Universidad de las Ciencias Informáticas

Facultad 5.

Centro de Informática Industrial.

Título: “Manejador para la comunicación con dispositivos de campo mediante el protocolo DF1”

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor: Pedro Alberto Uriarte Rodríguez

Tutor: Antonio Cedeño Pozo

Ciudad de la Habana, Marzo 2010

DECLARACIÓN DE AUTORÍA

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

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

_____

Firma del Autor.

(Pedro Alberto Uriarte Rodríguez)

_____

Firma del Tutor.

(Ing. Antonio Cedeño Pozo)

DATOS DE CONTACTO

Ing. Antonio Cedeño Pozo

Email: acedeno@uci.cu

Graduado de Ingeniero Informático de la Universidad de las Ciencias Informáticas en el 2009. Cinco años de experiencia en el desarrollo de software.

AGRADECIMIENTOS

A mis padres, quienes me han brindado todo su cariño, amor y amistad. Gracias, por su apoyo incondicional y por confiar siempre en mí. Es un privilegio tener padres como ustedes.

A mis abuelos, quienes han sido junto a mis padres quienes más me han nutrido con su sabiduría y sus principios.

A mi familia por haberme rodeado toda la vida de un ambiente lleno de amor, respeto y cariño.

A mi novia Suneidis, por darme todo su amor y apoyo en estos años, sin ti no hubiera logrado lo que he hecho hasta hoy.

A mis compañeros y profesores del Centro de Informática Industrial, en especial a Dr. Trujillo, Johnson, Luis Enrique y Adrián, por mostrarme lo mucho que me falta por aprender y ayudarme a ser un mejor profesional.

A mi tutor Antonio, quien me ha apoyado desde que comencé en el proyecto y me ha obligado a prepararme cada día más.

A mi equipo de trabajo y amigos, en especial a Yunaime y Amides. Espero que sigamos trabajando juntos mucho tiempo, es un honor contar con ustedes.

A todos los profesores que contribuyeron en mi formación desde mi niñez, a todos llegue mi más sincero agradecimiento.

A nuestra Revolución, por darme la oportunidad de estudiar en una Universidad tan especial.

DEDICATORIA

A mis padres, Pedro y Silvana.

A mis abuelos, Clara y Orlando y a los que ya no están.

A mis hermanas, Lisnay y Aylen.

A mi familia.

RESUMEN

En el presente trabajo se describe el desarrollo de un manejador para la comunicación con el protocolo industrial DF1. La necesidad del desarrollo del mismo surge debido a que el SCADA “Guardián del Alba” no cuenta con un mecanismo que le permita la comunicación con los dispositivos de campo que se comunican a través de este protocolo. Esta situación es desfavorable para el SCADA, debido a que en las instalaciones de PDVSA abundan los dispositivos que utilizan el DF1, por ejemplo: el PLC-5 Allen Bradley.

El proceso de desarrollo comienza con un estudio de los principales conceptos enmarcados dentro del proceso de adquisición de datos en los sistemas SCADA y de las especificaciones del protocolo en cuestión. Se analizan las características esenciales y elementos de actualidad de algunos de los sistemas SCADA más reconocidos, así como de los subsistemas con que cuenta el SCADA “Guardián del Alba”, haciendo énfasis en el de adquisición.

Seguidamente se lleva a cabo un proceso de selección de tecnologías y herramientas de desarrollo para determinar cuáles utilizar en el diseño e implementación del manejador. De los requisitos generales previamente definidos para el módulo de driver del SCADA “Guardián del Alba” se seleccionan los que guiarán el diseño y la implementación de las capas de Driver y Protocolo que conforman el manejador. Por último se concluirá el proceso de desarrollo con las pruebas que garanticen la calidad y el funcionamiento correcto del manejador.

PALABRAS CLAVE

Adquisición de datos, DF1, manejador, PLC, protocolo industrial, SCADA.

Índice

Índice de Figuras

Índice de Tablas

INTRODUCCIÓN

Con el desarrollo de los procesos industriales y el incremento sustancial del papel que juega la automática dentro estos procesos, se ha hecho necesario mantener una supervisión constante de su funcionamiento para garantizar la seguridad de estos ambientes tan complejos. Dándole respuesta a estas necesidades de supervisión, surgen los sistemas SCADA, acrónimo de Supervisory Control and Data Acquisition, en español, Control Supervisor y Adquisición de Datos.

El término SCADA denota un sistema informático para la gestión de datos. Es un sistema muy complejo y utilizado en una gran variedad de industrias, debido a su eficiencia. Su papel fundamental es la recopilación de información proveniente de dispositivos, que transmiten dichos datos a una estación central que supervisa los procesos y administra la información adquirida. Estos sistemas están concebidos para prevenir y alertar situaciones inesperadas en cualquier industria. Además de la recuperación de datos, el sistema puede realizar otras funciones que incluyen la supervisión y control de información en tiempo real la cual es posible gracias a sensores y actuadores que se instalan en la industria. Esta información es transmitida al sistema en determinados momentos pudiendo ser vista y modificada por los operadores del sistema.

Petróleos de Venezuela Sociedad Anónima (PDVSA) es la corporación estatal de la República Bolivariana de Venezuela dedicada a la exploración, producción, manufactura, transporte y mercadeo de los hidrocarburos. PDVSA contaba hasta diciembre del 2002 en sus instalaciones con varios sistemas SCADA suministrados por compañías extranjeras. Estos SCADA eran imprescindibles para mantener la confiabilidad y eficiencia de los procesos tecnológicos de la industria petrolera venezolana, pero tenían como desventaja que las compañías abastecedoras exigían un elevado costo para el mantenimiento del software y obligaban al cliente a ser dependientes de sus servicios.

La industria petrolera venezolana fue víctima de un sabotaje entre diciembre del año 2002 y 2003, que estuvo marcada por el bloqueo de los sistemas de automatización existentes para la supervisión y el control de los procesos dentro de la industria petrolera. En este bloqueo de los sistemas tuvieron mucho que ver algunas de estas compañías proveedoras. A raíz de este acontecimiento el Gobierno Bolivariano tomó la decisión de comenzar a trabajar para lograr una independencia tecnológica, siendo la tarea de establecer el uso de software libre y estándares abiertos en los sistemas, proyectos y servicios

informáticos de la Administración Pública Nacional de Venezuela y el desarrollo de un SCADA en plataforma libre, algunos de sus primeros pasos para conseguirla.

En el 2006, comenzó el proyecto SCADA PDVSA conocido en la actualidad como “Guardián del Alba” o “Galba”. El mismo consiste en el desarrollo de un SCADA para la industria petrolera venezolana que garantice la soberanía tecnológica de Venezuela y le ahorre millones de dólares en licencias de productos propietarios. Este proyecto está dividido arquitectónicamente en varias líneas de investigación y desarrollo, donde podemos encontrar la línea de manejadores (driver en inglés) encargada de la comunicación del SCADA con los dispositivos de campo y la responsable de que los datos registrados por estos dispositivos puedan ser accedidos desde el sistema. El intercambio de datos con estos instrumentos se realiza mediante protocolos de comunicación, que no son más que el “lenguaje” mediante el cual se realizan las peticiones de escritura y lectura de las variables configuradas y de relevancia para el sistema SCADA.

Actualmente el SCADA “Galba” no puede comunicarse con una amplia gama de dispositivos de campo tales como: PLC-5, Micro-Logix 1000, SLC-500, SLC-5/03, SLC-5/04, 1774-PLC, PLC-2 y PLC-3, con los cuales el intercambio de información se realiza mediante el protocolo de comunicación DF1. Muchos de estos equipos se encuentran en funcionamiento en varias instalaciones de la industria petrolera venezolana y reemplazarlos constituye una operación muy costosa y compleja. Que el SCADA “Galba” no sea capaz de establecer una comunicación con esta gama de equipos, tiene como consecuencia que en las instalaciones de PDVSA donde estén presentes algunos de estos, se verá reducida su capacidad de recolección. Esta limitación constituye un obstáculo en la implantación del “Galba” dentro de la industria petrolera y evita que este reemplace totalmente a los SCADA propietarios pertenecientes a compañías extranjeras, de las que aún depende PDVSA para gran parte del funcionamiento de su industria.

Ante esta situación problemática surge el siguiente **problema científico**:

¿Cómo intercambiar información con los dispositivos de campo que se comunican mediante el protocolo DF1 en el SCADA “Guardián del ALBA”?

Para resolver este problema se planteó como **objeto de estudio** el proceso de adquisición de datos en los sistemas SCADA.

El **objetivo general** es desarrollar un manejador para la lectura y escritura de las variables que son configuradas en los dispositivos que se comunican mediante el protocolo DF1.

El **campo de acción** son los mecanismos para el acceso a la información de los dispositivos de campo que se comunican a través del protocolo DF1.

Para dar cumplimiento al objetivo general se definieron las siguientes **Tareas de Investigación**:

- Búsqueda y estudio de la bibliografía que hace referencia al protocolo DF1 para seleccionar la especificación más completa del estándar en cuestión.
- Estudio de los manejadores implementados para el "Guardián del ALBA" y de otras implementaciones del protocolo DF1, para la reutilización de patrones y decisiones de diseño puestas en práctica en las soluciones mencionadas con anterioridad.
- Estudio de la Especificación IG 5.0 y de las bibliotecas DriverCore y TransportProvider para llevar a cabo el diseño e implementación de las capas de Driver y de Protocolo pertenecientes al manejador DF1.
- Estudio y selección de estructuras compatibles con la mayoría de los compiladores de C++ y con los sistemas operativos más utilizados, para lograr un producto multiplataforma.
- Diseño de la capa de protocolo.
- Implementación de la capa de protocolo.
- Diseño de la capa de Driver.
- Implementación de la capa de Driver.

Se utilizarán varios métodos científicos de investigación como:

Analítico-Sintético e Inductivo-Deductivo: utilizados en el análisis de las teorías y documentos que existen alrededor del desarrollo de manejadores y selección de los elementos más importantes relacionados con el protocolo DF1.

Modelación: se realizarán diagramas y modelos que permitan estructurar teóricamente el sistema (diagramas: de clases del diseño, de despliegue, etc.).

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 SISTEMAS SCADA

Se nombra SCADA (*Supervisory Control And Data Acquisition* o Control con Supervisión y Adquisición de Datos) a cualquier software que permita el acceso y control de datos remotos de un proceso, utilizando las herramientas de comunicación necesarias. Es un software especialmente diseñado para funcionar sobre computadoras dedicadas al control de producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, etc.) y controlando el proceso de forma automática desde la computadora. Provee toda la información que se genera en el proceso productivo a diversos usuarios, tanto del mismo nivel como de otros usuarios supervisores dentro de la empresa (supervisión, control calidad, control de producción, almacenamiento de datos, etc.). En este tipo de sistemas usualmente existe un ordenador, que efectúa tareas de supervisión y gestión de alarmas, así como tratamiento de datos y control de procesos. La comunicación se realiza mediante buses especiales o redes LAN. Todo esto se ejecuta normalmente en tiempo real, y están diseñados para dar al operador de planta la posibilidad de supervisar y controlar dichos procesos. (1)

1.1.1 Prestaciones

En los sistemas SCADA deben estar presentes de forma general una serie de prestaciones y características que le brindan al cliente un mejor producto:

- Una arquitectura abierta, capaz de adaptarse a las condiciones cambiantes de las variedades de industrias donde puede ser instalado.
- La comunicación con el usuario, los equipos de planta y otras empresas debe ser fácil de realizar.
- El despliegue e instalación de los paquetes del SCADA debe ser sencillo, al igual que las exigencias de hardware no deben ser excesivas y las interfaces que se les brinde a los usuarios deben ser amigables, fáciles de entender y usar.

1.1.2 Requisitos

Un SCADA debe cumplir con varios requisitos:

- Ser sistemas de arquitectura abierta, capaces de crecer o adaptarse según las necesidades cambiantes de la empresa.
- 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).
- Ser programas sencillos de instalar, sin excesivas exigencias de hardware, fáciles de utilizar y con interfaces amigables con el usuario.

1.1.3 Principales Fabricantes

En el mundo existen varias empresas que se dedican a desarrollar SCADA, a continuación se nombran algunas de las más destacadas:

Tabla 1 Principales fabricantes de sistemas SCADA.

Principales Fabricantes y sus productos.	
Fabricante	**Producto**
Siemens	WinCC
Rockwell Automation	RS-View32
Wonderware	InTouch
Progea	Movicon
GE-Fanuc	Cimplicity
Yokogawa	Fast/Tools

A continuación se describe el funcionamiento de los productos de dos de los principales fabricantes de sistemas SCADA:

- Rockwell Automation brinda a sus usuarios una familia de módulos que se integran para dar solución a las necesidades de los clientes, estos módulos están divididos en grupos funcionales, los cuales son: Diseño y Configuración, Administración de la Producción, Administración de datos, Calidad y Conformidad, Gestión de Activos y Rendimiento y Visibilidad. Específicamente el RSView32 es un componente base HMI, para monitorear y controlar procesos y maquinaria automatizada. Su compatibilidad con la tecnología cliente/servidor OPC le permite comunicarse con una amplia variedad de dispositivos de hardware. El producto se complementa con RSView32 Active Display System y RSView32 WebServer (el primero para ver y controlar los proyectos RSView32 desde localidades remotas y el segundo para que cualquier usuario autorizado pueda acceder a gráficas, etiquetas y alarmas mediante el uso de un navegador de internet convencional). Para el proceso de recolección y configuración de los dispositivos de campo se utilizan módulos como el RSLinx, que proporciona conectividad con los dispositivos y el acceso al RSView32 y el RSLogix5 para la comunicación y configuración del PLC-5 Allen-Bradley.

- Siemens a través del producto HMI WinCC, combina las características estándar (gráficos, alarmas, administración de recetas, etcétera) con otras avanzadas (reportes, referencias entre proyectos, diagnóstico de proceso, soporte multilingüe y redundancia completa). Con su producto WebCC, se logra que una gran variedad de información desde varias áreas de la planta se despliegue simultáneamente en una sola interfaz de usuario. Así, se facilita el control corporativo y la operación y monitoreo de las instalaciones de proceso y plantas de producción. El producto SIMATIC WinAC del mismo fabricante es una solución integrada para control, HMI, redes y procesamiento de datos, todos en la misma plataforma, con el cual es posible emular el funcionamiento de un PLC en una PC (la parte de control permite que se utilice una PC para emular a un PLC).

- El SCADA Fast/Tools de la compañía Yokogawa está soportado sobre módulos que se conectan con el bus de comunicación de datos llamado BUS/FAST. La base de datos en tiempo real es uno de los módulos que se conectan con el bus. Todos los módulos de Fast/Tools están basados en eventos que son enviados al bus. El bus envía esos eventos a los módulos que corresponden. Una configuración mínima del Fast/Tools compromete a los siguientes módulos:
  - BUS/FAST
  - ITEM/FAST: Administra la base de datos en tiempo real.
  - DATABASE/FAST: Administra todos los objetos y definiciones de alarmas.
  - EQUIPMENT/FAST: Administra toda la acción de los manejadores de entrada y salida.

  - De forma adicional otros módulos pueden conectarse al bus para enriquecer las funcionalidades del Fast/Tools:
  - ALARM/FAST: Maneja todas las alarmas.
  - HISTORY/FAST: Se hace cargo de salvar datos, alarmas, reportes y datos de auditoría.
  - AUDIT/FAST: Administra las definiciones de auditorías y los datos a ser auditados.
  - REPORT/FAST: Administra todas las definiciones de reportes y generaciones.
  - ACCESS/FAST: Administra la interfaz OPC.
  - USER/FAST: Suple la interfaz de configuración y reporte al usuario.

1.1.4 Evolución

En lo últimos años ha existido una evolución en los sistemas SCADA, orientada a ampliar su campo de aplicación. De una supervisión y control iniciales a nivel de máquina o de proceso, se ha pasado a una supervisión y control a nivel de planta. De una adquisición y registro de datos orientada a un control de proceso o de línea, se ha ampliado su utilidad a proveer información en tiempo real del estado de la planta o de la fábrica.

Una de las demandas más generalizadas y al mismo tiempo, una de las más críticas, es la capacidad de efectuar consultas trabajando con datos procedentes de diferentes fuentes: de diferentes aplicaciones o de bases de datos distintas y ubicadas en diferentes puntos del sistema. Disponer del conjunto de manejadores necesarios para intercomunicar los diversos componentes de la solución completa, configurarlos y activarlos de forma transparente, es un elemento esencial para disponer de una integración efectiva.

1.2 SCADA GALBA

Gracias a la arquitectura que posee este SCADA, es fácilmente configurable y extensible, lo cual garantiza la integridad y confidencialidad de la información, así como la capacidad de integración con otros sistemas. Ofrece funcionalidades para incorporar nuevos manejadores o protocolos de comunicación con dispositivos de forma dinámica en su módulo de Drivers, lo que permite trabajar con varios dispositivos, contando en estos momentos con más de una docena de manejadores incorporados.

1.2.1 Subsistema de Adquisición

Es el encargado de la transferencia de los datos entre el SCADA y los dispositivos que se encuentran en el campo, así como de su selección y procesamiento. Este subsistema está formado por tres componentes: Manejadores de Dispositivos, Módulo de Recolección y Base de Datos de Tiempo Real.

1.2.1.1 Manejadores de Dispositivos (Drivers)

Estos son módulos independientes en forma de bibliotecas dinámicas que permiten el intercambio de datos que provienen de disímiles equipos que pueden ser Autómatas, PLC, reguladores autónomos, sensores inteligentes, controladores, etc.

1.2.1.2 Módulo de Recolección

Es el responsable de construir las listas de los puntos, asociados a dispositivos de campo a consultar, agrupados según la frecuencia de recolección y entregar dichas listas a los manejadores para que estos construyan los bloques de encuesta.

1.2.1.3 Módulo BDTR

Encargado de manejar todo lo referente a la recepción, procesamiento y distribución de los datos provenientes del campo en tiempo real.

1.2.2 Subsistema Middleware

Se encarga de la comunicación entre los diferentes módulos que forman parte del sistema. Implementa dos formas de comunicación, el modelo de comunicación asincrónica, y el modelo sincrónico.

1.2.3 Subsistema de Visualización

Está compuesto por dos partes fundamentales, el ambiente de configuración editor y el ambiente de ejecución. El primero de estos permite configurar varios procesos, se definen y gestionan las variables, los drivers, los comandos, las alarmas y variadas opciones adicionales. El segundo ambiente se encarga de visualizar las animaciones y los objetos definidos en el editor, muestra lo que está ocurriendo en el campo en tiempo real, es el que envía los comandos a las estaciones remotas, quién recibe los valores de las variables, es el que interactúa con la mayoría de los operadores pues se emplea para supervisar el proceso de manera directa.

1.2.4 Subsistema de Históricos

Este subsistema es el encargado de almacenar la información del sistema con el objetivo de que esta pueda ser empleada luego en la generación de reportes, tendencias y gestión de producción.

1.2.5 Subsistema de Reportes

Se encarga de la configuración o edición de los informes y un posterior momento conocido como generación, que se basa en la información contenida en las plantillas de diseño para extraer los datos de las distintas bases de datos y conformar un documento, con la apariencia editada, y con los datos obtenidos de las fuentes de datos correspondientes.

1.2.6 Subsistema de Configuración

Es el encargado de almacenar, persistir y suministrar la información base para el funcionamiento de los demás módulos del SCADA (BDTR, BDH).

1.2.7 Subsistema de Seguridad

Provee las funcionalidades necesarias para garantizar el trabajo autorizado por usuarios y módulos, brinda las herramientas necesarias para la protección contra ataques maliciosos o involuntarios al sistema.

1.2.8 Subsistema de Integración con Terceros

Modelo de comunicación orientado a servicios, (SOA), los cuales pueden ser utilizados por un conjunto de aplicaciones externas al sistema.

1.3 CONTROLADORES LÓGICOS PROGRAMABLES (PLC)

Dispositivo electrónico operado digitalmente que utiliza la memoria programable para el almacenamiento interno de instrucciones a fin de implementar funciones específicas, tales como lógicas, secuenciales, tiempo y aritméticas y así controlar varios tipos de máquinas o procesos a través de módulos de entrada / salida analógicos o digitales. (2)

1.3.1 Funciones básicas

Los PLC debido a que operan en base a operaciones lógicas son normalmente usados para el control de procesos secuenciales, es decir, procesos compuestos de varias etapas consecutivas, en donde el PLC controla que las etapas se ejecuten sólo cuando se hayan cumplido una serie de condiciones fijadas en el programa. (3)

En general, estas funciones básicas pueden ser:

- **Detección**: Lectura de la señal de los captadores distribuidos.
- **Mando**: Elaborar y enviar las acciones al sistema.
- **Diálogo hombre máquina**: Mantener un diálogo con los operarios de producción, obedeciendo sus órdenes e informando del estado del proceso.
- **Programación**: Para introducir, elaborar y cambiar el programa de aplicación del autómata. El dialogo de programación debe permitir modificar el programa incluso con el autómata controlando la máquina.

1.4 PROTOCOLOS DE COMUNICACIÓN

Un protocolo de comunicación es un conjunto de reglas que permiten la transferencia e intercambio de datos entre los distintos dispositivos que conforman una red.

