura del transporte reimplementado.
Nombre:	disconnectHandler (unsigned long error)
Descripción:	Handler de desconexión del transporte reimplementado.
Nombre:	expiresHandler (unsigned long error)
Descripción:	Handler del timer asíncrono reimplementado.
Nombre:	executeTransaction ()
Descripción:	Ejecuta las transacciones del endpoint.
Nombre:	endTransaction ()
Descripción:	Termina la transacción
Nombre:	onTransaction ()
Descripción:	Se ejecuta después de haber leído un mensaje de respuesta del dispositivo correspondiente a una solicitud.
Nombre:	onReadTransaction ()
Descripción:	Se ejecuta después de haber leído un mensaje de respuesta del dispositivo correspondiente a una solicitud
Nombre:	onWriteTransaction ()
Descripción:	Se ejecuta después de haber leído un mensaje de respuesta del dispositivo correspondiente a una solicitud
Nombre:	getMessageSize ()
Descripción:	Devuelve la cantidad de bytes de la respuesta del dispositivo a partir de la interpretación de la cabecera de la trama.

3.3.2. Clase UP1LCDMessage

Esta clase es la encargada de ensamblar y desensamblar las tramas que conforman el protocolo del UP1LCD, así como de comprobar la suma de chequeo para cada trama. Con ella existe una relación de composición a partir de la clase UP1LCDCommand, porque esta última es la encargada de englobar las características comunes de las tramas de escritura y lectura. De ella heredan otras clases que personalizan el cuerpo de la trama para cada comando del protocolo.

Tabla 5: Descripción de la clase UP1LCDMessage

Nombre: UP1LCDMessage	
Tipo de clase: Entidad	
Atributo:	Tipo:
size	unsigned long
slaveID	unsigned char
readBuffer	unsigned char*
writeBuffer	unsigned char*
currentPos	unsigned long
commandsMap	std::map <std::string,UP1LCDCommand*>
errorsMap	std::map <std::string, unsigned long>
Para cada responsabilidad:	
Nombre:	UP1LCDMessage (const unsigned char* writeBuffer, const unsigned char* readBuffer, const unsigned char slaveID)
Descripción:	Constructor de la clase
Nombre:	initCommandMap ()
Descripción:	Funcionalidad que inicializa el mapa contenedor de todos los comandos permitidos por el protocolo
Nombre:	initErrorMap ()
Descripción:	Funcionalidad que inicializa el mapa contenedor de los posibles errores provocado por la respuesta del dispositivo ante tramas de escritura o error
Nombre:	setSlaveID (const unsigned char newSlaveID)

Descripción:	Establece el identificador del dispositivo con el que nos conectaremos
Nombre:	getSlaveID ()
Descripción:	Devuelve el identificador del dispositivo que se está usando.
Nombre:	writeByte (const unsigned char value)
Descripción:	Escribe un byte en el buffer de escritura.
Nombre:	readByte ()
Descripción:	Devuelve un byte desde el buffer de lectura.
Nombre:	readNBytes (int n)
Descripción:	Devuelve el valor que hay en los próximos N Bytes
Nombre:	writeString (std::string str)
Descripción:	Escribe pedazos en la trama
Nombre:	int calculateCRC(const unsigned char *str, const int endPos)
Descripción:	Calcula la suma de chequeo
Nombre:	assembleRead (CommandsEnum command)
Descripción:	Ensambla una trama de lectura
Nombre:	assembleWrite (CommandsEnum command, const std::vector<int>& data)
Descripción:	Ensambla una trama de escritura
Nombre:	disassembleRead (std::vector<int>& data)
Descripción:	Desempaqueta una trama del buffer de lectura
Nombre:	disassembleWrite ()
Descripción:	Desempaqueta una trama del buffer de escritura

Nombre:	getSize ()
Descripción:	Retorna el tamaño del buffer de lectura
Nombre:	setSize (unsigned long size)
Descripción:	Cambia el tamaño del buffer de lectura
Nombre:	getMinimumSize (CommandsEnum command)
Descripción:	Retorna el tamaño mínimo de la trama en cuestión

3.3.3. Clase UP1LCDDriver

Esta clase implementa el concepto de Manejador UP1LCD. Hereda de la clase Driver que brinda el DriverCore y posee las características y comportamientos del manejador UP1LCD.

Tabla 6: Descripción de la clase UP1LCDDriver

Nombre: UP1LCDDriver	
Tipo de clase: Controladora	
Atributo:	**Tipo:**
packageDensity	int
library	DriversLibrary*
provider	ITransportProvider*
transport	ISerialTransport*
Para cada responsabilidad:	
Nombre:	UP1LCDDriver (ReadHandler readHandler, WriteHandler writeHandler, SnifferHandler snifferHandler)
Descripción:	Mediante el mismo se crean las instancias de los manejadores con los handler de las funciones de lectura, escritura y el apuntador funcional al sniffer

