INFORME DE TRABAJO DE GRADO

Gateway para la integración de RTU con protocolos propietarios, a una red de adquisición de datos sobre iFix®

Autor: Luis M. Ortiz L.

Trabajo presentado ante
la ilustre Universidad Central
de Venezuela para optar al titulo
de Ingeniero Electricista.

CARACAS, NOVIEMBRE DE 2003

Dedicatoria:

A Dios y mis padres que son la razón misma de mi existencia.

De manera especial dedico este trabajo a mi madre **Joaquina López de Ortiz**, por guiarme en todas las decisiones que le dan sentido a mi vida.

A mis hermanos con quien siempre puedo contar y confiar en los momentos difíciles y de angustias.

A mis familiares y amigos que verdaderamente se preocuparon de manera desinteresada en prestarme su ayuda para lograr esta meta.

Agradecimientos:

A la Universidad Central de Venezuela por la formación profesional suministrada.

A los profesores que se encargaron impartir el conocimiento adquirido.

A mi tutor Alexander Cepeda por la dedicación e interés mostrado para hacer posible este trabajo.

A mis compañeros de estudios que colaboraron con este desarrollo; José Luis González por la horas de ayuda e instrucción brindadas, Roger Rivero y Aldo Valentini por la ayuda incondicional en todos los aspectos, Angelo Parisi, Hector Arcia, Soiram Silva, Fausto Longo, Ferdinando Ciavatini, Andreina, Victor y los que no nombro pero formaron parte de este trabajo.

AUTOR:	Luis Miguel Ortiz López
TITULO ACADÉMICO A OBTENER:	"Ingeniero Electricista" Mención Industrial
UNIVERSIDAD, FECHA:	Universidad Central de Venezuela 2003-10-30
TUTOR ACADEMICO:	Ing. Alexander Cepeda Q., M. Sc.
TITULO DEL TRABAJO:	**"Gateway para la integración de RTU con protocolos propietarios, a una red de adquisición de datos sobre iFix®"**

RESUMEN:

Este trabajo se realizó en el marco de la integración de dispositivos de campo (RTU) con protocolos propietarios, a una red de adquisición de datos sobre iFIX, para un desarrollo SCADA en la Ciudad Universitaria de Caracas de la Universidad Central de Venezuela. Se desea implementar un laboratorio de simulación y supervisión de sistemas de distribución, para monitorear el comportamiento de la Ciudad Universitaria de Caracas.

Las RTU poseen capacidad de comunicación bajo protocolos de comunicación propietarios, a los cuales es imposible acceder con las herramientas disponibles por el sistema SCADA. Es por esto que surgió la necesidad de desarrollar una interfase tipo Gateway, capaz de convertir dichos protocolos propietarios en un protocolo de comunicaciones que el sistema SCADA reconozca y opere.

La primera parte constó del desarrollo de una interfase tipo Gateway bajo protocolo MODBUS TCP, cuya capacidad comprobada y amplia difusión hacen de éste un protocolo robusto y versátil para las redes de campo.

La segunda parte abarcó el desarrollo de una interfase tipo Gateway similar a la anterior, bajo protocolo de comunicación para redes de distribución eléctrica de arquitectura abierta, robusta, eficiente y de dominio público, denominado DNP (*Distributed Network protocol*) basado en el estándar IEC 870-5.

Las interfaces fueron desarrolladas en lenguaje C^{++} compatible con cualquier sistema operativo Windows con núcleo NT, usando conceptos de multitarea, API WIN32 para comunicaciones seriales, API WINSOCKETS para desarrollo de servicios de red; todo ello enmarcado en un modelo de tipo modular, mostrando procesos dependientes del usuario (interfaz) y procesos independientes (transparentes para el usuario) en segundo plano. El diseño de la interfaz de usuario se basó en controles gráficos tipo Windows, orientado a la plataforma final de trabajo bajo iFIX.

Las interfaces desarrolladas cuentan con un sistema adicional, especializado en la detección automática de las RTU bajo estudio, usando un proceso de exploración sucesiva por sondeo.

Con base a lo desarrollado y las pruebas de laboratorio realizadas, se comprobó la factibilidad de introducir RTUs con protocolos diferentes en un mismo bus de campo, incrementando la flexibilidad en la instalación de estos equipos y por ende disminuyendo el grado de obsolescencia de una red de campo conformada por RTU como las estudiadas.

TABLA DE CONTENIDO:

Glosario.

- **ActiveX.** Son objetos de programación que encapsulan propiedades y ejecutan tareas específicas, son de uso frecuente por parte de los programadores.
- **AI.** Entrada de tipo analógico.
- **ANSI.** Instituto Nacional Americano de Estándares.
- **AO.** Salida de tipo analógico.
- **API.** Interfase para la programación de aplicaciones.
- **ASCII.** Estándar para la codificación de caracteres en algunos sistemas de comunicación de datos y sistemas operativos.
- **ASDU.** Unidad formada por un conjunto de bytes de dato de la capa de aplicación.
- **ASN1.** Notación abstracta de sintaxis para la capa de presentación de OSI.
- **ATM.** Protocolo o modo de transferencia asíncrona de la capa de enlace de datos de OSI.
- **BER.** Reglas básicas de codificación de la capa de presentación de OSI.
- **Bits.** Unidad de transmisión más pequeña utilizada en comunicación de datos digitales. Representa un estado binario lógico.
- **Bluetooth.** Sistema de comunicación inalámbrico para las comunicaciones cortas.
- **Borland.** Industria para desarrollo de compiladores y entornos de programación.
- **Bridge.** Puente de enlace entre dos redes distintas, la unión se presenta en la capa de enlace de OSI.
- **BSAP.** Protocolo de comunicación de datos en redes de campo.
- **Bus.** Topología de red donde todos los dispositivos se conectan de manera paralela a la misma línea de comunicación.
- **Buses.** Conjunto de varios bus de campo, comúnmente se utiliza para describir arquitecturas basadas en topologías tipo bus.
- **Byte.** Conjunto de bits en un orden específico.
- **C++ Builder.** Entorno de programación de aplicaciones bajo Windows.
- **Caracteres.** Conjunto de bytes alfanuméricos que respetan una codificación específica.
- **Checksum.** Tipo de chequeo de error usado en comunicación de datos.
- **Citect.** Software para sistemas SCADA bajo plataformas INTEL.
- **COM.** Tecnología basada en el desarrollo de componentes de tipo objeto.
- **CPU.** Unidad de procesamiento central de datos, es la unidad que ejecuta las instrucciones de un programa.
- **CRC.** Tipo de chequeo de error usado en comunicación de datos.
- **CSMA/CD.** Estándar para la comunicación en la capa de enlace contenido en IEEE 802.3.
- **CUC.** Ciudad Universitaria de Caracas (Sede principal de la Universidad Central de Venezuela).
- **DB.** Nombre genérico para conectores de tipo serial.
- **DCE.** Equipos de comunicación o terminación de circuito de datos.
- **DCS.** Sistemas de control distribuido.
- **DEC Alfa.** Arquitectura de computadores de Digital Equipment Corporation (DEC).
- **DI.** Entrada de tipo digital.

- **DNP.** Protocolo de comunicación para redes distribuidas.
- **DO.** Salida de tipo digital.
- **DOS.** Sistema operativo para PC de mucha difusión en los años 70 y 80.
- **DPLL.** Lazo de fijación de fase digital, dispositivo utilizado para transmisión síncrona.
- **Driver.** Software especializado para integración y control de sistemas, algunas veces funciona como una interfase entre dos o mas aplicaciones.
- **DTE.** Equipos terminales de datos.
- **DUCATI.** Marca registrada de productos en diversos rubros.
- **DUCBUS.** Protocolo de comunicación para redes seriales de productos DUCATI Energía S.P.A..
- **EIA.** Asociación de industrias electrónicas.
- **EPROM.** Memoria de programa programable eléctricamente.
- **Ethernet.** Estándar internacional para redes de datos, comprende su arquitectura y normas.
- **ETX.** Caracter ASCII de fin de trama.
- **FDDI.** Sistema de interfaz de red, en configuración simple o doble anillo, con paso de testigo.
- **FLASH ROM.** Memoria de programa con capacidad de programarse un numero determinado de veces, estas se programan y borran eléctricamente.
- **Front End.** Sistemas pertenecientes a una red de datos que permite unificar formatos de datos.
- **FSK.** Método para codificación por conmutación de frecuencia, usado en sistemas de transmisión.
- **FTP.** Protocolo de servicio de transferencia de archivos.
- **Full duplex.** Método utilizado para la comunicación donde se transmite y recibe de manera simultánea.
- **Gateway.** Interfase entre dos sistemas de comunicación de datos diferentes, de arquitecturas diferentes o protocolos diferentes.
- **Half duplex.** Método utilizado para la comunicación donde se transmite en ambas direcciones pero no simultáneamente.
- **Hamming.** Códigos especiales para aplicación de métodos de corrección de errores.
- **Handle.** Manejador de una aplicación en Windows, identifica a esta.
- **Hardware.** Dispositivos físicos que conforman una plataforma determinada.
- **HMI.** Interfaz ser humano – maquina, forma de interactuar el usuario con una aplicación SCADA.
- **HTTP.** Protocolo de servicio para transferencia de hipertexto o paginas Web.
- **Hub.** Sistema concentrador de datos en redes de datos.
- **IBM.** Corporación Industrial fabricante de soluciones de PC y hardware electrónico.
- **IEC.** Comisión para industrias electrotécnicas.
- **IEEE.** Instituto de ingenieros eléctricos y electrónicos.
- **iFIX.** Herramienta computacional de tipo software utilizada para implementar sistemas de control y adquisición de datos, es desarrollada por Intellution Dynamics.
- **Intel.** Desarrolladores de sistemas de hardware, electrónicos e integrados.
- **Intellution Dynamics.** Compañía desarrolladora de software para sistemas de control y adquisición de datos.

- **IP.** Protocolo de Internet.
- **ISO.** Organización internacional para la estandarización.
- **ITU-T.** Unión internacional de telecomunicaciones y estándares para este sector, antigua CCITT.
- **KEPware.** Software para sistemas SCADA bajo plataformas INTEL.
- **Kernel.** Núcleo o matriz que controla a un sistema operativo.
- **LAN.** Redes de área local.
- **Landline.** Sistemas de transmisión de datos a través de líneas terrestres.
- **LAPB.** Procedimiento de acceso de enlace balanceado, usado en la capa de enlace de datos de OSI.
- **LIFASA.** Marca registrada de equipos eléctricos y electrónicos.
- **LLC.** Control de enlace lógico.
- **Login.** Palabra alfanumérica que determina en un sistema a un único usuario o grupo de ellos. Comúnmente usado para seguridad en sistemas.
- **LPC.** Proceso de llamado a control de tipo local para pasar mensajes entre aplicaciones de Windows.
- **LRC.** Tipo de chequeo de error usado en comunicación de datos.
- **MAC.** Control de acceso al medio.
- **MACH 30.** Analizador de redes eléctricas fabricado por DUCATI.
- **Macintosh.** Industria fabricante de soluciones de tipo PC.
- **Manchester.** Codificación de los datos usada en redes Ethernet.
- **MAP.** Protocolo de comunicación del nivel de aplicación de OSI.
- **Master.** Dispositivo que funciona como maestro de un SCADA.
- **MCA.** Analizador de redes eléctricas fabricado por LIFASA.
- **Microsoft.** Industria desarrolladora de software y SO.
- **Mirror.** Sistema paralelo al SCADA capaz de asumir el control en caso de falla del sistema. También se le conoce como sistema redundante.
- **MMS.** Protocolo de comunicación de la capa de aplicación de OSI.
- **MODBUS ASCII.** Protocolo de comunicación para redes seriales.
- **MODBUS RTU.** Protocolo de comunicación para redes seriales.
- **MODBUS TCP.** Protocolo de comunicación para redes TCP/IP.
- **MTU.** Unidad terminal maestra.
- **NDOS.** Entorno de soporte o interfaz para DOS.
- **ODBC.** Sistema para interconectividad de bases de datos abiertas.
- **OPC.** Tecnología orientada al control a través de componentes de tipo objeto.
- **OS2.** Sistema operativo de alta difusión.
- **OSI.** Estándar para la interconexión de sistemas abiertos.
- **PC.** Computador personal.
- **PC104.** Sistemas modulares armables y compatibles para soluciones de hardware.
- **PCTOOLS.** Entorno de soporte o interfaz para DOS.
- **PDM.** Sistema de transmisión por modulación por duración de pulsos.
- **PDP11.** Hardware para SCADA de los años 70 de Digital Equipment Corporation.
- **PLC.** Controlador lógico programable.

- **Polling.** Método para interrogar dispositivos por sondeo, comúnmente usado en sistemas maestro – esclavo.
- **POO.** Programación Orientada a Objetos.
- **POST.** Auto prueba de encendido o autodiagnóstico que emplean los PC al encender.
- **QNX.** Estaciones de trabajo que funcionan bajo UNIX.
- **QOS.** Calidad de servicio para redes de datos.
- **RAM.** Memoria de acceso aleatorio, usada para escribir y leer datos provisionales.
- **RJ.** Nombre genérico para conectores telefónicos.
- **ROM.** Memoria de solo lectura, es la memoria donde reside el código de programa.
- **Router.** Puente de enlace entre dos redes distintas, la unión se presenta en la capa de red de OSI.
- **RPC.** (Remote Process Control) Cumple funciones entre procesos, por ejemplo, pasar mensajes entre un cliente y un servidor bajo Windows.
- **RS232.** Estándar internacional para las comunicaciones seriales.
- **RS485.** Estándar internacional para las comunicaciones seriales.
- **RSX11M.** Software tipo sistema operativo para SCADA de los años 70. de Digital Equipment Corporation.
- **RTS.** Solicitud de envío de algunos sistemas de transmisión.
- **RTU.** Unidades terminales remotas o dispositivos de campo.
- **SCADA.** Del ingles Supervisory Control and Data Acquisition, Sistema de control y adquisición de datos.
- **Shell.** Interfaz de usuario o entorno de un sistema operativo.
- **Single board.** Sistemas simples formados por un solo módulo de hardware.
- **Slew Rate.** Medida de la tasa de rapidez con la que cambia una señal binaria.
- **SMTP.** Protocolo de servicio de transmisión de correo simple.
- **SO.** Sistema operativo.
- **Sockets.** Estructura virtual para la conexión entre entidades clientes y servidoras.
- **Software.** Programa de computadora de tipo aplicación.
- **Solaris.** Marca de trabajo que funcionan bajo UNIX.
- **SONET.** Sistema de transmisión de la capa física que emplea una línea óptica sincrónica.
- **Stand-alone.** Sistemas independientes de control y adquisición con capacidad de comunicarse con un SCADA.
- **STP.** Tipo de cable apantallado usado en las comunicaciones.
- **STX.** Caracter ASCII de comienzo de trama.
- **TCP.** Protocolo control de transmisión.
- **TCP/IP.** Arquitectura de red de gran difusión a nivel mundial.
- **Telnet.** Aplicación para servicio de correo en Internet.
- **Timeouts.** Tiempos de espera para una transacción en un sistema.
- **Token bus.** Estándar para la comunicación en la capa de enlace contenido en IEEE 802.4.
- **Token ring.** Estándar para la comunicación en la capa de enlace contenido en IEEE 802.5.
- **Trama.** Conjunto de bytes de datos o cadena de caracteres.

- **UART.** Transmisor y receptor asincrónico universal.
- **UCV.** Universidad Central de Venezuela.
- **UDP.** Protocolo para datagramas de usuario.
- **UNIX.** Sistema operativo para plataformas de mucha difusión.
- **UTP.** Tipo de cable sin apantallar usado en las comunicaciones.
- **VBA.** Tecnología para desarrollo de aplicaciones en visual basic.
- **Visual Basic.** Entorno de programación bajo Windows.
- **VME.** Configuración industrial para equipos tipo PC.
- **VRC.** Tipo de chequeo de error usado en comunicación de datos.
- **VSAT.** Sistema de transmisión satelital.
- **WAN.** Redes de área amplia.
- **Watchdog timer.** Temporizador que detecta malos funcionamientos, fundamentalmente sirve para resetear el funcionamiento en unidades programables.
- **Wi-Fi.** Sistemas de transmisión en alta frecuencia.
- **Windows.** Sistema operativo de gran difusión mundial.
- **Wonderware.** Software para sistemas SCADA bajo plataformas INTEL.
- **X25.** Protocolo de comunicación de la capa de enlace de datos de OSI.
- **XTG.** Entorno de soporte o interfaz para DOS.

INDICE DE TABLAS:

INDICE DE FIGURAS:

1 INTRODUCCIÓN.

El departamento de Potencia de la Escuela de Ingeniería Eléctrica de la Universidad Central de Venezuela (UCV) se propuso la utilización de la red de distribución de energía eléctrica de la Ciudad Universitaria de Caracas de la UCV (CUC) como un laboratorio nacional de supervisión y simulación de sistemas eléctricos de distribución. La demanda de energía eléctrica de la CUC, presenta comportamientos similares a los perfiles típicamente registrados en clientes o sectores comerciales con altas densidades de carga, lo que le confiere a la CUC una posición privilegiada en cuanto a representatividad de la mayoría de los suscriptores, constituyéndose naturalmente en un campo de experimentación válido para todos los estudios de redes de distribución eléctrica que abarcan aspectos tales como: prueba de equipos, de diseño y prueba de programas (software) asociados a sistemas de distribución; prueba de sistemas de supervisión de carga, de políticas de gerencia de carga, de uso eficiente y de ahorro de energía, así como de sistemas llamados de edificios inteligentes; prueba de equipos destinados a la fiscalización de la calidad de la energía; de entrenamiento de personal en la supervisión de los sistemas de distribución, de adiestramiento de personal para la fiscalización de la calidad de la energía; así como, de desarrollo de software para la supervisión y la simulación de estos sistemas. En consecuencia, la implantación de un laboratorio de supervisión y simulación de sistemas de distribución, permitirá que los cursantes de estudios de pregrado y postgrado puedan aplicar y consolidar los conocimientos adquiridos en las diferentes asignaturas vinculadas a la Ingeniería Eléctrica tanto en las áreas de potencia, industrial y electrónica como las correspondientes a sistemas de redes de comunicación de datos.

Para la etapa inicial se han desarrollado o están en desarrollo diferentes trabajos que abarcaron la documentación y actualización de la red eléctrica de la CUC; la ingeniería conceptual del sistema de supervisión (SCADA) asociado al laboratorio nacional, la cual prevé una red de campo sobre el protocolo de comunicación para redes de distribución eléctrica de arquitectura abierta, robusto, eficiente y de dominio público, denominado DNP 3 (Distributed Network Protocol, basado en el estándar IEC 870-5); la ingeniería conceptual del nodo de comunicación correspondiente al SCADA, donde se ha definido parte de la ingeniería básica correspondiente a la red de campo a ser conformada por unidades terminales remotas de medición y transferencia de datos (RTU ó remotas) compatible con el protocolo de comunicación DNP, seudo protocolo MODBUS ó protocolos propietarios tales como DUCBUS de DUCATI, sobre redes de campo de arquitecturas compatibles con comunicación serial a través de la red telefónica existente en la CUC; la ingeniería básica y detalle correspondiente a los nodos de adquisición de datos y la base de datos en línea, implementado con el sistema de adquisición de datos iFIX de INTELLUTION, sobre una plataforma bajo arquitectura ETHERNET a través del protocolo TCP/IP; así como la base de datos de estudio y sus correspondientes interfases hombre máquina (HMI).

Para la implantación de la red de campo se cuenta en la actualidad con una veintena de remotas con capacidad de comunicación serial RS485 (tales como la modelo MACH 30, marca DUCATI y la MCA marca LIFASA); sin embargo, para la integración de la red de campo, conformada por remotas de diferentes protocolos de comunicación (DUCBUS, MODBUS y DNP), con el nodo de adquisición (sobre IFIX) es necesario que el nodo de comunicación realice la respectiva conversión de protocolos de comunicación (GATEWAY).

INTRODUCCIÓN.

Este trabajo está enmarcado dentro de los lineamientos previstos para la implantación y puesta en servicio de una red de medición distribuida en el campus universitario a través de unidades remotas de medición (RTU) con capacidad de comunicación a través del protocolo DNP, y con capacidad de integrar a subredes o nodos de medición preexistentes sobre protocolos de comunicación como DUCBUS y MODBUS; esta red de medición será capaz de interactuar con el nodo de adquisición de datos implementado sobre una plataforma iFIX de INTELLUTION DYNAMICS, a través de un nodo de comunicación sobre arquitectura ETHERNET con protocolo de comunicación TCP/IP.

Se presenta un marco teórico referencial, donde se establecen los conceptos involucrados en el desarrollo. Se manejan conceptos de SCADA, comunicación de datos, modelo OSI y sistemas operativos tipo Windows con núcleo NT.

Luego se presenta una descripción de los equipos (hardware) asociados al desarrollo, involucrando los equipos del sistema SCADA, dispositivos de campo, conexiones físicas entre dispositivos y soporte de hardware para la interfaz desarrollada.

Se realiza una descripción detallada de los programas involucrados en el desarrollo; software de programación, software de SCADA y software para diversas pruebas. Incluyendo las propiedades, procedencia y funciones en cada caso.

El desarrollo se dividió en cuatro etapas distintas: La primera etapa comprende las comunicaciones seriales a través del uso de la herramienta de programación conocida como API WIN32 para comunicaciones serie; para desarrollar un modulo independiente especializado que realice las funciones requeridas por la interfase. La segunda comprende el desarrollo de comunicaciones a través de Internet, usando programación con la herramienta Windows Sockets, para desarrollar un sistema de tipo servidor con filosofía Cliente – Servidor. La tercera involucra el desarrollo de la interfase tipo GATEWAY bajo protocolo de red MODBUS sobre TCP y la cuarta comprende un desarrollo similar pero con protocolo de red encapado DNP 3.0 sobre TCP.

El objetivo de este trabajo tendrá como alcance el desarrollo y puesta en servicio de dos GATEWAY multi-protocolo prototipo, tomando como base tanto la instrumentación existente de remotas y la arquitectura inicialmente prevista para la red de campo (red telefónica, a través de puertos seriales RS 485, sobre protocolos DUCBUS y MODBUS), la instrumentación futura de remotas (con capacidad de comunicación sobre protocolo DNP) como las características propias del programa de adquisición y control de datos y la arquitectura de la red computacional (iFIX sobre red ETHERNET compatible con TCP/IP) asociados al nodo de adquisición. El GATEWAY prototipo se realizará sobre una plataforma conformada por un computador personal, periféricos de comunicación y software desarrollado para la ejecución de la función de conversión de protocolos antes descritos.