1.4.1 Clasificación

Entre los protocolos propios de una red de área local podemos distinguir dos principales grupos. Por un lado están los protocolos de los niveles, físico y de enlace del modelo OSI, estos protocolos definen las funciones que se asocian con el uso del medio de transmisión. Se refieren al envío de los datos a nivel de bits y trama; y el modo de acceso de los nodos al medio. Estos protocolos están determinados sólo por el tipo de red que utilizan. En el segundo grupo de protocolos se encuentran aquellos que realizan las funciones de los niveles de red y transporte de OSI, estos son los que se encargan del encaminamiento de la información y garantizar una comunicación extremo a extremo, libre de errores. Estos protocolos transmiten la información a través de la red utilizando pequeños segmentos denominados paquetes. Cada

protocolo define su propio formato de paquetes, en el que se especifican elementos como: el origen, destino, longitud, tipo del paquete y la información redundante para el control de errores.

1.5 MANEJADORES DE DISPOSITIVOS

Un manejador es un programa o componente hardware que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz para usarlo. Puede ser el encargado de controlar uno o más dispositivos. Realiza algunas tareas como: la iniciación del enlace, la aplicación de formatos, el orden de las transferencias y la gestión del protocolo de comunicación.

1.5.1 Manejadores para protocolos industriales

La mayoría de las empresas utilizan estándares internacionales para definir los protocolos mediante los cuales será posible la comunicación con los elementos de hardware que fabrican. Las formas como los instrumentos fabricados por éstas firmas se comunican con otros dispositivos son muy variadas, típicamente tienen integradas interfaces de comunicación, tales como: RS232, RS485, RS222 y Ethernet (IEEE 802.3); sobre estos tipos de interfaces las comunicaciones se establecen utilizando diferentes protocolos o lenguajes de comunicación. (4)

En Cuba existen algunas empresas que poseen experiencia en el diseño y ensamblado de equipos para la automatización de procesos industriales, entre ellas se puede mencionar a la Empresa de Servicios Técnicos de Computación Comunicaciones e Informática del Níquel (SerCoNi), perteneciente a la Unión del Níquel en el municipio Moa de la provincia Holguín, y al Instituto Central de Investigaciones Digitales (ICID), ésta última institución trabaja en los campos de automatización y sistemas integrados, sistemas médicos de tecnología avanzada, computación, desarrollo y producción de equipos electrónicos y circuitos impresos. Ambas entidades se basan en estándares internacionales a la hora de diseñar e implementar los protocolos para sus dispositivos y muchas veces proveen los manejadores asociados a los mismos. (4)

1.6 INTERFAZ GENÉRICA

En la actualidad los SCADA que se desarrollan difícilmente contemplan en su código fuente todos los protocolos posibles para la gran cantidad de dispositivos con que estos sistemas se deben comunicar. Lo más común es crear un protocolo general y se le encarga la tarea de traducir de este protocolo general a los específicos a los manejadores, que se programan en módulos independientes al SCADA. 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. (5) Al mismo tiempo el diseño de una arquitectura general no debe tener un impacto negativo en el rendimiento de los manejadores, lo cual es de suma importancia en la supervisión y el control. Haciendo uso de esta interfaz se han desarrollado los manejadores para los protocolos Modbus TCP, Modbus RTU, Modbus ASCII, OPC, ABInterchange de Allen Bradley, BSAP Serie de Bristol y Ethernet/IP.

1.6.1 Soporte de múltiples modelos de cooperación

Se entiende por modelo de cooperación a la forma en que dos o más entidades de una red industrial llevan a cabo el intercambio de información. Los modelos que soporta la IG son el modelo Cliente/Servidor y Productor/Consumidor. El Cliente/Servidor es un modelo de cooperación mediante el cual una de las entidades (cliente) emite una demanda de servicio a la otra entidad (servidora), el servidor trata la demanda y envía la respuesta correspondiente al cliente. El Productor/Consumidor es un modelo de cooperación multipunto mediante el cual una de las entidades (productor) envía datos a un conjunto de entidades consumidoras. En este caso la emisión es iniciada por el productor. Llevado al plano del SCADA los dispositivos serían los productores y quienes iniciarían la comunicación. La IG tiene funciones dedicadas a ambos mecanismos que posibilitan que, incluso en el modelo productor / consumidor, no sea necesaria la creación de contextos de ejecución en el manejador. (5)

1.6.2 Arquitectura multicapas

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. (5)

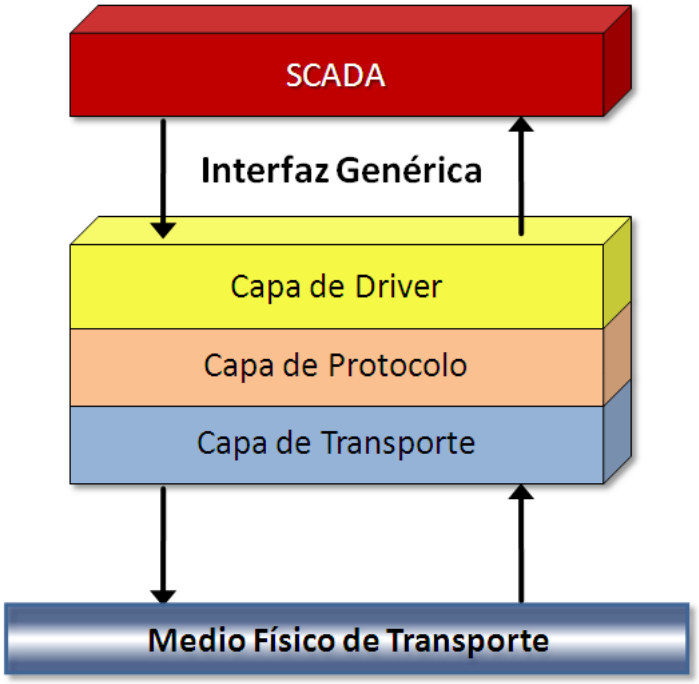

Fig. 1 Arquitectura multicapas.

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. (5)

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 implementación 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.

1.7 Comunicaciones Half-Duplex y Full-duplex

La transmisión half-duplex permite el envío de información en ambas direcciones pero con la limitación de que solo puede realizarse en una dirección a la vez. Tanto el transmisor como el receptor comparten una sola frecuencia. Un ejemplo típico de half-duplex es el radio de banda civil (CB) donde el operador puede transmitir o recibir, pero no puede realizar ambas funciones simultáneamente por el mismo canal. Cuando el operador ha completado la transmisión, la otra parte debe ser avisada que puede empezar a transmitir diciendo "cambio". (12)

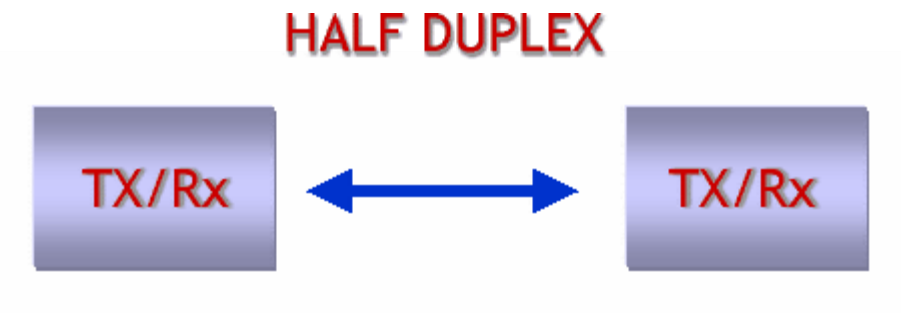

Fig. 2 Transmisión Half-Duplex.

La transmisión full-duplex (fdx) permite transmitir en ambas direcciones, pero simultáneamente por el mismo canal. Existen dos frecuencias una para transmitir y otra para recibir. Ejemplos de este tipo abundan en el terreno de las telecomunicaciones, el caso más típico es la telefonía, donde el transmisor y el receptor se comunican simultáneamente utilizando el mismo canal, pero usando dos frecuencias. (12)

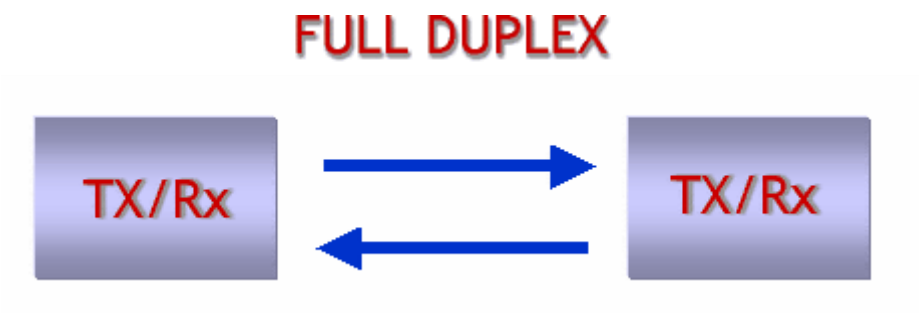

Fig. 3 Transmisión Full-Duplex.

1.8 TECNOLOGÍAS Y HERRAMIENTAS DE DESARROLLO

1.8.1 Biblioteca Boost

Boost es un conjunto de bibliotecas de clases, de código abierto y revisión por pares preparadas para extender las capacidades del lenguaje de programación C++. Su licencia permite que sea utilizada en cualquier tipo de proyecto, ya sea comercial o no. Varios fundadores de Boost pertenecen al Comité ISO de Estándares C++. (13)

Su diseño e implementación permite que sea utilizada en un amplio espectro de aplicaciones y plataformas. Abarca desde librerías de propósito general hasta abstracciones del sistema operativo. Con el objetivo de alcanzar el mayor rendimiento y flexibilidad se hace un uso intensivo de plantillas. Boost ha representado una fuente de trabajo e investigación en programación genérica y meta programación en C++. (13)

1.8.2 Biblioteca Asio

Asio es una biblioteca multiplataforma de código abierto para la implementación de aplicaciones de red, que brinda un desarrollo consistente de un mecanismo asíncrono de flujos de entrada y salida, usando un moderno enfoque de C++. Fue incluida en la biblioteca Boost el 30 de diciembre del 2005, aunque también puede ser utilizado de forma individual. En el 2006 se le envió al comité de C++ estándar una propuesta de un módulo de red basado en Asio para ser incluido en las extensiones de C++. (14)

Esta biblioteca ha sido probada en las siguientes plataformas:

- Window de 32 y 64 bit
- Linux (kernels 2.4 y 2.6)
- Solaris
- Mac OS X 10.4

1.8.3 Qt

Qt es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica como herramientas de la consola y servidores. Es producido por la división de software Qt de Nokia, que entró en vigor después de la adquisición por parte de Nokia de la empresa noruega Trolltech, el productor original de Qt, el 17 de junio del 2008. Utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación. El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de otros para el manejo de ficheros, además de estructuras de datos tradicionales. (15)

1.8.4 Posix

Posix es el acrónimo de Interfaz de Sistemas Operativos Portables. Es un estándar basado en el popular sistema operativo UNIX. Es una familia de estándares en evolución, cada uno de los cuales cubre

diferentes aspectos de los sistemas operativos. (16) Según la documentación de Posix, estos pueden ser agrupados en tres categorías:

- **Estándares base:** Definen la sintaxis y semántica de interfaces de servicios relacionados con diversos aspectos del sistema operativo. Estas interfaces permiten al programa de aplicación invocar directamente los servicios del sistema operativo. El estándar no especifica cómo se implementan estos servicios, sino únicamente su semántica; de este modo, los implementadores pueden elegir la implementación que estimen más conveniente siempre que sigan la especificación de la interfaz. La mayoría de los estándares base están especificados para el lenguaje de programación C.
- **Interfaces en diferentes lenguajes de programación ("bindings"):** Estos estándares proporcionan interfaces a los mismos servicios definidos en los estándares base, pero usando lenguajes de programación diferentes. Los lenguajes que se han usado hasta el momento son Ada y el Fortran.
- **Entorno de Sistemas Abiertos:** Estos estándares incluyen una guía al entorno POSIX y perfiles de aplicación. Los perfiles son subconjuntos de los servicios POSIX que se requieren para un determinado ámbito de aplicación. Los perfiles representan un importante mecanismo para definir de forma estándar un conjunto bien definido de implementaciones de sistemas operativos, adecuadas para áreas de aplicación específicas.

Las unidades de concurrencia del POSIX 1003.1 son los procesos con espacios de direcciones independientes. En la práctica, los procesos son relativamente voluminosos y poco eficientes debido al requisito de espacios de direcciones independientes y al estado asociado al proceso, que es voluminoso. Un cambio de contexto entre procesos es complejo, y normalmente requiere reprogramar la unidad de gestión de memoria, entre otras cosas. Por el contrario, las tareas de un núcleo de tiempo real de alta eficiencia comparten el mismo espacio de direcciones y tienen un estado asociado pequeño. Por tanto, con el propósito de incrementar la eficiencia, se introdujeron en el POSIX los threads o hilos, como un mecanismo de concurrencia de alta eficiencia. Bajo la Extensión de Threads, cada proceso POSIX puede tener múltiples flujos de control concurrentes, todos ellos compartiendo el mismo espacio de direcciones. El estándar proporciona servicios para la gestión, cancelación, planificación, y sincronización de hilos. (28)

1.8.5 IDE

Un entorno de desarrollo integrado o en inglés Integrated Development Environment (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Las principales características con que deben contar los IDE son: flexibilidad, estabilidad y documentación existente. (18)

1.8.5.1 QtCreator

Qt Creator es un IDE creado por Trolltech para el desarrollo de aplicaciones con la biblioteca Qt. Cuenta con herramientas como:

- Editor avanzado para C++.
- Diseñador de formularios (GUI) integrado.
- Herramientas para la administración y construcción de proyectos.
- Completado automático.
- Depurador visual.

Los sistemas operativos que soporta en forma oficial son:

- GNU/Linux 2.6.x, para versiones de 32 y 64 bits con Qt 4.x instalado. Además hay una versión para Linux con gcc 3.3.
- Mac OS X 10.4 o superior, requiriendo Qt 4.x
- Windows XP y Vista, requiriendo el compilador MinGW y Qt 4.4.3 para MinGW.

1.8.5.2 Eclipse

Eclipse fue desarrollado en un principio por IBM, en la actualidad lo mantiene la Fundación Eclipse, organización independiente que fomenta otros productos de código abierto. El IDE es multiplataforma, emplea módulos o plug-ins para la extensión de sus funcionalidades, gracias a los mencionados plug-ins es posible poder programar en C/C++, Python, PHP y Java, permite la integración con varias librerías de desarrollo como es el caso de Qt, da soporte para sistemas de gestión de bases de datos, control de versiones CVS y Subversion (SVN) y además para la realización de pruebas de unidad.

1.8.5.3 Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

A partir de la versión 2005 Microsoft ofrece gratuitamente las Express Editions. Estas son varias ediciones básicas separadas por lenguajes de programación o plataforma enfocadas para novatos y entusiastas. Estas ediciones son iguales al entorno de desarrollo comercial pero sin características avanzadas. Las ediciones que hay son:

- Visual Basic Express Edition.
- Visual C# Express Edition.
- Visual C++ Express Edition.
- Visual J# Express Edition (Desapareció en Visual Studio 2008).
- Visual Web Developer Express Edition (para programar en ASP.NET).

1.8.5.4 KDevelop

KDevelop es un entorno de desarrollo integrado para sistemas GNU/Linux y otros sistemas Unix, publicado bajo licencia GPL, está orientado principalmente al uso bajo el entorno gráfico KDE. Esta interfaz de desarrollo no cuenta con un compilador, por lo que depende de gcc para producir código binario.

Su última versión se encuentra actualmente bajo desarrollo y funciona con distintos lenguajes de programación como C, C++, Java, Ada, SQL, Python, Perl y Pascal, así como guiones para el intérprete de comandos Bash.

1.8.6 Lenguajes de Programación

Un lenguaje de programación es un idioma artificial diseñado para expresar operaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para expresar algoritmos con precisión o crear programas que controlen el comportamiento físico y lógico de una máquina. Está formado de un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

1.8.6.1 C

C es un lenguaje de programación creado en 1972 por Dennis M. Ritchie en los Laboratorios Bell. Su principal ventaja está en la eficiencia del código que produce. Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

1.8.6.2 C++

C++ es un lenguaje imperativo orientado a objetos derivado del C. En realidad 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. Lo mismo que en su ancestro, en el diseño del C++ primó sobre todo la velocidad de ejecución del código. (22)

1.8.6.3 Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. Utiliza muchas sintaxis de C y C++, pero usa un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

1.8.6.4 C Sharp

Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes.

1.8.7 Herramientas CASE

Las Herramientas CASE (Computer Aided Software Engineering o Ingeniería de Software Asistida por Ordenador) son aplicaciones informáticas utilizadas para aumentar la productividad en el desarrollo de software. Ayudan en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, entre otras.

Estas tienen como objetivos principales:

- Mejorar la productividad en actividades como el desarrollo y mantenimiento del software.

- Aumentar la calidad del software.
- Mejorar la planificación de un proyecto.
- Automatizar el desarrollo del software, la documentación, la generación de código, las pruebas de errores y la gestión del proyecto.
- Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
- Gestión global en todas las fases de desarrollo de software con una misma herramienta.
- Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

1.8.7.1 Visual Paradim

Visual Paradigm 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. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Presenta licencia gratuita y comercial. Es fácil de instalar, actualizar y compatible entre ediciones. (23)

1.8.7.2 Rational Software Architect (RSA)

Rational Software Architect proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad más rápidamente. Rational Software Architect es una herramienta de desarrollo y diseño integrada que fortalece el desarrollo dirigido por modelos con UML para la creación de servicios y aplicaciones con arquitecturas sólidas. Proporciona un soporte de desarrollo y diseño integrado para el desarrollo dirigido por el modelo con UML. Con Rational Software Architect, puede unificar todos los aspectos del desarrollo y el diseño de software. (24) Desarrolla aplicaciones de forma más productiva que:

- Utilice lo último de la tecnología de lenguajes de modelado.
- Refuerce la plataforma de modelado ampliable y abierto.

- Simplifique su solución de herramienta de desarrollo y diseño.

Puede ser utilizado en los sistemas operativos: GNU-Linux y Windows.

1.8.7.3 Rational Rose

Rational Rose es una herramienta de producción y comercialización establecida por Rational Software Corporation. Rose es un instrumento operativo conjunto que utiliza el Lenguaje Unificado (UML) como medio para facilitar la captura de dominio de la semántica, la arquitectura y el diseño. Esta herramienta no es gratuita, admite la integración con otras herramientas de desarrollo (IDEs).

Puede ser usado en sistemas operativos como:

- Windows 2000 Professional, Service Pack 4
- Windows XP Professional, Service Pack 2
- Windows 2000 and 2003 Server and Advanced Server, Service Pack 3 and 4
- Windows Vista
- Linux

1.8.8 Lenguaje de Modelado Unificado (UML)

UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Ofrece un estándar para describir un plano del sistema, incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Es un lenguaje de modelado para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software, pero no especifica en sí mismo qué metodología o proceso usar.

1.8.9 Metodologías de Desarrollo de Software

Se entiende por metodología de desarrollo una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software. (26) La finalidad de una metodología de desarrollo es garantizar la eficacia (cumplir los requisitos iniciales) y la eficiencia (minimizar las pérdidas de tiempo) en el proceso de generación de software.

En los últimos tiempos la cantidad y variedad de los procesos de desarrollo ha aumentado de forma impresionante, sobre todo teniendo en cuenta el tiempo que estuvo en vigor como ley única el famoso desarrollo en cascada. Se podría decir que en estos últimos años se han desarrollado dos corrientes en lo referente a los procesos de desarrollo, los llamados métodos pesados y los métodos ligeros. La diferencia fundamental entre ambos es que mientras los métodos pesados intentan conseguir el objetivo común por medio de orden y documentación, los métodos ligeros (también llamados métodos ágiles) tratan de mejorar la calidad del software por medio de una comunicación directa e inmediata entre las personas que intervienen en el proceso. (27)

1.8.9.1 Proceso Unificado de Racional (RUP)

RUP es uno de los procesos más generales de los existentes actualmente, ya que en realidad está pensado para adaptarse a cualquier proyecto, y no tan solo de software.

Un proyecto realizado siguiendo RUP se divide en cuatro fases:

- Inicio (puesta en marcha)
- Elaboración (definición, análisis, diseño)
- Construcción (implementación)
- Transición (fin del proyecto y puesta en producción)

En cada fase se ejecutarán una o varias iteraciones (de tamaño variable según el proyecto) y dentro de cada una de ellas seguirá un modelo de cascada para los flujos de trabajo que requieren las nuevas actividades anteriormente citadas.

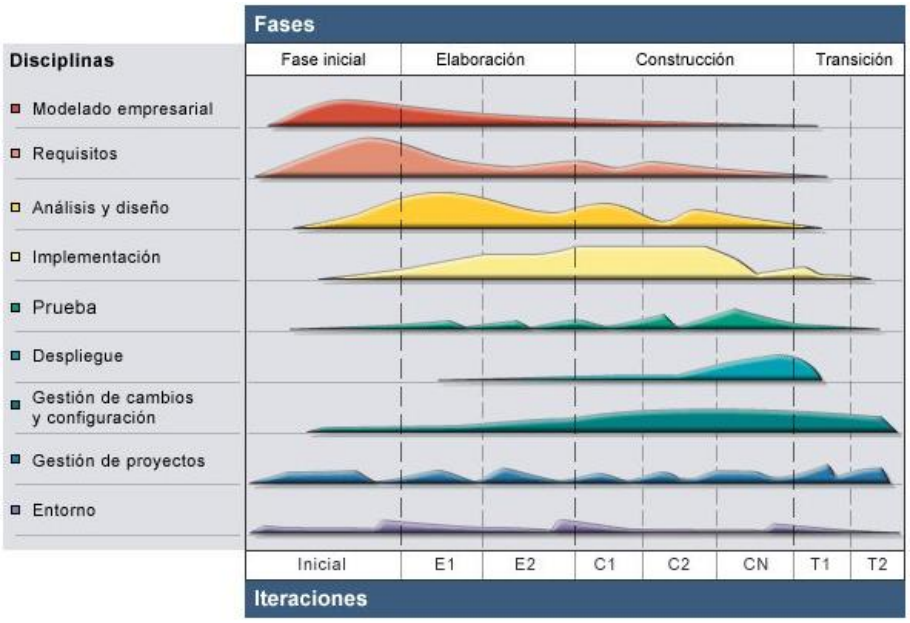

Fig. 4 Vista general de RUP.