Nombre:	resetDiagnosticParameters ()
Descripción:	Restablece los parámetros de diagnóstico.
Nombre:	getTransportProvider ()
Descripción:	Devuelve el provider que utiliza el driver
Nombre:	setPackageDensity (BaseClass* base, long long newPackageDensity)
Descripción:	Funcionalidad estática para el establecimiento de la propiedad de densidad de paquete
Nombre:	getPackageDensity (BaseClass* base)
Descripción:	Funcionalidad estática que devuelve el valor de la propiedad densidad de paquete del driver.
Nombre:	getErrorString (unsigned long errorCode, const char** ppErrorDescription
Descripción:	El método getErrorString permite obtener información textual comprensible a partir de un código de error del manejador. Si el valor de errorCode no se corresponde con uno de los valores admisibles se devuelve errInvalidErrorCode
Nombre:	getDeviceName (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con el nombre del dispositivo serial
Nombre:	setDeviceName (BaseClass* base)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con el nombre del dispositivo serial
Nombre:	setBaudRate (BaseClass* base, long long newBaudRate)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con la velocidad en baudios del dispositivo serial
Nombre:	getBaudRate (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada

	con la velocidad en baudios del dispositivo serial.
Nombre:	setDataBits (BaseClass* base, long long newDataBits)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con la cantidad de bits de datos por carácter a transmitir por el dispositivo serial.
Nombre:	getDataBits (BaseClass* base)
Descripción:	Función estática utilizada obtener el valor de la propiedad relacionada con la cantidad de bits de datos por carácter a transmitir por el dispositivo serial.
Nombre:	setParity (BaseClass* base, long long newParity
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con la paridad de los datos transmitidos por el dispositivo serial
Nombre:	getParity (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con la paridad de los datos transmitidos por el dispositivo serial.
Nombre:	setStopBits (BaseClass* base, long long newStopBits)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con la cantidad de bits de parada de los caracteres transmitidos por el dispositivo serial
Nombre:	getStopBits (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con la cantidad de bits de parada de los caracteres transmitidos por el dispositivo serial
Nombre:	setFlowControl (BaseClass* base, long long newFlowControl
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con el control de flujo del dispositivo serial.
Nombre:	getFlowControl (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con el control de flujo del dispositivo serial.

Nombre:	setUseTerminalServer (BaseClass* base, bool newUse)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con el uso del servidor de terminales
Nombre:	useTerminalServer (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con el uso del servidor de terminales
Nombre:	setTerminalServerIPAddress (BaseClass* base, string tsIpAddress)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con la dirección del servidor de terminales.
Nombre:	getTerminalServerIPAddress (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con la dirección del servidor de terminales
Nombre:	setTerminalServerPort (BaseClass* base, long long newTSPort)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con el puerto de comunicación del servidor de terminales.
Nombre:	getTerminalServerPort (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con el puerto de comunicación del servidor de terminales.

3.3.4. Clase UP1LCDDevice

La clase UP1LCDDevice implementa el concepto de Dispositivo UP1LCD. Hereda de Device que a la vez hereda de BaseClass, ambas de la biblioteca DriverCore. De estas clases obtiene lo referente a los parámetros de configuración tanto de diagnóstico como otros tipos de parámetros.

Tabla 7: Descripción de la clase UP1LCDDevice

Nombre: UP1LCDDevice	
Tipo de clase: Controladora	
Atributo:	**Tipo:**
library	DriversLibrary*

provider	ITransportProvider*
transport	ITransport*
deviceTimer	IAsyncTimer*
endPoint	UP1LCDEndPoint*
startBlockFile	unsigned long
usedRegistries	unsigned long

Para cada responsabilidad:	
Nombre:	UP1LCDDevice (unsigned long devScadaId, char* transferBuffer, unsigned long bufferSize, ITransportProvider* provider, SnifferHandler nsnifferHandler)
Descripción:	El constructor de la clase recibe como parámetro un identificador único para el SCADA y un buffer de transferencia que será utilizado para el envío de los datos que se lean desde el dispositivo hacia el recolector. Además de un provider mediante el cual se crea las instancias necesarias, del transporte o del timer asíncrono utilizados por el protocolo.
Nombre:	setNumberOfRetries (BaseClass* base, long long newNumberOfRetries)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con la cantidad de reintentos en caso de error del dispositivo.
Nombre:	getNumberOfRetries (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con la cantidad de reintentos en caso de error del dispositivo.
Nombre:	setRetriesInterval (BaseClass* base, long long newRetriesInterval)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con el tiempo de espera entre reintentos en caso de error.
Nombre:	getRetriesInterval (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con el tiempo de espera entre reintentos en caso de error.

Nombre:	setSlaveID (BaseClass* base, long long newSlaveID)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con el identificador del esclavo UP1LCD
Nombre:	getSlaveID (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con el identificador del esclavo UP1LCD
Nombre:	setResponseTimeOut (BaseClass* base, long long newResponseTimeOut)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con el tiempo de espera máximo para las lecturas del transporte
Nombre:	getResponseTimeOut (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con el tiempo de espera máximo para las lecturas del transporte.
Nombre:	setConnectionTimeOut (BaseClass* base, long long newConnectionTimeOut)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con el tiempo de espera máximo para las conexiones del transporte.
Nombre:	getConnectionTimeOut (BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con el tiempo de espera máximo para las conexiones del transporte.
Nombre:	getOperations (BaseClass* base, TDateTime* dateTime)
Descripción:	Función estática utilizada para obtener el valor de la propiedad de diagnóstico relacionada con la cantidad de operaciones realizadas.
Nombre:	getSatisfactoryOperations (BaseClass* base, TDateTime* dateTime)
Descripción:	Función estática utilizada para obtener el valor de la propiedad de diagnóstico relacionada con la cantidad de operaciones satisfactorias realizadas.