El desarrollo consta básicamente de dos Gateway prototipo. El primero una interfase capaz de comunicarse con un maestro bajo protocolo MODBUS TCP e interactuar con dispositivos de campo bajo protocolos seriales MODBUS RTU Y DUCBUS. El segundo otra interfase capaz de comunicarse con un maestro de red bajo DNP sobre TCP y realizar la recolección de los datos contenidos en los dispositivos de campo (protocolos MODBUS RTU y DUCBUS), este último con la idea de poder anexar la etapa inicial del SCADA a una red de adquisición de datos sobre un nodo DNP3.

2 MARCO TEÓRICO.

2.1 Sistemas SCADA.

2.1.1 Definición General de SCADA.

SCADA (supervisory control and data acquisition) es un sistema industrial de mediciones y control que consiste en una computadora principal o "Master" (generalmente llamada Estación Maestra, Master Terminal Unit o MTU); una o más unidades de control obteniendo datos de campo (generalmente llamadas estaciones remotas, Remote Terminal Units, o RTU's); y una colección de software estándar a la medida, usado para monitorear y controlar remotamente dispositivos de campo. Los sistemas SCADA contemporáneos exhiben predominantemente características de control a lazo abierto y utilizan comunicaciones generalmente interurbanas, aunque algunos elementos de control a lazo cerrado de comunicaciones de corta distancia pueden también estar presentes [1].

Sistemas similares a un sistema SCADA son vistos rutinariamente en fábricas, plantas de tratamiento, etc. Éstos son llamados a menudo como ***Sistemas de Control Distribuidos*** (DCS - Distributed Control Systems). Tienen funciones similares a los sistemas SCADA, pero las unidades de colección o de control de datos de campo se establecen generalmente dentro de un área confinada. Las comunicaciones pueden ser vía una red de área local (LAN – Local Area Network), y son normalmente confiables y de alta velocidad. Un sistema DCS emplea generalmente cantidades significativas de control a lazo cerrado. Por esto se puede considerar un DCS como un caso especifico de SCADA.

Un sistema SCADA por otra parte, generalmente cubre áreas geográficas más extensas, y normalmente depende de una variedad de sistemas de comunicación menos confiables que una LAN. El control a lazo cerrado en esta situación es menos deseable.

Un sistema SCADA se utiliza para vigilar y controlar la planta industrial o el equipamiento. El control puede ser automático, o iniciado por comandos de operador. La adquisición de datos es lograda en primer lugar por los RTU's que exploran las entradas de información de campo conectadas con ellos. Esto se hace generalmente a intervalos muy cortos de tiempo. La MTU entonces explorará los RTU's generalmente con una frecuencia menor. Los datos se procesan para detectar condiciones de alarma, y si una alarma estuviera presente, sería catalogada y visualizada en listas especiales de alarmas. Los datos pueden ser de tres tipos principales:

- **Datos analógicos** (por ejemplo números reales) que quizás sean presentados en gráficos.
- **Datos digitales** (on/off) que pueden tener alarmas asociadas a un estado o al otro.
- **Datos de pulsos** (por ejemplo conteo de revoluciones de un medidor) que serán normalmente contabilizados o acumulados.

La interfaz primaria al operador es una pantalla que muestra una representación de la planta o del equipamiento en forma gráfica. Los datos vivos (dispositivos) se muestran como dibujos o esquemas en primer plano sobre un fondo estático. Mientras los datos cambian en campo, el primer plano es actualizado (una válvula se puede mostrar como abierta o cerrada, etc.). Los datos analógicos se pueden mostrar como números, o gráficamente (esquema de un tanque con su nivel de líquido almacenado). El sistema puede tener muchas de tales pantallas, y el operador puede seleccionar los más relevantes en cualquier momento.

2.1.2 Unidades Maestras (MTU - Master Terminal Unit).

La parte más visible de un sistema SCADA es la estación central o MTU. Éste es el "centro neurálgico" del sistema, y es el componente del cual el personal de operaciones se valdrá para ver la mayoría de la planta. Una MTU a veces se llama HMI (Human Machine Interface), interfaz ser humano – máquina, aunque comprenden conceptos distintos[2].

2.1.2.1 Características de las Unidades Maestras

Todas las MTU de SCADA deben presentar una serie de características, algunas de estas son las siguientes:

- **Adquisición de datos:** Recolección de datos de las unidades terminales remotas (RTU's)
- **Gráficos de tendencia:** Salvar los datos en una base de datos, y ponerlos a disposición de los operadores en forma de históricos.
- **Procesamiento de Alarmas:** Analizar los datos recogidos de los RTU's para ver si han ocurrido condiciones anormales, y alertar a personal de operaciones sobre las mismas.
- **Control:** Control a Lazo Cerrado, e iniciados por operador.
- **Visualizaciones:** Gráficos del equipamiento actualizado para reflejar datos del campo.
- **Informes:** La mayoría de los sistemas SCADA tienen un ordenador dedicado a la producción de reportes conectado en red (LAN o similar) con el principal.
- **Mantenimiento del Sistema Espejo:** Se debe mantener un sistema idéntico con la capacidad segura de asumir el control inmediatamente si el principal falla.
- **Interfaces con otros sistemas:** Transferencia de datos hacia y desde otros sistemas corporativos para, por ejemplo, el procesamiento de órdenes de trabajo, de compra, la actualización de bases de datos, etc.
- **Seguridad:** Control de acceso a los distintos componentes del sistema.
- **Administración de la red:** Monitoreo de la red de comunicaciones.
- **Administración de la Base de datos:** Agregar nuevas estaciones, puntos, gráficos, puntos de cambio de alarmas, y en general, reconfigurar el sistema.
- **Aplicaciones especiales:** Casi todos los sistemas SCADA tendrán cierto software de aplicación especial, asociado generalmente al monitoreo y al control de la planta específica en la cual se está utilizando.
- **Sistemas expertos, sistemas de modelado:** Los más avanzados pueden incluir sistemas expertos incorporados, o capacidad de modelado de datos.

2.1.2.2 Hardware y Software.

Las MTU de sistemas SCADA se pueden implementar en la mayoría de las plataformas existentes. Los primeros sistemas existentes tendieron a ser propietarios y muy especializados; y donde fueron utilizados sistemas operativos de fines generales, tendieron a ser modificados ampliamente. Esto debido a que los requisitos de SCADA superaban los límites de la tecnología disponible en el momento, y por razones de desempeño, ya que tendieron a proporcionar sistemas gráficos por encargo, a usar bases de datos en tiempo real

(con gran parte de la base de datos en memoria), y a menudo el hardware debió ser modificado para estos requisitos particulares.

La serie Digital Equipment Corporation PDP11 y el sistema operativo RSX11M eran quizás la plataforma más común en los SCADA de los años 70 y principios de los 80. Posteriormente, UNIX comenzó a ser el sistema operativo de más frecuente elección.

Mientras la potencia de la PC aumentaba, los sistemas Intel llegaron a ser muy comunes, aunque las plataformas DEC Alfa, y otras estaciones de trabajo de fines elevados estén aún en uso. En épocas recientes Windows NT ha alcanzado alta aceptación dentro de la comunidad SCADA, aunque los sistemas muy grandes siguen siendo en la mayor parte de los casos estaciones de trabajo UNIX (QNX o Solaris), las cuales son más veloces en sus respuestas.

Actualmente la industria se está desarrollando claramente hacia estándares abiertos: ODBC, INTEL PCs, sistemas estándares de gráficos, e ínter conectividad a sistemas de computación corrientes. En años recientes ha aparecido en el mercado un importante número de sistemas SCADA sobre plataformas INTEL PC, ya que éstas están aumentando rápidamente su capacidad y desempeño. Ejemplos de ellos son Citect, FIX de Intellution, KEPWARE y WONDERWARE.

2.1.2.3 Adquisición de datos

La función de adquisición de datos de un sistema SCADA es obviamente una función preponderante. Hay un número de características asociadas a la adquisición de datos.

2.1.2.3.1 Interrogación, informes por excepción, y transmisiones iniciadas por RTU's.

Los primeros sistemas SCADA tenían RTU's muy simples y el sistema central debía utilizar un sistema de interrogación (polling) para tener acceso a sus datos. El Master controlaba todas las comunicaciones, y un RTU nunca respondía a menos que fuese interrogado. El Master preguntaba así a cada RTU alternadamente, pidiendo que le envíen sus datos.

La RTU haría lo necesario para recuperar los últimos datos de sus instrumentos (además de la conversión de señales analógicas a digitales) y después contestaría a la petición del Master. Al ser controladas las comunicaciones por el Master, éste registraba los datos con la hora de recepción, muchas veces muy distinta a la hora en que fueron generados.

Algunas variaciones en esto se han introducido para mejorar la eficacia de comunicaciones. El Master podía solicitar solamente algunos de los datos de un RTU en cada encuesta principal, y extraería los datos menos importantes en una segunda encuesta disparada con una frecuencia más baja.

Con RTU's más inteligentes, se podían explorar independientemente entradas de información, sobre una base continua, e incluso agrupar por hora los datos. El Master entonces preguntaría al RTU si tiene alguna cosa para informar. Si nada hubiera cambiado desde la vez última, la RTU respondería sin novedad, y el Master se movería al RTU siguiente. Para asegurarse de que un cierto acontecimiento no fue salteado, ocasionalmente el Master haría una encuesta completa como un chequeo de salud.

En otros casos, la RTU pudiera iniciar la transacción sin una solicitud previa del maestro, lo que se denomina reporte por excepción. Está claro lo que implica cuando una entrada de información digital ha cambiado, pero el uso del informe por excepción con valores analógicos significa que un cierto cambio del umbral está definido (típicamente 1-2%), y sobre éste se ha producido algún cambio.

El informe por excepción puede reducir dramáticamente el tráfico de comunicaciones, siempre y cuando los datos estén cambiando en forma relativamente lenta. Cuando se están midiendo parámetros altamente volátiles puede aumentar drásticamente el tráfico. En este caso una solución es poner estos parámetros volátiles en una encuesta rutinaria, sacrificando una cierta exactitud en la hora de registro en pos de la reducción del tráfico.

El acercamiento más sofisticado es permitir que el RTU reporte por excepción sin la encuesta previa por parte del Master. Esto significa que el sistema de comunicaciones no se está utilizando para las repetidas encuestas con sin novedad siendo esta la respuesta más frecuente. Esto permite que un sistema típico controle muchos más RTU's con la misma anchura de banda de comunicaciones. Como los asuntos asociados con parámetros altamente volátiles todavía existen, un chequeo de salud sigue siendo necesario, de otro modo un RTU podría salir de servicio y el sistema nunca se daría por enterado, esta constituye la más grande debilidad de este modo. Para utilizar esta técnica, el protocolo de comunicación debe tener la capacidad de proporcionar las direcciones de destino del mensaje, y de la fuente del mismo.

Este sistema también implica que dos RTU's pueden transmitir simultáneamente, interfiriendo uno con otro. Un sistema SCADA normalmente repetirá la transmisión si no recibe un acuse de recibo dentro de cierto tiempo. Si interfieren dos RTU's transmitiendo simultáneamente, y, luego si ambos poseen el mismo tiempo de reenvío, interferirán otra vez. Por esta razón, el acercamiento típico es repetir el envío después de un período aleatorio seleccionado.

El uso de timeouts (Tiempos de espera) al azar puede no ser suficiente cuando por ejemplo ha habido un apagón extenso. Incluso con recomprobaciones al azar, puede haber tanto tráfico que el RTU todavía no podrá conseguir realizar la transmisión. Por esta razón una mejora que es deseable es que después de varios intentos, el período de recomprobación se fije en un tiempo fijo.

2.1.2.3.2 Manejo de fallas de comunicaciones

Un sistema SCADA debe ser confiable. Los sistemas de comunicación para los sistemas SCADA se han desarrollado para manejar comunicaciones pobres de una manera predecible. Esto es importante donde está implicado el control, podría ser desastroso si las fallas de comunicaciones causaran que el sistema SCADA hiciera funcionar inadvertidamente el sector incorrecto de la planta.

Los sistemas SCADA hacen uso típicamente de las técnicas tradicionales de la paridad, del chequeo de sumas polinómicas, códigos de Hamming y demás. Sin embargo no confían simplemente en estas técnicas. La operación normal para un sistema SCADA, es esperar siempre, que cada transmisión sea reconocida. El sistema de interrogación que emplea tiene seguridad incorporada, en la que cada estación externa está controlada y debe periódicamente responder. Si no responde, entonces, un número predeterminado de recomprobaciones será gestionado. Las fallas eventualmente repetidas harán que el RTU en cuestión sea marcado como "fuera de servicio" (en un sistema de interrogación una falla de comunicación bloquea la

red por un período de tiempo relativamente largo, y una vez que se haya detectado una falla, no hay motivo para volver a revisar).

La exactitud de la transmisión de un SCADA se ha mirado tradicionalmente como tan importante que la aplicación SCADA toma directamente la responsabilidad sobre ella. Esto se produce en contraste con protocolos de comunicación más generales donde la responsabilidad de transmitir datos confiablemente se deja a los mismos protocolos. A medida que se utilicen protocolos de comunicación más sofisticados, y los proveedores de SCADA comiencen a tomar confianza con ellos, entonces la responsabilidad de manejar errores será transferida al protocolo.

2.1.2.3.3 Los protocolos de comunicación.

Se han desarrollado técnicas para la transmisión confiable sobre medios pobres, y es así que muchas compañías alcanzaron una ventaja competitiva respecto de sus competidoras simplemente debido al mérito técnico de sus protocolos. Estos protocolos por lo tanto tendieron a ser propietarios, y celosamente guardados.

Esto no representaba un problema al instalar el sistema, aunque sí cuando eran requeridas extensiones. Lo obvio y casi absolutamente necesario era acudir de nuevo al proveedor original. No era generalmente factible considerar el uso de un protocolo distinto, pues eran generalmente mutuamente excluyentes. Los progresos recientes han considerado la aparición de un número apreciable de protocolos "abiertos": IEC870/5, DNP3, MMS son algunos de éstos. Los mejores de estos protocolos son los multicapa completamente "encapsulados", y los sistemas SCADA que utilizan éstos pueden confiar en ellos para garantizar la salida de un mensaje y el arribo a destino. Un número de compañías ofrece los códigos fuente de estos protocolos, y otras ofrecen conjuntos de datos de prueba para comprobar la implementación del mismo. Por medio de estos progresos está llegando a ser factible, por lo menos a este nivel, considerar la interoperabilidad del equipamiento de diversos fabricantes

2.1.2.3.4 Las redes de comunicación.

SCADA tiende a utilizar la mayoría de las redes de comunicación disponibles. Los sistemas SCADA basados en transmisión radial son probablemente los más comunes. Éstos evolucionaron con el tiempo, y lo más básico es el uso de FSK (Frequency Shift Keying - codificación por conmutación de frecuencia) sobre canales de radio analógicos. Esto significa que aquellos 0 y 1 son representados por dos diversas frecuencias (1800 y 2100 Hz son comunes). Estas frecuencias se pueden sintetizar y enviar sobre una radio de audio normal. Velocidades de hasta 1200 bps son posibles. Una consideración especial necesita ser dada al retardo de RTS (request to send - petición de enviar) que normalmente se presenta. Esto se produce porque una radio se tomará algún tiempo después de ser encendida (on) para que la señal alcance niveles aceptables, y por lo tanto el sistema SCADA debe poder configurar estos retardos. La mayoría de las otras consideraciones con respecto a radio y SCADA se relacionan con el diseño básico de la red de radio.

2.1.2.3.5 Servicios basados en satélite:

Existes también servicios basados en satélites, pero la mayoría son muy costosos. Hay situaciones donde no hay alternativas. No obstante, existe un servicio basado en satélites que es económico: los sistemas VSAT: Very Small Aperture Terminal. Con VSAT, se alquila un segmento del espacio (64k o más), y los datos se envían de un sitio remoto a un dispositivo concentrador vía satélite. Hay dos tipos de dispositivos concentradores. El primero es un sistema proporcionado típicamente por un proveedor de servicios de VSAT. La ventaja es un costo fijo para los datos aunque su implementación puede tener costos elevados. La otra consideración para éstos es la necesidad de un enlace secundario desde un dispositivo concentrador central de SCADA. Esto puede ser de un costo considerable. El otro tipo de sistema utiliza un dispositivo concentrador pequeño (los clásicos de LAN estructuradas) que se puede instalar con el Master. Este es más barato, pero la administración del dispositivo concentrador es responsabilidad exclusiva del propietario de SCADA. La interfaz a cualquier tipo de sistema de VSAT implica el uso de protocolos utilizados por el sistema de VSAT - quizás TCP/IP.

2.1.2.3.6 Sistemas Landline (Líneas Terrestres).

Éstos son comúnmente usados, pero una gran cantidad de sistemas SCADA implican el uso de la radio para sustituir landlines ante una falla. Las termitas y el relámpago son problemas comunes para los landlines.

2.1.2.3.7 Procesadores de Comunicaciones Front End.

El "centro" de SCADA consiste típicamente en una colección de computadoras conectadas vía LAN (o LAN redundante). Cada máquina realiza una tarea especializada. La responsabilidad de la colección de datos básicamente puede residir en una de ellas (con un sistema espejo), las visualizaciones pueden ser manejadas por una segunda computadora, etcétera. Una función asignada típicamente a una computadora separada es la interfaz a la red de comunicaciones. Ésta manejará toda la interconexión especializada a los canales de comunicaciones, y en muchos casos realizará la conversión del protocolo de modo que el sistema principal pueda contar con datos entrantes en un formato estándar.

2.1.2.4 Procesamiento de alarmas

La característica del procesamiento de alarmas se ha asociado siempre a las funciones de las áreas de control de la planta. La computadora procesa todos los datos como vienen del campo, y considera si la variable ha entrado en alarma. Para los valores digitales, uno de los estados (0 o 1) se puede señalar como estado de alarma. Para valores analógicos es normal que se definan límites de alarmas tal que si el valor cae fuera de estos límites, considerarlo como en alarma. Las alarmas se clasifican normalmente en varios niveles de prioridad, con la prioridad más alta siendo a menudo reservada para las alarmas de seguridad. Esto permite que el operador seleccione una lista de las alarmas más importantes. Cuando un punto entra en alarma, debe ser validada por el operador. Un código es asociado a veces por el operador en ese momento para explicar la razón de la alarma. Esto ayuda en el análisis posterior de los datos. Es común tener cierto anuncio audible de la alarma, alguna señal sonora en la sala de

operaciones. Un problema común para los sistemas SCADA es la "inundación" de alarmas. Cuando ocurre un trastorno importante del proceso, a menudo un evento de alarma causa otro y así sucesivamente. A menudo en el entusiasmo inicial, los límites de alarma se especifican firmemente, y aún en valores que no son realmente importantes. La inundación de alarmas resultante puede abrumar al personal de operaciones, y ocultar la causa inicial del problema.

2.1.2.4.1 Características

Los recursos de alarmas incluyen la capacidad de identificar al personal de operaciones por su login, y exhibir solamente las alarmas relevantes a su área de responsabilidad, y de suprimir alarmas, por ejemplo, cuando la planta está bajo mantenimiento. Algunos sistemas sofisticados pueden resolver la inundación de alarmas identificando secuencias de causas y efectos.

2.1.2.4.2 Chequeos

Cuando los sistemas SCADA no interrogan regularmente todos los sitios, sino que por el contrario confían en la transmisión iniciada por el RTU, si se detectara una condición de error o un cambio significativo en un valor, existe la posibilidad de que el RTU o las comunicaciones puedan fallar, y el evento pase desapercibido. Para solucionar esto, se dispara un "chequeo de salud" en segundo plano, en el cual cada RTU es interrogado con una frecuencia determinada por el tiempo que se considere prudente en que una alarma no sea detectada.

2.1.2.5 Comunicaciones

La característica distintiva de los sistemas SCADA es su capacidad de comunicación. Como ya se ha dicho, comparado a los DCS (Distributed Control Systems - sistemas de control distribuido) considerados a menudo dentro de una planta o de una fábrica, un sistema SCADA cubre generalmente áreas geográficas más grandes, y utiliza muchos medios de comunicaciones diversos (y a menudo relativamente no fiables). Un aspecto importante de la tecnología de SCADA es la capacidad de garantizar confiablemente la salida de datos al usar estos medios. Los sistemas SCADA utilizaron inicialmente enlaces de comunicación lentos. Cálculos cuidadosos debieron ser hechos para evaluar los volúmenes de datos probables esperados, y asegurar que la red de comunicaciones fuera capaz de resolver las demandas

2.1.3 Unidades Terminales Remotas (Remote Terminal Units)

2.1.3.1 Fundamentos

La RTU es una robusta computadora que proporciona inteligencia en el campo para permitir que el Master se comunique con los instrumentos. Es una unidad independiente (stand-alone) de adquisición y control de datos. Su función es controlar el equipamiento de proceso en el sitio remoto, adquirir datos del mismo, y transferirlos al sistema central SCADA.

Hay dos tipos básicos de RTU, de un solo módulo (single boards), compactos, que contienen todas las entradas de datos en una sola tarjeta, y modulares que tienen un modulo

CPU separado, y pueden tener otros módulos agregados, normalmente enchufándolos en una placa común (similar a una PC con una placa madre donde se montan procesador y periféricos).

Un RTU single board tiene normalmente I/O fijas, por ejemplo: 16 entradas de información digital, 8 salidas digitales, 8 entradas de información analógicas, y 4 salidas analógicas. No es normalmente posible ampliar su capacidad.

Un RTU modular se diseña para ser ampliado agregando módulos adicionales. Los módulos típicos pueden ser un módulo de 8 entradas análogas, un módulo de 8 salidas digitales.