RUP define nueve flujos de trabajo a realizar en cada fase del proyecto. El proceso define una serie de roles que se distribuyen entre los miembros del proyecto y que definen las tareas de cada uno y el resultado (artefactos) que se espera de ellos.

Flujos de trabajo:

- Modelado del negocio.
- Requisitos.
- Análisis y diseño.
- Implementación.
- Prueba.
- Despliegue.
- Gestión de cambios y configuración.
- Gestión de proyectos.
- Gestión del entorno.

RUP se basa en casos de uso para describir lo que se espera del software y está muy orientado a la arquitectura del sistema, basándose en UML como herramienta principal. Es un proceso muy general y muy grande, por lo que antes de usarlo habrá que adaptarlo a las características de la empresa. Por suerte ya hay muchos procesos descritos en internet que son versiones reducidas del RUP.

1.8.9.2 Programación Extrema (XP)

Mientras que el RUP intenta reducir la complejidad del software por medio de estructura y la preparación de las tareas pendientes en función de los objetivos de la fase y actividad actual, XP, como toda metodología ágil, lo intenta por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción. XP intenta minimizar el riesgo de fallo del proceso por medio de la disposición permanente de un representante competente del cliente a disposición del equipo de desarrollo. Este representante debería estar en condiciones de contestar rápida y correctamente a cualquier pregunta del equipo de desarrollo de forma que no se retrase la toma de decisiones, de ahí lo de competente. XP define UserStories como base del software a desarrollar. Estas historias las escribe el cliente y describen escenarios sobre el funcionamiento del software, que no solo se limitan a la GUI si no también pueden describir el modelo, dominio, etc. A partir de las UserStories y de la arquitectura perseguida se crea un plan de entregables entre el equipo de desarrollo y el cliente. (27)

Para cada entregable se discutirán los objetivos de la misma con el representante del cliente y se definirán las iteraciones necesarias para cumplir con los objetivos del entregable. El resultado de cada iteración es un programa que se transmite al cliente para que lo juzgue. En base a su opinión se definen las siguientes iteraciones del proyecto y si el cliente no está contento se adaptará el plan de entregables e iteraciones hasta que el cliente dé su aprobación y el software esté a su gusto. (27)

En XP se programará solo la funcionalidad que es requerida para el entregable actual. Es decir, una gran flexibilidad y capacidad de configuración soló será implementada cuando sea necesaria para cumplir los requerimientos del entregable. Se sigue un diseño evolutivo con la siguiente premisa: conseguir la funcionalidad deseada de la forma más sencilla posible. De ahí una variación educada del famoso KISS (mantén las cosas tan sencillas como sea posible). Este diseño evolutivo hace que no se le dé apenas importancia al análisis como fase independiente, puesto que se trabaja exclusivamente en función de las necesidades del momento. (27)

1.8.9.3 Desarrollo Basado en Funcionalidades (FDD)

FDD es un proceso diseñado por Peter Coad, Erich Lefebvre y Jeff De Luca y se podría considerar a medio camino entre RUP y XP.

FDD está pensado para proyectos con tiempo de desarrollo relativamente cortos. Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorizar. Las iteraciones se deciden en base a funcionalidades, que son pequeñas partes del software con significado para el cliente. (27)

Un proyecto que sigue FDD se divide en 5 fases:

- Desarrollo de un modelo general.
- Construcción de la lista de funcionalidades.
- Plan de entregables en base a las funcionalidades a implementar.
- Diseñar en base a las funcionalidades.
- Implementar en base a las funcionalidades.

Las primeras tres fases ocupan gran parte del tiempo en las primeras iteraciones, siendo las dos últimas las que absorben la mayor parte del tiempo según va avanzando el proyecto, limitándose las primeras a un proceso de refinamiento. (27)

1.8.10 Selección de tecnologías y herramientas.

Respondiendo a las necesidades que se presentaron en el desarrollo del manejador en cuando a trabajo con hilos de ejecución, comunicación por puerto serie, contenedores, punteros a funciones y la necesidad de obtener un producto libre y multiplataforma, se decidió utilizar las bibliotecas Boost y Asio. Fueron seleccionadas debido a que constituyen herramientas potentes, que poseen soluciones, que ya han sido debidamente avaladas y probadas, por ejemplo: boost/tokenizer, boost/bind, asio::thread y asio::serial_port.

Para los entornos de desarrollo los requisitos de selección se centraron en un IDE que permitiera la programación en el lenguaje C++, que cuente con herramientas integradas para la gestión de la configuración tanto del código fuente como de la documentación, con un buen completamiento de código y

que brinde facilidades en la vinculación de bibliotecas y en la configuración del compilador. El desarrollo se realizará tanto en GNU-Linux como en Windows, por lo que se hace necesario que permita el trabajo en ambos sistemas operativos. Luego de una exhaustiva búsqueda e investigación se escogieron los IDE de desarrollo Eclipse y Visual Studio.

Para el desarrollo de los manejadores se necesita un lenguaje de programación que sea potente, rápido y que posea un amplio soporte en varios sistemas operativos. El lenguaje seleccionado fue el C++, algunos de los motivos de su selección fueron: su alta velocidad de ejecución y la buena calidad de su compilador libre para sistemas operativos GNU: g++.

Como herramienta CASE se utilizó el Visual Paradim, las principales características que se necesitaban y que en específico cumple esta herramienta son que: es multiplataforma y modela todos flujos de trabajo de RUP.

Como metodología de desarrollo de software fue escogido RUP por caracterizarse como proceso iterativo e incremental, centrada en la arquitectura y dirigida por los casos de uso, que sirve para guiar el proceso de desarrollo de software en el proyecto que se lleva a cabo. Además, con esta metodología se genera una serie de artefactos que facilita que se elabore una documentación detallada del proyecto. Sin embargo XP es ligera, genera poca documentación y requiere de una presencia constante del cliente en el proceso de desarrollo y en cuanto a FDD aunque su documentación es aceptable, necesita rigurosamente de personal especializado en el tema, además de pequeñas debilidades potenciales para una buena obtención de requisitos del sistema.

CAPÍTULO 2 : CARACTERÍSTICAS DEL SISTEMA

2.1 PROTOCOLO DF1

2.1.1 Especificaciones del protocolo

Un protocolo es un conjunto de reglas de programación para la interpretación de las señales transmitidas a través de una conexión física. Un protocolo como el DF1:

- Lleva un mensaje libre de errores desde un extremo del enlace a otro. No tiene preocupación por el contenido, la función o el objetivo final del mensaje. Ejemplo, con el protocolo DF1 Full-dúplex, se logra esto adjuntando un chequeo de caracteres BCC o CRC al final de cada comando y respuesta. El dispositivo recibe el comando o respuesta y verifica el BCC o CRC y retorna un ACK si el BCC o CRC es correcto, o un NAK si son incorrectos.
- Indica fallo con un código de error. Internamente el protocolo de enlace debe delimitar mensajes, detectar y señalar errores, hacer reintentos ante errores y controlar flujo de mensajes.
- Utiliza dos capas de software para posibilitar las comunicaciones, estas son: la capa de enlace de datos (data link layer) y la capa de aplicación (aplication layer).

2.1.1.1 Protocolo DF1 Half-duplex

Es una versión multi-niveles del protocolo para un maestro y uno o más esclavos. Se pueden tener de 2 a 255 nodos conectados simultáneamente en un único enlace, este enlace opera con todos los nodos interconectados a través de módems Half-duplex.

Para implementar el protocolo Half-dúplex, se utilizan los siguientes caracteres de comunicación:

- 8 bits por carácter.
- no paridad.
- 1 bit de parada.

Cuando se usa este protocolo, el ambiente de comunicación utilizado es un enlace multipuntos con todos los nodos a través de una interface de módems Half-dúplex. A menos que haya un solo esclavo conectado directamente a un maestro, debe utilizar un módem.

En este tipo de comunicación se designa un nodo como maestro para controlar cual nodo tiene acceso al enlace. Todos los otros nodos son esclavos y tienen que esperar por el permiso del maestro antes de transmitir. Cada nodo esclavo tiene un único número de nodo entre 0 y 254 (decimal).

El maestro puede enviar y recibir mensajes hacia y desde cada nodo en el enlace multipuntos y hacia y desde cada nodo de la red de enlaces conectada al enlace multipuntos.

Varios maestros no están permitidos, excepto cuando uno actúa como respaldo del otro y no se comunica a menos que el primario se apague.

2.1.1.2 Protocolo DF1 Full-duplex

El protocolo Full-dúplex se usa:

- Sobre un enlace punto a punto que permite dos formas de transmisión simultánea.
- Sobre un enlace multipuntos donde módulos de la interfaz son capaces de transmisiones arbitrarias en el enlace.
- Para aplicaciones de alto rendimiento en las que es necesario obtener el más alto rendimiento posible del medio de comunicación.

2.1.1.3 Símbolos de transmisión

Ambos, DF1 Half-dúplex y DF1 Full-dúplex son orientados a caracteres. Estos usan los caracteres de control ASCII que se muestran en las tablas siguientes, extendidos con ocho bits, adicionando cero en el bit 7.

Tabla 2 Caracteres DF1 Half-Duplex.

Abreviatura	Valor hexadecimal
STX	02
SOH	01
ETX	03
EOT	04
ENQ	05
ACK	06
DLE	10
NAK	0F

Tabla 3 Caracteres DF1 Full-Duplex.

Abreviatura	Valor hexadecimal
STX	02
ETX	03
EOT	04
ENQ	05
ACK	06
DLE	10
NAK	0F

Un símbolo es una secuencia de uno o más bytes que tienen un significado específico para el enlace del protocolo. Los caracteres que componen un símbolo deben ser enviados uno detrás del otro sin otros caracteres entre ellos. El protocolo DF1 combina los caracteres listados en la tabla anterior en símbolos de control y de datos:

- Los símbolos de control son símbolos fijados requeridos por el protocolo DF1 para leer un mensaje.
- Los símbolos de datos son símbolos variables, de contenido de datos de la aplicación para un determinado mensaje.

Tabla 4 Símbolos de transmisión DF1 Half-Duplex.

Símbolo	Tipo de símbolo	Descripción
DLE SOH	Control	Indica el comienzo de un mensaje del maestro.
DLE STX	Control	Separa los datos de la cabecera de comunicación.
DLE ETX BCC/CRC	Control	Indica la terminación del mensaje.
DLE ACK	Control	Respuesta que indica que el mensaje se recibió satisfactoriamente.
DLE NAK	Control	Mensaje sólo enviado por el maestro. Esto provoca que los esclavos cancelen los mensajes que estén listos para transmitir y devuelvan el mensaje con un código de error al que lo inició.
DLE ENQ	Control	Se envía sólo por el maestro e indica el comienzo de un mensaje encuesta.
DLE EOT BCC	Control	Respuesta que dan los esclavos a los mensajes encuesta enviados por el maestro cuando no tienen mensajes que transmitir.
STN	Datos	Indica el número del nodo esclavo en un vínculo Half-dúplex.
APP DATA	Datos	Caracteres simples que tienen valores 00-0F y 11-FF. Incluye datos de la capa de aplicación, programas de usuario y rutinas de aplicaciones comunes. Un dato 10^{16} es enviado como un 10 10(DLE DLE).
DLE DLE	Datos	Representa el valor del dato o el valor de STN de 10^{16}. Ver APP DATA.

Tabla 5 Símbolos de transmisión DF1 Full-Duplex.

Símbolo	Tipo de símbolo	Descripción
DLE STX	Control	Indica el comienzo de un mensaje.
DLE ETX BCC/CRC	Control	Indica la terminación de un mensaje.
DLE ACK	Control	Respuesta que indica que el mensaje se recibió satisfactoriamente.
DLE NAK	Control	Respuesta que indica que el mensaje no se recibió satisfactoriamente.
DLE ENQ	Control	Respuesta que se le envía al receptor en espera de símbolo de respuesta.
APP DATA	Datos	Caracteres simples que tienen valores entre 00-0F y 11-FF. Incluye datos de la capa de aplicación, programas de usuario y rutinas de aplicaciones comunes. Un dato 10^{16} es enviado como un 10 10 (DLE DLE).
DLE DLE	Datos	Representa el valor del dato de 10^{16}.

2.1.2 Transmisión de mensajes DF1 Half-duplex

En teoría a la capa de enlace de dato del protocolo no le concierne el contenido o forma del mensaje que es transferido. Sin embargo, Half-dúplex pone las siguientes restricciones en el mensaje, necesarias para la transmisión:

- Tamaño mínimo de la capa de enlace de datos es 6 bytes.
- Tamaño máximo de la capa de enlace de datos, depende de la capa de comandos de la aplicación.
- Algunas implementaciones requieren que el primer byte del mensaje concuerde con la dirección del nodo, o de lo contrario el receptor ignora el mensaje que no contiene la dirección correcta. Este filtrado puede ser realizado de forma opcional.
- Como parte del algoritmo detección de mensaje duplicado, el receptor chequea la fuente (SRC), comando (CMD) y el campo de transacción (TNS) de cada mensaje. Debe ser diferente al menos

uno de estos bytes entre un mensaje y el anterior. De otra forma, el mensaje es clasificado como retransmisión del mensaje anterior.

Las siguientes figuras muestran los eventos que ocurren en varias interfaces. Los bytes de la capa de enlace de datos están representados por “xxxx” y los datos corruptos son representados por “???”. Estas figuras muestran lo que ocurre en la capa de datos cuando un mensaje es enviado; no se muestran los pasos que realiza el receptor cuando envía una respuesta.

2.1.2.1 Transferencia de un mensaje normal

- Los bytes de datos son enviados desde la fuente al maestro y este los transmite al esclavo.
- La pila envía un mensaje comunicando que no está llena.
- Los datos son almacenados en la pila y se envía un DEL ACK al maestro.
- El maestro le comunica a la fuente que el mensaje fue entregado.

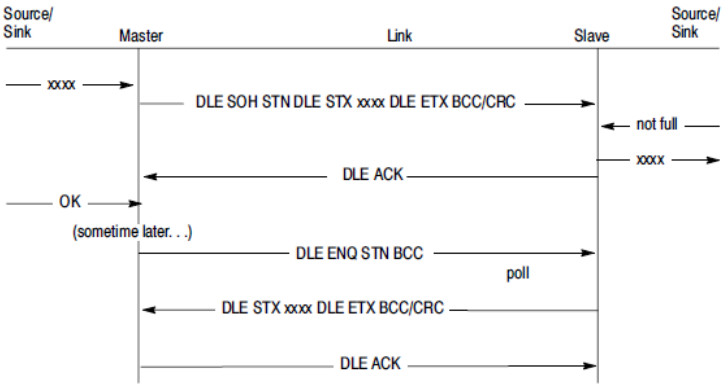

Fig. 5 Transferencia de un mensaje normal.

2.1.2.2 Transferencia de un mensaje con BCC/CRC inválido

- El maestro envía el mensaje con el BCC/CRC inválido.
- El maestro no recibe el ACK y expira el tiempo de espera.

- El maestro retransmite el mensaje con un BCC/CRC válido y se procede con la transferencia de un mensaje normal.

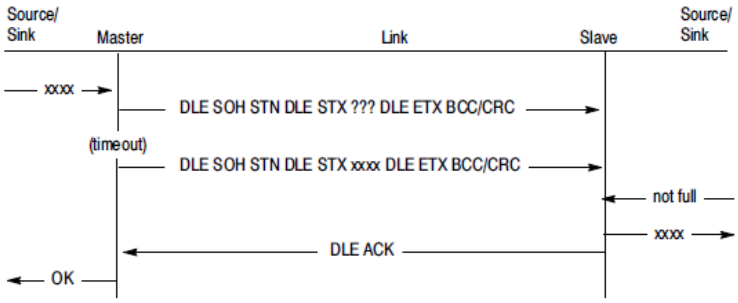

Fig. 6 Transferencia de un mensaje con BCC/CRC inválido.

2.1.2.3 Mensaje con ACK incorrecto

- El maestro envía el mensaje.
- El esclavo acepta el mensaje y transmite un DLE ACK, pero con el ACK incorrecto.
- El maestro retransmite y se procede con la transferencia de un mensaje normal.
- Si la detección de mensajes duplicados está activa, el mensaje se descarta.

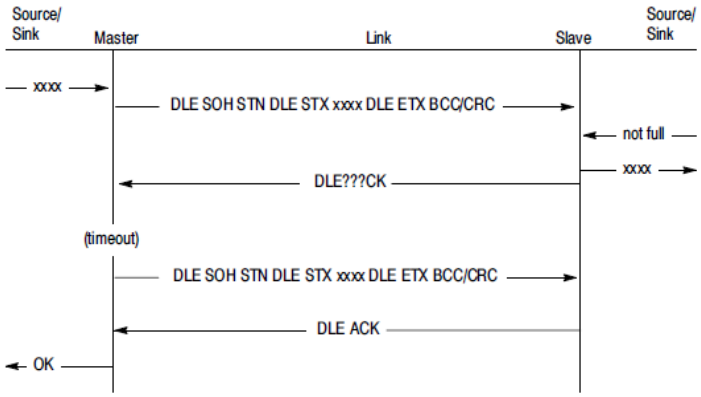

Fig. 7 Mensaje con ACK incorrecto.

2.1.2.4 Encuesta con mensaje no disponible

- El maestro envía un mensaje de encuesta al esclavo.
- El esclavo no tiene mensajes de respuesta.
- El esclavo devuelve un DLE EOT.

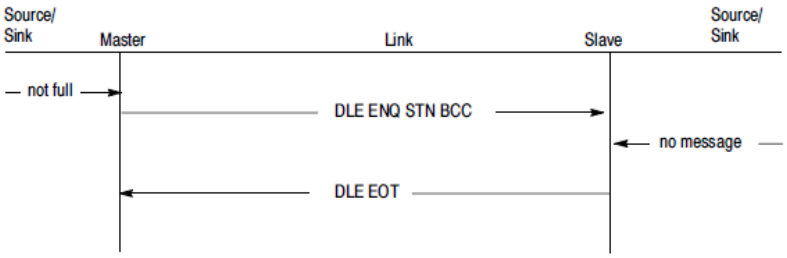

Fig. 8 Encuesta con mensaje no disponible.

2.1.2.5 Encuesta con un mensaje devuelto

- El maestro envía un mensaje encuesta al esclavo y este le devuelve un mensaje con datos defectuosos.
- El esclavo no recibe un DLE ACK en su primer intento por lo que retransmite el mensaje.
- El maestro responde un DLE ACK.

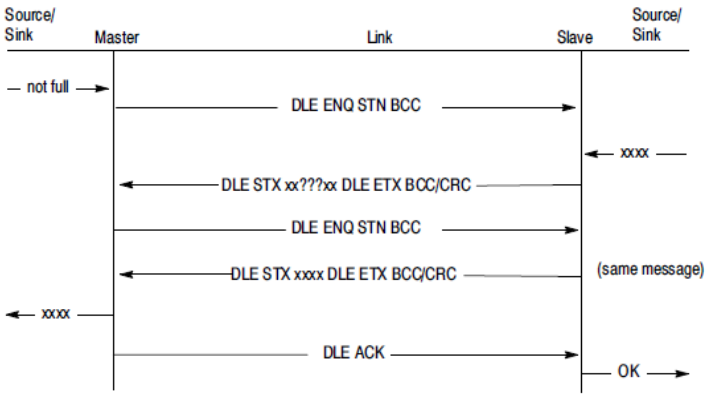

Fig. 9 Encuesta con un mensaje devuelto.

2.1.2.6 Mensaje duplicado

- El esclavo no recibe un DLE ACK del maestro después de que el mismo recibiera un mensaje del esclavo.
- El maestro envía un mensaje encuesta al esclavo y este le envía el mismo mensaje.
- El maestro responde un DLE ACK pero descarta el mensaje duplicado.

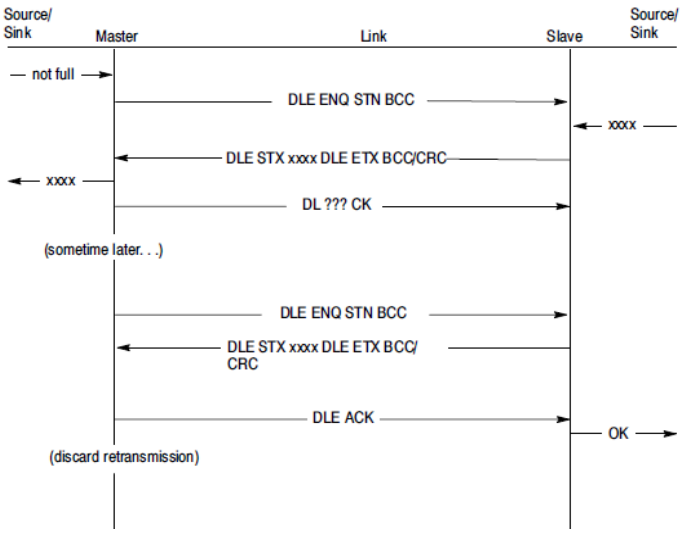

Fig. 10 Mensaje duplicado.