Nombre:	getErrConnectionTimeOut (BaseClass* base, TDateTime* dateTime)
Descripción:	Función estática utilizada para obtener el valor de la propiedad de diagnóstico relacionada con la cantidad de errores de time out de conexión.
Nombre:	getErrConnectionRefused (BaseClass* base, TDateTime* dateTime)
Descripción:	Función estática utilizada para obtener el valor de la propiedad de diagnóstico relacionada con la cantidad de errores de conexión rehusada.
Nombre:	getErrMsgTimeOut (BaseClass* base, TDateTime* dateTime)
Descripción:	Función estática utilizada para obtener el valor de la propiedad de diagnóstico relacionada con la cantidad de errores time out de lectura.
Nombre:	getErrFrame (BaseClass* base,TDateTime* dateTime)
Descripción:	Función estática utilizada para obtener el valor de la propiedad de diagnóstico relacionada con la cantidad de errores de trama
Nombre:	resetDiagnosticParameters ()
Descripción:	Inicializa los contadores de la información de diagnóstico del endPoint relacionado con el device.
Nombre:	getEndPoint ()
Descripción:	Devuelve la instancia de la clase UP1LCDEndPoint
Nombre:	getDeviceTransport ()
Descripción:	Devuelve la instancia de la interfaz ITransport
Nombre:	getDeviceTimer ()
Descripción:	Devuelve la instancia del timer asíncrono creada.
Nombre:	incrementFailure ()
Descripción:	Esta función incrementa el contador de fallos de comunicación del

	dispositivo.
Nombre:	incrementSucess ()
Descripción:	Esta función incrementa el contador de éxitos en la comunicación del dispositivo
Nombre:	setTransport (ISerialTransport* transport)
Descripción:	Establece un nuevo apuntador a la instancia del transporte serial que posee el driver
Nombre:	getTransport ()
Descripción:	Devuelve el apuntador a la instancia del transporte establecida.
Nombre:	setCloseAfterTransaction (BaseClass* base, bool newValue)
Descripción:	Función estática utilizada para el establecimiento de la propiedad relacionada con cerrar el dispositivo serial o no después de realizada la transacción
Nombre:	closeAfterTransaction(BaseClass* base)
Descripción:	Función estática utilizada para obtener el valor de la propiedad relacionada con el cierre o no del dispositivo serial después de finalizada la transacción
Nombre:	createBlock (int blockStart, int blockEnd, bool forRead,VarLinkVector* varsVector)
Descripción:	Este método tiene como objetivo la creación de un Bloque de variables.
Nombre:	createIProtocolAddress (const char* address, bool forRead)
Descripción:	Crea internamente una instancia de UP1LCDBlock la dirección y el tipo de operación pasados por parámetros
Nombre:	startBlock (IProtocolAddress* address1, DeviceVarType varType, unsigned long arraySize, bool forRead,unsigned long addressSize)
Descripción:	La función startBlock marca en startBlockAddress la dirección en forma de unsigned long de la primera variable

Nombre:	fitInBlock (IProtocolAddress* address2, DeviceVarType varType, unsigned long arraySize, bool forRead, unsigned long addressSize)
Descripción:	Esta función determina si una variable UP1LCD puede alojarse en un bloque
Nombre:	compare (IProtocolAddress* address1, IProtocolAddress* address2)
Descripción:	Esta función compara dos direcciones UP1LCDIProtocolAddress.

3.3.5. Clase UP1LCDBlock

La clase UP1LCDBlock implementa el concepto conjunto consecutivo de variables, que pueden ser accedidos por medio de en un solo mensaje. Hereda de la clase Block de la biblioteca DriverCore y de la UP1LCDHandler, de la cual reimplementa los callback de lectura y escritura para cuando se completen dichas tareas por parte del endpoint.

Tabla 8: Descripción de la clase UP1LCDBlock

Nombre: UP1LCDBlock	
Tipo de clase: Controladora	
Atributo:	**Tipo**
minPos	unsigned long
maxPos	unsigned long
dateTime	TDateTime
isMasked	bool
isWrite	bool
refAddress	unsigned long
Data	std::vector<int>
Address	IProtocolAddress*
mapa	std::map<int, int>

dir	unsigned int
Index	unsigned int
Para cada responsabilidad	
Nombre:	UP1LCDBlock (Device* device, unsigned long blockStart, unsigned long blockEnd, bool forRead, VarLinkVector* container)
Descripción:	Constructor de la clase
Nombre:	readInteger (IProtocolAddress* address, TDateTime* deviceTimeStamp, unsigned short* quality)
Descripción:	Esta función lee un valor entero del bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	readText (IProtocolAddress* address, unsigned long arraySize, char* value, TDateTime* deviceTimeStamp, unsigned short* quality)
Descripción:	Esta función lee una cadena del bloque a partir de un IProtocolAddress.
Nombre:	writeInteger (IProtocolAddress* address, long long value)
Descripción:	Escribe un valor entero en el bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	beginWrite ()
Descripción:	Realiza las operaciones necesarias para inicializar la escritura del Bloque
Nombre:	asyncMakeReadRequest ()
Descripción:	Responde por el despacho asíncrono de la solicitud de lectura.
Nombre:	asyncMakeWriteRequest ()
Descripción:	Responde por el despacho asíncrono de la solicitud de escritura
Nombre:	readHandler (CommandsEnum command, std::vector<int> data, unsigned long error)
Descripción:	Callback invocado por el EndPoint cuando finaliza una lectura realizada.