2.1.3.2 Funcionalidad del Hardware de un RTU.

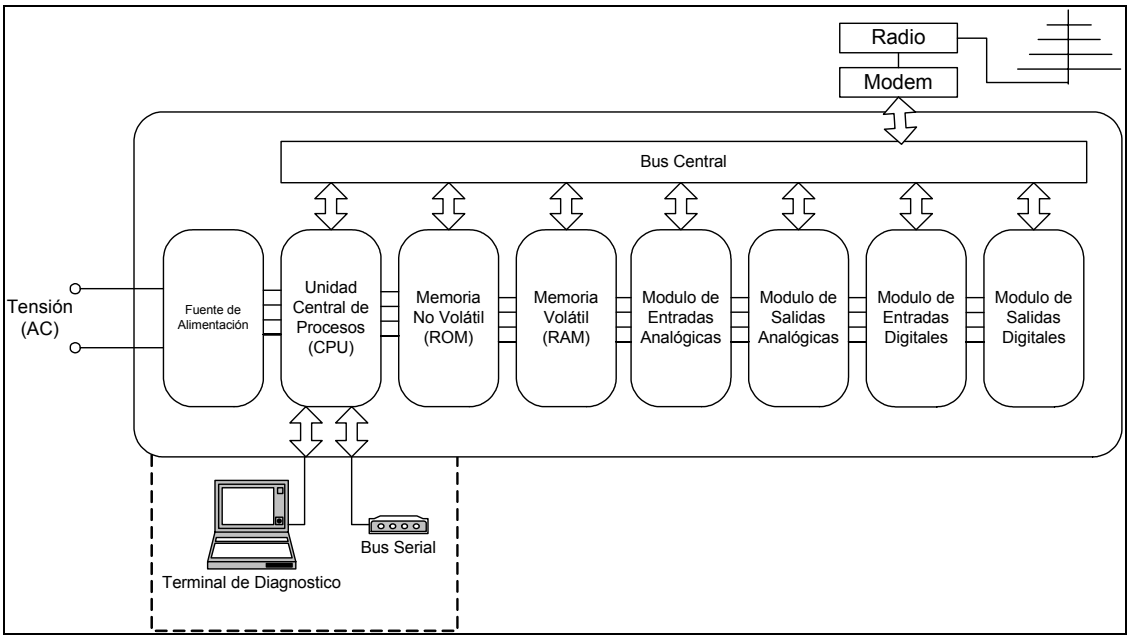


Figura 1- Diagrama general de una unidad terminal remota (RTU)

El hardware de un RTU tiene los siguientes componentes principales (Véase la Figura 1):

- CPU y memoria volátil (RAM).
- Memoria no volátil para grabar programas y datos.
- Capacidad de comunicaciones a través de puertos seriales o a veces con módem incorporado.
- Fuente de alimentación segura (con batería de respaldo).
- Perro Guardian (Watchdog timer) que asegure reiniciar el RTU si algo falla.
- Protección eléctrica contra fluctuaciones en la tensión.
- Interfaces de entrada-salida a DI/DO/AI/AO's.
- Reloj de tiempo real.

2.1.3.3 Funcionalidad del Software de un RTU

Todos los RTU's requieren la siguiente funcionalidad. En muchos RTU's éstas se pueden mezclar y no necesariamente ser identificables como módulos separados.

- Sistema operativo en tiempo real.
- Manejador para el sistema de comunicaciones con el Master.
- Drivers de dispositivo para el sistema de entrada-salida a los dispositivos de campo.
- Programa de aplicación SCADA para exploración de entradas de información, procesamiento y el grabado de datos, respondiendo a las peticiones del Master sobre la red de comunicaciones.
- Algún método para permitir que las aplicaciones de usuario sean configuradas en el RTU. Ésta puede ser una simple configuración de parámetros, habilitando o deshabilitando entradas-salidas específicas que invalidan o puede representar un ambiente de programación completo para el usuario.
- Programa de diagnóstico y verificación.
- Algunos RTU's pueden tener un sistema de archivos con soporte para descarga de archivo, tanto programas de usuario como archivos de configuración.

2.1.3.4 Operación Básica

El RTU operará la exploración de sus entradas de información, normalmente con una frecuencia bastante alta. Puede realizar algún procesamiento, por ejemplo cambios de estado, chequeo de cambios, y almacenaje de datos que aguardan el polling del Master. Algunos RTU's tienen la capacidad de iniciar la transmisión de datos al Master, aunque es más común la situación donde el Master encuesta a los RTU's preguntando por cambios. El RTU puede realizar un cierto procesamiento de alarmas. Cuando es interrogado el RTU deber responder a la petición, la que puede ser tan simple como dame todos tus datos, o una compleja función de control para ser ejecutada.

2.1.3.5 RTU´s pequeños contra RTU´s grandes

Los RTU's son dispositivos especiales fabricados a menudo por pequeños proveedores en pequeños lotes de algunos cientos, normalmente para los mercados domésticos. Por lo tanto no todos los RTU's soportan toda la funcionalidad descrita. Un RTU's más grande puede ser capaz de procesar centenares de entradas de información, y aún controlar el funcionamiento de "sub RTU's" más pequeños. Éstos son obviamente más costosos. La potencia de procesamiento de un RTU se extiende desde pequeños procesadores de 8 bits, con memoria mínima, hasta sofisticados RTU's más grandes, capaces de recolectar datos en el orden del milisegundo.

2.1.3.6 Algunos tipos de RTU´s

- Sistemas stand-alone minúsculos que emplean las mismas baterías por un año entero o más. Estos sistemas registran los datos en la EPROM o FLASH ROM y descargan sus datos cuando son accedidos físicamente por un operador. A menudo estos sistemas usan procesadores de chip simple con memoria mínima y pueden no ser capaces de manejar un protocolo de comunicaciones sofisticado.
- Sistemas stand-alone pequeños que pueden accionar periódicamente a los sensores (o radios) para medir y/o reportar. Generalmente las baterías son mantenidas por energía

solar. Estos sistemas tienen generalmente bastante capacidad para un esquema mucho más complejo de comunicaciones.

- Sistemas medios. Computadores industriales single board dedicados, incluyendo IBM-PC o compatibles en configuraciones industriales tales como VME, MultiBus, megabus, PC104, etc.
- Sistemas grandes. Completo control de planta con todas las alarmas visuales y sonoras. Éstos están generalmente en DCS en plantas, y se comunican a menudo sobre LAN de alta velocidad. La sincronización entre sus componentes puede ser muy crítica.

2.1.3.7 Estándares

Como fuera indicado, los RTU's son dispositivos especiales. Ha habido una carencia de estándares, especialmente en el área de comunicaciones, y los RTU's provenientes de un fabricante no se pueden mezclar generalmente con RTU's de otro. Una industria ha crecido desarrollando conversores y emuladores de protocolos. Algunos estándares intentan solucionar estas diferencias para RTU's, como por ejemplo DNP e IEC870 para comunicaciones o IEC1131-3 para programar RTU's.

2.2 Comunicación de datos.

2.2.1 Concepto y evolución.

La comunicación de datos es el proceso de transferir información digital (normalmente binaria), entre dos o más puntos. La información se define como el conocimiento o forma de conocimiento. La información que se procesa y se organiza se llama datos. Los datos pueden ser cualquier información alfabética, numérica o simbólica, incluyendo los símbolos alfa numéricos codificados en binarios, códigos operacionales del microprocesador, códigos de control, direcciones de usuario, datos del programa o información de base de datos. En la fuente y el destino los datos están en forma digital. Sin embargo durante la transmisión, los datos pueden estar en forma digital o analógica [3].

Una red de comunicación de datos puede ser tan sencilla como dos computadoras personales conectadas, entre sí, por medio de una red telefónica pública, o puede abarcar una red compleja de una o más computadoras tipo servidores y cientos de terminales remotas.

Si se considera la implementación de señales eléctricas para la comunicación de datos todo comenzó con la invención del telégrafo y el desarrollo del *código Morse* en 1837, por Samuel F.B. Morse. Con el telégrafo los puntos y rayas (análogos a los unos y ceros binarios) se transmiten a través de un alambre utilizando la inducción electromecánica. Varias combinaciones de estos puntos y rayas se usaron para representar los códigos binarios para números, letras y puntuación. Los Bell laboratories desarrollaron la primera computadora de uso especial, en 1940, usando relevadores electromecánicos. La primera computadora de uso general fue una calculadora controlada por secuencia automática desarrollada por la Universidad de Harvard y la Internacional Business Machine Corporation (IBM). La computadora UNIVAC, construida en 1951 por Corporation Remington Rand (ahora Sperry Rand), fue la primera computadora electrónica producida masivamente. Después de años de

nuevos desarrollos monopolizados por grandes empresas, fue entonces para el año de 1968 cuando una resolución de la corte suprema, conocida como la *Resolución de Carterfone*, permitió que las compañías que no fuesen la Bell se interconectaran a la gran red de comunicación AT&T. Con esta decisión comenzó la *industria de interconexión*, la cual ha permitido ofertas competitivas de comunicación de datos por un gran número de compañías independientes.

2.2.2 Transmisión de datos.

La transmisión de datos es el intercambio de datos (en forma de ceros y unos) entre dos dispositivos a través de alguna forma de transmisión (como un cable). La transmisión de datos se considera local si los dispositivos están en el mismo edificio o en un área geográficamente restringida y se considera remota si los dispositivos están separados por una distancia considerable.

Para que la transmisión de datos sea posible, los dispositivos de comunicación deben ser parte de un sistema de comunicación formado por hardware y software. La efectividad del sistema de comunicación de datos depende de tres características fundamentales:

- **Entrega:** El sistema debe entregar los datos en el destino correcto. Los datos deben ser recibidos por el dispositivo o usuario adecuado y solamente por ese dispositivo o usuario.
- **Exactitud:** El sistema debe entregar los datos con exactitud. Los datos que se alteran en la transmisión son incorrectos y no se pueden utilizar.
- **Puntualidad:** El sistema debe entregar los datos con puntualidad. Los datos entregados tarde son inútiles. Para este caso los más críticos son los sistemas conocidos como *sistemas en tiempo real*.

Un sistema de transmisión de datos esta formado básicamente por cinco componentes (Véase la Figura 2)

1. **Mensaje:** Es la información (datos) a comunicar. Puede estar formado por texto, números, gráficos, sonido, vídeo o una combinación de los anteriores.
2. **Emisor:** Es el dispositivo que envía los datos del mensaje. Puede ser una computadora, una estación de trabajo, un teléfono, una videocámara u otros.
3. **Receptor:** Es el dispositivo que recibe el mensaje. Puede ser una computadora, una estación de trabajo, un teléfono, una videocámara u otros.
4. **Medio:** El medio de transmisión es el camino físico por el cual viaja el mensaje del emisor al receptor. Puede estar formado por un cable de par trenzado, un cable coaxial, un cable de fibra óptica, un láser u ondas de radio (terrestres o microondas de satélite).
5. **Protocolo:** Es un conjunto de reglas que gobiernan la transmisión de los datos. Representa un acuerdo entre los dispositivos que se comunican. Sin un protocolo, dos dispositivos pueden estar físicamente conectados pero no comunicarse, igual que dos personas que hablen idiomas diferentes.

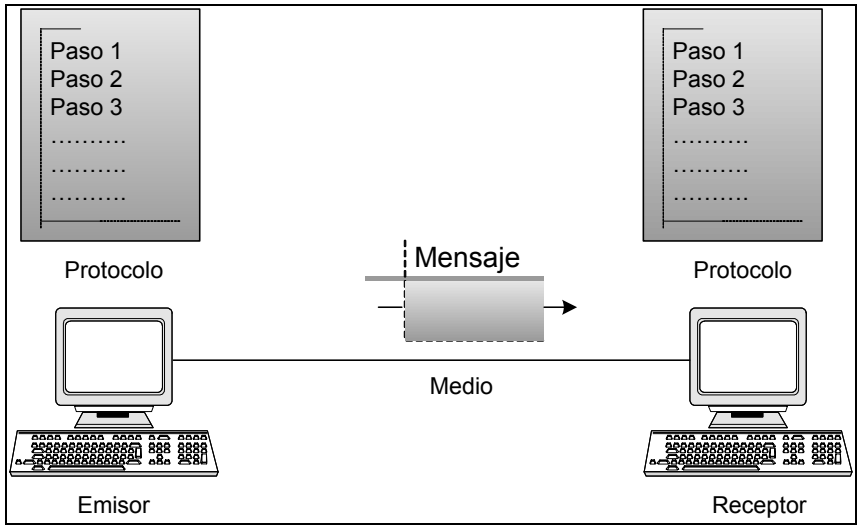


Figura 2 Componentes de un sistema de transmisión de datos.

En transmisión de datos, además de la información de usuario, es necesaria la adición de una serie de datos de control para asegurar la fiabilidad de la transmisión y la correcta extracción de la información en el otro extremo, como bloques de control de errores, control de sincronía, etc.

La transmisión de datos directos (transmisión punto a punto) se puede producir en tres modos diferentes, dependiendo de la dirección del flujo de datos:

- Simples: Si la comunicación se realiza en un solo sentido, desde un equipo emisor a uno receptor. Es el modo de comunicación más sencillo (por ejemplo en conexión de periféricos como un ratón a un ordenador personal).
- Semi-dúplex (*half-duplex)*: Si la comunicación se realiza en ambos sentidos, pero no simultáneamente. En este caso el canal de comunicación es el mismo para las transmisiones en ambos sentidos, por lo que se deben utilizar protocolos que regulen quién accede al canal común en cada momento.
- Dúplex completo (*full duplex*): La comunicación se puede realizar en ambos sentidos simultáneamente. Para ello debe existir un medio físico de transmisión en cada sentido.

La transmisión de datos directos (punto a punto), es útil en muchos casos, pero resulta costosa y poco eficiente cuando se desea establecer sistemas más complejos de comunicación que involucren un número mayor de equipos. En estos casos se recurre al empleo de un medio de transmisión común, que comparten los diferentes equipos que intervienen en el sistema de comunicación. Tal es el caso de las *redes de comunicación de datos*, que agrupan dispositivos de comunicación de datos empleando un canal común, que a su vez pueden formar parte de estructuras de mayor extensión, mediante la interconexión de éstas, a las que se denominan subredes. Las diferentes formas de conexión y organización del canal se conocen como **topologías**, que serán revisadas en el capítulo dedicado al estudio del modelo OSI.

2.2.2.1 Transmisión de los bits de datos.

Existen dos formas de transmitir los bits de datos entre equipos, en función del número de hilos empleados para la transmisión de un mensaje:

- Transmisión en paralelo: se emplea un hilo para cada bit del mensaje. Sólo se utiliza cuando las distancias de transmisión son pequeñas (dentro de un equipo o con periféricos cercanos), y supone un retardo muy pequeño en el envío de cada palabra. Su coste sería muy elevado para redes de comunicación con longitudes de cables superiores a unos pocos metros.
- Transmisión en serie: se emplea un solo par de hilos para enviar los bits de un octeto uno por uno a intervalos fijos de tiempo.

2.2.2.2 Modos de transmisión.

Normalmente la transmisión de datos se realiza en grupos de longitud fija, habitualmente de 8 bits (bytes u octetos). El receptor de estos datos deberá ser capaz de identificar:

- Dónde comienza cada bit, para muestrear la señal eléctrica que lo codifique en el centro de cada celda.
- Dónde empieza y termina cada byte, para obtener el mismo patrón de 8 bits que transmitió el emisor.
- Dónde empieza y termina cada trama o bloque de mensaje completo.

Esto es lo que se conoce como **sincronización de reloj o de bit**, **sincronización de byte o carácter** y **sincronización de bloque o trama** respectivamente. La sincronización entre equipos se puede realizar de dos maneras, dependiendo de que los relojes del emisor y el receptor sean independientes o estén sincronizados, de modo que la transmisión puede ser:

- Asíncrona. Los relojes son independientes y el receptor se resincroniza al principio de cada carácter recibido para conseguir la sincronía de reloj o bit.
- Síncrona. Todo el mensaje o trama de datos se transmite como una cadena de bits contiguos, y el receptor debe mantener la sincronía en cada bit hasta el final de la trama.

2.2.2.3 Transmisión asíncrona.

La transmisión asíncrona se emplea cuando los datos se transmiten a intervalos aleatorios, de modo que el receptor debe ser capaz de volver a sincronizarse al inicio de cada nuevo carácter que reciba. Para ello cada carácter debe ir precedido de un bit de inicio y seguido de uno o más bits de parada, como se muestra en la Figura 3.

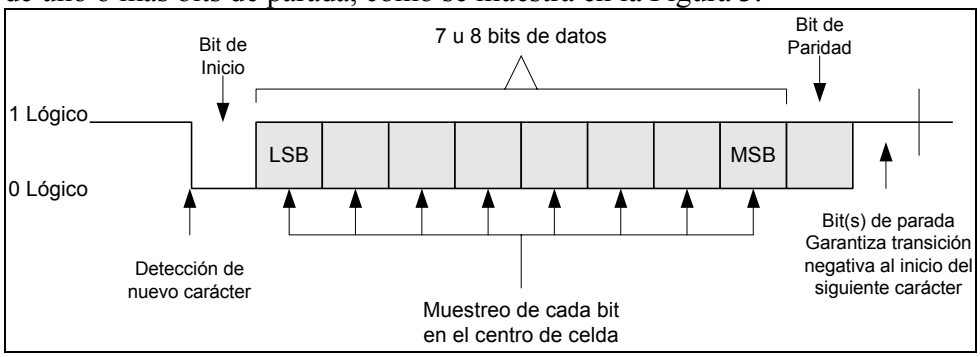


Figura 3 Transmisión asíncrona

Este modo es el empleado, por ejemplo, en la transmisión de caracteres entre un teclado y un computador, así como en la norma de interfaz RS 232 para la conexión de DTE

con DCE o incluso para la comunicación de datos entre dos ordenadores (DTE-DTE) que se explica en la sección Comunicaciones RS-232. (pag. 28). En este caso, a continuación de cada bit de parada seguiría el bit de inicio del siguiente carácter sin retardo adicional hasta completar el bloque completo de datos.

Para comprender el funcionamiento de este modo de transmisión describiremos el proceso seguido desde la generación de los bytes en el DTE hasta su recepción y procesamiento en el segundo dispositivo DTE (ver Figura 4).

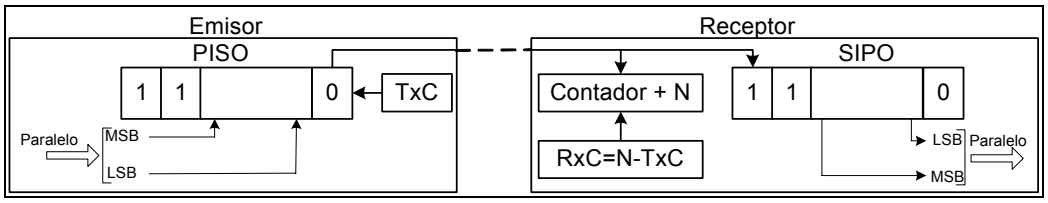


Figura 4 Principio de funcionamiento en transmisión asíncrona.

En el enlace de datos los bits son transmitidos en serie agrupados en conjuntos de 8 bits (bytes). Por el contrario, en el DTE los datos son tratados en paralelo, por lo que los circuitos de interfaz de la capa física deberán realizar las siguientes funciones:

- Conversión paralelo a serie de cada byte que se desee transmitir por el enlace de datos. Dicha conversión se realiza mediante un registro de desplazamiento de entrada en paralelo y salida serie (PISO: *parallel-in, serial-out*).
- Conversión serie a paralelo de cada carácter recibido por el enlace de datos. Dicha conversión se realiza mediante un registro de desplazamiento con entrada serie y salida paralelo (SIPO: *serial-in, parallel-out*).
- Generación de los bits de inicio, parada y paridad, necesarios para la sincronización de bit y byte, así como para la detección de errores si se aplica.

En inactividad, la línea permanece en estado de marca (‘1’ lógico), y el inicio de cada byte se detecta por la transición 1→0 provocada por el bit de inicio, que origina un flanco descendente para informar al receptor del comienzo de la transmisión de ese byte. A continuación el receptor muestreará lo más cerca posible del centro de cada bit, para lo cual emplea un reloj interno N veces mayor a la velocidad de transmisión, previamente acordada. A continuación, y mediante el empleo de un contador, se cuentan N/2 pulsos de reloj en el receptor para muestrear en el centro del primer bit, y para los siguientes se realizan conteos de N bits. Con ello se garantiza un muestreo que coincida aproximadamente con los centros de bits (hay que tener en cuenta que el receptor funciona de manera asíncrona respecto del emisor). El bit de parada (siempre nivel lógico ‘1’) garantiza el regreso de la línea al estado de marca para que el bit de inicio del siguiente carácter vuelva a provocar un flanco descendente, de modo que se produzca de nuevo la sincronización y se repita todo el proceso.

Al producirse sincronización al principio de cada carácter se evita que las posibles desviaciones entre los relojes del emisor y el receptor se acumulen durante la transmisión de tramas completas de datos.

Cuanto mayor sea la frecuencia del reloj del receptor, se podrá muestrear con mayor precisión en el centro de cada bit. Este modo de operación limita la velocidad de transmisión máxima, que no suele superar los 19,2 kbps.

2.2.2.4 Sincronización de la trama en transmisión asíncrona.

Cuando se transmiten bloques de bytes, también denominados tramas de información, es necesaria, además de la sincronización de bit y de byte o caracter, la sincronización de trama; el receptor debe saber cuándo comienza y termina la trama, compuesta por una serie de caracteres consecutivos. Para ello, es habitual aunque no necesario, utilizar un carácter de inicio de trama **STX** (Start Transmition) que indique dónde comienza el bloque, y un carácter de fin **ETX** (End Transmition) para delimitar el final.

Este método, ilustrado en la Figura 5, es válido únicamente en el caso de que se estén transmitiendo caracteres imprimibles (caracteres alfanuméricos). Pero no existe mucha diferencia en el caso que la representación no sea imprimible. Tal es el caso de transmisión de datos binarios, que pueden corresponder con ficheros, números en formato entero o de tipo mantisa-exponente, que son directamente enviados por el enlace de datos y los receptores utilizan otros métodos para sincronizar las tramas; como longitud fija del numero de bytes o un byte especial de cantidad de bytes recibidos, en fin esta tarea depende mas del protocolo de comunicación que se utilice. Aun así este método ilustra de manera simple como se procesan las tramas.

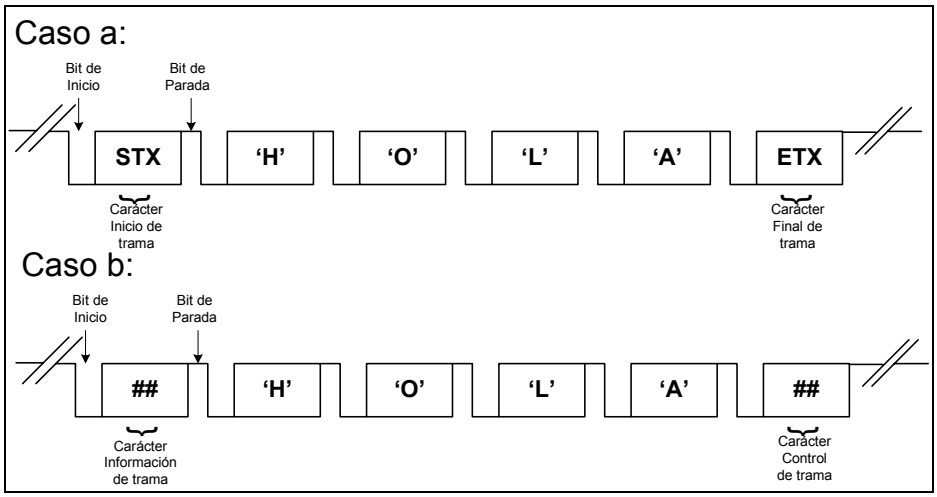


Figura 5 Sincronización de trama en transmisión asíncrona.