2.1.3 Transmisión de mensajes DF1 Full-duplex

Con el protocolo Full-dúplex, el enlace usa dos circuitos físicos para dos formas de transmisión simultánea de mensajes (paquetes de mensajes de comando y respuesta). Estos dos circuitos físicos proveen comunicación en cuatro caminos lógicos:

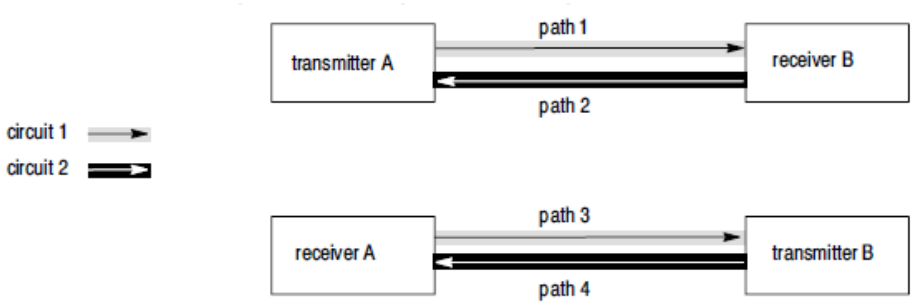

Fig. 11 Envío de datos de manera simultánea en dos sentidos.

- En el primer circuito, el transmisor A envía mensajes al receptor B (path 1) y el receptor A envía un símbolo de control de respuesta (DLE ACK, del NAK) al transmisor B (path 3).
- En el segundo circuito, el transmisor B envía mensajes al receptor A (path 4) y el receptor B envía un símbolo de control (DLE ACK, DLE NAK) al transmisor A (path 2).
- Todos los mensajes y símbolos en el circuito viajan en la misma dirección (nodo A al nodo B) y todos los mensajes y símbolos en el segundo circuito viajan en la dirección opuesta (B a A).

Teóricamente a la capa de enlace de dato del protocolo no le concierne el contenido o forma del mensaje que es transferido. Sin embargo, Full-dúplex pone las siguientes restricciones en el mensaje, necesarias para la transmisión:

- Tamaño mínimo de la capa de enlace de datos es 6 bytes.
- Tamaño máximo de la capa de enlace de datos, depende de la capa de comandos de la aplicación.
- Algunas implementaciones requieren que el primer byte del mensaje concuerde con la dirección del nodo, o de lo contrario el receptor ignora el mensaje que no contiene la dirección correcta.
- Como parte del algoritmo de detección de mensaje duplicado, el receptor compara el segundo, tercero, quinto y sexto bytes de la capa de enlace con los mismos bytes en el mensaje anterior. Si no hay diferencia entre este grupo de bytes, el mensaje es clasificado como una retransmisión del mensaje anterior.

Las siguientes figuras muestran los eventos que ocurren en varias interfaces. Los bytes de la capa de enlace de datos están representados por “xxxx” y los datos corruptos son representados por “???”.

2.1.3.1 Transferencia de un mensaje normal

- El transmisor envía los datos al receptor.
- La pila envía un mensaje comunicando que no está llena.
- El receptor envía los datos a la pila y envía un DEL ACK al transmisor.
- El transmisor le comunica a la fuente que el mensaje fue entregado.

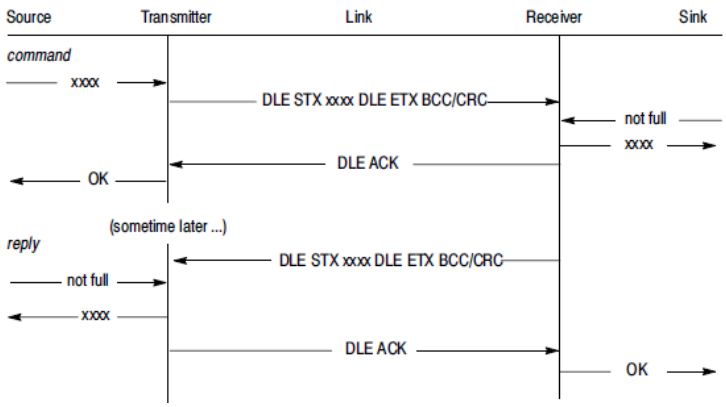

Fig. 12 Transferencia de un mensaje normal.

2.1.3.2 Transferencia de un mensaje con NAK

- El transmisor envía los datos corruptos al receptor y este le responde con un DLE NAK.
- El transmisor retransmite los datos y la transmisión es un éxito.
- El transmisor le comunica a la fuente que el mensaje fue entregado.
- El receptor envía un DLE ACK al transmisor.

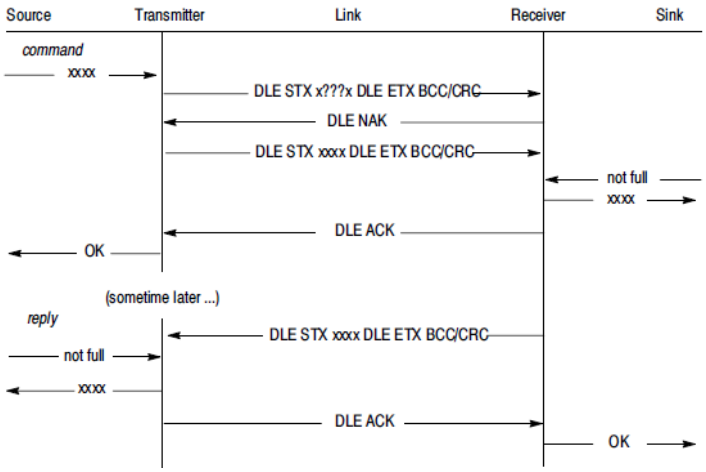

Fig. 13 Transferencia de un mensaje con NAK.

2.1.3.3 Transferencia de un mensaje con time out y ENQ

- El receptor recibe la transmisión pero el DLE ACK que envía es corrupto.
- El transmisor da un time out esperando por el DLE ACK y envía un DLE ENQ.
- El receptor envía un DLE ACK al transmisor.

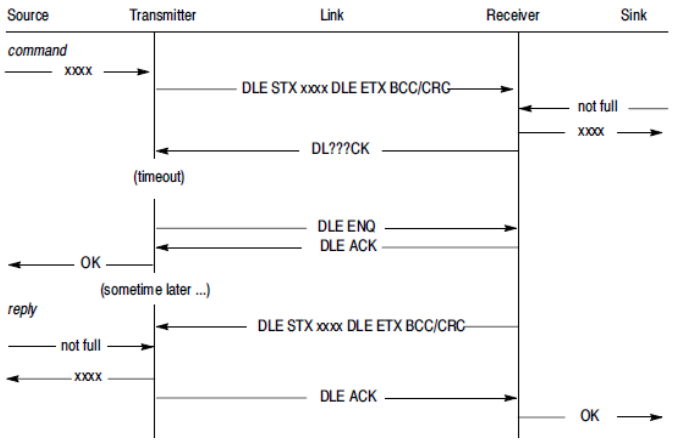

Fig. 14 Transferencia de un mensaje con time out y ENQ.

2.1.3.4 Transferencia de mensaje con retransmisión

- El ruido modifica DLE ACK y al mismo tiempo se producen caracteres inválidos en el receptor.
- Debido a los caracteres no válidos, el receptor cambia su última variable de respuesta por un DLE NAK.
- Desde que el DLE ACK fue modificado, el transmisor envía un DLE ENQ y el receptor responde un DLE NAK.
- El transmisor retransmite el mensaje y el receptor le envía un ACK.
- El receptor descarta los mensajes duplicados (si la detección de mensajes duplicados está habilitada en el módulo).

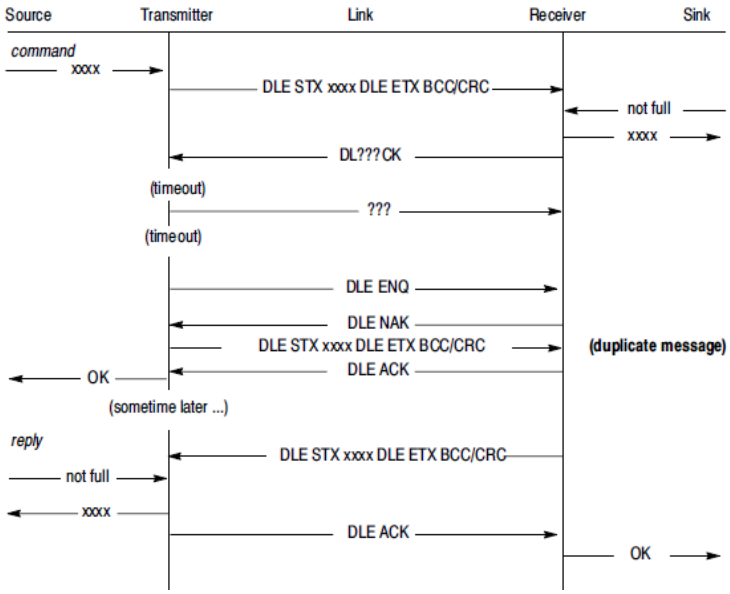

Fig. 15 Transferencia de mensaje con retransmisión.

En esta transferencia el receptor no tiene forma de saber que el DLE ACK que envió al transmisor fue modificado. Si el tiempo de envío del ACK del transmisor es lo suficientemente largo, es posible que la respuesta (ej. DLE STX “xxxx” DLE ETX BCC/CRC) regrese antes de que el transmisor envíe el DLE ENQ. Por tanto la respuesta viene antes que el DLE ACK sea recibido por el transmisor.

2.1.3.5 Transferencia de mensaje con pila llena

- El transmisor envía un mensaje al receptor pero este tiene la pila llena y le responde con un DLE NAK.
- El transmisor retransmite el mensaje y la pila no está llena, el receptor le envía un DLE ACK.
- El receptor descarta los mensajes duplicados (si la detección de mensajes duplicados está habilitada en el módulo).

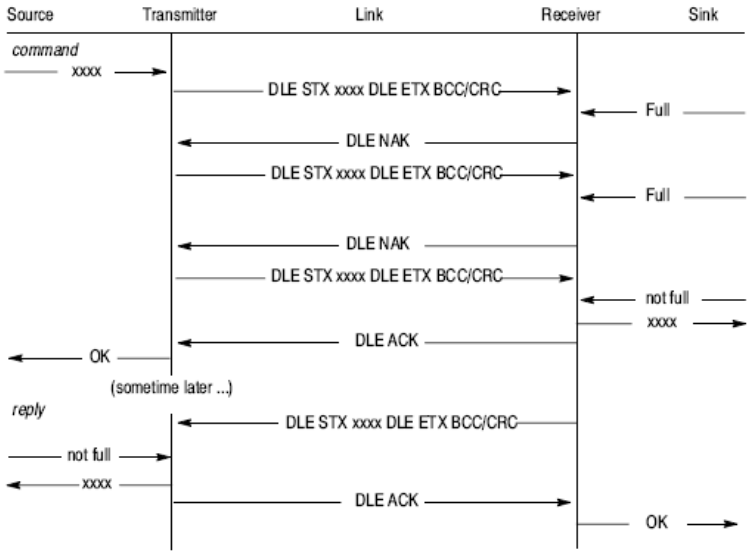

Fig. 16 Transferencia de mensaje con pila llena.

2.1.3.6 Transferencia de mensaje con NAK como respuesta

- El mensaje es transmitido con éxito por la red.
- La respuesta es corrupta y el transmisor responde con un DLE NAK.
- La respuesta es enviada de nuevo con éxito.

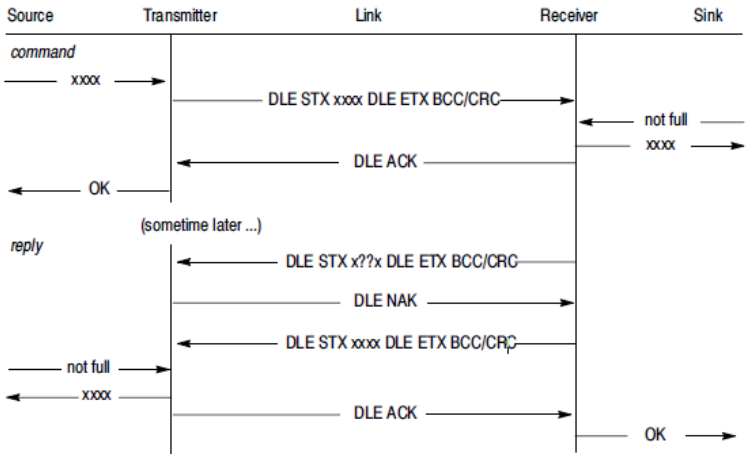

Fig. 17 Transferencia de mensaje con NAK como respuesta.

2.1.3.7 Transferencia de un mensaje con un ENQ como respuesta

- El mensaje es transmitido con éxito por la red.
- El receptor envía una respuesta por la red pero el transmisor responde con un DLE NAK corrupto.
- El receptor da time out esperando por DLE ACK y envía un DLE ENQ.
- El transmisor envía de regreso un DLE ACK.

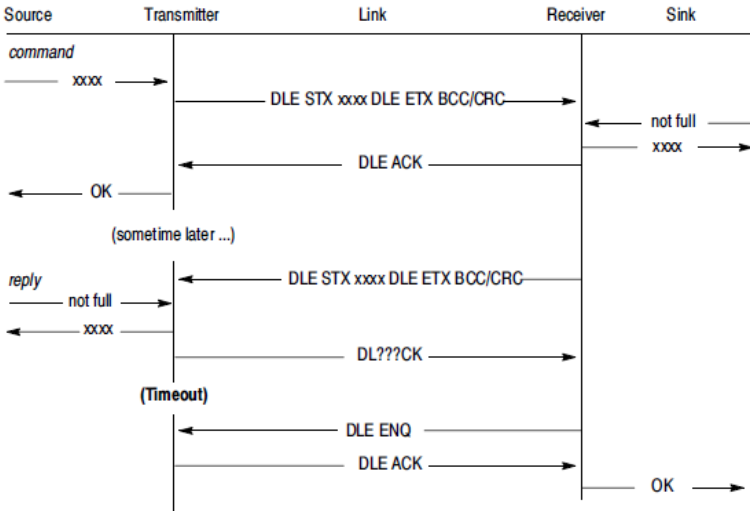

Fig. 18 Transferencia de mensaje con ENQ como respuesta.

2.1.4 Estructura de la capa de enlace de datos de los mensajes

En la capa de enlace de datos del mensaje:

- El protocolo Half-dúplex usa tres tipos de transmisiones.
- El protocolo Full-dúplex implementa sus campos de mensajes en diferentes capas de red.
- Al final de cada marco de encuesta y de mensaje, hay un campo BCC que es un byte o un campo CRC de dos bytes.

2.1.4.1 Estructura de mensajes del DF1 Half-Duplex

Usa tres tipos de transmisiones:

- De encuesta.
- De mensaje del maestro.
- De mensaje del esclavo.

El nodo maestro transmite ambos, mensaje de encuesta y mensaje de maestro y el nodo esclavo transmite el mensaje de esclavo. El mensaje de esclavo tiene el mismo formato del mensaje Full-dúplex. El mensaje del maestro es el mismo del esclavo excepto por el prefijo DLE SOH y la dirección para especificar la estación del esclavo (STN).

Nota: Incluso si se usa un CRC, cuando se envía un mensaje de encuesta se usa en este un solo byte de BCC.

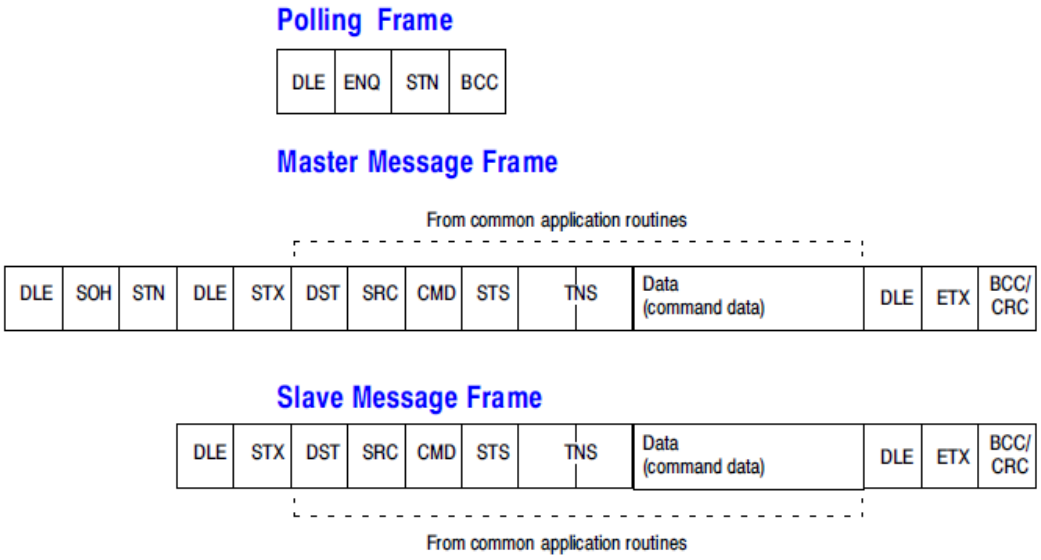

Fig. 19 Estructura del mensaje Half-Duplex.

El campo BCC contiene el 2do complemento de la suma de los 8 bit (módulo de la suma aritmética 256) de la dirección de la estación del esclavo (STN) y todos los bytes de datos en el mensaje. Para el mensaje de encuesta el BCC es solo el 2do complemento de STN. El BCC no incluye ningún otro símbolo del mensaje de encuesta o símbolo de respuesta.

2.1.4.2 Estructura de mensajes del DF1 Full-duplex

Full-dúplex implementa diferentes mensajes en dependencia de la capa de red. Esta figura muestra el formato del mensaje full-duplex y los elementos que conforman cada una de sus capas:

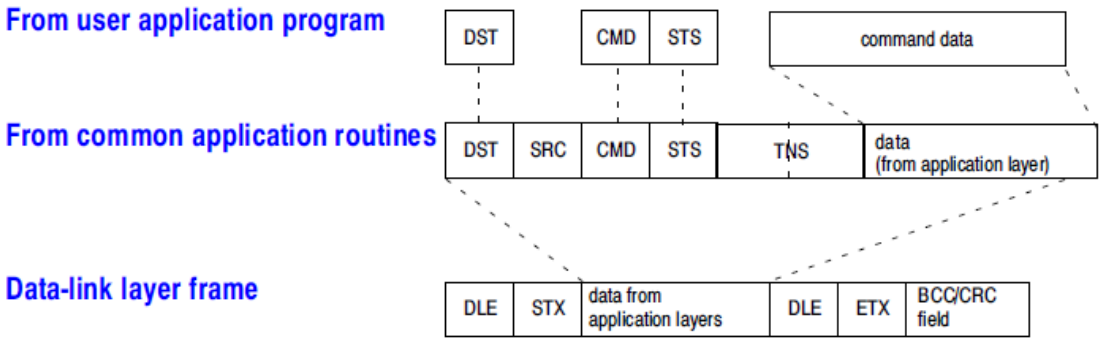

Fig. 20 Estructura del mensaje Full-Duplex.

El campo BCC/CRC contiene el 2do complemento de la suma de 8 bit (módulo de la suma aritmética 256) de toda la capa de datos de la aplicación entre el DLE STX y el DLE ETX BCC. Sin incluir ningún símbolo de respuesta. Se puede usar el campo BCC/CRC para uno de los siguientes tipos de chequeos de error:

- Chequeo de bloques de caracteres (BCC).
- Chequeo de redundancia cíclica de 16 bit (CRC-16).

2.2 DIRECCIONAMIENTO DEL PLC-5 ALLEN-BRADLEY

Para el desarrollo del manejador del protocolo DF1, el dispositivo que PDVSA pone a disposición para realizar las pruebas es un PLC5 Allen-Bradley. Esto se debe a que este dispositivo es uno de los que más abunda en la infraestructura de la Industria Petrolera Venezolana que se comunican usando el protocolo DF1 y que el direccionamiento de la gran mayoría de los otros dispositivos que utilizan este protocolo es muy similar a la de este PLC. Por lo que es interés de PDVSA que la puesta a punto del manejador se realice específicamente con este dispositivo.

La memoria en el PLC5 se divide en la memoria de programas y la memoria de las variables. La memoria de programas contiene las instrucciones que deben ser ejecutadas y generalmente no puede ser cambiada, mientras el PLC se encuentra en modo de ejecución. La memoria de las variables cambia constantemente mientras el PLC ejecuta el programa interno, reflejando por un lado los valores obtenidos de las entradas físicas del PLC y por otro los cálculos intermedios y variables de estado del PLC.

En el PLC5, la memoria es organizada en bloques de hasta 1000 elementos consecutivos que se denominan “files” o ficheros. Existen ocho ficheros de datos definidos por defecto (ver Fig. 21) pero se pueden crear ficheros adicionales si son necesarios por el programa del PLC.