Nombre:	writeHandler (CommandsEnum command, unsigned long error)
Descripción:	Callback invocado por el EndPoint cuando finaliza una escritura realizada

3.3.6. Clase UP1LCDIProtocolAddress

La función de la clase UP1LCDIProtocoloAdress es representar las direcciones de las variables soportadas por el protocolo del UP1LCD. La misma hereda de la clase IProtocolAddress proporcionada por el DriverCore, permitiendo de esta manera encapsular el concepto de dirección de protocolo.

Tabla 9: Descripción de la clase UP1LCDIProtocolAddress

Nombre: UP1LCDIProtocolAddress	
Tipo de clase: Controladora	
Atributo:	Tipo:
Device	UP1LCDDevice*
Valid	bool
File	int
subFile	int
Para cada responsabilidad:	
Nombre:	UP1LCDIProtocolAddress (UP1LCDDevice* device, const char* address)
Descripción:	Constructor de la clase que recibe un apuntador al dispositivo y la dirección de la variable a leer.
Nombre:	getValid ()
Descripción:	Funcionalidad que devuelve un valor booleano si la dirección es válida o no de acuerdo a las relas específicas del protocolo

Nombre:	getFile()
Descripción:	Funcionalidad de devuelve el file que inicia la dirección
Nombre:	getSubFile()
Descripción:	Funcionalidad de devuelve el subFile de la dirección en cuestión
Nombre:	checkAddress (const char* address)
Descripción:	Funcionalidad que chequea dada una dirección, si es correcta o no.

3.4. Diagrama de Despliegue del Manejador UP1LCD

En este diagrama de despliegue se observa como haciendo uso del manejador, como software, se puede establecer una comunicación con el dispositivo. Para este caso el medio de comunicación sería el serial.

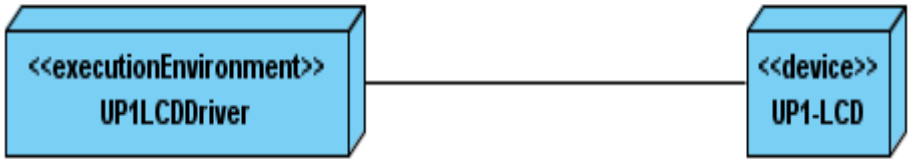

Figura: 24 Diagrama de Despliegue del Manejador UP1LCD

3.5. Diagrama de Componentes del Manejador UP1LCD

El manejador de manera general lo componen tres elementos. Las bibliotecas DriverCore y TransportProvider proveen un conjunto de funcionalidades utilizadas por el manejador, el cual como producto final constituye otra biblioteca más, estereotipado '.so' para sistema operativo Linux y '.dll' para Windows.

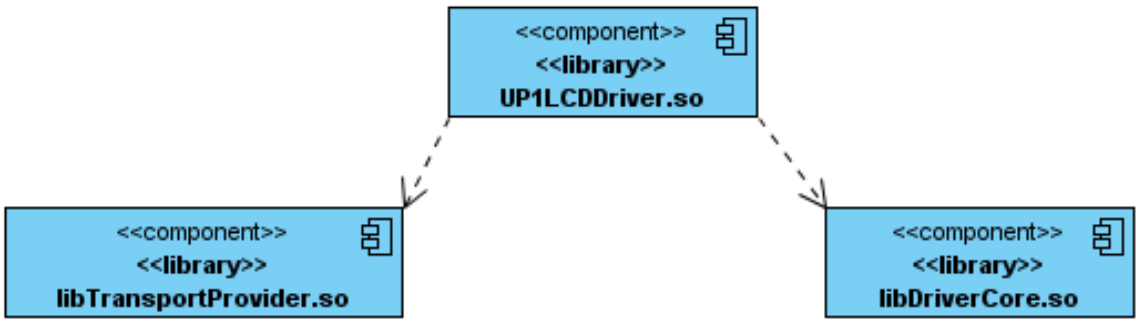

Figura: 25 Diagrama de Componentes del UP1LCD

3.6. Estándar de codificación

El estándar de codificación utilizado en el desarrollo del presente manejador fue definido en el documento "Estándar de codificación, Proyecto SCADA Nacional" v1.0. Haciendo uso del mismo permite establecer un conjunto de reglas de programación que no están dirigidas a la lógica del software, sino a su estructura y apariencia física, facilitando así su lectura, comprensión y mantenimiento del código.

Entre los principales elementos de este estándar se encuentran:

Idioma del código:

El código implementado es totalmente en idioma inglés, ya que es el más utilizado en el mundo de la programación.