Caso a) Caracteres alfanuméricos. Caso b) Datos binarios.

Para el caso b estamos en presencia de un byte de información de trama, que en muchos casos contiene la información pertinente del resto de la trama, los datos de la trama están expresados como datos alfanuméricos, pero no necesariamente deben serlo, al final de la trama suele incluirse un byte de control, que en algunos casos funciona como control de errores. En estos dos casos se resume las características usuales de transmisión asíncrona, utilizándose también combinaciones de ambos que el protocolo de comunicaciones utilice.

2.2.2.5 Transmisión Síncrona.

El modo de transmisión asíncrono presenta una serie de inconvenientes que lo hacen poco apropiado cuando se pretende conseguir velocidades de transmisión elevadas con un aprovechamiento eficiente del canal de comunicación. Entre otras:

- Son necesarios una serie de bits adicionales de sincronía para cada carácter enviado (un bit de inicio y entre uno y dos bits de parada), que suponen, como mínimo, un 20% de los datos. Lo que la hace especialmente ineficiente en la transmisión de grandes bloques de datos.
- El método de sincronización obliga a emplear un reloj en el receptor con frecuencia N veces mayor a la del emisor, lo que limita considerablemente la tasa máxima de transferencia.

Por ello es muy frecuente el empleo de técnicas de transmisión síncrona en redes de comunicación de datos. En este caso **el reloj del receptor funciona de manera síncrona con la señal recibida**, al contrario que ocurría en modo asíncrono. Para ello se utiliza una de las dos siguientes técnicas:

- Bien el sistema de codificación integra información de la señal de reloj, que el receptor extrae para generar su reloj de recepción. Tal es el caso de la codificación de tipo Manchester empleada en redes *Ethernet*.
- O bien el receptor tiene un reloj interno que se mantiene sincronizado mediante un dispositivo denominado **lazo de fijación de la fase digital (DPLL**: *Digital Phase lock loop*), que aprovecha las transiciones de 1→0 y de 0→1 de la señal recibida para mantener la sincronía de bit.

2.2.2.6 Sincronización de la trama en transmisión síncrona.

Para la sincronización de trama o bloque se utilizan técnicas similares a las descritas en transmisión asíncrona. El bloque se delimita por un carácter de inicio (STX) y un carácter de fin (ETX) en el caso de caracteres imprimibles. Si se transmiten datos binarios se inserta un caracter especial definido por el protocolo usado de manera similar a la utilización de STX o ETX.

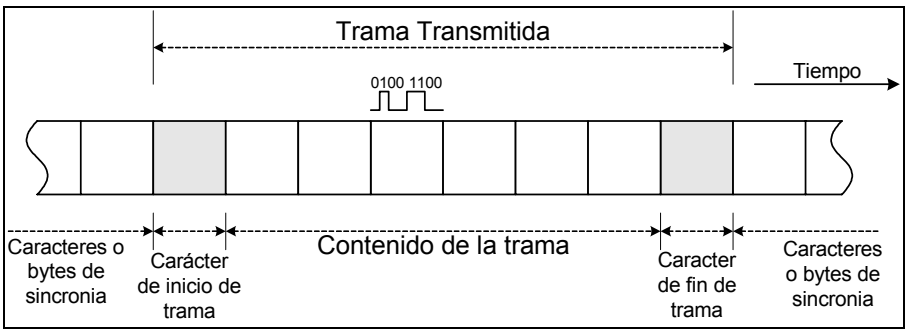


Figura 6 Transmisión síncrona.

Pero en el caso de transmisión síncrona es necesario mantener en todo momento la sincronía, de modo que es fundamental que antes del inicio de un bloque de datos los dispositivos se encuentren sincronizados. Para ello se puede emplear una de las dos siguientes técnicas:

- Incluir, antes del inicio de trama, uno o dos bytes especiales de sincronización, para que el receptor recupere la sincronía de bit y de byte (ver Figura 6).

- Realizar transmisión de caracteres continuamente en los periodos de inactividad entre las transmisiones para mantener la sincronía de bit y byte en el receptor. A esto se le conoce como inactividad síncrona.

Finalmente se puede decir que las técnicas aplicadas a la transmisión síncrona son más eficientes en cuanto a grandes bloques de datos se refiere. Para comunicación en redes de datos es muy utilizada, la desventaja radica en el elevado costo de los circuitos que forman parte de este sistema y que no es adecuada para redes con altos periodos de inactividad.

2.2.3 Protocolos y estándares.

La información transmitida entre la unidad remota y la unidad maestra va "empaquetada" en un lenguaje y un formato conocido tanto por el emisor como por el receptor. Es la única forma de que un extremo pueda interpretar correctamente el mensaje por el otro extremo.

Para que pueda tener éxito la comunicación debe haber compatibilidad en dos niveles básicos: el que se refiere a las señales eléctricas presente en el medio de comunicación y el que se refiere a la interpretación de tales señales como información.

En sistemas prácticos, se utilizan más de dos niveles para incluir mayor información (por ejemplo direccionamiento en ambiente de redes).

La interfaz entre los diferentes equipos se define dividiendo el manejo de información en niveles o capas, cada una usando los servicios suministrados por las capas debajo de ellas y agregando información o capacidad adicional.

El intercambio de información en un sistema de transmisión de datos exige una serie de pasos bien definidos o diálogo entre las estaciones transmisoras y receptoras.

Estos pasos son:

- La determinación o selección de un circuito dado entre las estaciones.
- El pedido o demanda para transmisión o recepción de información.
- La verificación de que la información no contiene errores.
- La repetición de un bloque de información que ha sido recibido con errores,
- La finalización de la transmisión.
- La supervisión, control y sincronización de las estaciones en el caso de transmisión sincrónica.

Un protocolo de comunicación es un conjunto de reglas y procedimientos que permite a las unidades remotas y maestra el intercambio de información.

Los protocolos empleados dependen si el control se efectúa mediante caracteres especiales o por conteo de bytes o bit por bit. Típicamente, en los sistemas SCADA los protocolos utilizados son orientados a bit debido a la naturaleza de la secuencia de intercambio de mensajes entre la estación maestra y las RTUs, son del tipo direccionamiento y envío.

En el mercado existe una diversidad considerable de protocolos, muchos de los cuales han sido desarrollados en forma individual por cada fabricante (denominados protocolos propietarios), y otros han sido producto del consenso de varias empresas en búsqueda de una estandarización (denominados protocolos abiertos) obteniendo la posibilidad de integración de

dispositivos de diversas marcas comerciales. A continuación se presentan las características básicas de algunos protocolos utilizados por Unidades Remotas de los Sistemas SCADA:

MODBUS: Trabaja bajo la filosofía Maestro – Esclavo, donde el maestro controla toda la actividad de transmisión de datos, interrogando en un instante de tiempo distinto a cada unidad esclava (proceso conocido como polling). Es muy utilizado en los Controladores Lógicos Programables (PLC), y debido a su naturaleza, puede facilitar el uso de PLC como Unidad Remota de un Sistema SCADA. Puede direccionar hasta 254 esclavos. Opera frecuentemente sobre una interfaz de conexión basada en el estándar IEEE RS-485, que define el número y disposición física de los conductores, así como los valores de tensión correspondientes a cada nivel lógico.

Distributed Network Protocol (DNP): Permite la implementación de sistemas Maestro – Esclavo que posean una o más estaciones Maestras, así como la operación en diversos modos que permiten la transmisión de datos desde la Unidades Terminales Remotas (RTU) a petición de la Unidad Maestra, o exclusivamente cuando se presenten estados de alarma. Es por ello que su uso se ha generalizado sobre las RTU.

Bristol Standard Asynchronous Protocol (BSAP): Es un protocolo propietario orientado a bit en el cual la estación central siempre interroga a la RTU, y ésta puede contestar según el estado de sus variables. La información es transmitida por Modulación de Duración de Pulsación (PDM).

Existe gran cantidad de protocolos de comunicación de datos, así como también existe dependencia del tipo de protocolo con el tipo de red de comunicación en el cual es utilizado. Existen protocolos destinados desde niveles de comunicación en entornos industriales hasta redes de datos de alta jerarquía. Lo que supone que existen diversos fabricantes de equipos para la comunicación de datos los cuales deben estar de acuerdo en la manera en la que estos interactuarán, por esta razón nacen los estándares y las organizaciones que los fundamentan.

Un ***estándar*** proporciona un modelo de desarrollo que hace posible que un producto funcione adecuadamente con otros sin tener en cuenta quien lo ha fabricado. Los estándares son esenciales para crear y mantener un mercado abierto y competitivo entre los fabricantes de los equipos, y además para garantizar la interoperatividad nacional e internacional de los datos, la tecnología y los procesos de telecomunicaciones. Proporciona guías a los fabricantes, vendedores, agencias del gobierno y otros proveedores de servicios, para asegurar el tipo de interconectividad necesario en los mercados actuales y en las comunicaciones internacionales.

Los estándares de transmisión de datos se pueden clasificar en dos categorías: *de facto* (que quiere decir “de hecho” o “por convención”) y *de jure* (que quiere decir “por ley” o “por regulación”) (Véase Figura 7)

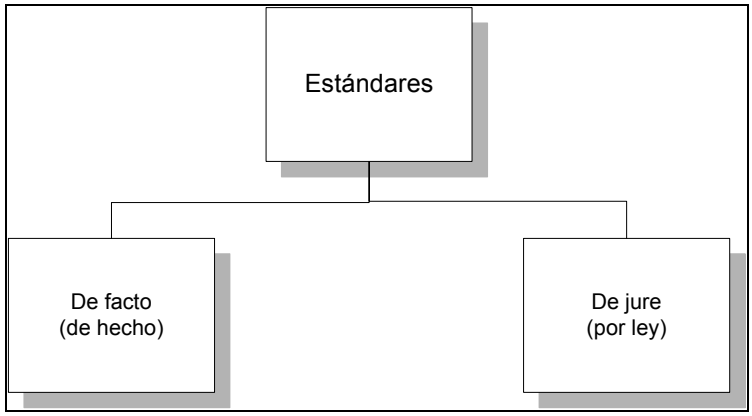


Figura 7 Categorías de estándares.

Los **estándares de jure** son aquellos que han sido legislados por un organismo oficialmente reconocido. Los estándares que no han sido aprobados por una organización reconocida pero han sido adoptados como estándares por su amplio uso son estándares de facto. Los estándares de facto suelen ser establecidos a menudo por fabricantes que quieren definir la funcionalidad de un nuevo producto de tecnología.

Los **estándares de facto** se pueden subdividir en dos clases: *propietario y no propietario*. Los estándares de propietario son aquellos originalmente inventados por una organización comercial como base para el funcionamiento de sus productos. Se llaman de propietario porque son propiedad de la compañía que los inventó. Estos estándares también se les conoce como estándares cerrados, porque cierran o entorpecen las comunicaciones entre sistemas producidos por distintos vendedores. Los estándares no propietario son aquellos originalmente desarrollados por grupos o comités que los han transferido al dominio público; también se les llama estándares abiertos porque abren las comunicaciones entre distintos sistemas [4].

2.2.4 Organizaciones de estandarización.

Los estándares son desarrollados mediante la cooperación de comités de creación de estándares, foros y agencias reguladoras de los gobiernos.

Los ***comités de creación de estándares*** se dedican a la definición y establecimiento de estándares para datos y comunicaciones, en Norteamérica se confía fundamentalmente en aquellos publicados por los siguientes:

- The International Standards Organization (ISO).
- The International Telecommunications Union – Telecommunication Standards Sector (ITU-T anteriormente el CCITT).
- The American National Standards Institute (ANSI).
- The Institute of Electrical and Electronics Engineers (IEEE).
- The Electronic Industries Association (EIA).

The **International Standards Organization** (ISO) es una organización dedicada a acuerdos mundiales sobre estándares internacionales en una amplia variedad de campos.

The **International Telecommunications Union – Telecommunications** (ITU–T) es una organización de estandarización internacional relacionada con las naciones unidas que desarrolla estándares para telecomunicaciones.

El **Instituto Nacional para la Estandarización** (ANSI) a pesar de su nombre es una organización completamente privada sin fines de lucro y es el representante con derecho a voto de los Estados Unidos tanto en ISO como en ITU-T.

El **Instituto de Ingenieros Eléctricos y Electrónicos** (IEEE) es el grupo profesional mas grande a nivel mundial involucrado en el desarrollo de estándares para computación, comunicación, ingeniería eléctrica y electrónica.

La **Asociación de Industrias Electrónicas** (EIA) es una asociación también sin fines de lucro de fabricantes de electrónica de los Estados Unidos.

Los ***foros*** trabajan con las universidades y con los usuarios para probar, evaluar y estandarizar nuevas tecnologías. Concentrando sus esfuerzos en una tecnología en particular, los foros son capaces de acelerar la aceptación y el uso de esa tecnología en la comunidad de las telecomunicaciones. Los foros presentan sus conclusiones a los organismos de estandarización. Por ser específicos los foros no existe una preferencia marcada como en el caso de los comités para estandarización, por lo que se señalan algunos que son de interés en nuestro caso.

El **Grupo de Usuarios de DNP** (www.dnp.org) es un foro para soporte de Distributed Network Protocol (DNP) o protocolo para redes distribuidas, cuya función principal es proveer las especificaciones del protocolo de comunicación DNP 3 e interactuar con los usuarios a manera de realizar mejoras en dicho protocolo (sin realizar modificaciones a su base), también se encarga de proveer certificaciones a empresas y dispositivos que funcionen bajo su norma.

El **Grupo de Usuarios de MODBUS** (www.modbus.org) es un foro para proveer las especificaciones totales del protocolo MODBUS en todas sus variantes, su misión principal es reglar y certificar productos que funcionen bajo este protocolo de comunicaciones, se compone de un grupo de usuarios que una vez registrados participan de los foros para ayudar al desarrollo de tecnologías asociadas a ellos.

Las ***Agencias Reguladoras*** son agencias de los gobiernos que regulan los aspectos que están sujetos a ámbitos de legalidad y de interés para cada nación, su función principal es proteger el interés público y conceder permisos, así como restringir al que no se sujete a su norma. Como ejemplo se puede nombrar a la **Comisión Nacional para las Telecomunicaciones** (CONATEL) de Venezuela y la **Comisión Federal de Comunicaciones** (FCC) en los Estados Unidos.

2.2.5 Control de errores.

En todo sistema de comunicación de datos se utilizan *métodos de detección y corrección de errores*, estos métodos hacen fiable la comunicación entre dispositivos. Las señales que representan datos, en muchos casos eléctricas, circulan por medios físicos sometidos a ruido y perturbaciones que pueden hacer que el receptor interprete erróneamente los niveles recibidos. Por ello son necesarios sistemas que permitan detectar cuando se

producen este tipo de errores para poner en marcha los mecanismos necesarios para corregirlos.

Si se realiza una división dependiendo de la técnica empleada en protocolos para recuperar los errores de transmisión se pueden distinguir *protocolos autodetectores* y *protocolos autocorrectores*. Para los protocolos autodetectores se incluye información para que el receptor sea capaz de detectar que se ha producido un error; estos protocolos se denominan también de corrección hacia atrás, puesto que en caso de error es necesario volver a transmitir los datos erróneos. Tal es el caso del control de paridad, suma de comprobación o códigos de redundancia cíclica. Para los protocolos autocorrectores el receptor puede detectar y corregir el error automáticamente, para ello el mensaje debe contener información redundante que permita recuperar la información a pesar de los datos erróneos; estos protocolos son también denominados de corrección hacia delante, ya que no requieren la retransmisión del mensaje erróneo para recuperar los datos, como ejemplo se puede tener códigos Hamming.

Los protocolos autocorrectores requieren una gran cantidad de información adicional para la corrección de errores. Este hecho, unido a que las tasas de error en redes de comunicación de datos son muy pequeñas, hace que este tipo de técnicas sea poco utilizada frente a los protocolos autodetectores o de corrección hacia atrás.

A continuación se revisaran algunas técnicas más empleadas para corrección de errores en los sistemas de comunicación de datos [2].

2.2.5.1 Sistemas autodetectores (corrección hacia atrás)

Estos protocolos permiten al equipo receptor comprobar si se ha producido un error en el mensaje transmitido. Para ello se realizan las siguientes operaciones:

1. El emisor toma el mensaje a enviar y calcula un valor de comprobación que enviará como parte de la trama.
2. El receptor toma el mensaje mas el valor de la comprobación y realiza los mismos cálculos para obtener dicho campo de comprobación.
3. Si los valores de comprobación recibido y calculado coinciden se supone que la transmisión ha sido correcta.

En el caso de que dichos valores de comprobación (recibido y calculado) difieran, sabemos con un 100% de probabilidad que se ha producido un error en la transmisión, pero si dichos valores coinciden no es seguro que el mensaje recibido coincida con el original, aunque si muy probable. Esto es así porque el campo de comprobación también se puede ver afectado por errores en la transmisión de tal modo que dichas alteraciones puedan inducir a validar datos erróneos.

Estos métodos de detección de errores serán tanto mas fiables cuanto menor sea la probabilidad de que se produzcan errores no detectados.

Revisaremos los principales sistemas que son: control de paridad, suma de comprobación y códigos de redundancia cíclica.

2.2.5.1.1 Control de paridad.

Los errores en la transmisión se detectan utilizando uno o varios bits de paridad. Existen cuatro tipos de control de paridad:

1. Par (Even). El número de bits de datos a "1" lógico más el bit de paridad debe ser par.
2. Impar (Odd). El numero de bits de datos a "1" lógico más el bit de paridad debe ser impar.
3. Marca (Mark). El bit de paridad siempre vale "1".
4. Espacio. El bit de paridad siempre vale "0".

Con este sistema el receptor sólo detectará un error en la transmisión si el número de bits que cambia como consecuencia de dicho error es impar. Por ejemplo, para el carácter codificado en binario con el valor "0110 1111" la paridad par valdría "0" y la impar "1".

Nótese que si cambiamos solo dos bits cualesquiera del carácter el método fallaría en la detección. De este modo, si fallaran los ocho bits un error no se detectaría. Se trata por lo tanto de un sistema poco fiable que suele ser relegado al nivel físico de transmisión. Sin embargo, en los sistemas actuales la fiabilidad de la transmisión es tan alta que es improbable que se produzca más de un bit de error en el mismo byte, por lo que en muchos casos es una técnica aplicable.

Existen dos formas de aplicar el control de paridad, dependiendo de si se quiere obtener un bit de paridad por byte o un byte de comprobación completo para un bloque de datos:

- Paridad vertical (VRC Vertical Redundancy Check). Consiste en obtener un bit de paridad para cada byte o carácter transmitido.
- Paridad longitudinal u horizontal (LRC Longitudinal Redundancy Check). Obtiene un byte de comprobación para un mensaje o bloque de datos completo. Cada bit se determina calculando la paridad de los bits de igual peso en cada uno de los bytes del mensaje.

2.2.5.1.2 Suma de comprobación (Checksum).

La suma de comprobación se emplea para obtener un campo de comprobación de errores para los diferentes datos que componen la trama. Para ello se suman todos los bytes a transmitir en un único registro de 8, 16 ó 32 bits, dependiendo del tamaño deseado para el campo de comprobación. Si la suma supera el valor máximo del registro, el valor quedará truncado al número de bits empleado en el mismo.

Este sistema presenta también algunos inconvenientes. En primer lugar, no se detectan errores por alteración del orden en los bytes de la trama (la suma de comprobación para "ABCD" es la misma que para "BADC"). Y en segundo, si se produce un fallo sistemático en un bit varios fallos lo pueden hacer parecer correcto, esto se produce como consecuencia del efecto combinado de sumar dos o más errores que por su valor posicional desborden la suma en el máximo valor posible del byte.

No obstante, este sistema de detección es mucho más fiable que el control de paridad, y es muy empleado en redes de comunicación de datos por su simplicidad.

2.2.5.1.3 Códigos de redundancia cíclica.

Los sistemas de detección anteriores son apropiados para aplicaciones en las que se producen errores aleatorios de un solo bit. Cuando se presentan ráfagas de errores es necesario emplear técnicas más complejas. Una ráfaga de errores se define como el número de bits entre dos bits erróneos sucesivos, incluidos estos. Además, para determinar la longitud de la ráfaga, el último bit erróneo de una ráfaga y el primero de la siguiente deben estar separados n bits, siendo n la longitud de la ráfaga.

Los códigos de redundancia cíclica utilizan polinomios generadores para la obtención del campo de comprobación. La teoría matemática en que se basan queda fuera del alcance de esta explicación, por lo que limitaremos la descripción a solo su funcionamiento, que se puede resumir así:

- Se emplea aritmética de modulo 2 para las operaciones, por lo que la resta de polinomios equivale a una operación XOR (OR exclusivo). La obtención del campo de comprobación, denominado CRC (Cyclic Redundancy Code) o FCS (Trama Check Sequence) se puede realizar mediante operaciones XOR y desplazamientos.
- El mensaje de k bits a transmitir se representa por un polinomio *M(x)* de grado *k-1*, donde cada coeficiente representa un bit del mensaje:

$$M(x) = b_k.x^{k-1} + b_{k-1}.x^{k-2} + ... + b_0.x^0$$

- Para generar el CRC se emplea un polinomio generador *G(x)* de grado *n* con los bits de mayor y menor peso a 1. *G(x)* tiene *n+1* bits, siendo *n* el numero de bits de CRC a generar, es decir, el polinomio generador tiene un bit más que el número de bits del campo del CRC que queramos obtener.

Para el cálculo del campo de comprobación de CRC (o FCS) se realizan las siguientes operaciones:

- Añadir r ceros al final del mensaje, representado por *M(x)*, que equivale a multiplicar por x^n:

$$M(x) \rightarrow x^n.M(x)$$

- Dividir por el polinomio generador: $\frac{x^n.M(x)}{G(x)} \rightarrow R(x)$, operación de la que se obtiene el resto *R(x)*, que representa el campo de CRC.
- El emisor transmite $T(x) = x^n.M(x) - R(x)$, que representa el mensaje de datos al que se ha añadido el campo de CRC:
  - *T(x)* contiene el mensaje en su parte de mayor peso.
  - El CRC en los r bits de menor peso.
  - Al haber restado *R(x)*, *T(x)* es divisible por el generador *G(x)*.

- Si no hay error se recibirá el mensaje transmitido, *T(x)*. Como dicho mensaje es divisible por *G(x)*, el receptor solo tendrá que realizar la operación:

$$\frac{T(x)}{G(x)} \rightarrow R^1(x)$$

Si el resto obtenido es 0, la recepción ha sido correcta, si no, se habrá producido un error.