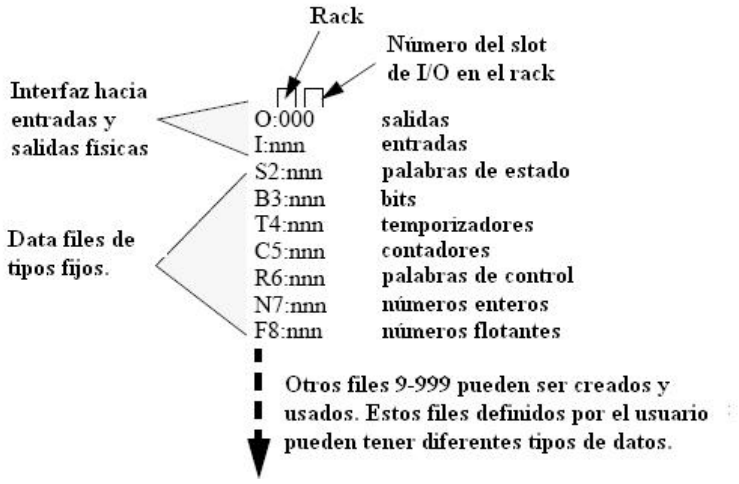

Fig. 21 Organización de memoria de variables del PLC-5.

De esta manera, cada variable en el PLC5 se referencia como mínimo por tres elementos. Estos elementos son: el tipo del fichero, que se expresa por la primera letra de la dirección, el número del fichero, que es el número que sigue a esa primera letra y el desplazamiento de la variable dentro de fichero, que se expresa por un número separado de la primera parte de la dirección por el carácter de dos puntos ":". Ejemplos de direcciones posibles son N27:5 o F41:12. Si el tipo del fichero es un tipo estructurado como los temporizadores o contadores, se pueden añadir como sufijo, el nombre del elemento de la estructura para referenciar la variable.

En la siguiente tabla se muestran algunos de los posibles tipos de datos de los ficheros. Se omiten deliberadamente otros tipos que no tienen interés para el funcionamiento del SCADA. Debe subrayarse que no todos estos tipos están presentes en todos los modelos.

Tabla 6 Tipos de datos principales de las variables del PLC-5.

Prefijo	Descripción
O	Salidas físicas del PLC.
I	Entradas físicas del PLC.
S	Palabras de estado del PLC de 16 bits.
B	Palabras de 16 bits que pueden almacenar bits independientes.
T	Temporizadores.
C	Contadores.
R	Palabras de control.
N	Números enteros con signo de 16 bits.
F	Números flotantes de 32 bits.
A	ASCII.
D	Números enteros en formato BCD.
L	Números enteros con signo de 32 bits.
MG	Mensajes.
ST	Cadenas de caracteres.
PD	PID.

2.3CARACTERÍSTICAS DEL ACCESO A LOS DATOS

Para acceder a los datos de las estructuras de tipo: Timer (T), Counter (C), SFC Status (SC) y PIDControl (PD), que son las que están soportadas en el manejador, es necesario especificar el campo de la estructura a que se desea acceder.

Para acceder al valor acumulado de la una estructura de tipo temporizador (timer) como T88:5 se debe usar la dirección “T88:5.ACC”. Los campos a los que tenemos acceso en esta estructura son: EN, TT, DN, PRE, ACC.

El acumulador y el valor de preselección de un contador se acceden mediante los sufijos .ACC y .PRE respectivamente. De esta manera si se desea asociar una variable al valor de preselección del contador

C12:4 se debe usar la dirección “C12:4.PRE”. Los campos de esta estructura son: CU, CD, DN, OV, UN, PRE, ACC.

En el caso del SFC Status sus campos son: SA, FS, LS, OV, ER, DN, PRE, TIM y el acceso a estos campos es de la misma forma que las demás estructuras.

La cantidad de palabras (2 bytes) de los tipos de datos es la siguiente:

Tabla 7 Tamaño de direcciones del PLC-5.

Tipos de datos	Tamaño en bytes
I/O Image(I/O)	1 word(2 bytes)
Status File(S)	1 word(2 bytes)
Binary(B)	1 word(2 bytes)
Floating Point(F)	2 word(4 bytes)
Integer(N)	1 word(2 bytes)
Timer(T)	3 word(6 bytes)
Counter(C)	3 word(6 bytes)
SFC Status(SC)	3 word(6 bytes)
PID Control(PD)	82 word(164 bytes)

2.4DECISIONES DE DISEÑO

En la implementación de este manejador se tomaron un grupo de decisiones encaminadas a responder a las necesidades del cliente, al papel que va a jugar este manejador en el SCADA Galba y a obtener un producto de calidad, con posibilidades de continuar desarrollándolo e incrementando sus funcionalidades.

2.4.1 Comandos de lectura y escritura

El protocolo DF1 ofrece para su implementación más de cuarenta comandos para disímiles funcionalidades. Estos se agrupan en: comandos de lectura y escritura de direcciones, comandos de carga y descarga de memoria del PLC y los de configuración.

El interés del cliente se centra en que el manejador pueda realizar solo operaciones de lectura y escritura. Esto descarta en la implementación a los grupos de comandos que no sean para este propósito, reduciendo así el número de comandos candidatos a incluir en la implementación a quince.

Para lograr optimizar al máximo las capacidades de lectura del protocolo, una solución es recuperar la mayor cantidad de valores en una transacción simple. En vista de que el protocolo brinda varios comandos, con distintas posibilidades para recuperar una cantidad máxima de datos por transacción, se realizó un estudio sobre las características específicas de cada uno de estos quince comandos. Este estudio arrojó como resultado, que para lograr un uso eficiente de las capacidades del protocolo en cuanto a lectura y escritura de variables, es necesaria la implementación de los siguientes comandos:

- **Word Range Read o lectura de un rango de variables:** Lee un archivo de direcciones a partir de una dirección específica. Un caso especial de este comando es la lectura de una sola dirección. Fuera de este caso el número de direcciones a leer debe ser par. Puede leerse un máximo de 244 bytes de datos.
- **Word Range Write o escritura de un rango de variables:** Escribe un archivo de direcciones a partir de una dirección específica. Un caso especial de este comando es la escritura de una sola dirección. Fuera de este caso el número de direcciones a escribir debe ser par. Puede escribir un máximo de 240 bytes de datos.
- **Read File o lectura de archivo:** Lee los datos a partir de un archivo o bloque de direcciones. La dirección de comienzo debe apuntar a un archivo de 2 bytes. Puede leerse un máximo de 244 bytes de datos.

- **Write File o escritura de archivo:** Escribe los datos a partir de un archivo o bloque de direcciones. La dirección de comienzo debe apuntar a un archivo de direcciones de 2 bytes de tamaño cada una. Puede escribir un máximo de 239 bytes de datos.

2.4.2 Densidad de los paquetes

En virtud de que mediante el protocolo podemos obtener varias direcciones en una transacción, si necesitáramos leer en un instante de tiempo las direcciones N3:1 y N3:23 podríamos optar por hacerlo en una transacción leyendo las 23 direcciones consecutivas a partir del N3:1. Por supuesto que en este caso estamos forzando al PLC a entregarnos información no útil, como son las direcciones N3:2, N3:3,…., N3:22, lo cual también tiene un costo. En el documento "Guía de Implementación del Manejador Modbus" se introduce el concepto de densidad de un paquete o bloque de registros que en esencia refleja la proporción de datos útiles, con respecto al total, en el paquete solicitado. Una densidad muy baja en los paquetes podría tener un impacto negativo en el rendimiento, al incrementar innecesariamente la cantidad de datos a transferir y ocupar el CPU del PLC en transferencias innecesarias. Por otra parte obligar el uso de densidades muy altas puede provocar fragmentación en las solicitudes lo que también puede disminuir el rendimiento. Para mantener un equilibrio y lograr un mayor rendimiento en la lectura, se decidió incluir la densidad de los paquetes como un parámetro de la configuración del manejador. De esta forma se podrá adaptar el rendimiento del manejador a las necesidades del medio en que se utilice.

2.4.3 Bloques de direcciones

Un bloque de direcciones podrá estar formado únicamente por direcciones del mismo tipo de datos y pertenecientes al mismo fichero. Esto se debe a que con los comandos con que cuenta el protocolo sólo es posible realizar operaciones de lectura y escritura en una transacción simple, sobre conjuntos de direcciones secuenciales que sean de un tipo de dato único y además que se encuentren dentro del mismo fichero. Ejemplos correctos e incorrectos de bloques de direcciones:

- El conjunto de direcciones desde N2:0 hasta N2:122 (Correcto).
- El conjunto de direcciones desde N3:1 hasta N4:23 (Incorrecto, se encuentran en ficheros diferentes de la memoria del PLC).

Por último un bloque tendrá como capacidad máxima 244 bytes. Las direcciones más pequeñas ocupan un tamaño de 2 bytes. Ejemplos correctos e incorrectos de bloques de direcciones:

- El conjunto de direcciones desde N3:1 hasta N3:260 (Incorrecto, sobrepasa el máximo de bytes que se pueden obtener en una transacción simple).
- El conjunto de direcciones desde N3:1 hasta N3:122 (Correcto, las direcciones de tipo entero ocupan 2 bytes).

2.4.4 Implementación de versión Full-Duplex

La comunicación con el PLC5 con que cuenta PDVSA se realizará utilizando la versión Full-Duplex del protocolo. Esto se debe a que para utilizar la versión Half-Duplex es necesaria la existencia de módems que controlen la comunicación entre un dispositivo maestro y los dispositivos esclavos. En la infraestructura de PDVSA no existe este tipo de hardware para la comunicación con estos dispositivos, por lo que no es posible en este momento este tipo de comunicación.

Con el objetivo de poder extender en el futuro el desarrollo del manejador a la comunicación Half-Duplex, se diseñó una clase genérica para la creación de las tramas que reúne las características comunes de los dos tipos de tramas del protocolo y dos clases especializadas con las características específicas de las tramas para la comunicación Half-Duplex y Full-Duplex respectivamente. En el caso de las clases que controla la lógica de comunicación en la capa de mensaje, se diseñaron dos clases: una genérica para las características generales de la comunicación y otra especializada que implementa la lógica de la comunicación Full-Duplex y que utiliza la clase especializada en la creación de la trama Full-Duplex que se mencionaba anteriormente. Este diseño permite que si en algún momento es necesario incluir la comunicación Half-Duplex, solo tendría que agregarse otra clase especializada en la lógica de este tipo de comunicación y utilizar la clase que se diseño para el envío de las tramas Half-Duplex. Evitando así tener que conocer como está implementada la otra clase para la comunicación Full-Duplex y que tengan que realizarse modificaciones en el código fuente.

2.5 ESPECIFICACIÓN DE REQUERIMIENTOS

Para los manejadores que se desarrollan para el “Galba”, existen un conjunto de requerimientos, tanto funcionales como no funcionales, previamente definidos y que fueron aprobados por el cliente. Estos se encuentran especificados y descritos en el documento: “Especificación de Requerimientos de Software del Módulo de Drivers”, en su versión 1.3. El propósito de este documento es formalizar un conjunto acordado de requerimientos que conforman el desarrollo de la línea de manejadores de dispositivos según el Anexo 11 del convenio PDVSA-ALBET S.A. para el desarrollo del SCADA “Galba” y esbozar la posible interacción de estos manejadores con la capa de recolección. (6) De los requerimientos que se especifican en el documento antes mencionado, el manejador para el protocolo DF1 debe incluir en su funcionamiento, específicamente los siguientes requerimientos:

Requerimientos funcionales:

- Parametrización del protocolo, de los dispositivos y redes.
- Configuración de dispositivos, redes y manejadores.
- Validación de las direcciones admisibles.
- Acceso a información de diagnóstico.
- Soporte para la configuración de variables de diagnóstico.
- Funciones de lectura.
- Funciones de escritura.
- Estampado del tiempo.
- Calidad de los valores.
- Mensajes de Error.

En el caso de los requerimientos no funcionales, se tendrán en cuenta para este manejador todos los que se mencionan en el documento de especificación de requerimientos antes mencionado.

2.6 BIBLIOTECA DRIVER CORE

La implementación de los manejadores puede ser llevada a cabo de dos formas, la primera consiste en hacerlo directamente implementando de la IG y la segunda, haciendo uso de las clases que brinda la biblioteca DriverCore.

La biblioteca dinámica DriverCore es una implementación de la Interfaz Genérica 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. Esta clase es un singleton, existe solo una por cada biblioteca que se carga. La biblioteca DriverCore carga de manera opcional una biblioteca de transporte asíncrona (basada en Asio 1.4.1) que contiene fábricas de transporte TCP y serie.

Haciendo uso del DriverCore se tiene acceso a las ventajas que brindan sus áreas funcionales, estas son: la introspección de manejadores y dispositivos, el manejo de las direcciones, clasificación de las variables en bloques de lectura o escritura y la entrada y salida asíncrona.

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 (29). 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.

CAPÍTULO 3 DISEÑO, IMPLEMENTACIÓN Y PRUEBA

3.1 ARQUITECTURA DE SOFTWARE

Según define la IEEE Std 1471-2000: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”. (32)

La arquitectura de software tiene la responsabilidad de:

- Definir los módulos principales.
- Definir las responsabilidades que tendrá cada uno de estos módulos.
- Definir la interacción que existirá entre dichos módulos:
    - Control y flujo de datos.
    - Secuenciación de la información.
    - Protocolos de interacción y comunicación.
    - Ubicación en el hardware.

La Arquitectura del Software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. (33)

3.1.1 Patrones de arquitectura

Los patrones arquitectónicos tratan sobre aspectos fundamentales de la estructura de un sistema. Especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. (34)

3.1.1.1 Arquitecturas en Capas

Este patrón define cómo organizar el modelo de diseño en capas, que pueden estar físicamente distribuidas, lo cual quiere decir que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Este patrón es importante porque simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores. Además, nos ayuda a identificar qué se puede reutilizar y proporciona una estructura que nos ayuda a tomar decisiones sobre qué partes comprar y qué partes construir. (43)

Principales estilos de este patrón arquitectónico:

- Arquitecturas de dos capas.
- Arquitecturas de tres capas.
- Arquitecturas de n capas.

Para el diseño del manejador se utiliza específicamente el estilo de arquitectura tres capas, ya que el modelo de diseño está organizado en las capas de Driver, Protocolo y Transporte, como se muestra a continuación:

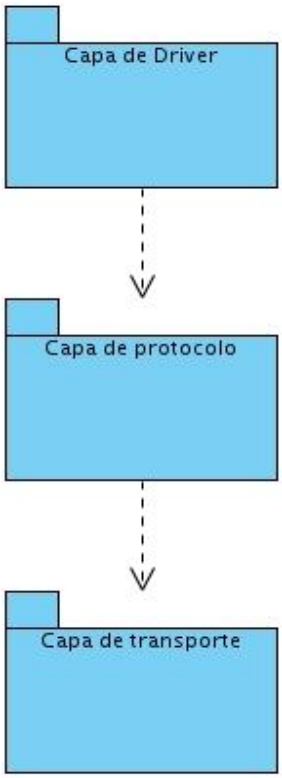

Fig. 22 Arquitectura tres capas del manejador.

Capa de Driver

En la capa de Driver, 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 para el cual se esté llevando a cabo el manejador. Para lograr que cada manejador tenga capacidad de introspección, es necesario crear cuatro nuevas clases descendientes, respectivamente de las clases del DriverCore. Estas clases son: DriverMetaClass, DeviceMetaClass, Driver y Device. La clase DriverMetaClass es una clase abstracta que sirve de base a las metaclases de los descendientes de la clase Driver. Le añade, con respecto a su ancestro BaseMetaClass, el hecho de que "conoce" la metaclase de los dispositivos asociados al Driver, expone la estructura DriverInfo y además contiene una factoría para instanciar la clase descendiente de Driver correspondiente. DeviceMetaClass sirve de ancestro común para todas las metaclases de los dispositivos. Agrega solamente a su ancestro BaseMetaClass el registro del parámetro de diagnostico STATE. Driver y Device permiten modelar las características comunes de los manejadores y los dispositivos respectivamente.

Las clases encargadas del manejo de las direcciones en el DriverCore son: Address, SimpleAddress e IProtocolAddress. La primera de estas, define una lista de direcciones simples (Simple Address) con el formato que se les definan según el protocolo que se implemente, esta lista puede tener un máximo de 16 direcciones simples; la segunda, además de que se le define su formato según las características del protocolo, puede ir acompañada de una máscara que contiene el bit de comienzo de la dirección y el tamaño en bits del segmento. Para la validación de las direcciones según las características del protocolo se crea una clase descendiente de IProtocolAddress. Para la formación de los bloques de variables se tiene en cuenta una serie de restricciones. Estas restricciones se definen en funciones que se encuentran en la clase Device y son redefinidas en sus descendientes.

La clase Block implementa el concepto de conjunto "consecutivo" de direcciones que tienen el mismo periodo 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, esta clasificación la regula el atributo addressType. Como las variables de un bloque son "consecutivas", para conocer las variables que integran el mismo sólo se necesita conocer el índice de la primera variable (firstVarIndex), el de la última variable (lastVarIndex) y el correspondiente contenedor que es interno y es definido en la implementación de cada manejador. Una vez que se efectúa la lectura física el bloque tiene que retornar el valor de una dirección y de un tipo de dato. Para ello 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

La capa de protocolo tiene como objetivo garantizar la lógica de comunicación entre el manejador y los dispositivos, definiéndose para esto una clase EndPoint. En esta clase se controlan 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. Cuando las tramas tienen algún nivel de complejidad, se crea una clase Mensaje que abstrae al EndPoint de la necesidad de conocer como se crean o cuáles son los elementos que constituyen una trama.

La clase Mensaje se encarga entonces de la construcción de las tramas y de conocer el comportamiento que debe tener cada uno de los elementos que la conforman, brindándole al EndPoint solamente los datos que se reciben en la trama, que son de interés para el funcionamiento de las capas superiores del manejador. Para llevar a cabo toda esta secuencia de pasos es necesario el uso de un transporte, que nos permita el envío y recepción de las tramas a través del medio físico que utilice el protocolo.

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 unos 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 de TCP existen las clases ITCPTransport y TCPTransport, para UDP son la IUDPTransport y UDPTransport y por último para el transporte Serie están definidas las clases ISerialTransport y SerialTransport. La interfaz ITransport encapsula las funcionalidades asíncronas de los diferentes tipos de transporte que brinda la librería. La clase hereda de ITransportObject, define además de las funcionalidades asíncronas, otras funcionalidades comunes para las interfaces de transporte, como lo son el establecimiento de algunas propiedades y otras funcionalidades necesarias. 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. De esta manera se avisa en forma de callback cuando se culmina una lectura, escritura o conexión.

De manera adicional la clase TransportProvider brinda dos componentes de gran utilidad. El primer componente es un temporizador asíncrono definido en la clase AsyncTimer que a su vez hereda de su interfaz IAsyncTimer. El segundo un Aceptor que tiene como única tarea aceptar conexiones TCP, siendo muy útil para un servidor TCP. Para el Aceptor fue definida una clase Acceptor que hereda de una interfaz IAcceptor de la cual implementa sus funciones.

3.2DIAGRAMA DE CLASES DEL DISEÑO

3.2.1Diagrama de clases de la Capa de Driver

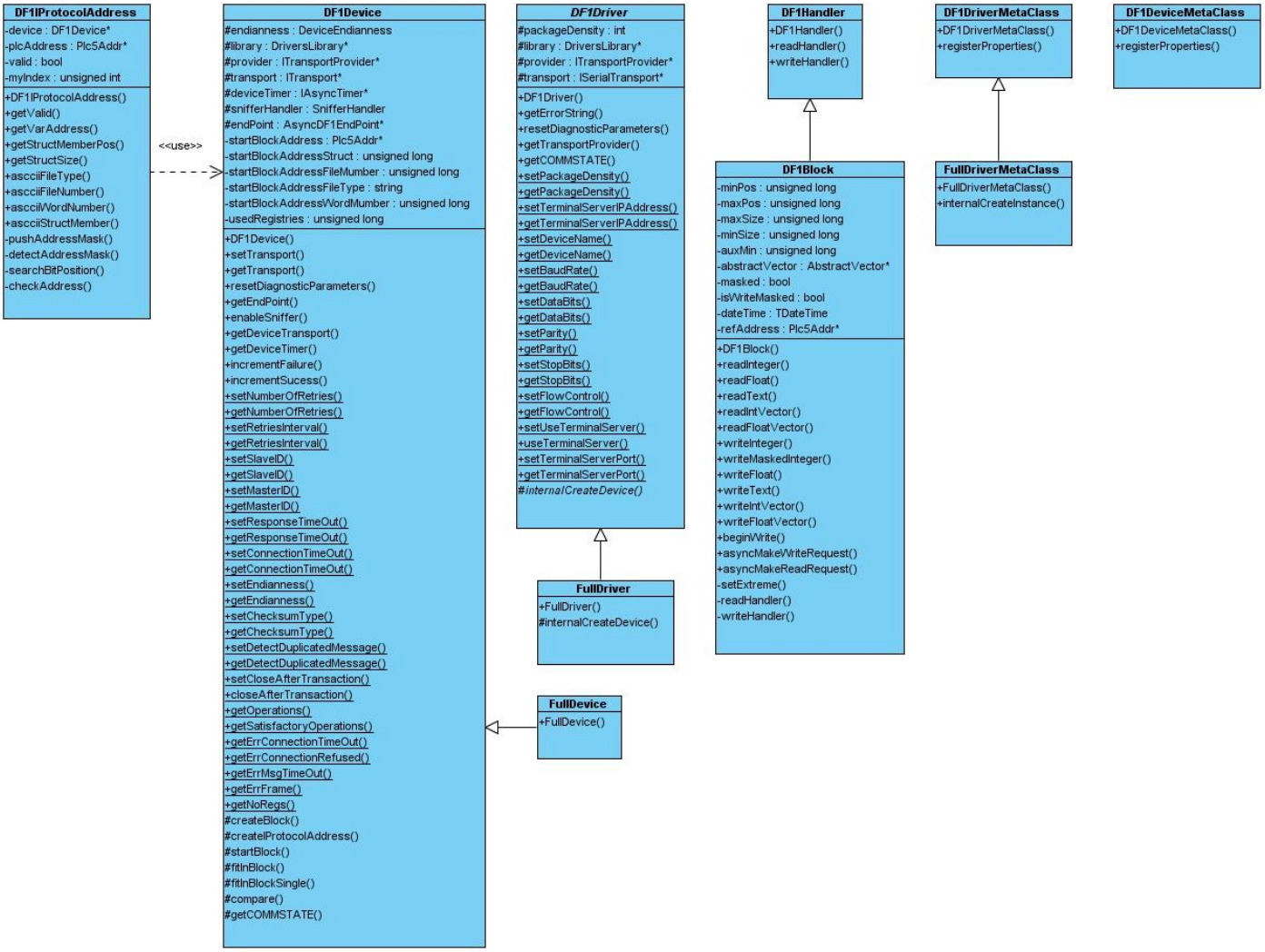

Fig. 23 Diagrama de clases del Diseño Capa de Driver.