Nombre de los ficheros:

Los nombre de los ficheros .h y .cpp comienzan con mayúscula, en el caso de ser un nombre formado por más de una palabra, cada una debe comenzar con mayúscula igualmente y las letras siguientes en minúsculas. Por ejemplo:

NameOfUnit.h

NameOfUnit.cpp

Nombre de clases, métodos y variables:

Para el caso de los nombres de clases se sigue un estilo similar al de los ficheros. Nunca seleccionar nombres que comiencen con uno o más caracteres de subrayado (underscore).Por ejemplo:
MyClass.

Los nombres de las funciones y las variables comenzaran con minúscula, si está compuesto por más de una palabra entonces estas comenzarían con mayúsculas a partir de la segunda. Exceptuando los constructores y destructores. Por ejemplo:
funtion ()
myFunction ()
La implementación de una función no será ubicada en la especificación de una clase a menos que la función vaya a ser inline.

3.7. Conclusiones parciales del capítulo

En este capítulo se abordaron aspectos específicos del diseño e implementación del manejador. Por ejemplo, la representación de los principales diagramas utilizados en el desarrollo del mismo, así como sus clases más importantes. De esta manera se logra una mejor comprensión estructural y funcionalmente del software desarrollado. Siendo esto punto muy importante porque permitiría la agregación de funcionalidades con mayor facilidad en caso de ser necesario.

Capítulo 4: Pruebas

4.1. Prueba de software

Cierto es que un desarrollo podrá estar marcado por características variables como sus dimensiones, tecnología, sector o criticidad, pero todos ellos comparten una característica común, lo desarrollan personas. Las personas cometen errores y o cambian de ideas, por eso hay que probar y volver a probar (18)

El único instrumento adecuado para determinar el status de la calidad de un producto software es el proceso de pruebas. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. En las pruebas se usan casos de prueba, especificados de forma estructurada mediante Técnicas de prueba. (19)

Para la realización de las pruebas, se utilizó la aplicación Recolector Gráfico, desarrollada en la línea de manejadores del proyecto Guardián del Alba. La misma posibilita la recolección de variables de acuerdo a sus períodos de encuesta. Es de gran ayuda para la puesta a punto y detección de fallas de los manejadores existentes. Así como la corrección de los errores de configuración que imposibilitan la recolección correcta de las variables configuradas.

Para realizar las pruebas al manejador en cuestión se utilizó la aplicación llamada Recolector Grafico, desarrollada en la línea de manejadores del proyecto Guardián del Alba. La misma posibilita la recolección de variables de acuerdo a sus períodos de encuesta. Es de gran ayuda para la puesta a punto y detección de fallas de los manejadores existentes y futuros y la corrección de los errores de configuración que imposibilitan la recolección correcta de las variables configuradas.

4.2. Diseño de Casos de Pruebas realizadas al manejador

A continuación se describen los casos de pruebas efectuados al manejador. Para el diseño de los mismos se tomaron en consideración los requisitos funcionales a cumplir. La ejecución de las pruebas posibilita la corrección de errores que afectaban el correcto funcionamiento de la aplicación final.

- **Nombre de caso de prueba:** Lectura de variable

Descripción general: Este caso de prueba la funcionalidad de lectura del manejador. Muchas son las variables que se pueden leer, en dependencia del comando de trama que se especifique. En este caso se escoge el comando “Rdf”, para leer el firmware del dispositivo.

Condiciones de ejecución: Debe haber creado un dispositivo de tipo UP1-LCD, y configurar sus parámetros específicos tanto del protocolo como del transporte. Debe estar creada la variable a leer en la paleta “Capturas”.

Tabla 10: Lectura de variable

Entrada	Respuesta del Sistema	Resultado de la prueba
Nombre: firmware Dirección (Lectura): 1:1 Tipo (Lectura): string	Se espera el valor <SOH>FIRMWARE UP1-LCD 1.2.65 b095<EOT>	La aplicación mostro el valor esperado: “UP1-LCD 1.2.65b”

Además de este caso de prueba con una variable se decidió probar la funcionalidad de lectura durante un tiempo aproximado de 8 horas. En ese tiempo se mantuvo la recolección de 12 variables, pertenecientes a 3 bloques diferentes. Arrojando como promedio una lectura estable, aunque en ocasiones por cuestión de rendimiento del equipo y de la máquina en que se probó la operación no era satisfactoria.