- Si hay error, se recibe *T(x)+E(x)* (mensaje mas error), y como *T(x)* es divisible por *G(x)*, el resto se debe a la componente de error *E(x)*, por lo tanto, si *E(x)* es divisible por *G(x)* el error no se detecta. Por ejemplo, si el error es de un bit, $E(x)=x^i$, y si *G(x)* tiene mas de un termino no puede dividir a *E(x)* y se detectarán todos los errores de un bit.

En general, un polinomio generador de n bits detectará:

- Todos los errores de 1 bit.
- Todos los errores de 2 bits.
- Todos los errores de bit en número impar.
- Todas las ráfagas de errores de longitud menor que de *n*.
- La mayor parte de las ráfagas de errores mayor que *n*.

La forma de representar el polinomio generador consiste en indicar las posiciones de los unos binarios como potencias de *x*. Algunos de los polinomios normalizados que se utilizan en comunicación de datos son:

- CRC–16: $x^{16} + x^{15} + x^2 + 1$
- CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$
- CRC-32: $x^{32} + x^{26} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- CRC-DNP $x^{15} + x^{13} + x^{10} + x^9 + x^7 + x^5 + x^4 + x^3 + x^{2.}$

2.2.6 Medios de transmisión.

Los medios de transmisión son los elementos por los que se transporta la información, haciendo que llegue con la menor cantidad de ruido y distorsión a todos los nodos (o estaciones) involucrados en el proceso de comunicación. A nivel de campo deben permitir mucha flexibilidad en cuanto a manejo físico del mismo y al incremento del número de nodos de manera simple. Se presenta a continuación los medios de transmisión utilizados en los entornos industriales.

De manera ineludible, asociado a los medios de transmisión se encuentran los conectores que permiten realizar la unión entre los nodos, elementos de la red y el medio de transmisión, debiendo ser transparentes al funcionamiento de la misma, sin entorpecer o atenuar el flujo de señales. Dependiendo del tipo de red a instalar, a menudo estos conectores suelen ser específicos, aunque existen conectores de uso general como los conectores DB9, DB15 y DB25 habitualmente empleados en transmisión de señales eléctricas.

Existen dos clases principales de medios de transmisión, los *medios guiados*, y los *medios no guiados*. En el primer caso existe un medio material por donde se transmite la información (cableado en general), y el segundo tipo utiliza el aire como medio de transmisión, es decir, suelen ser sistemas de transmisión inalámbricos. La elección de un tipo de red local conlleva la elección del medio de transmisión pues cada fabricante suele recomendar un tipo de medio de transmisión que mejor se adapta a la red, o bien aconseja varios medios de transmisión dependiendo de las distancias, velocidades de transmisión, ancho de banda, entorno de trabajo, etc.

2.2.6.1 Medios guiados.

La característica principal de un medio guiado es la existencia de un cable consistente en una envoltura de uno o más hilos conductores eléctricos u ópticos. Entre los medios de transmisión de señal eléctrica, uno de los parámetros a tener en cuenta es la impedancia característica, que debe mantenerse en todo el cable para asegurar una correcta transmisión, y

la atenuación del cable, medida en dB (decibelios) por unidad de distancia. Respecto al cable óptico, se debe asegurar una buena transmisión del haz óptico, especialmente a través de los múltiples empalmes que suelen realizar en cualquier instalación.

2.2.6.1.1 Par trenzado.

Es el medio de transmisión más antiguo. Está formado por dos hilos de cobre aislado entrelazados en forma helicoidal, uno para transmisión de datos y el otro referenciado a tierra. La utilización de la forma entrelazada tiene por objeto la reducción de la interferencia eléctrica con respecto a los pares de hilos más cercanos, ya que como es sabido dos cables paralelos podemos asimilarlos a una antena. Existen los cables apantallados (STP, Shielded Twisted Pair) y no apantallados (UTP, Unshielded Twisted Pair); este último se clasifica en diferentes categorías (de la 2 a la 5) dependiendo de la calidad del mismo, actualmente ya existen cables de categorías 6 y 7, pues a veces es necesaria mayor calidad para soportar velocidad y ancho de banda. También existen numerosas variantes de cables dependiendo de la envoltura plástica para adaptarse a cualquier aplicación o emplazamiento (exteriores, temperaturas extremas, etc.) sin perjudicar la calidad de la transmisión.

Existen cables con diferentes números de pares en su interior, ya que dependiendo de la aplicación o tipo de red son necesarios más o menos pares. Su utilización tiene cabida tanto en aplicaciones digitales como analógicas. La aplicación más conocida es la transmisión telefónica, con cable STP de dos pares y conectores RJ45 y RJ11. Otra aplicación habitual es la de redes para transmisión de datos (Ethernet, por ejemplo) con el uso de pares trenzados y conectores RJ45 y RJ11.

2.2.6.1.2 Cable coaxial.

El cable coaxial está formado por un núcleo de cobre rodeado de material aislante y un conductor exterior trenzado denominado comúnmente maya, que se dispone en una estructura concéntrica. Cubriendo a todo el conjunto encontramos externamente una cubierta protectora de material plástico. Existen dos tipos principales, de banda base y de banda ancha, aunque este último, a pesar de poseer mejores cualidades, es menos empleado dada su mayor complejidad de instalación y mayor coste. Este tipo de cable suele ser robusto ante interferencias, sus aplicaciones mas conocidas son para señales de televisión y redes de datos.

2.2.6.1.3 Fibra óptica.

Constituida por un núcleo muy fino de fibra de vidrio circular (existen diferentes materiales plásticos que dotan a la fibra óptica de diferentes propiedades y calidades), que al tener un elevado índice de refracción permite conducir la energía óptica en su interior. Este núcleo esta envuelto por un recubrimiento opaco que aísla la fibra óptica de posibles interferencias. A diferencia del caso anterior, la transmisión no es digital sino analógica, por lo que se necesita disponer de amplificadores que refuercen la señal de forma periódica. Es el medio idóneo si se necesitan altas velocidades de transmisión, gran ancho de banda o cubrir largas distancias, pues la luz es más inmune a las interferencias electromagnéticas y posee tiempos de transmisión menores. Existen tres tipos básicos de fibra óptica, fibra monomodo, multimodo de índice gradual, y multimodo de índice discreto o escalonado, con diferentes grados de atenuación, velocidades de transmisión, y ancho de banda. Debido a la complejidad de la instalación y sus dispositivos asociados, resulta una opción muy cara, por lo que solo se instala en lugares donde no sea posible otra alternativa.

2.2.6.2 Medios no guiados.

En emplazamientos donde resulta complicado trazar un tendido de cable, es conveniente utilizar un enlace inalámbrico. Actualmente, este tipo de enlaces esta teniendo una gran auge debido a la aceptación de las tecnologías como sistemas de enlace Wi-fi (IEEE 802.11b) y Bluetooth, que resuelven las comunicaciones entre dispositivos en distancias cercanas, pero donde se centra gran parte de las necesidades de los usuarios (por ejemplo en una industria). Sin embargo, los enlaces mediante medios no guiados ya se vienen realizando con anterioridad mediante ondas de radio para distancias cercanas, y mediante enlaces de microondas , usados generalmente en enlaces punto a punto que deben cubrir largas distancias (se usan en comunicaciones terrestres y vía satélite).

2.2.7 Estándares de comunicación básicos.

Entre los estándares de comunicación mas usados en aplicaciones electrónicas industriales tenemos a el estándar RS-232 y RS422/485. Esto es debido especialmente a la relativa sencillez en la intercomunicación, la incorporación de manera casi estándar dentro de los equipos, y a su probada robustez y adaptabilidad para satisfacer las necesidades de gran numero de aplicaciones, sin necesidad de tener que recurrir a instalaciones más complejas. RS-232 está diseñado para comunicaciones punto a punto, mientras que RS422/485 permite una conexión de varios dispositivos sobre el mismo bus (punto – multipunto).

2.2.7.1 Comunicaciones RS-232.

El estándar RS-232 fue propuesto por primera vez en 1962, aunque ha sufrido diversas revisiones desde entonces. Actualmente, el nombre oficial es EIA/TIA-232-E, haciendo referencia al organismo que lo define (EIA, Electronics Industry Association y Telecommunication Industry Association) y a la ultima revisión (E, quinta revisión), el termino RS hace referencia a su descripción inicial (Recommended Standard, estándar recomendado), pero que actualmente sigue siendo el más empleado. Además, el nombre que define el estándar es: “Interfaz entre equipos terminales de datos y equipos de terminación de circuito de datos empleando intercambio de datos binarios tipo serie”. Es decir, que ofrece un estándar de intercomunicación entre los comúnmente conocidos como DTE (equipos terminales de datos) y DCE (equipos de comunicación o de terminación de circuito de datos); los primeros, suelen ser los equipos encargados de generar y controlar los datos a transmitir (una computadora o un autómata programable), y los segundos hacen referencia al dispositivo periférico encargado de recibir esos datos, ejemplos típicos de este tipo de dispositivos son un MODEM, una impresora, etc. La Figura 8 muestra esta descripción [5].

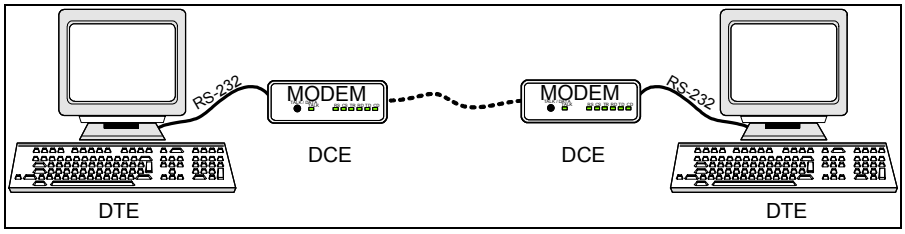


Figura 8 Enlace RS-232 entre equipo terminal (DTE) y de comunicaciones (DCE).

La especificación RS-232 describe tres ámbitos de la comunicación: Los niveles de tensión de las señales, el patillaje de las señales y la información de control que debe existir

entre los equipos. Contrariamente a como hacen otros estándares, RS-232 describe, además de las características eléctricas, las características mecánicas y funcionales.

2.2.7.1.1 Características eléctricas.

Estas características definen los niveles de tensión, tiempos de bajada y de subida de niveles, e impedancia de línea. Dado que en 1962 todavía no se había definido la lógica TTL, los niveles de tensión definidos no coinciden, pero este estándar deja un rango de tensiones amplio para la definición de niveles, con un espacio de margen de tensión entre niveles suficientemente amplio como para evitar confusiones. El nivel lógico alto (conocido como "espaciado") esta definido entre +5 y +15 Voltios representando el "0" lógico, y el nivel bajo (conocido como "marcado") entre -5 y -15 Voltios representando el "1" lógico, pero un receptor puede aceptar un nivel alto a partir de +3V, y un nivel bajo a partir de -3V. En lo que respecta a las transiciones, para evitar problemas de interferencias, se impone un tasa de transición (Slew Rate) máximo de 30 V/µs y una transferencia máxima de 20 kbits/segundo, aunque hoy en día existen numerosos dispositivos que ofrecen velocidades mayores (hasta 350 kbits/segundo). Inicialmente, existía una restricción en la longitud máxima del cable, pero en las últimas revisiones se ha cambiado por un límite máximo en la capacidad del cable (2500pF), así, dependiendo del cable: capacidad mutua entre conductores, capacidad de entrada del driver y capacidad por unidad de longitud, será posible alcanzar diferentes longitudes de conexión. Finalmente, también se impone una impedancia de carga para el driver (transmisor o receptor) de entre 3kΩ y 7kΩ. La Figura 9 muestra algunas características que definen a RS-232.

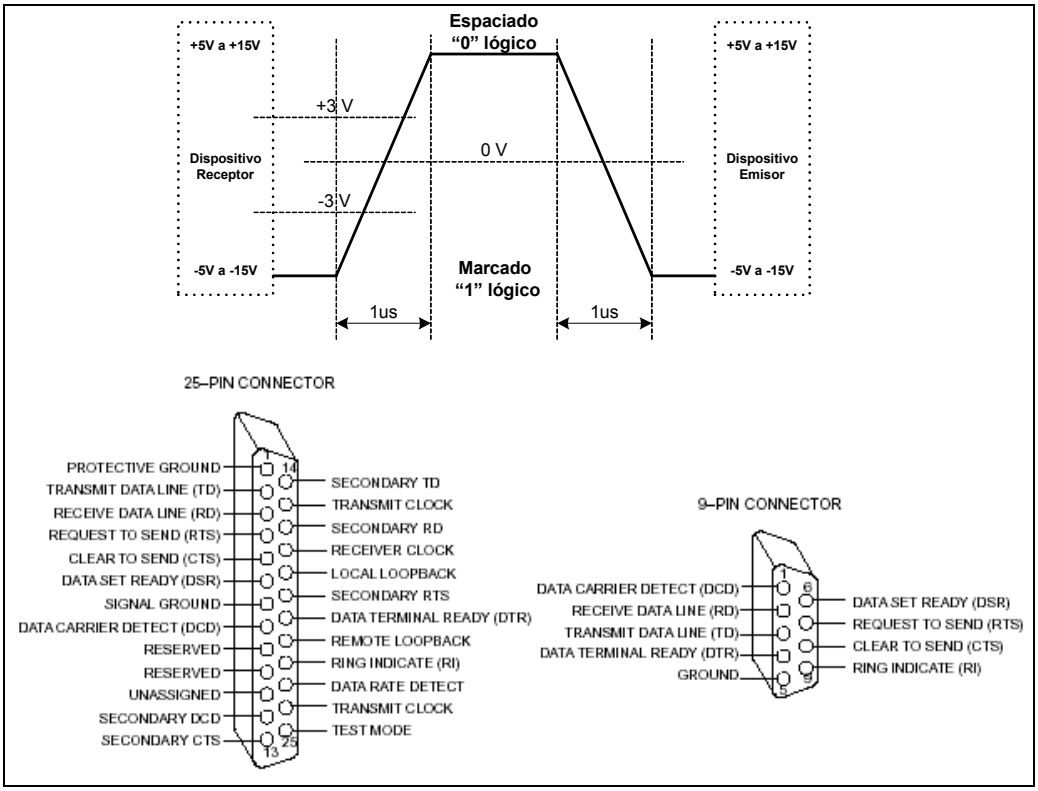


Figura 9 Características eléctricas y mecánicas de RS-232.

Dado que estos niveles no son compatibles con la lógica TTL ó CMOS actual, resulta obligada una conversión de niveles capaz de compatibilizar ambos sistemas. Para ello, existen circuitos integrados tales como el DS232A ó MAX232A que a partir de una sola tensión de alimentación (+5V que es la habitualmente disponible) y mediante elevadores de tensión internos, son capaces de convertir los niveles lógicos habituales a niveles RS-232, y viceversa. De este modo, en conjunción con las llamadas UART (Universal Asynchronous Receiver / Transmitter) permiten el enlace con computadores, y en general, cualquier dispositivo al que se le quiera incorporar la transmisión serie.

2.2.7.1.2 Características funcionales y mecánicas.

Existen cuatro tipos de señales: de control, comunes, datos y temporización. Esto proporciona un conjunto muy amplio de señales para poder adaptarse a una gran variedad de aplicaciones, en total, 24 señales están disponibles. Sin embargo, en las aplicaciones habituales se emplea un conjunto bastante reducido de estas, pues para transmisiones asíncronas con envío y recepción, suele ser suficiente con cuatro señales, una línea de envío, otra de recepción, y dos para el control de la transmisión, a veces, incluso no son necesarias las líneas de control y solo se emplean datos, en resumen solo depende de la aplicación el uso de las líneas.

Inicialmente, dado el gran numero de líneas definidas, la conexión se realizaba mediante un conector de 25 patillas (generalmente del tipo DB25), pero en la gran mayoría de aplicaciones se emplea una conexión de 9 líneas mediante un conector del tipo DB9(ver Figura 9).

2.2.7.2 Comunicaciones RS422/RS485.

Estos estándares surgieron como necesidad para ampliar la capacidad de comunicación de RS232 en aspectos como la velocidad de transmisión, inmunidad al ruido, longitud de conexión y conexiones multipunto. Ambos tienen como principal característica la de trabajar con señales digitales de voltaje balanceado (también llamado voltaje diferencial), evitando así problemas de referencia de niveles respecto a una tierra que puede variar entre dispositivos, produciendo errores. Esto requiere el empleo de dos líneas (cables) para cada señal, generalmente un par trenzado. Si llamamos a una línea A y otra B, se tomará que se ha recibido un nivel bajo si A>B, y nivel alto si A<B, siempre y cuando esta diferencia supere un cierto umbral (típicamente 200 mV). De este modo, dado que la mayoría del ruido se induce por igual en ambas líneas (ruido en modo común), éste queda compensado a la hora de realizar la diferencia entre ambos y no produce alteraciones. Habitualmente, un emisor que quiere transmitir genera una diferencia de tensión de 2 a 5 voltios entre las líneas [6].

A pesar de la aparición de múltiples sistemas de transmisión, RS422/485 sigue siendo uno de los más empleados en comunicaciones industriales debido a su robustez y sencillez en la instalación. Ha sido empleado en infinidad de aplicaciones, desde la conexión entre varios puntos de venta en supermercados al enlace entre diferentes dispositivos industriales como drivers de motores, electroválvulas y autómatas programables, hasta los asientos de pasajeros de los aviones. La amplia difusión de estos estándares se debe en gran parte a su libertad absoluta en cuanto al protocolo de transmisión, ya que se limitan a especificar las características eléctricas de la línea así como las limitaciones que se imponen. Esto ha permitido que sobre el mismo medio físico, dependiendo del fabricante, se puedan encontrar

multitud de protocolos diferentes. Incluso protocolos recientes emplean este medio de transmisión en numerosas aplicaciones.

RS422 y RS485 poseen características muy similares en los aspectos de definición de líneas de control y datos, pero la principal diferencia reside en que RS485 proporciona un mayor nivel de funcionalidad ya que en RS422 solo es posible una comunicación multipunto de modo que solo puede existir un emisor y hasta 10 receptores. En cambio, RS485 permite hasta 32 nodos en la misma red (actualmente, debido a la mejora de la circuitería electrónica, se llegan a admitir hasta 255 nodos), siendo posible que todos ellos ejerzan como iniciadores de transmisión hacia el resto, es decir, tiene capacidad multimaestro. Estas capacidades añadidas que incorpora RS485 obligan a introducir elementos de gestión de red, pues es posible la existencia de colisiones entre varios nodos, y a nivel de hardware, es necesario añadir elementos triestado para disponer de alta impedancia en los nodos (en RS422 los nodos siempre están activos). Dado que estos estándares suponen una extensión de RS232, la conversión de una línea RS232 a RS422/485 es relativamente sencilla y existen muchos circuitos desarrollados para este fin particular.

Otra diferencia importante entre ambos estándares reside en el número de líneas de transmisión necesaria para implementar la comunicación. En RS422, al igual que en RS232, es necesario dos líneas de datos (4 cables en caso de RS422), una para transmisión y otra para recepción, mientras que en RS485 existen dos posibilidades, emplear una única línea compartida para transmisión y recepción, o bien utilizar una implementación similar a RS422, lo que permite transmitir y recibir al mismo tiempo, incrementando la velocidad de transmisión. En el caso de una sola línea de comunicación, la sencillez del cableado es mayor, aunque existen mayores problemas en la gestión de las comunicaciones.

Finalmente se puede comparar las características de ambos estándares en la Tabla 1 donde se describen las principales características.

Especificaciones	RS485	RS422
Numero total de emisores y receptores en una línea	32 (Receptores o Emisores)	1 Emisor 10 Receptores
Longitud máxima del cable	1200 metros	1200 metros
Velocidad de transferencia de datos máxima	10 Mbit/s	100kbit/s
Tensión de salida máxima	-7V a +12V	-0.25V a +6V
Nivel mínimo de salida con carga	+/- 1.5V	+/- 2.0V
Nivel máximo de salida con carga	+/- 6V	+/- 6V
Impedancia de carga (Ohmios)	54	100
Rango de tensión de entrada del receptor	-7V a +12V	-10V a +10V
sensibilidad del receptor	+/- 200mV	+/- 200mV
Resistencia de entrada del receptor (Ohms)	>=12k	4k mínimo

Tabla 1 Comparativa entre estándares RS422 y RS485.

Junto con RS422, existe otro estándar denominado RS423, con idénticas especificaciones eléctricas, pero empleando una conexión no balanceada. RS422 es la extensión directa de la conexión RS232, por ello, su estructura sólo admite que en cada línea (una línea está constituida por un par trenzado) exista un emisor y varios receptores. Actualmente, el nombre del estándar RS422 es EIA/TIA-422-B (también lo describe el estándar ITU-T v11), y para RS485, EIA/TIA 485-A (también definido en el estándar ISO 8482).

Para una correcta adaptación, es necesario disponer de una terminación de las líneas de transmisión (típicamente, 100 – 120Ω). La velocidad máxima de transmisión depende de la

longitud de la línea, a mayor longitud, menor velocidad de transmisión. En general, todos los elementos hardware que cumplen con el estándar RS485 son compatibles con sistemas basados en RS422, en cambio, no es posible incluir elementos RS422 en instalaciones RS485, ya que existirán problemas de funcionamiento. Dado que RS485 es el bus más genérico y gran parte de su descripción es aplicable a RS422, se realizará una explicación mas detallada del mismo.

2.2.7.2.1 Comunicaciones RS485.

A pesar de que la instalación de líneas de comunicación RS485 resulta sencilla deben tenerse en cuenta ciertas consideraciones para evitar la aparición de problemas en la comunicación.