3.2.2Diagrama de clases de la Capa de Protocolo

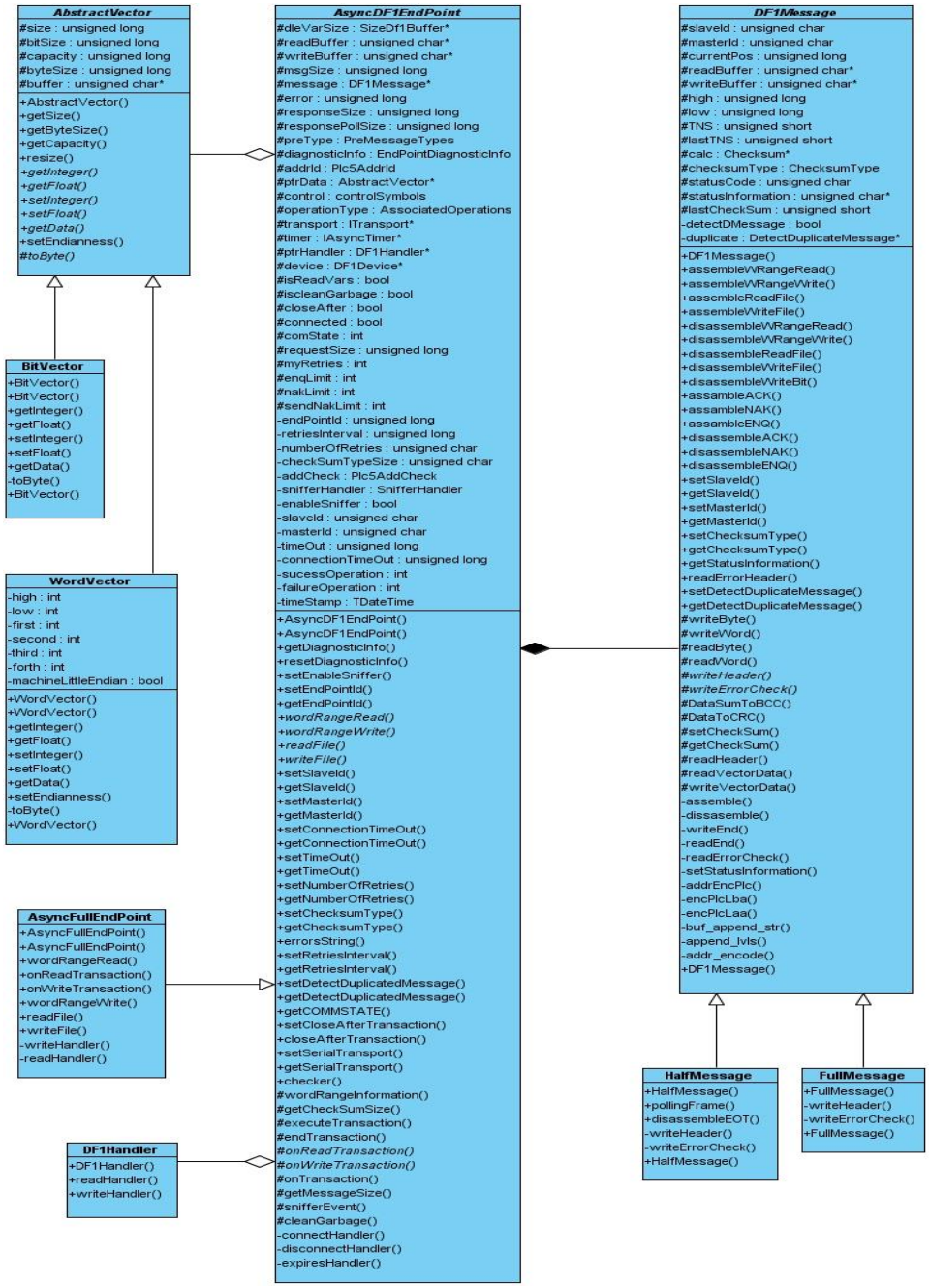

Fig. 24 Diagrama de clases del Diseño Capa de Protocolo.

3.2.3Diagrama de clases del manejador DF1

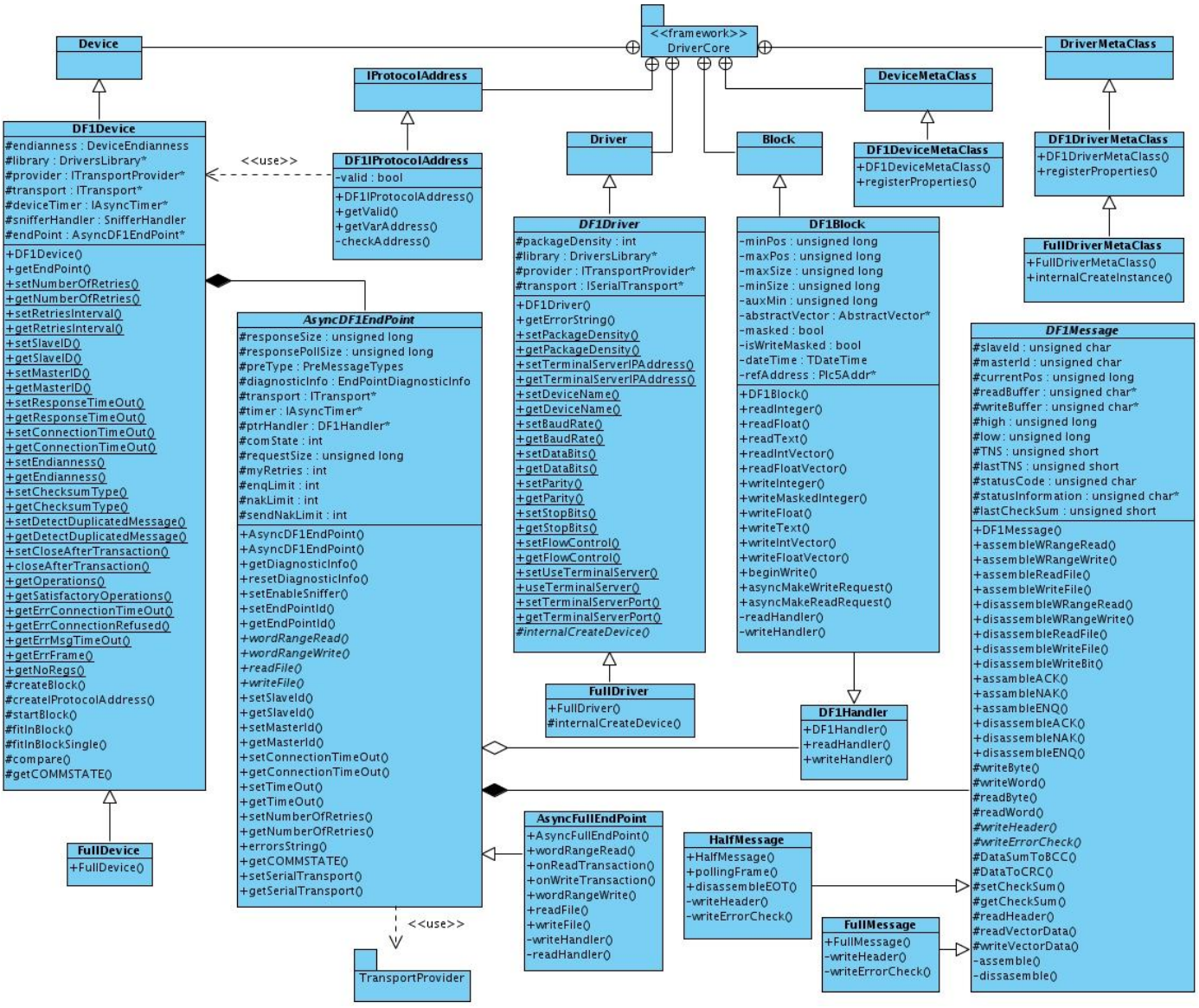

Fig. 25 Diagrama de clases del Diseño del Driver DF1.

3.3 DESCRIPCIÓN DE CLASES DEL DISEÑO

3.3.1 Descripción de la clase AsyncDF1EndPoint

Esta clase encapsula la lógica de comunicación general con el dispositivo. Permite obtener y configurar algunas propiedades de los dispositivos. En ella se establecen los comandos que se usarán para la comunicación que luego serán implementados en las clases específicas para cada tipo de DF1 (Half-Duplex y Full-Duplex). La lógica de comunicación y los comandos para la variante Full-Duplex serán implementados en la clase AsyncFullEndPoint. En versiones posteriores para implementar la lógica de comunicación de la versión Half-Duplex, solo se necesita heredar de AsyncDF1EndPoint y reimplementar el funcionamiento de los comandos.

Tabla 8 Descripción de clase AsyncDF1EndPoint.

Nombre: AsyncDF1EndPoint	
Tipo de clase: Entidad	
Atributo	**Tipo**
readBuffer	unsigned char*
writeBuffer	unsigned char*
message	DF1Message*
error	unsigned long
responseSize	unsigned long
responsePollSize	unsigned long
diagnosticInfo	EndPointDiagnosticInfo
Transport	ITransport*
Timer	IAsyncTimer*
ptrHandler	DF1Handler*
comState	int
requestSize	unsigned long
enqLimit	int
nakLimit	int
sendNakLimit	int

Para cada responsabilidad:	
Nombre:	AsyncDF1EndPoint(DF1Device* ndevice, SnifferHandler nsniffer)
Descripción:	Constructor de la clase AsyncDF1EndPoint.
Nombre:	getDiagnosticInfo()
Descripción:	Devuelve una estructura con la información de diagnóstico.
Nombre:	resetDiagnosticInfo()
Descripción:	Reinicia toda la información de diagnóstico a cero.
Nombre:	setEnableSniffer(bool enable)
Descripción:	Habilitar y desabilitar el Sniffer.
Nombre:	setEndPointId(unsigned long newEndPointId)
Descripción:	Cambia el identificador del EndPoint.
Nombre:	getEndPointId()
Descripción:	Devuelve el identificador del EndPoint.
Nombre:	wordRangeRead(Plc5Addr* address, AbstracVector* data, DFHandler* handler)
Descripción:	Lee los registros desde un dispositivo PLC-5 previamente determinado.
Nombre:	wordRangeWrite(Plc5Addr* address, AbstracVector* dataWrite, DFHandler* handler)
Descripción:	Escribe los registros en un dispositivo PLC-5 previamente determinado.
Nombre:	readFile(Plc5Addr* address, AbstracVector* data, DFHandler* handler)
Descripción:	Lee los registros desde un dispositivo PLC-5, PLC-3 y PLC5/250 previamente determinado.
Nombre:	writeFile(Plc5Addr* address, AbstracVector* dataWrite, DFHandler* handler)
Descripción:	Escribe los registros en un dispositivo PLC-5, PLC-3 y PLC5/250 previamente determinado.
Nombre:	SetSlaveId(unsigned char newSlaveId)
Descripción:	Establece el identificador en la trama del dispositivo con el que se establecerá la comunicación.
Nombre:	getSlaveId()
Descripción:	Devuelve el identificador en la trama del dispositivo con el que se establecerá la comunicación.

Nombre:	setMasterId()
Descripción:	Establece el identificador del manejador en la trama.
Nombre:	getMasterId()
Descripción:	Devuelve el identificador del manejador en la trama.
Nombre:	setConnectionTimeOut(unsigned long timeOut)
Descripción:	Establece el tiempo de espera por la conexión con el dispositivo.
Nombre:	getConnectionTimeOut()
Descripción:	Devuelve el tiempo de espera por la conexión con el dispositivo.
Nombre:	setTimeOut(unsigned long timeOut)
Descripción:	Establece el tiempo que se esperará para recibir un mensaje del dispositivo.
Nombre:	getTimeOut()
Descripción:	Devuelve el tiempo que se esperará para recibir un mensaje del dispositivo.
Nombre:	setNumberOfRetries(unsigned char number)
Descripción:	Establece el número de reintentos a llevar a cabo cuando se recibe un mensaje de NAK o de ENQ.
Nombre:	getNumberOfRetries()
Descripción:	Devuelve el número de reintentos a llevar a cabo cuando se recibe un mensaje de NAK o de ENQ.
Nombre:	setChecksumType(ChecksumType checksum)
Descripción:	Establece el tipo de chequeo de error que se desee usar en la trama.
Nombre:	getChecksumType()
Descripción:	Devuelve el tipo de chequeo de error que se usará en la trama.
Nombre:	errorString(const unsigned long error)
Descripción:	Llena el buffer de errores según el código de error que se genere.
Nombre:	getCOMMSTATE()
Descripción:	Devuelve el estado de la comunicación con el dispositivo.
Nombre:	setSerialTransport(ISerialTransport* ntransport)
Descripción:	Establece el transporte serial.
Nombre:	getSerialTransport()
Descripción:	Devuelve el transporte serial usado.

3.3.2 Descripción de la clase DF1Message

Esta clase es la encargada de manejar la capa de mensaje del protocolo. En ella se implementan los métodos necesarios para llevar a cabo el ensamblado y desensamblado genérico de los mensajes, según los comandos que se deseen enviar o recibir. También es responsable de verificar la calidad de las tramas recibidas del dispositivo mediante el chequeo de error y la captura o inserción de los datos en las tramas. De esta clase genérica heredan dos clases especializadas. La primera clase hija se nombra FullMessage y es la encargada de reimplementar los métodos que son para ensamblar partes del mensaje con características específicas para la versión Full-Duplex del protocolo DF1 y la segunda HalfMessage, la cual realiza las mismas tareas, pero para la versión Half-Duplex del protocolo.

Tabla 9 Descripción de clase DF1Message.

Nombre: DF1Message	
Tipo de clase: Entidad	
Atributo	**Tipo**
slaveId	unsigned char
masteId	unsigned char
readBuffer	unsigned char*
writeBuffer	unsigned char*
high	unsigned long
low	unsigned long
TNS	unsigned short
lastTNS	unsigned short
statusCode	unsigned char
statusInformation	unsigned char*
lastChecksum	unsigned short
Para cada responsabilidad:	
Nombre:	DF1Message(unsigned char* pwriteBuffer, unsigned char* preadBuffer, unsigned char pslaveId)
Descripción:	Constructor de la clase DF1Message.

Nombre:	assembleWRangeRead(const unsigned short packetOffset, const unsigned short totalTrans, const unsigned char size, const Plc5Addr* address)
Descripción:	Ensambla las tramas de lectura en una cantidad de hasta 244 bytes.
Nombre:	assembleWRangeWrite(const unsigned short packetOffset, const unsigned short totalTrans, AbstracVector* dataWrite, const Plc5Addr* address)
Descripción:	Ensambla las tramas de escritura en una cantidad de hasta 244 bytes.
Nombre:	assembleReadFile(const unsigned short packetOffset, const unsigned short totalTrans, unsigned char size, const Plc5Addr* address)
Descripción:	Ensambla las tramas de lectura de datos comenzando desde un archivo símbolo o un bloque de direcciones.
Nombre:	assembleWriteFile(const unsigned short packetOffset, const unsigned short totalTrans, AbstracVector* dataWrite, const Plc5Addr* address)
Descripción:	Ensambla las tramas de escritura de datos comenzando desde un archivo símbolo o un bloque de direcciones.
Nombre:	disassembleWRangeRead(AbstractVector* dataWrite)
Descripción:	Desensambla las tramas de lectura.
Nombre:	disassembleWRangeWrite()
Descripción:	Desensambla las tramas de escritura.
Nombre:	disassembleReadFile(AbstractVector* dataWrite)
Descripción:	Desensambla las tramas de lectura de archivo.
Nombre:	disassembleWriteFile()
Descripción:	Desensambla las tramas de escritura de archivo.
Nombre:	writeHeader()
Descripción:	Escribe la cabecera de la trama en el buffer de escritura.
Nombre:	writeErrorCheck(const unsigned long bodySize)
Descripción:	Escribe los datos de chequeo de errores en el buffer de escritura.
Nombre:	readHeader()
Descripción:	Devuelve los datos de la cabecera de la trama desde el buffer de lectura.
Nombre:	assemble(const unsigned char commandCode)
Descripción:	Ensambla las partes comunes de los mensajes.

Nombre:	disassemble(const unsigned char commandCode)
Descripción:	Desensambla las partes comunes de los mensajes.
Nombre:	writeEnd()
Descripción:	Escribe el fin de las tramas en el buffer de escritura.
Nombre:	readEnd()
Descripción:	Lee el fin de las tramas en el buffer de escritura.

3.3.3 Descripción de la clase DF1Device

La clase DF1Device implementa el concepto de dispositivo DF1. Hereda de la clase Device que brinda la biblioteca DriverCore para modelar las características comunes de los dispositivos. De esta clase se obtiene lo referente a los parámetros de configuración tanto de diagnóstico como de otros tipos. DF1Device posee las características y comportamientos comunes de los dispositivos.

Tabla 10 Descripción de clase DF1Device.

Nombre: DF1Device	
Tipo de clase: Controladora	
Atributo	**Tipo**
endianness	DeviceEndianness
library	DriversLibrary*
provider	ITransportProvider*
transport	ITransport*
deviceTimer	IAsyncTimer*
snifferHandler	SnifferHandler
endPoint	AsyncDF1EndPoint*
Para cada responsabilidad:	
Nombre:	DF1Device(unsigned long devScadaId, char* transferBuffer, unsigned long bufferSize, ITransportProvider* Provider, SnifferHandler nsnifferHandler)
Descripción:	Constructor de la clase DF1Device.
Nombre:	createBlock(int blockStart,int blockEnd, bool forRead, VarLinkVector* varsVector)

Descripción:	Crea un bloque de variables.
Nombre:	createIProtocolAddress(const char* address, bool forRead)
Descripción:	Crea una instancia de IProtocolAddress a partir de la representación textual de la dirección dada por el parámetro address. El objeto IProtocolAddress responde por almacenar los parámetros de la dirección que son específicos de cada protocolo.
Nombre:	startBlock(IProtocolAddress* address1, DeviceVarType varitype, unsigned long arraySize, bool forread, unsigned long addressSize)
Descripción:	Establece una dirección como dirección de comienzo de un bloque.
Nombre:	startBlock(IProtocolAddress* address1, DeviceVarType varitype, unsigned long arraySize, bool forread, unsigned long addressSize)
Descripción:	Establece una dirección como dirección de comienzo de un bloque.
Nombre:	fitInBlock(IProtocolAddress* address2, DeviceVarType varType, unsigned long arraySize, bool forRead, unsigned long addressSize)
Descripción:	Debe determinar si el rango de direcciones definido entre la dirección establecida por startBlock y la dirección que se define por los parámetros del método, puede alojarse en un bloque del protocolo, o sea puede ser recuperado o modificado con un solo mensaje del protocolo.
Nombre:	fitInBlockSingle(IProtocolAddress* address1, IProtocolAddress* address2, DeviceVarType varType, unsigned long arraySize, bool forread, unsigned long addressSize)
Descripción:	Debe determinar si una variable puede alojarse en un bloque. Esta determinación se hace a través de las direcciones address1 y address2 que representan las direcciones de protocolo, de la menor y mayor de las direcciones simples de la variable respectivamente.
Nombre:	compare(IProtocolAddress* address1, IProtocolAddress* address2)
Descripción:	Compara dos direcciones de protocolo de acuerdo a las reglas del protocolo.

3.3.4 Descripción de la clase DF1Driver

Esta clase implementa el concepto de manejador DF1. Hereda de la clase Driver que brinda la biblioteca DriverCore que posee las características y comportamientos comunes de los manejadores.

Tabla 11 Descripción de clase DF1Driver.

Nombre: DF1Driver	
Tipo de clase: Controladora	
Atributo	**Tipo**
packageDensity	int
library	DriversLibrary*
provider	ITransportProvider*
transport	ISerialTransport*
Para cada responsabilidad:	
Nombre:	DF1Driver(ReadHandler readHandler, WriteHandler writeHandler, SnifferHandler snifferHandler)
Descripción:	Constructor de la clase DF1Driver.
Nombre:	getErrorString(unsigned long errorCode, const char** ppErrorDescription)
Descripción:	Permite obtener información textual comprensible a partir de un código de error del manejador.
Nombre:	internalCreateDevice (unsigned long devScadaId, char* transferBuffer, unsigned long bufferSize)
Descripción:	Crea internamente un dispositivo DF1.
Nombre:	unsigned long setPackageDensity(BaseClass* base, long long newPackageDensity)
Descripción:	Establece la propiedad de densidad de paquete.