En la imagen de a continuación se observa la lectura de variables, haciendo uso del Recolector Gráfico, este caso es la variable firmware, con el valor “UP1-LCD 1.2.65b”

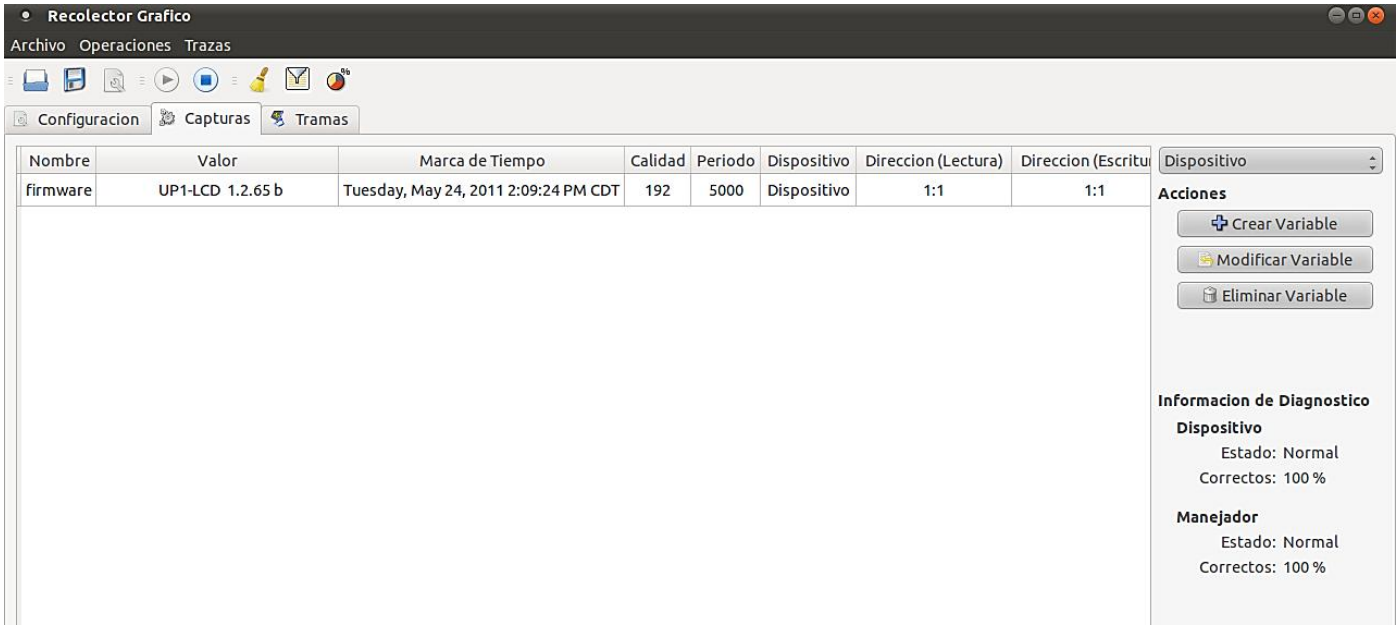

Figura: 26 Lectura de Variables

- **Nombre de caso de prueba:** Validación de direcciones admisibles

Descripción general: Este caso de prueba se valida las direcciones admisibles para el manejador en cuento a las características del protocolo. Se tomara la misma variable del caso anterior, pero con una dirección incorrecta.

Condiciones de ejecución: Debe haber creado un dispositivo de tipo UP1-LCD, y configurar sus parámetros específicos tanto del protocolo como del transporte.

Tabla 11: Validación de direcciones admisibles

Entrada	Respuesta del Sistema	Respuesta de la prueba
Nombre: firmware Dirección (Lectura): 1:2 Tipo (Lectura): string	La aplicación deberá mostrar la variable creada de color rojo, esto significa que la dirección no es correcta.	Dado los valores de entrada, la aplicación mostró lo esperado.