1. **Puesta a tierra.** Cada dispositivo puede estar conectado a una tierra diferente, pero debe tenerse en cuenta que existe un máximo de diferencia de potencial entre las tierras establecido por el estándar de -7V a +12V. Opcionalmente (aunque recomendado), existe la posibilidad de introducir una línea adicional de tierra para igualar niveles, y reducir las posibilidades del funcionamiento incorrecto.
2. **Protección contra cortocircuito.** Se debe establecer una protección para corrientes mayores de 150mA entre A ó B y tierra, o de 250mA entre A y B.
3. **Protección contra sobre tensiones y descargas.** En entornos industriales resulta habitual que existan cambios de tensión bruscos que pueden llegar a las centenas o miles de voltios, debido a la puesta en marcha de maquinaria, alumbrado, o descargas electrostáticas (ESD). Por ello también resulta apropiado incluir supresores de transitorios de tensión para eliminar estos transitorios momentáneos.
4. **Apantallamiento.** La utilización de cable apantallado no suele ser necesaria si el cable no esta sometido a grandes campos que puedan afectar el funcionamiento, pues el uso de señales diferenciales evita gran parte del ruido que puede introducirse en la línea. Por tanto, solo si el cable se emplaza junto a líneas de alimentación, o maquinaria de alto consumo de energía (robots de soldadura, motores de gran potencia, etc.), no es estrictamente necesario un apantallado. De todos modos, es recomendable considerar la decisión antes de la instalación para evitar problemas futuros.
5. **Terminación de la línea.** Este es un aspecto muy importante a la hora de evitar reflexiones de señal que introducen ruido en la línea y ello redunda en la posibilidad de obtener menores prestaciones (baja velocidad de transmisión y menores longitudes). Existen varias técnicas de emplazamiento de resistencias de terminación: Sin terminación, terminación paralela, terminación bidireccional y terminación AC. Si no se introduce terminación, las presentaciones bajan, pero para líneas cortas y sin muchos nodos puede ser una solución sencilla. La terminación paralela necesita de una única resistencia al final de la línea, pero solo es apta en caso de que exista un único emisor. La terminación bidireccional permite que cualquier nodo sea emisor o receptor, pero necesita dos resistencias (una en cada extremo), lo que se puede traducir en un mayor consumo de potencia. La terminación AC consiste en la introducción de un condensador en serie con la resistencia de terminación que funcione como elemento bloqueador DC, para tener un menor consumo de potencia, esta técnica es menos usada debido a que puede introducir transitorios no deseados, pero bien calculada es una técnica excelente para presentaciones donde el consumo energético es importante. Estas diferentes terminaciones se pueden apreciar en la Figura 10 que muestra las cuatro técnicas.

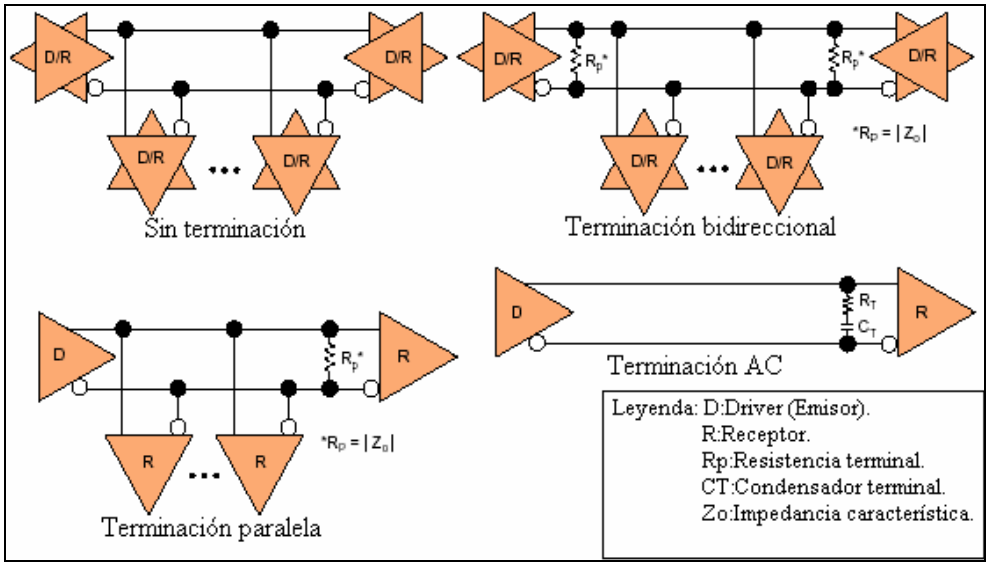


Figura 10 Métodos de terminación usados en redes RS485.

6. **Topologías.** Ciertas topologías de red no son admisibles debido a las reflexiones de la línea, por tanto, topologías como estrella o anillo no son admisibles. La mejor topología es la de bus.
7. **Estados inactivos de la línea.** Cuando un emisor no emplea el bus, éste pone su salida en alta impedancia dejando el bus al aire (flotante), lo que puede conducir que se introduzca ruido considerado como señales. Por ello, habitualmente se pone una línea a tierra (A) y la otra alimentación (B) a través de resistencias (pull-down y pull-up) para evitar eso. Pero estas resistencias también influyen a la hora de la transmisión, ya que producen variaciones en la impedancia de la línea. Para mejorar esto, es conveniente emplazar varias resistencias de pull-up y pull-down a lo largo de la línea en lugar de utilizar una sola resistencia de cada tipo para toda la línea.

2.3 El modelo OSI (Open System Interconnection).

2.3.1 El modelo.

Cuando se produce un intercambio de datos entre equipos a través de un sistema de bus es preciso definir el sistema de transmisión y el método de acceso, así como informaciones relativas al establecimiento de los enlaces. Por este motivo, la *International Standards Organization* (ISO) especificó el **modelo de referencia ISO/OSI**, convertido en un estándar esencial a la hora de describir redes de comunicación y sus diferentes partes en las que se divide. Este modelo propone una serie de niveles o capas para intentar reducir la complejidad de comprensión de estos sistemas. El estándar describe siete capas, de tal modo que una se fundamenta en la anterior, aunque no es necesario emplear todas ellas para construir un sistema de comunicación ya que eso depende de su complejidad y aplicación. Esta separación estructurada permite que exista una independencia de cada capa, de tal modo que cada una puede ser modificada internamente sin afectar al resto, siendo responsable de extraer la información de control contenida en los datos recibidos y necesaria para esa capa, así como de enviar los datos a la siguiente capa. Dentro de cada capa la comunicación se lleva a cabo siguiendo reglas y convenciones predefinidas, que constituyen lo que generalmente se conoce

por **protocolo**. Entre las capas adyacentes debe existir un interfaz que permite el intercambio de información, lo que se conoce como **especificaciones de servicio**. El conjunto total de capas y protocolos constituye la arquitectura de una red.

Este modelo es válido tanto para grandes flujos de información (intercambio de datos entre entidades bancarias) como aplicaciones muy sencillas (transmisión de estado de sensores), por ello, no se establecieron restricciones de tiempo, ya que la prioridad principal es la exactitud de los datos recibidos. Esto supone una limitación para las aplicaciones industriales, pues en estos casos, además de la exactitud de los datos, resulta necesaria una caracterización temporal (condiciones de tiempo crítico), por lo que bajo el modelo OSI han nacido estándares que incluyen dichas restricciones de tiempo en la transmisión. También es necesario comentar que este modelo no es de obligado cumplimiento, sino que constituye un "manual de buenas prácticas" para que el sistema pueda formar parte de los "Sistemas Abiertos". Estas capas del modelo OSI son las que deben ser implementadas en cada nodo de la red, donde la capa 1 constituye el medio físico de transmisión, y la capa 7 es la formada por la aplicación o interfaz de usuario. La Tabla 2 muestra una breve descripción de estas capas [4].

Capa	Nombre	Función	Características
7	**Capa de Aplicación** *Application layer*	Funciones de usuario. Intercambio de variables. Servicios de comunicación específicos de usuario.	Servicios de comunicación: Read/Write, Start/Stop
6	**Capa de Presentación.** *Presentation layer*	Representación de los datos. Conversión del tipo de presentación del sistema de comunicación en un formato adecuado al equipo. Diagnóstico.	
5	**Capa de Sesión.** *Session layer*	Sincronización. Requerimientos de respuestas. Establecimiento, disolución y vigilancia de una sesión.	Coordinación de la sesión.
4	**Capa de Transporte** *Transport layer*	Establecimiento / disolución de enlace. Formación, repetición y clasificación de paquetes.	Transmisión asegurada de paquetes.
3	**Capa de Red** *Network layer*	Direccionamiento de otras redes y control de flujo. Rutas de comunicación.	Comunicación entre dos subredes.
2	**Capa de enlace de Datos** *Data link layer*	Método de acceso. Gestión de colisiones. Limitación de los bloques de datos, transmisión asegurada, detección y eliminación de errores.	Chequeo CRC. CSMA/CD Token
1	**Capa Física** *Physical layer*	Medio físico de transmisión. Test de errores a nivel de bit.	Cable coaxial / triaxial. Cable óptico. Cable bifilar. ITP

Tabla 2 Capas y descripción del modelo OSI.

2.3.2 Funciones de los niveles.

Nivel 1: Capa Física *(Physical Layer)*. Este nivel procura la transmisión transparente de bits a través del soporte físico en el orden definido por el nivel de enlace (capa 2). Se definen las características eléctricas y mecánicas de la línea de transmisión (bus), así como conectores o medios de enlace hardware. También define los sistemas de modulación y demodulación de la señal transmitida/recibida, las señales de control que determinan la temporización y el orden de transmisión, además realiza un diagnóstico de errores a nivel de bit. Entre otros estándares usados en este nivel, los más conocidos son el RS-232 y el RS-422. El cable de conexión no

pertenece a este nivel ya que el modelo sólo se aplica a los nodos de la red y no a la red misma.

Nivel 2: Capa de enlace de datos (**Data Link Layer**). Este nivel tiene como función asegurar la transmisión de la cadena de bits entre dos sistemas. Este nivel es el encargado de recoger los datos del nivel de red (capa 3) para formar las tramas de envío (añadiendo datos de control), y viceversa. También impone los métodos de direccionamiento, detección y recuperación de errores, reenvío de tramas perdidas y regulación del tráfico de información en cuanto a velocidades de transmisión. En redes locales, el nivel de enlace procura también el acceso exclusivo al soporte de transmisión (acceso al medio). Para ello, dicho nivel se divide en dos subniveles, *Medium Access Control* (MAC) y *Logic Link Control* (LLC), que se designan también como niveles 2a y 2b respectivamente. Las normas más conocidas para los métodos de acceso aplicados en el subnivel MAC son IEEE 802.3 (Ethernet, CSMA/CD), IEEE 802.4 (Token Bus), IEEE 802.5 (Token Ring). Para el subnivel LLC se aplica generalmente la norma IEEE 802.2, aunque debido a las características de tiempo real exigidas normalmente a sistemas de bus de campo, éstos utilizan métodos de acceso considerablemente modificados.

Nivel 3: Capa de red (Network Layer). Este nivel se encarga de la operatividad de la red, controlando la ruta de la comunicación de datos entre sistemas finales (nodos y caminos), entendiendo por sistemas finales el emisor y el receptor de una información cuyo recorrido puede llevar bajo circunstancias a través de diversos sistemas de tránsito. Por ello, el nivel de red debe seleccionar la ruta a seguir, lo que normalmente se denomina encaminamiento *(Routing*). Las estaciones, por medio de este nivel añaden una cabecera indicando la dirección de destino, asegurando que el encaminamiento de los paquetes de datos es apropiado para poder llegar hasta su destino. Este nivel es encargado de traducir nombres lógicos en direcciones físicas y controlar la congestión en la red. Conforme la red posee una topología más compleja, esta tarea resulta más complicada. En un enlace punto a punto no entra en juego este nivel.

Nivel 4: Capa de Transporte (Transport Layer). Este nivel entra en juego una vez que se ha producido el enlace entre nodos en la red. La comunicación es ya independiente de la red, siendo el nivel que enlaza lo que quiere transmitir el usuario con la información que hay que enviar. Este nivel tiene como misión ofrecer al usuario un enlace entre nodos fiable, entregando datos libres de error al nivel 5. Puede dividir la conexión para hacerla más rápida (varias conexiones al nivel de transporte). Los servicios ofrecidos incluyen el establecimiento del enlace de transporte, la transmisión de datos, así como la disolución del enlace. Para ello el usuario puede exigir, en general, una determinada calidad en el servicio (Qo*S, Quality of Servic*e). Parámetros de calidad son, por ejemplo, la velocidad de transferencia y la tasa de errores residuales.

Nivel 5: Capa de Sesión (Session Layer). La tarea principal del nivel de sesión es sincronizar las relaciones de comunicación, es decir, permitir establecer una sesión de comunicación entre dos capas de aplicación (nivel 7), una para cada nodo. El inicio de una sesión implica un conjunto de acciones de comunicación para establecer un proceso unitario (como transmitir un fichero, por ejemplo) que se distribuye en: control de comunicaciones uni ó bidireccional, administración del testigo, evitando que ambos lados traten de realizar la misma operación simultáneamente y establecimiento de puntos de chequeo en la información (puntos de

sincronización). En caso de error sólo es necesario retransmitir de nuevo desde el último chequeo. También permite configurar el tipo de diálogo (full-duplex o semi-duplex), así como realizar ciertas verificaciones de seguridad. Esta capa no aparece en numerosos sistemas de comunicación.

Nivel 6: Capa de Presentación (Presentation Layer). Resuelve el problema de semántica y sintaxis de la información transmitida. Generalmente, al intercambiar datos, diferentes sistemas utilizan lenguajes distintos. El nivel de presentación traduce los diversos lenguajes de las estaciones de comunicación a un lenguaje unificado con una sintaxis abstracta para permitir un diálogo entre diferentes sistemas. Así, este nivel convierte los datos del nivel 7 a un lenguaje que es el acordado para la transmisión (aquí también podría incluirse la encriptación y compresión de datos), y modifica los datos recibidos para que la aplicación reciba los datos conforme a su criterio. Para ello se utiliza en la mayor parte de los casos el *Abstract Syntax Notation One* (ASN.1) definido en ISO 8824 y las *Basic Encoding Rules* (BER) asociadas.

Nivel 7: Capa de Aplicación (Application Layer). El nivel de aplicación comprende los servicios específicos de enlace con las diferentes aplicaciones de comunicación. Como existen multitud de aplicaciones, es particularmente difícil establecer estándares unificados, puesto que las aplicaciones propiamente dichas no forman parte del modelo. Habitualmente incluye protocolos de uso general tales como la forma de iniciar y cerrar una sesión de comunicaciones. Existen numerosas propuestas de protocolos orientados a determinados tipos de aplicaciones. Para aplicaciones de automatización se tiene el *Manufacturing Message Specification* (MMS), que describe los servicios y protocolos del nivel de aplicación (MAP, *Manufacturing Automation Protocol*). Los sistemas de bus de campo modernos se orientan fuertemente en MMS a la hora de diseñar el nivel de aplicación.

Para lograr un entendimiento suficiente y seguro son imprescindibles los niveles 1, 2 y 4. El nivel 1 define las condiciones físicas, entre otras, los niveles de tensión y corriente. El nivel 2 define el mecanismo de acceso y el direccionamiento de la estación, para que en un determinado instante sólo pueda enviar datos una de las estaciones del bus. La seguridad y coherencia de los datos se garantiza gracias a la función del nivel 4, el de transporte. Este nivel también se ocupa de tareas de control de flujo de datos, de seccionar bloques o paquetes y de los mecanismos de acuse de recibo o confirmación.

En resumen podemos decir que los niveles OSI 1 y 2 proporcionan el transporte de datos básico para una red simple. Los niveles 3 y 4 extienden estas funciones para una red compleja compuesta de muchas redes simples con diferentes propiedades. Los niveles 5 y 6 proporcionan un marco de trabajo para establecer y negociar las comunicaciones orientadas por el usuario y finalmente el nivel 7 proporciona los medios para comunicar la aplicación final con los procesos de envío y recepción.

Se puede considerar que el flujo de los datos en un sistema de comunicación experimenta un tratamiento o “empaquetado” similar al de un objeto que se desea enviar por correo: a cada nivel del modelo OSI corresponde un tratamiento similar a las diversas fases de embalaje del objeto. La transmisión a través de la red corresponde entonces al envío del paquete, mientras que a la recepción, cada nivel del modelo OSI se encarga de desempaquetar la información agregada al embalaje, procediendo en sentido inverso, iniciando del envoltorio externo a los más internos. Cada nivel a la recepción se ocupa de desempaquetar lo que fue agregado a los datos originales al momento de la transmisión del nivel correspondiente.

2.3.3 Topologías de red.

Se llaman topologías de red a las diferentes estructuras de intercomunicación en que se pueden organizar las redes de transmisión de datos entre dispositivos. Cuando componentes de automatización autónomos tales como sensores, actuadores, autómatas programables, robots, etc., intercambian información, éstos deben interconectarse físicamente con una estructura determinada. Cada topología de red lleva asociada una topología física y una topología lógica. La primera (topología física), es la que define la estructura física de la red, es decir, la manera en la que debe ser dispuesto el cable de interconexión entre los elementos de la red (Figura 11). La topología lógica es un conjunto de reglas normalmente asociado a una topología física, que define el modo en el que se gestiona la transmisión de los datos en la red. La utilización de una topología influye en el flujo de información (velocidad de transmisión, tiempos de llegada, etc.), en el control de la red, y en la forma en la que ésta se puede expandir y actualizar.

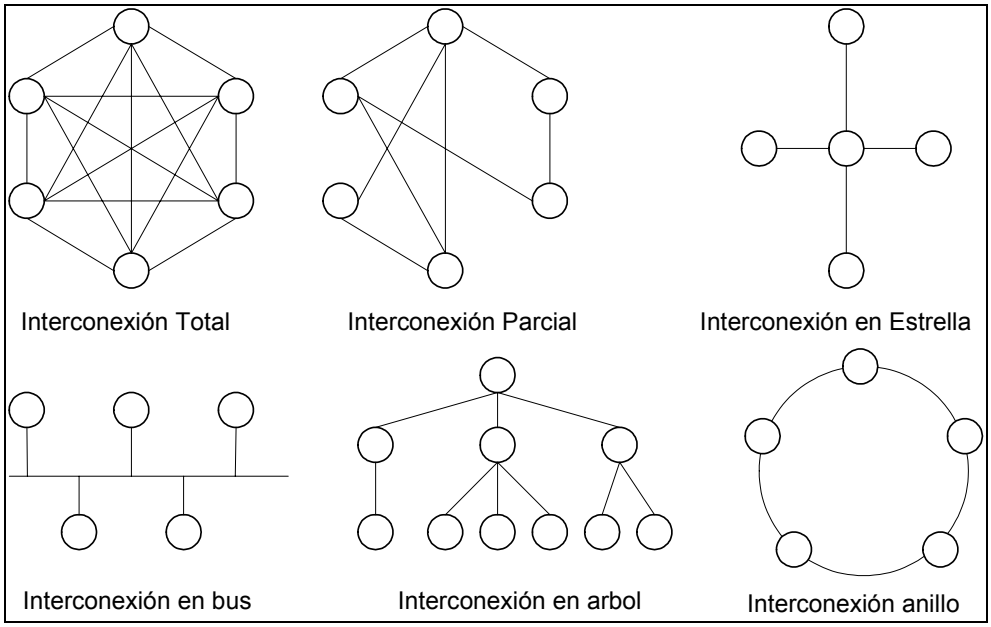


Figura 11 Topologías de red.

- **Interconexión total y parcial.** Este tipo de interconexión proporciona múltiples enlaces físicos entre los nodos de la red, de tal modo que no existen varios canales de comunicación compartidos y múltiples caminos de interconexión entre dos nodos. La interconexión es total cuando todos los nodos están conectados de forma directa entre ellos, existiendo siempre un enlace punto a punto para su intercomunicación. La interconexión es parcial cuando no todos los nodos pueden conectarse mediante un enlace punto a punto con cualquier otro nodo de la red.
- **Interconexión en estrella.** Cada nodo se conecta a un nodo central encargado del control de acceso a la red por el resto de nodos (colisiones, errores, etc.). En esta topología adquiere una importancia decisiva el nodo central que se encarga de controlar toda la comunicación, pues cualquier perturbación en el mismo conduce, generalmente, al fallo de la red completa. Su implementación puede ser una decisión factible en el caso de que los nodos de la red no se encuentren muy distanciados del nodo central debido al coste que supone cablear cada nodo hasta el nodo central.

- **Interconexión en bus.** Todos los nodos se conectan a un único medio de transmisión utilizando adaptadores, encargados de controlar el acceso al bus. Los mensajes se envían por el bus y todos los nodos escuchan, aceptando los datos sólo en el caso de que vayan dirigidos a él (reconocimiento de su propia dirección). Esta topología permite la adición y sustracción de nodos sin interferir en el resto, aunque un fallo en el medio de transmisión inutiliza por completo la red (rotura del cable, por ejemplo). Suelen ser necesarios terminadores de red para poder adaptar impedancias y evitar reflexiones de las ondas transmitidas y recibidas. Los nodos se deben conectar a la línea de bus principal mediante segmentos cortos pues ello influye directamente en la velocidad de transmisión y recepción de datos para ese nodo. Esta es una de las topologías más utilizadas habitualmente. Puede cubrir largas distancias empleando amplificadores y repetidores. Poseen un coste reducido, siendo las más sencillas de instalar. La respuesta es excelente con poco tráfico, siendo empleadas en redes pequeñas y con poco tráfico
- **Interconexión en árbol.** Esta topología puede interpretarse como el encadenamiento de diferentes estructuras en bus de diferente longitud y de características diferenciadas, constituyendo diferentes ramas de interconexión. En este caso adquieren gran importancia los elementos que permiten duplicar y enlazar las diferentes líneas, ya que actúan como nodos principales de manera análoga a como lo hace el nodo principal de la interconexión en estrella. Dado que existen varias estructuras de bus, cada una debe incorporar sus terminadores y elementos asociados, así como los elementos de enlace.
- **Interconexión en anillo.** Los nodos se conectan en serie alrededor del anillo. Sería equivalente a unir los extremos de una red en bus. Los mensajes se transmiten en una dirección (actualmente ya existen topologías en red con envío en ambos sentidos), pasando por todos los nodos necesarios hasta llegar a su destino. No existe un nodo principal y el control de la red queda distribuido entre todos los nodos. Cuando la red es ampliada o reducida, el funcionamiento queda interrumpido, y un fallo en la línea provoca la caída de la red. También se la conoce como red *"testigo en anillo"* o *"Token ring"*. Posee una relación coste – modularidad buena, en general, la instalación es complicada, aunque es fácil variar el número de estaciones. No influyen los fallos en las estaciones si no condicionan la capacidad del interfaz del anillo. Es muy sensible a fallos en los módulos de comunicaciones (interfaz) y en el medio de comunicación. El retardo es grande para un número de estaciones elevado.

Además de las topologías mencionadas, pueden conformarse diferentes topologías que mezclan varios tipos básicos de interconexión mediante la inclusión de elementos de enlace como repetidores, concentradores (hub), puentes (bridge), pasarelas (gateway) o encaminadores (router). Estos elementos pueden incluir cierto nivel de computación en el manejo de la información para poder adaptar dos tipos de redes diferentes, o bien pueden consistir en meros retransmisores de la señal a otros segmentos de la red.

El **repetidor** (***repeater***) copia la información que recibe de un lado en el otro y amplifica su nivel. El repetidor es transparente a todos los niveles de las estaciones en comunicación, es decir, los niveles físicos de ambas redes deben ser idénticos. Por ello, los repetidores no se utilizan para acoplar subredes diferentes, sino para amplificar o prolongar una subred existente como por ejemplo una interconexión de bus (ver Figura 12 Caso a).