3.3.5 Descripción de la clase DF1Block

La clase DF1Block implementa el concepto de conjunto consecutivo de direcciones DF1 que puede ser accedido a través de un solo mensaje del protocolo. Hereda de la clase Block que brinda la biblioteca DriverCore y de DF1Handler, de esta última reimplementa los callback de lectura y de escritura para cuando se completen dichas tareas por parte del endpoint.

Tabla 12 Descripción de clase DF1Block.

Nombre: DF1Block	
Tipo de clase: Controladora	
Atributo	**Tipo**
minPos	unsigned long
maxPos	unsigned long
maxSize	unsigned long
minSize	unsigned long
auxMin	unsigned long
abstractVector	AbstractVector*
masked	bool
isWriteMasked	bool
dateTime	TDateTime
refAddress	Plc5Addr*
Para cada responsabilidad:	
Nombre:	DF1Blook(Device* device, unsigned long blockStart, unsigned long blockEnd, bool forRead, VarLinkVector* container)
Descripción:	Constructor de la clase DF1Blook.
Nombre:	readInteger(IProtocolAddress* address, DeviceVarType varType, TDateTime* deviceTimeStamp, unsigned short* quality)
Descripción:	Lee un valor entero del bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	readFloat (IProtocolAddress* address, DeviceVarType varType, TDateTime*

	deviceTimeStamp, unsigned short* quality)
Descripción:	Lee un valor flotante 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:	Lee una cadena del bloque a partir de un IProtocolAddress.
Nombre:	readIntVector(IProtocolAddress* address, DeviceVarType varType, unsigned long arraySize, long long* value, TDateTime* deviceTimeStamp, unsigned short* quality)
Descripción:	Lee un arreglo de ordinales (booleanos o enteros) a partir de un IProtocolAddress.
Nombre:	readFloatVector(IProtocolAddress* address, DeviceVarType varType, unsigned long arraySize, long long* value, TDateTime* deviceTimeStamp, unsigned short* quality)
Descripción:	Lee un arreglo de flotantes a partir de un IProtocolAddress.
Nombre:	writeInteger(IProtocolAddress* address, DeviceVarType varType, long long value)
Descripción:	Escribe un valor entero en el bloque a partir de un IProtocolAddress.
Nombre:	writeFloat (IProtocolAddress* address, DeviceVarType varType, long long value)
Descripción:	Escribe un valor flotante en el bloque a partir de un IProtocolAddress.
Nombre:	writeMaskedInteger(IProtocolAddress* address, DeviceVarType varType, long long value, long long mask)
Descripción:	Permite modificar bits específicos en el bloque a partir de un IProtocolAddress.
Nombre:	writeText(IProtocolAddress* address, unsigned long arraySize, char* value)
Descripción:	Escribe una cadena en el bloque a partir de un IProtocolAddress.
Nombre:	writeIntVector(IProtocolAddress* address, deviceVarType varType, unsigned long arraySize, long long* value)
Descripción:	Permite modificar un arreglo de ordinales (booleanos y enteros) en el bloque a partir de un IProtocolAddress.
Nombre:	writeFloatVector(IProtocolAddress* address, deviceVarType varType, unsigned long arraySize, long long* value)
Descripción:	Permite modificar un arreglo de flotantes en el bloque a partir de un

	IProtocolAddress.
Nombre:	beginWrite()
Descripción:	Realiza las operaciones necesarias para inicializar la escritura del bloque.
Nombre:	asyncMakeWriteRequest()
Descripción:	Realiza una escritura asíncrona al dispositivo.
Nombre:	asyncMakeReadRequest()
Descripción:	Realiza una lectura asíncrona al dispositivo.
Nombre:	setExtreme()
Descripción:	Establece las fronteras del bloque.
Nombre:	readHandler(Plc5Addr*, AbstractVector*, unsigned long error)
Descripción:	Callback invocado por el DF1EndPoint cuando finaliza una lectura. Restablece el valor de la variable de error de lectura del bloque y avisa al Core que finalizó la realización de la lectura a través de la función Block::asyncReadCompleted.
Nombre:	writeHandler(Plc5Addr*, unsigned long error)
Descripción:	Callback invocado por el DF1EndPoint cuando finaliza una escritura. Restablece el valor de la variable de error de escritura del bloque y avisa al Core que finalizó la realización de la escritura a través de la función Block::asyncWriteCompleted.

3.3.6 Descripción de clase DF1IProtocolAddress

La clase DF1IProtocolAddress representa a las direcciones de las variables del protocolo DF1. Hereda de la clase IProtocolAddress que brinda la biblioteca DriverCore para encapsular el concepto de dirección de protocolo o punto de medición en el campo.

Tabla 13 Descripción de clase DF1IProtocolAddress.

Nombre: DF1IProtocolAddress	
Tipo de clase: Controladora	
Atributo	**Tipo**
valid	bool
plcAddress	Plc5Addr*

Para cada responsabilidad:	
Nombre:	DF1IProtocolAddress(DF1Device* device, const char* address)
Descripción:	Constructor de la clase DF1IProtocolAddress.
Nombre:	getValid()
Descripción:	Expresa si la dirección es válida o no de acuerdo a las reglas específicas del protocolo.
Nombre:	getVarAddress()
Descripción:	El método getVarAddress nos permite acceder al atributo wordAddress.
Nombre:	checkAddress(const char* newAddress)
Descripción:	Dado un registro verifica si corresponde a una dirección correcta.

3.4 ESTÁNDAR DE CODIFICACIÓN

El manejador está totalmente programado en inglés, debido a que en este idioma no se acentúan las palabras y es muy utilizado en el mundo de la informática. A continuación se describe la estructura que se utiliza en la escritura de los nombres de los ficheros, clases y métodos.

Nombre de los ficheros.

Se nombraran los ficheros .h y .cpp de la siguiente manera.

- NameOfUnit.h
- NameOfUnit.cpp

Clases.

Los nombres de las clases deben tener la siguiente forma: MyClass En un fichero .h solo debe ser definida una clase en caso de que sea necesario y a dicha definición le debe corresponder un fichero .cpp donde se implemente las funcionalidades de la clase definida. Solo se podrán implementar en los .h funciones miembros con el comando o palabra reservada “inline”.

Métodos o función miembro.

Los métodos de clases deben seguir la siguiente estructura:

- myFunction

A excepción de los constructores y destructores todos los métodos deben empezar con minúscula.

3.5 DIAGRAMA DE DESPLIEGUE

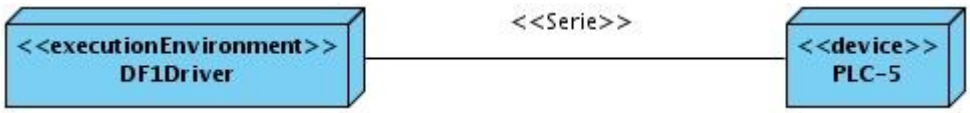

Fig. 26 Diagrama de despliegue.

3.6 DIAGRAMA DE COMPONENTES

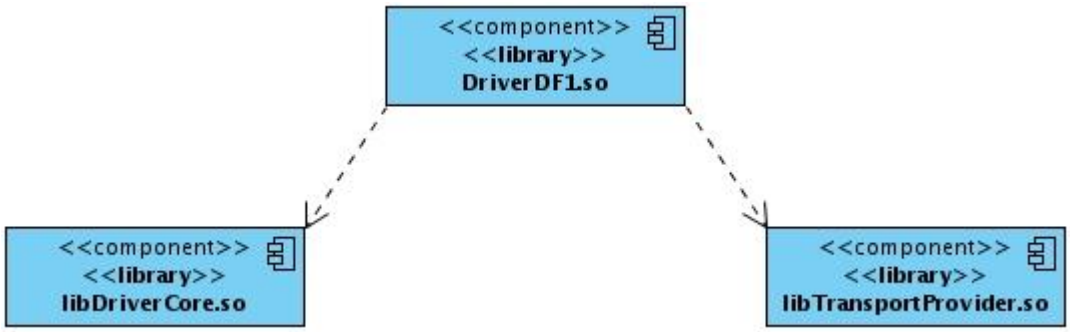

Fig. 27 Diagrama de componentes.

3.7 PRUEBA

Las pruebas son procesos que se llevan a cabo con el objetivo de verificar la calidad de un producto detectando los posibles errores que puede tener un software.

3.7.1 Tipos de pruebas

Aunque no hay una clasificación oficial o formal acerca de los diversos tipos de pruebas de software, existen dos vertientes fundamentales:

Caja Negra: Son las pruebas que se le realizan a una aplicación, verificando las funcionalidades que debe tener, sin tener en cuenta el funcionamiento del código fuente. Generalmente se utiliza una interfaz visual para realizarlas.

Caja Blanca: Son las pruebas que se realizan verificando los caminos lógicos del software, por lo que se requiere el conocimiento de la estructura interna del programa.

Para probar el funcionamiento de los módulos del SCADA Galba existe un equipo especializado en Venezuela, que posee la experiencia y las herramientas necesarias para obtener resultados satisfactorios en la detección de errores. De manera adicional, durante el desarrollo del manejador se realizan pruebas de funcionamiento de la capa de protocolo y al manejador como un módulo independiente cuando se ha concluido el flujo de trabajo de implementación. Para realizar estas pruebas se utiliza un Recolector Gráfico el cual simula el funcionamiento del recolector del SCADA. Esta aplicación es un GUI que carga los manejadores y permite realizarle la configuración y las operaciones de lectura y escritura de variables.

3.7.2 Pruebas realizadas al manejador

A continuación se exponen algunas de las pruebas de caja negra que se le han realizado al manejador para el protocolo DF1 con el SCADA Galba en Venezuela. En las que se probaron las funciones de lectura de enteros y flotantes y de bits.

3.7.2.1 Pruebas de recolección, procesamiento y publicación de puntos analógicos

Pre-condiciones

- El Operador está registrado en el sistema y pertenece a un grupo operacional de privilegios.
- El punto puede estar configurado con permiso de solo lectura o lectura/escritura. Se configuran los puntos PtoEnteroDF1, PtoRealDF1.

- Los dispositivos de control son previamente configurados en el sistema, según la configuración descrita.
- El protocolo DF1 es definido previamente en el Sistema.

Post-condiciones

- Se observan los puntos publicados en el JDesktop.

Procedimiento

Para esta prueba se hará uso de los puntos analógicos, creados durante la fase de Configuración de Ambiente de Prueba. Para dar inicio a la prueba se debe:

- Iniciar los módulos: Capa de comunicaciones, Configuración, Seguridad y JConfig.
- Crear la configuración.
- Iniciar Adquisición 2.0, BDH y el driver.
- Iniciar sesión en JDesktop.

Paso 1: Verificar la lectura del dato.

- Tomando los Puntos PtoEnteroDF1, PtoRealDF1.
- Verificar si el recolector está adquiriendo dicho punto por medio del sumario de puntos analógicos en el JDesktop.

Tabla 14 Resultado de prueba de lectura de datos analógicos.

Descripción de la salida esperada	Valores de la salida esperada
En el sumario de puntos analógicos del JDesktop los puntos deben ser publicados con calidad 192.	**PtoEnteroDF1** Value_INT: 50 Quality: 192

	PtoRealDF1 Value_Double:55.55 Quality: 192

Paso 2: Verificar la calidad del dato.

- Tomando los Puntos PtoEnteroDF1, PtoRealDF1.
- Desconectar el cable red.
- Verificar que el Procesamiento de puntos esté publicando el dato con calidad 24, para ello se puede visualizar desde el sumario de puntos analógicos en el JDesktop.
- Se debe verificar que la salida corresponda con los valores de la salida esperada que se describe a continuación.
- Debe estar desactivado el Network Manager. Para esto puede utilizar los siguientes pasos. */etc/init.d/network-manager stop ; dhclient eth0 .*

Tabla 15 Resultado de prueba calidad de lectura de datos analógicos.

Descripción de la salida esperada	**Valores de la salida esperada**
Los puntos deben ser publicados con calidad 24 en el sumario de puntos analógicos del JDesktop.	Quality: 24

3.7.2.2 Pruebas de recolección, procesamiento y publicación de puntos digitales.

Pre-condiciones.

- El Operador está registrado en el sistema y pertenece a un grupo operacional de privilegios.
- El punto puede estar configurado con permiso de solo lectura o lectura/escritura. Configurar el punto **PtoDigitalDF11** y **PtoDigitalDF12.**

- Los dispositivos de control son previamente configurados en el Sistema, según la configuración descrita.
- El protocolo DF1 es definido previamente en el Sistema.

Post-condiciones.

- Se observan los puntos publicados en el JDesktop.

Procedimiento.

Para esta prueba se hará uso del punto analógico, creados durante la fase de Configuración de Ambiente de Prueba. Para dar inicio a la prueba se debe:

- Iniciar los módulos: Capa de comunicaciones, Configuración, Seguridad y JConfig.
- Crear la configuración.
- Iniciar Adquisición 2.0, BDH y el driver.

Paso 1: Verificar la lectura del dato.

- Tomando los Puntos PtoDigitalDF11 y PtoDigitalDF12.
- Colocar en el dispositivo, en la dirección de entrada del punto el valor 1.
- Verificar en las trazas si el recolector está adquiriendo dicho punto, utilizando para ello el sumario de puntos del JDesktop.

Tabla 16 Resultado de prueba de lectura de datos digitales.

Descripción de la salida esperada	**Valores de la salida esperada**
En el sumario de puntos analógicos del JDesktop los puntos deben ser publicados con calidad 192.	**PtoDigitalDF11** Valor Actual: 1 Quality: 192

	PtoDigitalDF12 Valor Acual: 1 Quality: 192

Paso 2: Verificar la calidad del dato.

- Desconectar el cable red.
- Verificar que el procesamiento de puntos esté publicando el dato con calidad 24.

Debe estar desactivado el Network Manager. Para esto puede utilizar los siguientes pasos. /etc/init.d/network-manager stop ; dhclient eth0.

Tabla 17 Resultado de prueba calidad de lectura de datos digitales.

Descripción de la salida esperada	**Valores de la salida esperada**
Los puntos deben ser publicados con calidad 24 en el sumario de puntos analógicos del JDesktop.	Quality: 24

3.7.2.3Funciones probadas

Las funcionalidades implementadas y probadas en el manejador se exponen en la siguiente tabla:

Tabla 18 Funcionalidades implementadas y probadas.

Funcionalidad	**Implementada**	**Probado**
Comunicación con los PLC-5	Si	Si
Comunicación con los PLC2	Si	No
Comunicación con los PLC3	Si	No
Comunicación con los PLC5/250	Si	No

Comunicación con los PLC5/VME	Si	No
Lectura de tipos enteros de 16 bits (N)	Si	Si
Escritura de tipos enteros de 16 bits (N)	Si	Si
Lectura de tipos flotantes de 32 bits (F)	Si	Si
Escritura de tipos flotantes de 32 bits (F)	Si	Si
Lectura de tipos Binary (B)	Si	Si
Escritura de tipos Binary (B)	Si	Si
Lectura de tipo Output: (O), Input: (I), y Status(S)	Si	Si
Escritura de tipo Output: (O)	Si	Si
Lectura de Timers (T), Counter (C), SFCStatus (SC) y PIDControl (PD).	Si	Si
Escritura de Timers (T), Counter (C), SFCStatus (SC) y PIDControl (PD).	Si	Si

3.7.2.4 Prueba de rendimiento

Se efectuó una prueba de rendimiento con 12200 variables, a un solo PLC. Estas variables ocupaban un total de 24400 bytes, en un solo mensaje pueden ser solicitados hasta 244 bytes de datos por lo que estas 12200 variables fueron solicitadas en 100 mensajes. El tiempo de demora total para la solicitud de las 12200 variables en los 100 mensajes fue de 16.44 segundos, por lo que podemos concluir que para una lectura de 122 variables en un mensaje, el tiempo de llegada de los datos es 0.16 segundos aproximadamente. Luego se realizó otra prueba donde se leyeron la misma cantidad de variables (12200), pero esta vez en cada mensaje se pidieron 61 variables y se enviaron 200 mensajes, la lectura del total de

todas las variables se realizó en 22 segundos, dando así un tiempo de 0.11 segundos por mensaje aproximadamente. En el caso de la escritura con la misma cantidad de variables y en 100 mensajes el tiempo fue de 20 segundos, mientras que con 200 mensajes fueron 25 segundos. Según los resultados que se arrojan en estas pruebas podemos concluir que se debe crear bloques de variables lo más cercanos al límite que permita cada mensaje en un envió ya que esta será la forma más rápida de obtención de los datos. Estos valores pico se alcanzarán si las direcciones de las variables están agrupadas convenientemente para su recuperación en grandes bloques. No se considera la carga del CPU del ordenador usado para los test ya que esta no fue significativa.

CONCLUSIONES

Como producto final se obtuvo un manejador para la comunicación con dispositivos de campo a través del protocolo industrial DF1 Allen Bradley, que cumple con los requerimientos definidos previamente para el módulo de Driver del SCADA “Galba”.

Este manejador fue diseñado e implementado en base a una arquitectura multicapas conformada por: Capa de Driver, Capa de Protocolo y Capa de Transporte. En la Capa de Driver se reimplementaron las clases y métodos del DriverCore que permiten que el manejador pueda ser utilizado por el SCADA “Galba” a través de la Interfaz Genérica. En la Capa de Protocolo se definió la lógica de acceso a los dispositivos y la semántica de las tramas, según las especificaciones del protocolo, esta capa puede ser utilizada de forma independiente por cualquier otro sistema que necesite comunicarse con dispositivos que utilicen el estándar DF1. Finalmente en la Capa de Transporte se utilizó la biblioteca TransportProvider, la que provee el transporte serie necesario para la comunicación.

Actualmente este manejador está siendo utilizado de manera satisfactoria por el SCADA “Guardián del Alba” para lograr la comunicación con dispositivos de campo que utilizan el protocolo DF1.

RECOMENDACIONES

Se recomienda incluir en la capa de Driver del manejador, la implementación de la versión Half-Duplex del protocolo de comunicación, usando para esto las clases de la capa de Protocolo que ya se encuentran disponibles para este tipo de comunicación.

BIBLIOGRAFÍA

1. **Electronics, ABC.** Acerca de nosotros: abc electronic. *sitio web de abc electronic.* [En línea] citrus cloud, 2005. [Citado el: 11 de Febrero de 2010.] http://www.automatizacionycnc.com.

2. *Controladores Lógicos Programables.* **Ramírez Cortés, Christian.** Santiago de Chile : Universidad de Chile, 2001.

3. **Lagos, Carolina.** Acerca de nosotros: Revista Electro Industrial. *Revista Electro Industrial.* [En línea] Septiembre de 2006. [Citado el: 11 de Febrero de 2010.] http://www.emb.cl/electroindustria/articulo.mv?xid=562&rank=1.

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

5. **Trujillo, Dr. Rafael.** *Especificación de la interfaz genérica con el SCADA versión 4.1.* Ciudad de la Habana : Entregable según Anexo 13 del convenio Marco PDVSA ALBET, 2007.

6. —. *Especificación de Requerimientos de Software de la Línea de Manejadores de Dispositivos versión 1.3.* Ciudad de la Habana : Entregable según Anexo 13 del convenio Marco PDVSA ALBET, 2006.

7. *Logix5000 Data Access.* **Automation, Rockwell.** s.l. : http://www.ab.com, 2000. 1756-RM005A-EN-E.

8. automatas.org. *sitio web de Autómatas Industriales.* [En línea] 2 de marzo de 2006. [Citado el: 10 de febrero de 2010.] http://www.automatas.org.

9. **Mendoza Jiménez, Ing. Silverio, Guillén García, Ing. Yolanda y Orizaba, Catedráticos I.T.** cetis143.edu.mx. *cetis143.* [En línea] 2007. [Citado el: 10 de Febrero de 2010.] http://www.cetis143.edu.mx/revista/expressa12/plc.pdf.

10. *SCADAs para redes multiautómatas.* **Orcajo Campillo, Francisco J.** 255, Barcelona : s.n., 2004.

11. **Romagosa Cabús, Jaume, Gallego Navarrete, David y Pacheco Porras, Raúl.** *Sistemas SCADA.* Cataluña : ETI, Especialidad en Electrónica Industrial, 2004.

12. **Martinez, Evelio.** eveliux. *eveliux.* [En línea] 11 de Julio de 2007. [Citado el: 12 de Febrero de 2010.] http://www.eveliux.com/mx/modos-simplex-half-duplex-y-full-duplex.php.

13. **Community, Boost, Dawes, Beman y Abrahams, David.** Biblioteca Boost C++. *Biblioteca Boost C++.* [En línea] 7 de junio de 2009. [Citado el: 11 de Febrero de 2010.] http://www.boost.org/.

14. **Kohlhoff, Christopher.** Boost.Asio. *Boost.Asio.* [En línea] Distributed under the Boost Software License, 1 de Febrero de 2010. [Citado el: 12 de Febrero de 2010.] http://www.boost.org/doc/libs/1_42_0/doc/html/boost_asio.html.

15. **Corporation, Nokia.** Qt. *Qt.* [En línea] Nokia, Enero de 2010. [Citado el: 12 de Febrero de 2010.] http://qt.nokia.com/products.

16. **Gomis, Francisco Pastor.** *Librería POSIX-RT.* 2004.

17. **Institute of Electrical and Electronics Engineers, Inc.** *IEEE POSIX Testing Policy General Information.* New York : IEEE, 1998. NJ 08851-1331.