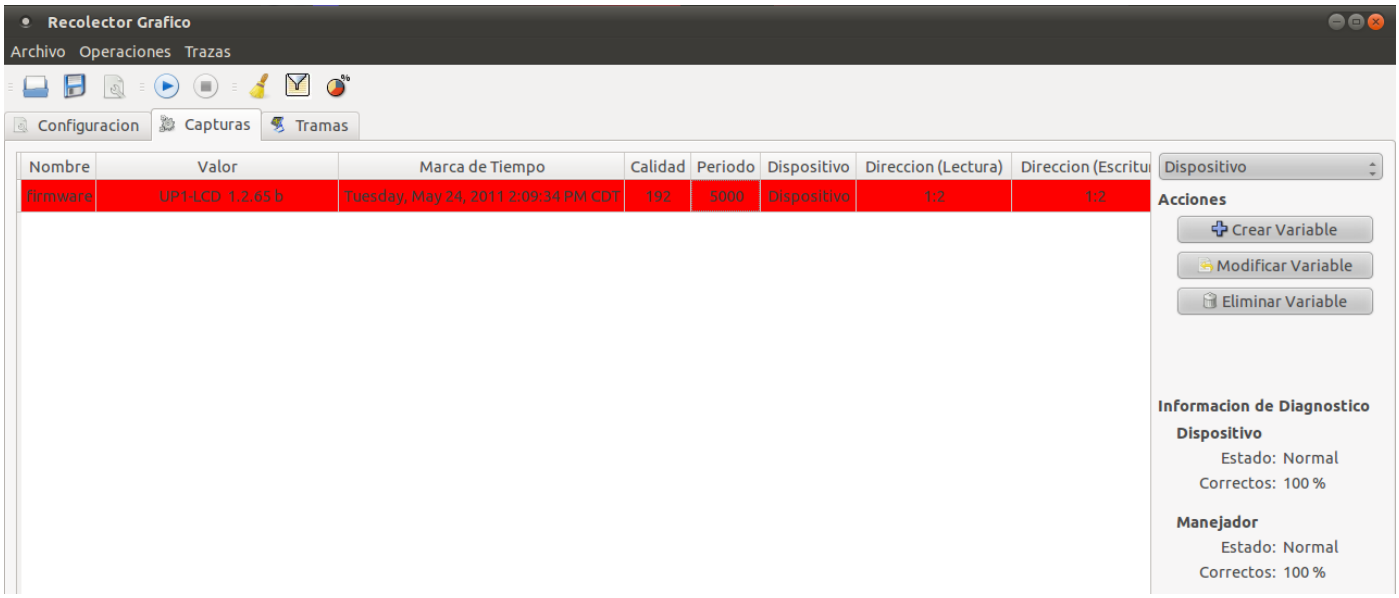

Figura: 27 Validación de dirección

- **Nombre de caso de prueba:** Estampado de tiempo

Descripción general: En el caso específico del estampado de tiempo nos auxiliaremos de las ventajas que nos ofrece el recolector gráfico, pues a la hora de crear una variable, este muestra dentro de la información de la misma, la requerida en este caso de prueba.

Condiciones de ejecución: Debe haber creado un dispositivo de tipo UP1-LCD, y configurar sus parámetros específicos tanto del protocolo como del transporte. Además debe estar creada una variable y puesta en función, tomaremos la variable firmware nuevamente.

Tabla 12: Estampado de tiempo

Entrada	Respuesta del Sistema	Respuesta de la prueba
Nombre: firmware Dirección (Lectura): 1:1 Tipo (Lectura): string	Una vez leída la variable, la aplicación muestra información referente a esta, entre la que se encuentra estampado de	Dado los valores de entrada, la aplicación mostró lo esperado, la

	tiempo.	fecha y hora del momento de la ejecución de la lectura

Figura: 28 Estampado de Tiempo

4.3. Pruebas de Rendimiento

Como prueba de rendimiento se realizó la puesta en marcha del manejador con 40 variables en modo lectura, por un tiempo aproximado de 12 horas. Se realizó una comparación entre los parámetros iniciales y finales de la estación donde se efectuaron las pruebas y se observó un equilibrio entre ambos. No se detectó interrupción del proceso durante la puesta en marcha y el estado de los parámetros del manejador y dispositivo era superior al 60 %.

4.4. Conclusiones parciales del capítulo

Este capítulo se concibió para registrar las pruebas de laboratorios efectuadas al manejador. Las mismas arrojaron los resultados esperados, al comprobar el correcto funcionamiento del manejador. Fueron catalogadas de forma general como positivas, aunque se detectó lentitud de respuesta por parte del dispositivo ante mayores peticiones de diferentes comandos o menor período de encuesta de variables.

Conclusiones Generales

Se obtuvo como producto final un manejador capaz de comunicarse e intercambiar información con el dispositivo UP1-LCD. El mismo podrá acoplarse al SCADA GECTEL como módulo independiente al sistema, posibilitando el manejo de información importante para la toma de decisiones de la empresa ante situaciones excepcionales.

El manejador cumple con los requisitos funcionales y no funcionales definidos para su implementación. Los mismos fueron comprobados satisfactoriamente durante la etapa de pruebas. Este producto debido a sus características, en caso de ser necesario puede reutilizarse en otros sistemas o entornos que requieran de sus funcionalidades.

Se garantiza su correcto funcionamiento de distribuciones Ubuntu y Debian de sistema operativo Linux.

Es importante señalar que el equipo no se comporta bien ante cargas de trabajo elevadas, dígase carga de trabajo a una cantidad de bloques de variables distintos excesiva. Por lo que para mejorar la comunicación se puede configurar un período de lectura superior a 2 segundos, porque un tiempo menor puede provocar un desbordamiento en el dispositivo trayendo como consecuencia que este genere información incorrecta, no obstante, en periodos mayores a 5 segundos se detectó que el equipo en ocasiones no responde a la petición enviada. Aparentando error en la comunicación

Recomendaciones

Se recomienda probar el manejador en un despliegue real, que permita el manejo de información valida y directa. Esto se hace con el objetivo de probar correctamente todas las funcionalidades implementadas en el manejador. No obstante las pruebas efectuadas en laboratorio, arrojaron resultados satisfactorios, a pesar de contar con un dispositivo defectuoso y de baja calidad.

Referencias Bibliográficas

1. **Forcade Gómez, Natasha; y otros.** *Sistemas de guías de laboratorio utilizando los entornos de desarrollo SCADA WinCC® e InTouch®.*

2. ETECSA. [Online] [Cited: 11 23, 2010.] http://www.etecsa.cu/index.php?sel=nuestraempresa.

3. **Autómatas.** Autómatas. *Autómatas.* [Online] Autómatas, 2006. http://www.automatas.org/redes/scadas.htm.

4. **Herrera Vázquez, Moisés.** *Diseño y Desarrollado de Sistema Automatizado de Inmersión Temporal.*

5. **Cedeño Pozo, Antonio.** *Módulos de adquisición y análisis para la interacción con dispositivos de campo de un SCADA.* La Habana : s.n., 2009.

6. *Revista Técnica de la Empresa de Telecomunicaciones de Cuba, S.A.* **Vidal García, Vilma.** Ciudad de la Habana : s.n. ISSN: 1813-5056.