Los **puentes** (***bridge***) se utilizan para acoplar subredes que trabajan con el mismo protocolo en el nivel de enlace (Logical Link Control, LLC). Los soportes de transmisión y los métodos de acceso al bus (Medium Access Control, MAC) de las subredes a enlazar pueden

ser diferentes. Los puentes se utilizan principalmente para unir redes locales que tienen diferente topología o cuando, en base a aplicaciones especiales, es necesario añadir determinadas estructuras a subredes. Ese tipo de puentes se utilizan en subredes que, si bien utilizan un soporte de transmisión diferente (cable bifilar, fibra óptica), tienen la misma estructura (ver Figura 12 Caso b).

El **encaminador** (***router, enrutador***) sirve para enlazar redes OSI con niveles 1 y 2 diferentes. El encaminador determina además el camino óptimo (ruta de comunicación) de una información a través de una red existente (***routing***). Criterios para definir el camino óptimo pueden ser, por ejemplo, la longitud del recorrido o el retardo de transmisión mínimo. Para cumplir su tarea, el encaminador modifica las direcciones de origen y destino del nivel de la red de los paquetes entrantes antes de volver a transmitirlos. Como los encaminadores tienen que ejecutar tareas sensiblemente más complejas que los puentes, trabajan a menor velocidad (ver Figura 12 Caso c).

Una **pasarela** (***gateway, puerta de enlace***) se utiliza para acoplar redes con diferentes arquitecturas, es decir, permite interconectar dos subredes cualesquiera. En base al modelo de referencia OSI, una pasarela tiene como misión convertir los protocolos de comunicación de todos los niveles. Permite también acoplar una red ISO con una no conforme a esta norma. Los enlaces de red materializados mediante pasarela suponen complicaciones y ofrecen una velocidad más reducida (ver Figura 12 Caso d).

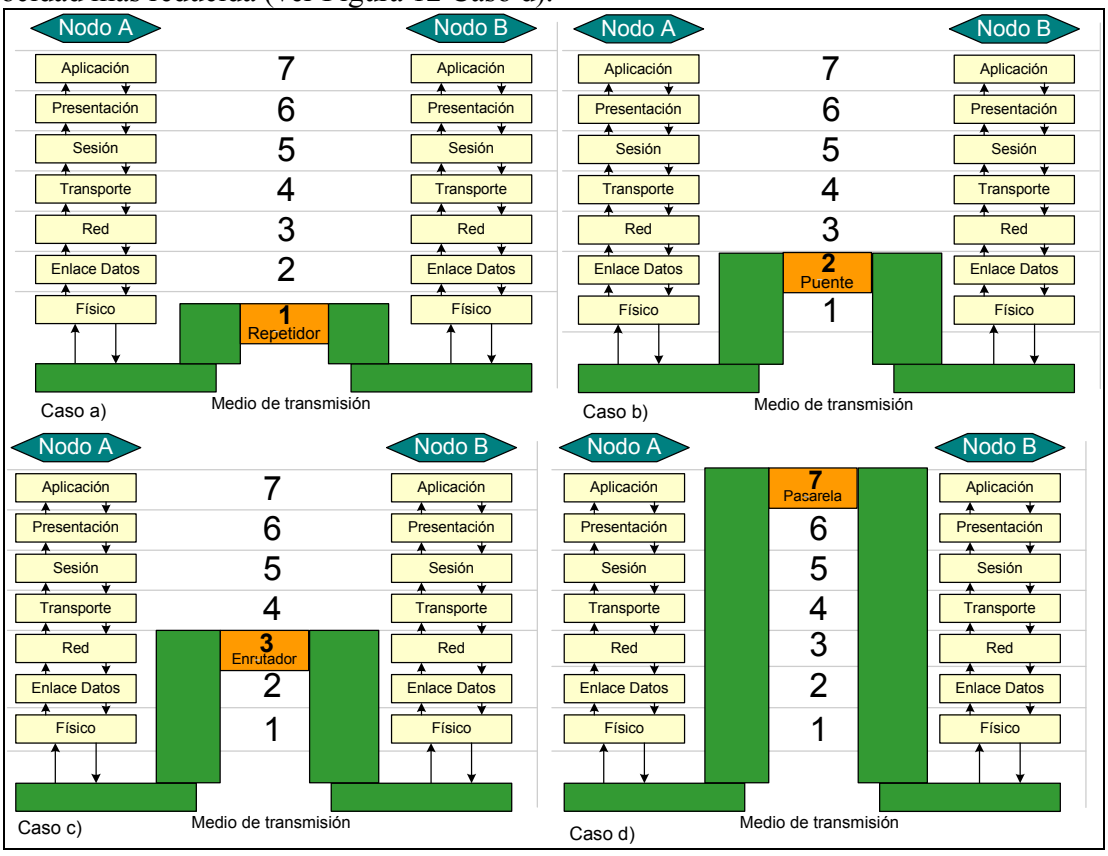


Figura 12 Elementos de enlace entre nodos o estaciones: repetidor, puerta de enlace, enrutador y pasarela.

2.3.4 Familia de protocolos TCP/IP

La familia de protocolos TCP/IP, usados en Internet, se desarrolló antes que el modelo OSI. Por tanto, los niveles del **protocolo de control de transmisión / Protocolo de red** (**TCP/IP**, *Transport Control Protocol / Internet Protocol*) no coinciden exactamente con los del modelo OSI. La familia de protocolos TCP/IP esta compuesta por cinco niveles: físico, enlace de datos, red, transporte y aplicación. Los primeros cuatro niveles proporcionan estándares físicos, interfaces de red, conexión entre redes y funciones de transporte que se corresponden con los cuatro primeros niveles del modelo OSI. Sin embargo, los tres niveles superiores del modelo OSI están representados en TCP/IP mediante un único nivel denominado aplicación, donde funcionan las aplicaciones de usuario (Telnet, FTP, SMTP, HTTP).

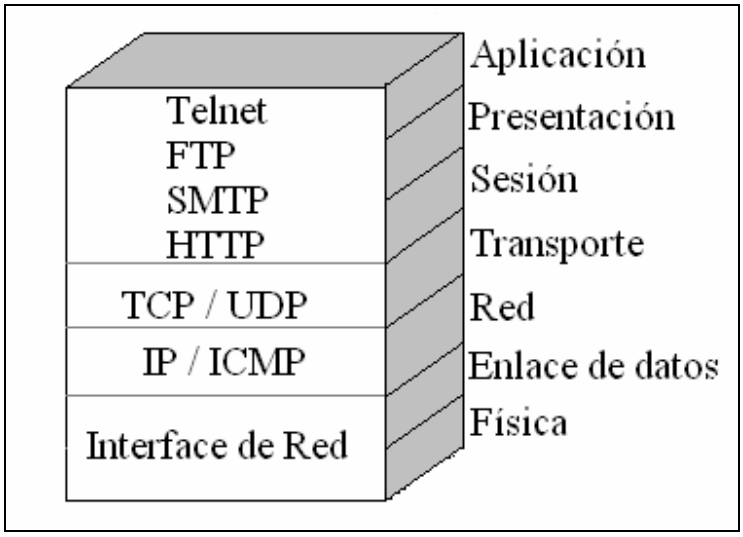


Figura 13 Modelo de capas TCP/IP.

TCP/IP es un protocolo jerárquico compuesto por módulos interactivos, cada uno de los cuales proporciona una funcionalidad específica, pero que no son necesariamente interdependientes. Mientras el modelo OSI especifica qué funciones pertenecen a cada uno de sus niveles, los niveles de la familia de protocolos TCP/IP contienen protocolos relativamente independientes que se pueden mezclar y hacer coincidir dependiendo de la necesidad del sistema. El término *jerárquico* significa que cada protocolo de nivel superior está soportado por uno o más protocolos de nivel inferior.

TCP/IP define dos protocolos en el nivel de transporte: Protocolo de Control de Transmisión (TCP) y Protocolo de Datagramas de Usuario (UDP). En el nivel de red, el principal protocolo definido por TCP/IP es el Protocolo de Redes (IP), aunque hay algunos otros protocolos que proporcionan movimiento de datos en este nivel como ICMP (véase Figura 13).

En los niveles: físico y enlace de datos, TCP/IP no define ningún protocolo específico. Soporta todos los protocolos estándares y propietarios existentes en estos niveles.

La Figura 14 muestra el encapsulado de las unidades de datos en niveles diferentes del conjunto de protocolos TCP/IP. La unidad de datos creada en la unidad de aplicación se denomina *Mensaje*. TCP o UDP agrega un encabezado para crear una unidad de datos denominada *Segmento o Datagrama de Usuario*. El nivel IP hace lo propio y crea una unidad

de datos denominada *Datagrama*. La transferencia de datagramas a través de Internet es responsabilidad del protocolo TCP/IP.

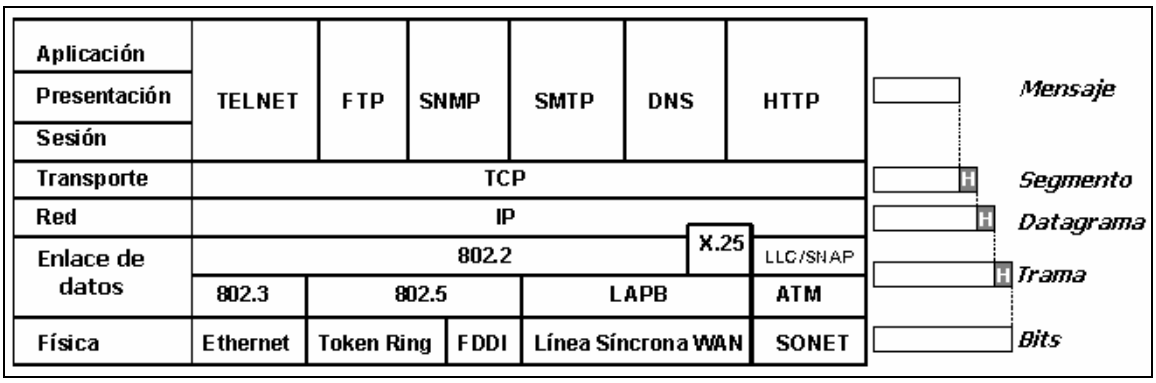


Figura 14 Características del modelo TCP/IP.

Para poder ser transferido físicamente de una red a otra, el datagrama debe encapsularse en una trama en el nivel de enlace de datos de la red subyacente, que puede estar fundamentada en el estándar conocido como *Proyecto* 802 (desarrollado por IEEE en 1985) o protocolos de enlace como X.25, ATM u otros. Finalmente será transferido por el medio de transmisión en forma de datos binarios, basando esta transferencia en los estándares físicos con su propio método de acceso al medio como Ethernet, Token Ring u otro [4].

2.4 Sistemas operativos.

2.4.1 Definición.

Un sistema Operativo (SO) es en sí mismo un programa de computadora. Sin embargo, es un programa muy especial, quizá el más complejo e importante en una computadora. El SO despierta a la computadora y hace que reconozca a la CPU, la memoria, el tecla do, el sistema de vídeo y las unidades de disco. Además, proporciona la facilidad para que los usuarios se comuniquen con la computadora y sirve de plataforma a partir de la cual se corran programas de aplicación.

Cuando se enciende una computadora, lo primero que ésta hace es llevar a cabo un autodiagnóstico llamado auto prueba de encendido (Power On Self Test, POST). Durante la POST, la computadora identifica su memoria, sus discos, su teclado, su sistema de vídeo y cualquier otro dispositivo conectado a ella. Lo siguiente que la computadora hace es buscar un SO para arrancar (boot).

Una vez que la computadora ha puesto en marcha su SO, mantiene al menos parte de éste en su memoria en todo momento. Mientras la computadora esté encendida, el SO tiene 4 tareas principales:

- Proporcionar ya sea una interfaz de línea de comando o una interfaz gráfica al usuario, para que este último se pueda comunicar con la computadora.
- Administrar los dispositivos de hardware en la computadora. Cuando corren los programas, necesitan utilizar la memoria, el monitor, las unidades de disco, los puertos de Entrada/Salida (impresoras, módems, etc.). El SO sirve de intermediario entre los programas y el hardware.

- Administrar y mantener los sistemas de archivo de disco. Los SO agrupan la información dentro de compartimientos lógicos para almacenarlos en el disco. Estos grupos de información son llamados archivos. Los archivos pueden contener instrucciones de programas o información creada por el usuario. El SO mantiene una lista de los archivos en un disco, y nos proporciona las herramientas necesarias para organizar y manipular estos archivos.
- Apoyar a otros programas. Otra de las funciones importantes del SO es proporcionar servicios a otros programas. Estos servicios son similares a aquellos que el SO proporciona directamente a los usuarios. Por ejemplo, listar los archivos, grabarlos a disco, eliminar archivos, revisar espacio disponible, etc. Cuando los programadores escriben programas de computadora, incluyen en sus programas instrucciones que solicitan los servicios del SO. Estas instrucciones son conocidas como "llamadas del sistema"

Las funciones centrales de un SO son controladas por el núcleo (kernel) mientras que la interfaz del usuario es controlada por el entorno (shell). Por ejemplo, la parte más importante del DOS es un programa con el nombre "COMMAND.COM" Este programa tiene dos partes. El kernel, que se mantiene en memoria en todo momento, contiene el código máquina de bajo nivel para manejar la administración de hardware para otros programas que necesitan estos servicios, y para la segunda parte del COMMAND.COM el shell, el cual es el interprete de comandos.

Las funciones de bajo nivel del SO y las funciones de interpretación de comandos están separadas, de tal forma que puedes mantener el kernel DOS corriendo, pero utilizar una interfaz de usuario diferente. Esto es exactamente lo que sucede cuando se carga Microsoft Windows, el cual toma el lugar del shell, reemplazando la interfaz de línea de comandos con una interfaz gráfica del usuario. Existen muchos shells diferentes en el mercado, ejemplo: NDOS (Norton DOS), XTG, PCTOOLS, o inclusive el mismo SO MS-DOS a partir de la versión 5.0 incluyó un Shell llamado DOS SHELL [7].

2.4.2 Categorías de sistemas operativos.

2.4.2.1 Multitarea.

El término multitarea se refiere a la capacidad del SO para correr mas de un programa al mismo tiempo. Existen dos esquemas que los programas de sistemas operativos utilizan para desarrollar SO multitarea, el primero requiere de la cooperación entre el SO y los programas de aplicación, y el segundo método es el llamado multitarea con asignación de prioridades..

En el primer caso los programas son escritos de tal manera que periódicamente inspeccionan con el SO para ver si cualquier otro programa necesita a la CPU, si este es el caso, entonces dejan el control del CPU al siguiente programa, a este método se le llama multitarea cooperativa y es el método utilizado por el SO de las computadoras de Macintosh y DOS corriendo Windows de Microsoft. En el segundo método el SO mantiene una lista de procesos (programas) que están corriendo. Cuando se inicia cada proceso en la lista el SO le asigna una prioridad. En cualquier momento el SO puede intervenir y modificar la prioridad

de un proceso organizando en forma efectiva la lista de prioridad, el SO también mantiene el control de la cantidad de tiempo que utiliza cualquier proceso antes de ir al siguiente. Con multitarea de asignación de prioridades el SO puede sustituir en cualquier momento el proceso que esta corriendo y reasignar el tiempo a una tarea de mas prioridad. UNIX OS-2 y Windows NT emplean este tipo de multitarea.

2.4.2.2 Multiusuario.

Un SO multiusuario permite a mas de un solo usuario accesar una computadora. Claro que, para llevarse esto a cabo, el SO también debe ser capaz de efectuar multitareas. UNIX es el Sistema Operativo Multiusuario más utilizado. Debido a que UNIX fue originalmente diseñado para correr en una mini computadora, era multiusuario y multitarea desde su concepción.

Actualmente se producen versiones de UNIX para PC tales como The Santa Cruz Corporation Microport, Esix, IBM,y Sunsoft. Apple también produce una versión de UNIX para la Machintosh llamada: A/UX.UNIX

UNIX proporciona tres maneras de permitir a múltiples personas utilizar la misma PC al mismo tiempo.

- Mediante Módems.
- Mediante conexión de terminales a través de puertos seriales
- Mediante Redes.

2.4.2.3 Multiproceso.

Las computadoras que tienen más de un CPU son llamadas multiproceso. Un sistema operativo multiproceso coordina las operaciones de las computadoras multiprocesadoras. Ya que cada CPU en una computadora de multiproceso puede estar ejecutando una instrucción, el otro procesador queda liberado para procesar otras instrucciones simultáneamente.

Al usar una computadora con capacidades de multiproceso incrementamos su velocidad de respuesta y procesos. Casi todas las computadoras que tienen capacidad de multiproceso ofrecen una gran ventaja.

Existen dos tipos de multiproceso en cuanto a la independencia de las CPU que describen comportamientos diferentes, estos son:

- Multiproceso asimétrico: Una CPU principal retiene el control global de la computadora, así como el de los otros procesadores. Esto fue un primer paso hacia el multiproceso pero no fue la dirección ideal a seguir ya que la CPU principal podía convertirse en un cuello de botella.
- Multiproceso simétrico: En un sistema multiproceso simétrico, no existe una CPU controladora única. La barrera a vencer al implementar el multiproceso simétrico es que los SO tienen que ser rediseñados o diseñados desde el principio para trabajar en un ambiente multiproceso. Las extensiones de UNIX, que soportan multiproceso asimétrico ya están disponibles y las extensiones simétricas se están haciendo disponibles. Windows NT y los SO de Microsoft que trabajan bajo el mismo núcleo soportan multiproceso simétrico.

2.4.3 Windows NT de Microsoft.

Antes de realizar la descripción propia del SO Windows NT, es de importancia señalar que los SO Windows 2000 y Windows XP, exhiben características similares a este; todos ellos forman parte de una nueva generación de SO que se desprenden por completo de la dependencia o coexistencia con el MS-DOS. La concepción de estos SO obedece a la necesidad de independizar los conceptos de trabajo e integrar dentro de si todos los servicios existentes.

Con Windows NT, Microsoft ha expresado su dedicación a escribir software no sólo para PC de escritorio sino también para poderosas estaciones de trabajo, servidores de red y bases de datos. Microsoft Windows NT no es necesariamente un sustituto de DOS ni una nueva versión de éste; es un nuevo SO diseñado desde sus bases para las máquinas mas modernas y capaces disponibles.

Windows NT de Microsoft ofrece características internas de construcción que ningún otro SO para PC ofrece, con excepción de UNIX. Además de las características tradicionales de estricta seguridad de sistema, red, servicios de comunicación y correo electrónico internos, herramientas de administración y desarrollo de sistema, y además una robusta interfaz de usuario. Windows NT puede correr directamente aplicaciones de Windows de Microsoft y de UNIX. Windows NT, al igual que el OS/2 versión 2.0 y algunas versiones de UNIX, es un SO de 32 bits, que puede hacer completo uso de los procesadores de estas características. Además de ser multitarea, está diseñado para tomar ventaja del multiproceso simétrico [8].

2.4.3.1 El núcleo (*Kernel).*

El núcleo es la base del sistema operativo, en donde reside el ejecutivo del NT por medio del cual se realizan las siguientes operaciones:

- Entradas y salidas de tareas al sistema.
- Proceso de interrupciones y excepciones.
- Sincronización de los multiprocesadores.
- Recuperación del sistema después de una caída.

2.4.3.1.1 Entradas y salidas de tareas al sistema.

Cada objeto de tipo tarea es creado como respuesta a una solicitud de una aplicación, cada una de las tareas puede encontrarse en los estados de ejecución, espera en cola, espera por recursos, lista para ejecución o finalizada. El kernel cuenta con un módulo llamado despachador que se encarga de permitir la entrada de los procesos y de darlos por terminados. El despachador igualmente examina la prioridad de los procesos para determinar en qué orden van a ser ejecutados; suspendiendo y activando los procesos.

2.4.3.1.2 Proceso de interrupciones y excepciones

En Windows NT se manejan las interrupciones como en cualquier sistema operativo. La llegada de señales por el bus debido a fallas de los programas o por peticiones de

entrada/salida de los periféricos son atrapadas por el núcleo. El paso de los subsistemas hacia los servicios del sistema, se hacen a través de mensajes y de atrapado de interrupciones.

2.4.3.1.3 Sincronización de los multiprocesadores.

Esta característica asegura que sólo una tarea puede acceder un mismo recurso a la vez. En un sistema basado en multiprocesadores con memoria compartida, dos o más procesadores pueden estar ejecutando tareas que necesiten acceder la misma página de memoria o realizar operaciones sobre un mismo objeto. El núcleo y el ejecutivo de NT proveen mecanismos para asegurar la integridad del sistema a través de la sincronización; en el caso del kernel la sincronización es manejada a través de candados colocados en puntos críticos de las instrucciones del nivel despachador, de esta manera, ningún otro procesador puede ejecutar código o acceder datos protegidos por uno de los candados hasta que éste es liberado. El ejecutivo del NT realiza la sincronización a través de la familia de los objetos de sincronización.

2.4.3.1.4 Recuperación del sistema

La última función del kernel consiste en la recuperación del sistema en caso de una caída. Cuando existe una falla de alimentación en un sistema NT se dispara una interrupción de alta prioridad la cual dispara a su vez una serie de tareas diseñadas para preservar la integridad del sistema operativo y de los datos tan rápido como sea posible.
El núcleo de Windows NT contiene una capa de abstracción que es el límite entre el ejecutivo del NT y el específico de la computadora. NT fue diseñado de tal manera que los cambios de código son mínimos para ser acoplados a las diferentes plataformas de hardware tomando como ejemplo los sistemas UNIX.

2.4.3.2 Arquitectura cliente/servidor

Windows NT tiene dos modos de operar, modo usuario y modo privilegiado (kernel). Programas de aplicaciones como una base de datos, una hoja de cálculo, o un sistema de reservaciones de un hotel, siempre son ejecutados en modo usuario. El ejecutivo de NT es el corazón del sistema. El ejecutivo de NT realiza tareas como el manejo de entradas y salidas, la memoria virtual, y todos lo procesos, además de controlar las ligas entre NT y el hardware de la computadora. El ejecutivo de NT es ejecutado en modo kernel, el cual es un modo de alta seguridad libre de interferencias de los procesos de los usuarios

En modo usuario, hay también los llamados subsistemas protegidos. Un ejemplo de estos es el Win32 API. Usando esta API los programadores no tienen porque preocuparse acerca del hardware donde el programa va a ser ejecutado y por otro lado protege al sistema de aquellos programadores que traten de modificar su memoria y hacer que falle el sistema. Adicionalmente el API tiene reglas de seguridad que protegen a los otros subsistemas de interferencias entre ellos.