18. **Nourie, Dana.** Getting Started with an Integrated Development Environment (IDE). *Getting Started with an Integrated Development Environment (IDE).* [En línea] Oracle, 25 de Marzo de 2005. [Citado el: 12 de Febrero de 2010.] http://java.sun.com/developer/technicalArticles/tools/intro.html.

19. **Foundation, Eclipse.** Organización Eclipse. *Organización Eclipse.* [En línea] Eclipse Foundation, 2010. [Citado el: 12 de Febrero de 2010.] http://www.eclipse.org.

20. **Corporation, Microsoft.** Microsoft Tech Net. *Microsoft Tech Net.* [En línea] 2010. [Citado el: 12 de Febrero de 2010.] http://technet.microsoft.com.

21. **Búrdalo, Miguel Ángel De Blas.** KDevelop. *KDevelop.* [En línea] KDevelop, 23 de Febrero de 2008. [Citado el: 12 de Febrero de 2010.] http://www.kdevelop.org/.

22. **Millán, A. J.** Zator Systems. *Zator Systems.* [En línea] Zator Systems, 14 de Agosto de 2008. [Citado el: 12 de Febrero de 2010.] http://www.zator.com.

23. **Pupo Polanco, Rosell y González Pérez, Yenier.** *Implementación del componente réplica de base de datos para Akademos v2.0.* Ciudad de la Habana : s.n., 2009.

24. **Moreno Borges, Adrian Carlos y Bridón Danger, Yordanis.** *Interfaz asíncrona para la comunicación con los Controladores Lógicos Programables utilizando el Protocolo Industrial EtherNet/IP.* Ciudad de la Habana : s.n., 2008.

25. **Group, Object Management.** OMG. *OMG.* [En línea] OMG, 15 de Diciembre de 2009. [Citado el: 14 de Febrero de 2010.] http://www.uml.org/.

26. **Cuaresma, Sergi Blanco.** Marble Station. *Marble Station.* [En línea] 2008. [Citado el: 14 de Febrero de 2010.] http://www.marblestation.com/?p=644.

27. **Molpeceres, Alberto.** java Hispano. *java Hispano.* [En línea] Asociación javaHispano, 5 de Marzo de 2003. [Citado el: 15 de Febrero de 2010.] http://www.javahispano.org/contenidos/es/procesos_de_desarrollo/.

28. **Harbour, Michael González.** *Tostadores y POSIX.* Cantabria : Universida de Cantabria, 2006.

29. **Trujillo, Dr. Rafael.** *Guía Implementación IG 5.0.* Mérida : s.n., 2009.

30. **Rockwell Automation.** Rockwell Automation. *Rockwell Automation.* [En línea] Rockwell Automation, 10 de Febrero de 2010. [Citado el: 14 de Febrero de 2010.] http://www.rockwellautomation.com/.

31. *Evaluación comparativa de herramientas CASE para UML desde el punto de vista notacional.* **Gonzalo Génova, Fuster, Miguel Fuentes, José y Valiente Blázquez, María Cruz.** 181, Madrid : Depto. de Informática, Universidad Carlos III, Madrid., 2006.

32. **IEEE Standards Association.** IEEE Std 1471-2000 . *Sitio Web de la IEEE.* [En línea] IEEE, 2000. [Citado el: 15 de Febrero de 2010.] http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html.

33. **Casanovas, Josep.** Usabilidad y arquitectura del software. *Usabilidad y arquitectura del software.* [En línea] desarrolloweb, 9 de Septiembre de 2004. [Citado el: 18 de Febrero de 2010.] http://www.desarrolloweb.com/articulos/1622.php.

34. **Buschmann.** *Pattern Oriented Software Architecture.* s.l. : John Wiley & Sons, 1996. 0471958697.

35. **Pressman, Roger S.** *Ingeneria de Software. Un enfoque practico.* La Habana : Felix Varela, 2005.

36. **Allen Bradley.** *DF1 Protocol and Command set.* s.l. : A-B, 1991. 1770-6.5.16.

37. —. *1785 PLC-5 Programmable Controllers Addressing Reference Manual.* s.l. : A-B. 6200–6.4.6.

ANEXOS

Anexo1: Descripción de la clase AbstractVector

Tabla 19 Descripción de la clase AbstractVector.

Nombre: AbstractVector	
Tipo de clase:	
Atributo	**Tipo**
size	unsigned long
bitSize	unsigned long
capacity	unsigned long
byteSize	unsigned long
buffer	unsigned char*
Para cada responsabilidad:	
Nombre:	AbstractVector()
Descripción:	Constructor de la clase AbstractVector.
Nombre:	getSize()
Descripción:	Devuelve la cantidad de elementos del vector.
Nombre:	getByteSize()
Descripción:	Devuelve la cantidad de byte que ocupa el vector.
Nombre:	getCapacity()
Descripción:	Devuelve la cantidad máxima de elementos del vector.
Nombre:	resize(unsigned long newSize)

Descripción:	Cambia el tamaño del vector.
Nombre:	getInteger(unsigned long index, DeviceVarType varType)
Descripción:	Devuelve un entero ubicado a partir del índice indicado.
Nombre:	getFloat(unsigned long index, DeviceVarType varType)
Descripción:	Devuelve un flotante ubicado a partir del índice indicado.
Nombre:	setInteger(unsigned long index, DeviceVarType varType, long long value)
Descripción:	Almacena un entero en la posición indicada.
Nombre:	setFloat(unsigned long index, DeviceVarType varType, double value)
Descripción:	Almacena un flotante en la posición indicada.
Nombre:	getData ()
Descripción:	Devuelve un arreglo de bytes con los datos del vector.
Nombre:	setEndianness(Endianness value)
Descripción:	Cambia el orden de los bytes al dispositivo.

Anexo2: Descripción de la clase BitVector

Tabla 20 Descripción de la clase BitVector.

Nombre: BitVector	
Tipo de clase:	
Para cada responsabilidad:	
Nombre:	BitVector(unsigned long size)
Descripción:	Constructor de la clase BitVector.
Nombre:	getInteger(unsigned long index, DeviceVarType varType)
Descripción:	Devuelve un entero ubicado a partir del índice indicado.
Nombre:	getFloat(unsigned long index, DeviceVarType varitype)
Descripción:	Devuelve un flotante ubicado a partir del índice indicado.
Nombre:	setInteger(unsigned long index, DeviceVarType varType, long long value)
Descripción:	Almacena un entero en la posición indicada.
Nombre:	setFloat(unsigned long index, DeviceVarType varType, double value)
Descripción:	Almacena un flotante en la posición indicada.
Nombre:	getData()
Descripción:	Devuelve un arreglo de bytes con los datos del vector.
Nombre:	toByte (unsigned long value)
Descripción:	Según la cantidad de elementos q se le introduzcan devuelve la cantidad de bytes que ocupan el total de estos elementos.

Anexo3: Descripción de la clase WordVector

Tabla 21 Descripción de la clase WordVector.

Nombre: WordVector	
Tipo de clase:	
Atributo	**Tipo**
high	int
low	int
first	int
second	int
third	int
forth	int
machineLittleEndian	bool
Para cada responsabilidad:	
Nombre:	WordVector()
Descripción:	Constructor de la clase WordVector.
Nombre:	WordVector(unsigned long size)
Descripción:	Constructor de la clase WordVector.
Nombre:	getInteger(unsigned long index, DeviceVarType varType)
Descripción:	Devuelve un entero ubicado a partir del índice indicado.
Nombre:	getFloat(unsigned long index, DeviceVarType varitype)
Descripción:	Devuelve un flotante ubicado a partir del índice indicado.

Nombre:	setInteger(unsigned long index, DeviceVarType varType, long long value)
Descripción:	Almacena un entero en la posición indicada.
Nombre:	setFloat(unsigned long index, DeviceVarType varType, double value)
Descripción:	Almacena un flotante en la posición indicada.
Nombre:	getData()
Descripción:	Devuelve un arreglo de bytes con los datos del vector.
Nombre:	toByte (unsigned long value)
Descripción:	Según la cantidad de elementos q se le introduzcan devuelve la cantidad de bytes que ocupan el total de estos elementos.
Nombre:	setEndianness(Endianness value)
Descripción:	Cambia el orden de los bytes al dispositivo.

Anexo3: Prueba de lectura de datos analógicos en el “Galba”

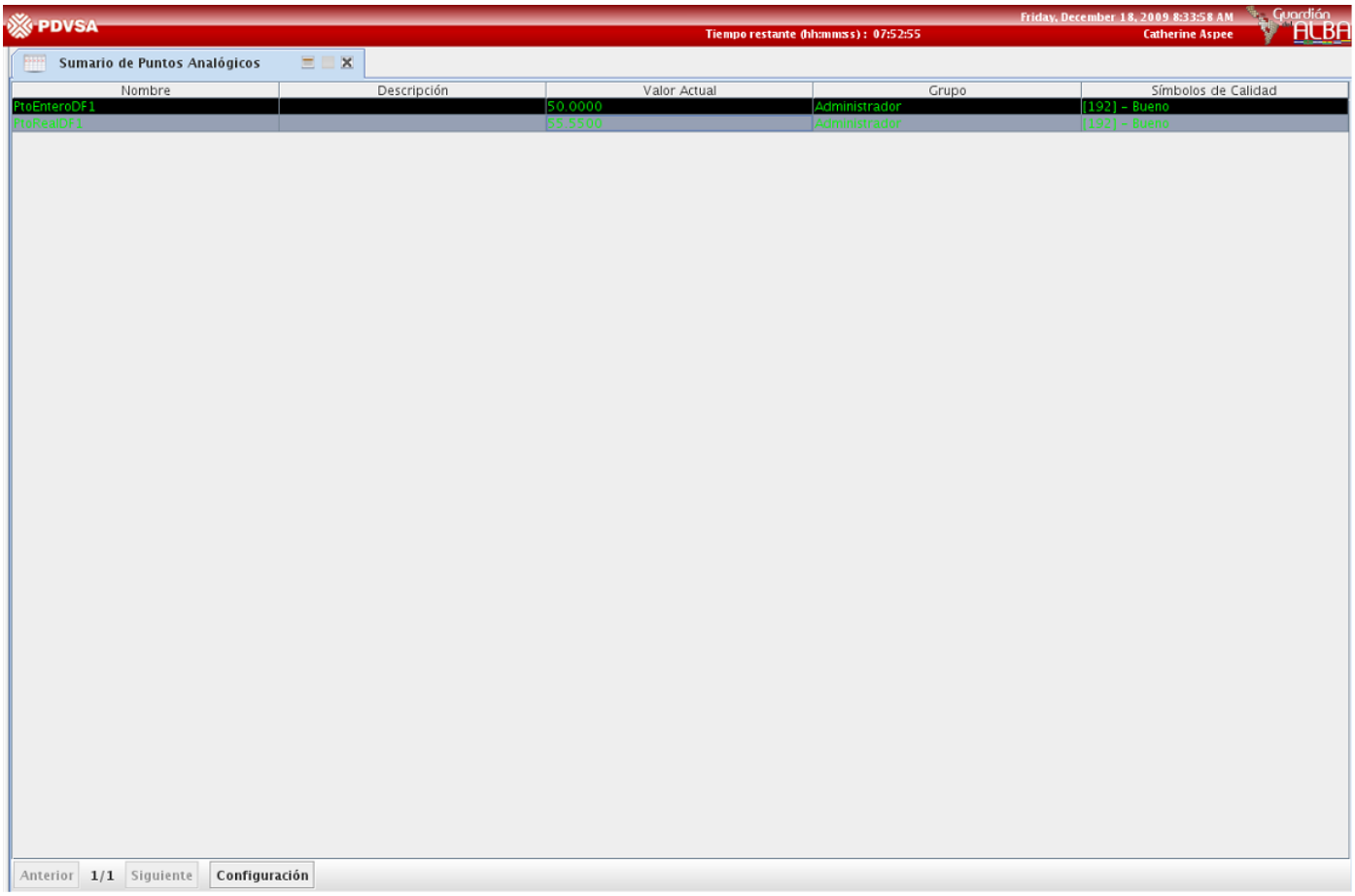

Fig. 28 Prueba de lectura de datos analógicos.

Anexo3: Prueba de calidad de datos analógicos en el “Galba”

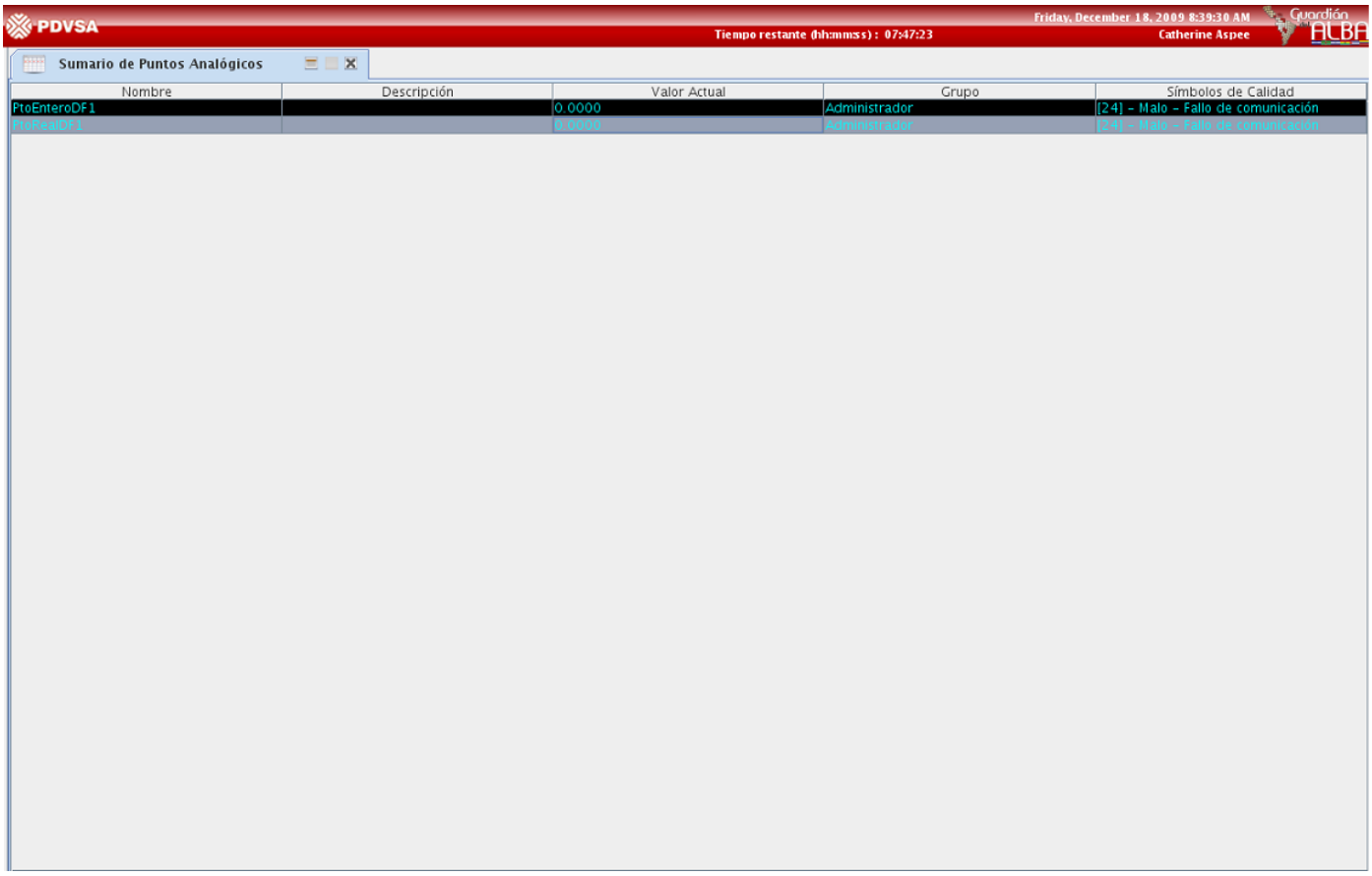

Fig. 29 Prueba de calidad de datos analógicos.

Anexo4: Prueba de lectura de datos.

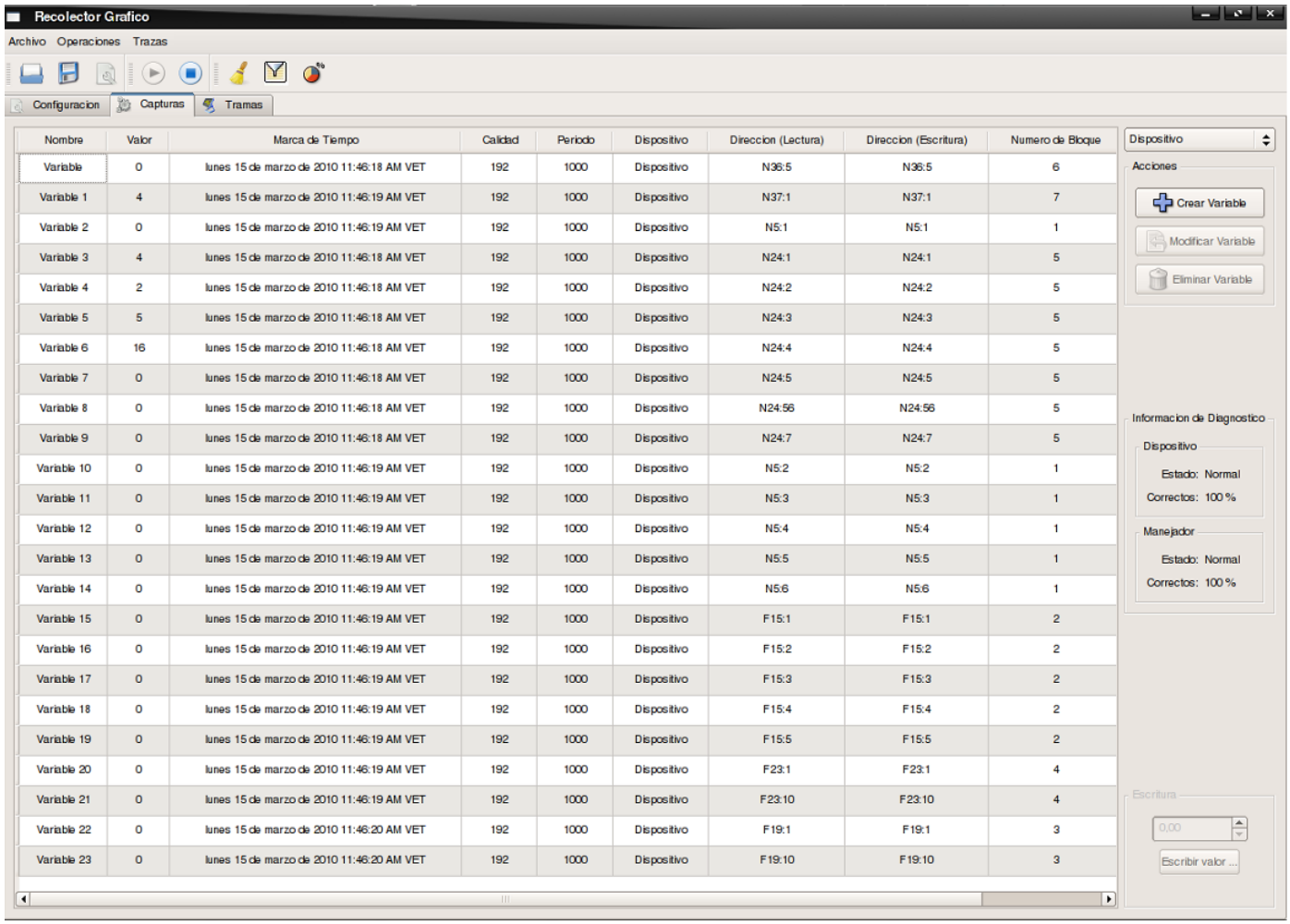

Fig. 30 Prueba de lectura de datos.

GLOSARIO

ASCII: (acrónimo inglés de American Standard Code for Information Interchange). Es un código estándar para el Intercambio de Información. Utiliza 7 bits para representar los caracteres, aunque inicialmente empleaba un bit adicional (bit de paridad) que se usaba para detectar errores en la transmisión.

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

Dispositivo de campo: Son los elementos físicos que miden, monitorean y, en algunos casos almacenan, los datos de las variables del proceso. Estos dispositivos no se conectan directamente al SCADA.

Framework: En los sistemas orientados a objeto un framework es un conjunto de clases que encapsulan diseños abstractos de soluciones a un determinado números de problemas en relación. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

GCC: Conjunto de compiladores creados por el proyecto GNU. GCC es software libre y se distribuye bajo la licencia GPL. Su versión para C++ se denomina G++.

GNU/Linux: El núcleo Linux se complementa con una serie de aplicaciones desarrolladas por el grupo GNU para conformar el sistema operativo software libre GNU/Linux. Linux/GNU es además Multiusuario, Multitarea, Multiprocesador, Multiplataforma, Multilingüe, nacido en la red de redes Internet.

Interfaz: Zona de contacto o conexión entre dos componentes de hardware; entre dos aplicaciones; o entre un usuario y una aplicación. Apariencia externa de una aplicación informática.

Multiplataforma: Término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en Windows en un procesador x86, en GNU/Linux en un procesador x86, y en Mac OS X en uno x86.

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

Software libre: Las cuatro reglas esenciales que deben cumplir las aplicaciones para ser consideradas como software libre son:

Libertad 0: Libertad de ejecutar el programa como quieras.

Libertad 1: Libertad de estudiar el código fuente y cambiarlo para realizar lo que desees.

Libertad 2: Libertad de realizar copias y distribuirlas cuando quieras.

Libertad 3: Libertad de distribuir o publicar versiones modificadas cuando desees.

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