7. **Stalling, William.** *Comunicación y redes de comunicación.*

8. **Noda Cid, Yunaime.** *Manejador Hitachi 912 para un sistema de Supervisión y Control de Procesos Automatizados.* 2010.

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

10. **Trujillo Codorníu, Rafael Arturo, Cedeño Pozo, Antonio and García Hernández , Luis Enrique.** *Interfaz Genérica de Manejadores de Dispositivos en el proyecto SCADA PDVSA.* 2007. v 5.0.

11. Alfresco. [Online] [Cited: enero 22, 2011.] http://alfresco.cedin.prod.uci.cu/alfresco/webdav/eXcriba/Workspace/Proyectos%20de%20Aplicaci%C3%B3n/Proyectos%20Cerrados/SCADA_ETECSA.tar.gz.

12. **González, Belkis Grissel y otros.** *Arquitectura de software proyecto SCADA- ETECSA v2.0.*

13. *Protocolo UP1-LCD v03-en.*

14. **CEDIN.** Alfresco. [Online] [Cited: febrero 13, 2011.] http://alfresco.cedin.prod.uci.cu/alfresco/webdav/eXcriba/Workspace/Proyectos%20de%20Desarrollo/Recolecci%C3%B3n/Expediente%20de%20Proyecto/1.%20ingenier%C3%ADa/1.1%20requisitos/CEDIN_Recoleccion_0113_ERS.doc.

15. **Uriarte Rodríguez, Pedro Alberto.** *Manejador para la comunicación con dispositivos de campo mediante el protocolo DF1.* 2010.

16. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* Buenos Aires : s.n., 2004.

17. **Facultad de Informática-Universidad Politécnica de Madrid.** *Patrones del Gang of Four.* Madrid : s.n.

18. **Raja Prado, Elena .** *Casi todas las pruebas de software.* Madrid : s.n. ISSN 1988-3455.

19. **www.pruebasdesoftware.com.** Pruebas de software. [Online] http://www.pruebasdesoftware.com.

Bibliografía

1. **Tecnocontrolli.** *UP1-LCD GSM Data Transmission system.*
2. *Sistema de adquisición supervisión y control para los pueblos del ALBA.* 2009.
3. **García Hernández, Luis Enrique, Gómez Johnson, Rubén and Moreno Borges, Adrián Carlos.** *Manejadores GE Fanuc y BSAP Serial para el proyecto SCADA PDVSA.* 2009.
4. **Pérez Pérez, Yosep Yasmany, Cedeño Pozo, Antonio and García Hernández, Luis Enrique.** *Framework para el desarrollo de manejadores de dispositivos para sistemas SCADA.* 2009.
5. **Trujillo Codorníu, Rafael , et al., et al.** *MANEJADOR ETHERNET/IP PARA LOS PLC CONTROLLOGIX EN EL PROYECTO SCADA PDVSA.*
6. **Garcia Hernandez, Luis Enrique.** *Framework para el desarrollo de manejadores de dispositivos, para el proyecto SCADA Guardián del ALBA.* 2009.
7. **Hung Gutiérrez, Eduardo Rafael .** *Manejador para la comunicación con sistemas de bases de datos.* 2010.
8. **Rodriguez Rodriguez, Amides.** *Manejador para el intercambio de información con dispositivos que se comuniquen a través del protocolo DNP.*
9. **Gomez Johnson, Ruben.** *Capa de acceso a datos sincrona y asincrona para dispositivos Modbus TCP.*
10. **Bridón Danger, Yordanis and Moreno Borges, Adrian Carlos.** *Interfaz asíncrona para la comunicación con los Controladores Lógicos Programables utilizando el.*
11. Portal CEDIN. [Online] [Cited: noviembre 16, 2010.] http://alfresco.cedin.prod.uci.cu/alfresco/d/d/workspace/SpacesStore/b3cfb98b-e3ec-4946-a742-c04b77b32390/Protocol_UP1-LCD_v03-en.pdf.

Glosario de Términos

- ✓ **Trama:** Es una unidad de envío de datos, viene a ser sinónimo de paquete de datos.
- ✓ **UP1-LCD:** Dispositivo fabricado por empresa italiana para la comunicación con grupos electrógenos marca GREYMO
- ✓ **Interfaz Genérica de los Driver:** Conjunto de funcionalidades que le permiten al SCADA comunicarse con cualquier tipo de dispositivo, sin importar el protocolo de comunicación.
- ✓ **Protocolo TCP/IP:**
- ✓ **Protocolo UDP:**
- ✓ **SCADA EROS:** SCADA monolítico desarrollado por la empresa SerCoNi, utilizado en varias instalaciones del país.
- ✓ **CEDAI:** Centro de Desarrollo de la automatización integral situado en la provincia de Villa Clara, mantiene una estrecha relación con la UCI.
- ✓ **IEEE**: Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. El propósito principal de la IEEE es fomentar la innovación tecnológica y excelencia en beneficio de la humanidad.
- ✓ **PDVSA**: Empresa venezolana dedicada al procesamiento del petróleo así como su comercialización. Es la principal entrada de ingresos de este país.