En el ambiente de NT los programas de aplicación de los usuarios son los clientes y los subsistemas protegidos son los servidores. Las aplicaciones (clientes) mandan mensajes a los subsistemas protegidos a través del ejecutivo de NT, el cual provee un conjunto de servicios

compartidos para todos los servidores. Y a su vez los servidores contestan a los clientes de la misma forma.

En NT, los servidores ejecutándose en un procesador local pueden mandar mensajes de sus clientes a otros servidores que estén siendo ejecutados en procesadores remotos sin que se necesite que el cliente sepa algo de los servidores remotos.

El modelo cliente/servidor hizo que el sistema operativo fuera más eficiente eliminando recursos duplicados y elevó el soporte que ofrece el sistema operativo para multiproceso y redes. Esta arquitectura permite que otros API's sean añadidos sin tener que aumentar un nuevo ejecutivo de NT para su manejo. Por otro lado cada subsistema es un proceso separado en su propias memorias protegidas, así, si uno de los subsistemas falla no hace que todo el sistema falle también.

El ejecutivo NT es un sistema operativo completo que no cuenta con interfaz y está compuesto de capas, siendo éstas las siguientes:

- Servicios del sistema: son las llamadas al sistema que sirven como medio de comunicación entre los modos de los procesos y los componentes del ejecutivo. La manera en que interactúan los dos componentes anteriormente mencionados es a través de llamadas al sistema; en otras palabras los servicios del sistema son el API para el modo de usuario.
- Componentes del ejecutivo: el ejecutivo de NT tiene seis componentes primarios cada uno de los cuales realiza el siguiente conjunto de operaciones críticas del sistema: manejador de objetos, monitoreo de la seguridad del sistema, manejador de procesos, facilidad para la llamada de procesos locales, manejador de la memoria virtual y manejador de las entradas y salidas.

2.4.3.3 Manejo de procesos en Windows NT

En la arquitectura de NT los procesos son segmentados en componentes más pequeños llamados 'threads'. Windows NT soporta varias tareas al mismo tiempo. Existen dos tipos de multitarea, el apropiativo (preemptive) y el no apropiativo (no preemptive). Con la multitarea apropiativa la ejecución de un 'thread' puede ser suspendida después de un tiempo determinado (time slice) por el sistema operativo para permitir que otro thread sea ejecutado. Mientras que con la multitarea no apropiativa, es el thread el que determina cuándo le regresará el control al sistema operativo para permitir que otro thread sea ejecutado. NT así como OS/2 y UNIX usan apropiativo multitarea para soportar la ejecución "simultánea" de varios procesos.

2.4.3.3.1 Manejador de Procesos.

El manejador de procesos es un componente ambiental que crea y destruye procesos y tareas, como el manejador de objetos, el manejador de procesos ve los procesos como si fueran objetos, en efecto el manejador de procesos puede ser considerado como una instancia específica del manejador de objetos porque dicho manejador crea, maneja y destruye un sólo tipo de objetos.

Se puede únicamente distinguir una funcionalidad adicional al manejador de objetos con la que cuenta el manejador de procesos que consiste en el manejo de la estadía de cada uno de los procesos (ejecutar, suspender, reiniciar, terminar una tarea).

Las llamadas a procedimientos locales (LPC, *Call Process Local*) son usadas para pasar mensajes entre dos diferentes procesos corriendo dentro de un mismo sistema NT, estos sistemas fueron diseñados utilizando como modelo las llamadas a procedimientos remotos (RPC, *Call Process Remote*); los RPC consisten en una manera estandarizada de pasar mensajes entre un cliente y un servidor a través de una red. Similarmente los LPC's pasan mensajes de un procedimiento cliente a un procedimiento servidor en un mismo sistema NT.

Cada proceso cliente en un sistema NT que tiene capacidad de comunicación por medio de LPC's debe tener por lo menos un objeto de tipo puerto asignado a él, este objeto tipo puerto es el equivalente a un puerto de TCP/IP en un sistema UNIX.

2.4.4 Lenguajes de programación.

Los lenguajes de programación cierran el abismo entre las computadoras, que sólo trabajan con números binarios, y los humanos, que preferimos utilizar palabras y otros sistemas de numeración [10].

Mediante los programas se indica a la computadora qué tarea debe realizar y como efectuarla, pero para ello es preciso introducir estas ordenes en un lenguaje que el sistema pueda entender. En principio, el ordenador sólo entiende las instrucciones en código máquina, es decir, el específico de la computadora. Sin embargo, a partir de éstos se elaboran los llamados lenguajes de alto y bajo nivel.

Los lenguajes de bajo nivel utilizan códigos muy cercanos a los de la máquina, lo que hace posible la elaboración de programas muy potentes y rápidos, pero son de difícil aprendizaje y retardan el desarrollo de aplicaciones debido a lo complicado de su entorno de funcionamiento.

Los lenguajes de alto nivel por el contrario, son de uso fácil, ya que en ellos un solo comando o instrucción puede equivaler a millares en código máquina. El programador escribe su programa en alguno de estos lenguajes mediante secuencias de instrucciones. Antes de ejecutar el programa la computadora lo traduce a código máquina de una sola vez (lenguajes compiladores) o interpretándolo instrucción por instrucción (lenguajes intérpretes). Ejemplos de lenguajes de alto nivel: Pascal, Cobol, Basic, Fortran, C++, etc. Un Programa de computadora, es una colección de instrucciones que, al ser ejecutadas por el CPU de una máquina, llevan a cabo una tarea ó función específica. Este conjunto de instrucciones que forman los programas son almacenados en archivos denominados archivos ejecutables puesto que, al teclear su nombre (o hacer clic sobre el icono que los identifica) logras que la computadora los cargue y corra, o ejecute las instrucciones del archivo. El contenido de un archivo ejecutable no puede ser entendido por el usuario, ya que no está hecho para que la gente lo lea, sino para que la computadora sea quien lo lea.

Los archivos de programas ejecutables contienen el código máquina, que la CPU identifica como sus instrucciones. Son lo que conocemos como Programas Objeto. Dado que sería muy difícil que los programadores crearan programas directamente en código de máquina, usan lenguajes más fáciles de leer, escribir y entender para la gente.

El programador teclea instrucciones en un editor, que es un programa parecido a un simple procesador de palabras, estas instrucciones son almacenadas en archivos denominados programas fuentes (código fuente). Si los programadores necesitan hacer cambios al programa posteriormente vuelven a correr el editor y cargan el programa fuente para modificarlo.

El proceso de conversión de programas fuente a programas objeto se realiza mediante un programa denominado compilador. El compilador toma un programa fuente y lo traduce a programa objeto y almacena este último en otro archivo [9].

El programa fuente es el programa escrito en alguno de los lenguajes y que no ha sido traducido al lenguaje de la maquina, es decir el programa que no está en código de máquina y que por lo tanto no puede ser ejecutable. El programa objeto es aquel programa que se encuentra en lenguaje máquina y que ya es ejecutable por esta.

2.4.4.1 Programación orientada a objetos.

La programación orientada a objetos no es un concepto nuevo, sus inicios y técnicas de programación se iniciaron a principios de los 70. Se puede definir programación orientada a objetos (POO) como una técnica de programación que utiliza objetos como bloque esencial de construcción. La POO, es un tipo de programación más cercana al razonamiento humano. La POO surge como una solución a la programación de grandes programas, y para solventar el mantenimiento de dichas aplicaciones, ya que en la programación no estructurada el más mínimo cambio supone la modificación de muchas funciones relacionadas, en cambio con la POO solo es cuestión de añadir o modificar métodos de una clase o mejor, crear una nueva clase a partir de otra (Herencia). Dos lenguajes destacan sobre el resto para programar de esta forma, Smalltalk y C++. A continuación se definen algunas partes importantes de este tipo de programación:

- Concepto de Objeto: Desde un punto de vista general un Objeto es una estructura de datos de mayor o menor complejidad con las funciones que procesan estos datos. Dicho de otra forma, serían datos más un código que procesa estos datos. A los datos se les denomina miembros dato y a las funciones miembros o miembros funciones. Los datos están ocultos y sólo se puede acceder a ellos mediante las funciones miembros.
- Clases: Las Clases son como plantillas o modelos que describen como se construyen ciertos tipos de Objeto. Cada vez que se construye un Objeto de una Clase, se crea una instancia de esa Clase ("instance"). Una Clase es una colección de Objetos similares y un Objeto es una instancia de una Clase. Se puede definir una Clase como un modelo que se utiliza para describir uno o más Objetos del mismo tipo.
- Herencia: Una característica muy importante de los Objetos y las Clases es la Herencia, una propiedad que permite construir nuevos Objetos (Clases) a partir de otras ya existentes. Esto permite crear "Sub-Clases" denominadas Clases Derivadas que comparten las propiedades de la Clase de la cual derivan (Clase base). Las Clases derivadas heredan código y datos de la clase base, asimismo incorporan su propio código y datos especiales. Se puede decir que la herencia permite definir nuevas Clases a partir de las Clases ya existentes.
- Polimorfismo: En un sentido literal, Polimorfismo significa la cualidad de tener más de una forma. En el contexto de POO, el Polimorfismo se refiere al hecho de que una simple operación puede tener diferentes comportamientos en diferentes objetos. En otras palabras, diferentes objetos reaccionan al mismo mensaje de modo diferente. Los primeros lenguajes de POO fueron interpretados, de forma que el Polimorfismo se contemplaba en tiempo de ejecución. Por ejemplo, en C++, al ser un lenguaje compilado, el Polimorfismo se admite tanto en tiempo de ejecución como en tiempo de compilación.

- La encapsulación es la creación de módulos autosuficientes que contienen los datos y las funciones que manipulan dichos datos. Se aplica la idea de la caja negra y un letrero de "prohibido mirar adentro". Los objetos se comunican entre sí intercambiando mensajes. De esta manera, para armar aplicaciones se utilizan los objetos cuyo funcionamiento está perfectamente definido a través de los mensajes que es capaz de recibir o mandar. Todo lo que un objeto puede hacer está representado por su interfase de mensajes. Para crear objetos, el programador puede recurrir a diversos lenguajes como el C++, el Smalltalk, el Visual Objects y otros. Si se desea solamente utilizar los objetos y enlazarlos en una aplicación por medio de la programación tradicional se puede recurrir al Visual Basic, al CA-Realizer, al Builder, etc.

2.4.4.2 Compilador.

Es un programa que traduce un lenguaje de alto nivel al lenguaje máquina. Un programa compilado indica que ha sido traducido y está listo para ser ejecutado. La ejecución de los programas compilados es más rápida que la de los interpretados, ya que el interprete debe traducir mientras está en la fase de ejecución (saca todos los errores). Un compilador es un programa que traduce el programa fuente (conjunto de instrucciones de un lenguaje de alto nivel, por ejemplo BASIC o Pascal) a programa objeto (instrucciones en lenguaje máquina que la computadora puede interpretar y ejecutar). Se requiere un compilador para cada lenguaje de programación. Un compilador efectúa la traducción, no ejecuta el programa. Una vez compilado el programa, el resultado en forma de programa objeto será directamente ejecutable. Presentan la ventaja considerable frente a los intérpretes de la velocidad de ejecución, por lo que su uso será mejor en aquellos programas probados en los que no se esperan cambios y que deban ejecutarse muchas veces. En caso de que se opte por un interpretador se debe considerar que el intérprete resida siempre en memoria.

El intérprete o traductor de lenguajes de programación de alto nivel, ejecuta un programa línea por línea. El programa siempre permanece en su forma original (programa fuente) y el interprete proporciona la traducción al momento de ejecutar cada una de las instrucciones. Un intérprete es un programa que procesa los programas escritos en un lenguaje de alto nivel, sin embargo, está diseñado de modo que no existe independencia entre la etapa de traducción y la etapa de ejecución. Un intérprete traduce cada instrucción o sentencia del programa escrito a un lenguaje máquina e inmediatamente se ejecuta. Encuentran su mayor ventaja en la interacción con el usuario, al facilitar el desarrollo y puesta a punto de programas, ya que los errores son fáciles de detectar y sobre todo de corregir.

2.4.4.3 Comparación de compiladores.

El siguiente texto intenta explicar cuales son las principales diferencias entre los entornos de programación mas utilizados de la actualidad y en que momento según el proyecto conviene uno u otro. En todo documento escrito de esta índole se expresa una opinión y como todas las opiniones son subjetivas, más de un lector podría no estar de acuerdo con lo aquí expresado.

Primero que nada debemos convencernos que no existe ninguna herramienta que cubra todos los intereses o sea la mejor en todos los aspectos; es por esto que empresas como Microsoft o Inprise (Borland) tienen entre sus productos mas de un entorno. En ciertos casos, según el tipo de proyecto, el tiempo que disponemos, el tipo de aplicación, el ciclo de vida del

proyecto, y otras variables tendremos que decidir que herramienta es la que mejor se adapta a nuestra necesidad. En este trabajo evaluaremos los siguientes entornos:

- Microsoft Visual Basic.
- Microsoft Visual C++
- Java Workshop/Microsoft Visual J++.
- Borland C++ Builder.
- Borland Delphi.

Claro que no son todos los entornos que se utilizan en la actualidad, pero cubre la mayor parte de los usos y son las herramientas mas utilizadas de la actualidad en plataformas Windows.

El Visual Basic fue uno de los primeros lenguajes en acercar a personas no dedicadas a la programación a desarrollar aplicaciones sobre plataformas Windows. Si bien en sus primeras versiones el entorno corría en DOS, el producto se hizo popular en sus versiones Windows. Programar en Visual Basic aplicaciones sencillas es muy fácil, se aprende rápidamente, el diseño de sus formularios es sencillo, no es necesaria la declaración de variables. Visual Basic es uno de los lenguajes con el que mas rápido obtenemos soluciones, se avanza muy velozmente gracias a su sencilla sintaxis y a su permisividad, pasamos a modo de ejecución en sólo segundos. Visual Basic es interpretado, el performance de su código exe es bastante bajo, sin embargo es lo suficientemente buena para la mayoría de las aplicaciones de escritorio, además con el avance en velocidad de los microprocesadores esto cada vez pesa menos. Visual Basic al ser un lenguaje interpretado requiere de otra aplicación (Ej: msvbvm60.dll) que entienda el código generado por el entorno, claro que esto es transparente para el usuario final, sin embargo puede ser un problema si queremos realizar una aplicación que sea totalmente transportable, como por ejemplo un instalador. En Visual Basic no existen los punteros, esto es causante de no poder utilizar ciertas librerías o ciertas funciones del API de Windows. Además al no existir punteros no podemos manejar la asignación de memoria en forma dinámica, con lo que perdemos control y probablemente consumamos memoria de más que la misma solución programada en otro lenguaje. El Visual Basic no es 100% orientado a objetos, si bien esto cambiará en las versiones superiores (Visual Basic .NET), actualmente puede ser un problema para algunos desarrolladores que deseen implementar una solución basada en diagramas de clase UML, en donde se aplique herencia, encapsulamiento y polimorfismo.

El Visual C++ es uno de los entornos más serios y poderosos para desarrollar aplicaciones comerciales potentes. El utilizar el lenguaje C/C++ es uno de los mas populares y extendidos de todas las plataformas, la posibilidad de realizar operaciones de bajo nivel en un lenguaje de alto nivel fue la característica mas importante que lo hizo tan popular (con el C se realizaron muchísimas soluciones que antes hubiesen requerido utilizar Assembler). Con Visual C++ se puede realizar casi cualquier tipo de proyecto, desde una aplicación Windows tradicional, hasta un servicio de Windows NT/2000/XP, un control ActiveX, librería DLL, etc. Entorno poderoso y liviano: El entorno de Visual C++ es uno de los más poderosos que existe en el mercado, es estable, intuitivo y ágil. Compilador ultra veloz: El Visual C++ es el compilador más rápido de C/C++ que existe actualmente en el mercado, es casi cinco veces más veloz que su “competidor” C++ Builder. Si no utilizamos librerías especiales o controles ActiveX, el Visual C++ permite crear un ejecutable en donde no se necesite ningún otro componente para ser ejecutado. Realizar una aplicación rica en interfaz de usuario con Visual C++ es desesperante. Su editor de formularios es muy poco flexible. Los controles de la MFC

(*Microsoft Fundattion Clases*) tienen pocas propiedades y menos aún editarlos en tiempo de diseño, además no pueden contener a otros objetos. Es cierto que es posible utilizar controles ActiveX dentro de Visual C++, pero por cada control que insertemos el entorno debe crear una clase para poder acceder fácilmente a sus propiedades y si actualizamos la versión del control (si se agregan métodos, por ejemplo) es muy engorroso indicarle al compilador que debe actualizar esta clase. El Visual C++ dice poder compilar código Ansi C/C++, sin embargo esto no es del todo cierto.

El Java es un lenguaje que prometió revolucionar la Web, objetivo que cumplió solo a medias gracias a la aparición de diversos componentes (como el Macromedia Flash) que lo suplantaron en una gran cantidad de usos para los cuales Java era una posibilidad. Sin embargo, no todo es mostrar animaciones y aplicaciones gráficas interactivas en un cuadro, Java es un lenguaje poderoso de propósito general que puede ser utilizado para un sinfín de aplicaciones donde el Macromedia Flash nada tiene que hacer. El Java es multiplataforma, es decir, el código generado por un compilador Java, conocido como "ByteCodes", puede ser ejecutado en gran variedad de tipos de computadores sin recompilar ni cambiar un solo byte. Los bytecodes deben ser interpretados por una máquina virtual JVM (*Java Virtual Machine*). Esta máquina virtual es una aplicación que debe existir en toda plataforma en donde deseemos correr una aplicación Java. Hoy día, casi cualquier sistema respetable posee su implementación de JVM (Microsoft Windows 95/98/Me/NT/2000/XP, Linux, Sun Solaris, Digital UNIX, MacOs, etc). La sintaxis del Java está basada en C++. Aunque no se puedan manejar los punteros directamente, se puede hacer uso de referencias que brindan casi la misma funcionalidad. Además posee un motor concentrador que permite gestionar la memoria mas fácilmente. Las aplicaciones de Web Java son seguras, en cuanto a que no pueden acceder a recursos internos de la PC sin que se le brinden privilegios específicamente. Además una aplicación descargada desde un servidor no puede querer conectarse a otro. Toda ventaja tiene su contraparte, el hecho que las aplicaciones Java deban ser interpretadas por otra aplicación situada entre nuestro programa y el sistema operativo, resulta en una disminución de performance considerable. Además el uso de la memoria en aplicaciones Java es por lo general mayor. El equipo en donde deseemos correr la aplicación Java debe poseer debidamente instalada una Java Virtual Machine (esto no sucede en todos los casos). Desgraciadamente Microsoft no apoya las soluciones Java, por ser una creación de una empresa competidora (SUN). En un principio Microsoft adhirió a la idea y creó su compilador de Java (Microsoft Visual J++ 1.0), pero comenzó a extender el lenguaje para que desde el mismo se pudiera utilizar las API de Windows. Claro que si no se tenía la suficiente precaución, los bytecodes generados por la aplicación de Microsoft sólo correrían en Windows, cosa que iba en contra de la filosofía del leguaje, por lo que SUN le inició un juicio que terminó con un arreglo fuera del juzgado. Luego Microsoft creó su propio lenguaje muy parecido a Java y dejó descontinuado el Visual J++. Además Microsoft no incluye en su nuevo sistema operativo XP, la instalación de la JVM.

El C++ Builder surgió como una alternativa al Visual C++ luego del gran éxito del Delphi. Se está utilizando cada vez más, aunque está lejos de hacerle sombra al monstruo de Microsoft. Además, es claro que el C++Builder fue creado a partir del Delphi y de hecho usa prestado su gran gama de componentes. Esto hace que en ciertas situaciones a uno le parece que la gente de Borland no se tomó el tiempo suficiente a para crear un entorno totalmente nuevo y adaptado al uso de C++. Una de las principales ventajas del Builder es su buen entorno para crear formularios. Posee muchos controles que a su vez poseen una gran cantidad de propiedades, permite además importar controles ActiveX (carácterística que, a diferencia

de otros entornos, funciona muy bien). El compilador del Builder respeta el estándar Ansi de manera más fiel que el compilador de Microsoft. El entorno de Borland, permite crear una gran cantidad de proyectos. Los controles que utiliza Builder (VCL: Visual Component Library) y los controles ActiveX son una opción. Permite además crear servicios, librerías DLL y hasta proyectos MFC. El entorno del Builder, es mucho mas pesado que el de Microsoft y tarda bastante en cargarse a pesar de poseer una PC poderosa. Su depurador integrado no es tan bueno. Se puede pensar que es un producto aún no maduro y que faltan algunas versiones mas para que realmente logre tener un buen nivel. El compilador del C++ Builder es extremadamente lento si lo comparamos con el del Visual C++.

El Delphi es una gran alternativa para desarrolladores que generalmente utilicen Visual Basic como entorno de programación. En un principio se encontró en el punto medio entre el Visual Basic y el Visual C++, de hecho bien podría decirse que posee lo mejor de los dos mundos. Utiliza como lenguaje el Pascal que posee una gran flexibilidad y es altamente estructurado y es totalmente orientado a objetos desde un principio. El Pascal es un lenguaje poderoso, en el cual se pueden utilizar punteros y administrar la memoria eficientemente. Además es altamente estructurado, lo que evita en gran medida desordenes y vicios de los programadores (Ej: Las variables a utilizar no pueden ser declaradas en cualquier lado). La cantidad de controles que existe para Delphi es realmente grande, aunque es menor a la cantidad de controles ActiveX (que también puede utilizar). El entorno para crear formularios, que es el mismo que el de C++Builder, es muy potente pudiendo diseñar el aspecto visual de la aplicación casi por completo dentro de él. A diferencia del C++Builder, el Delphi posee un compilador pascal muy veloz. Muchos desarrolladores acostumbrados a utilizar C++ en sus proyectos podrían tomar como característica negativa el "volver" a programar en Pascal (Pascal tradicionalmente es utilizado como lenguaje de enseñanza). Por un lado es potente y flexible y por el otro no es tan difícil de aprender como el Visual C++. Sin embargo esto trae una contradicción que muchos anuncian: es mas difícil que Visual Basic y no llega a ser tan poderoso como el Visual C++ por lo que ¿para que molestarse en aprenderlo?.

A continuación se muestra la Figura 15 donde se señala que lenguaje, es mas conveniente para que tipo de proyecto. Con "conveniente" se quiere expresar que lenguaje está mejor dotado para hacer que tipo proyecto. Por ejemplo, no se recomienda el Visual Basic para realizar controles OCX aunque es cierto que si es posible hacerlo [11].

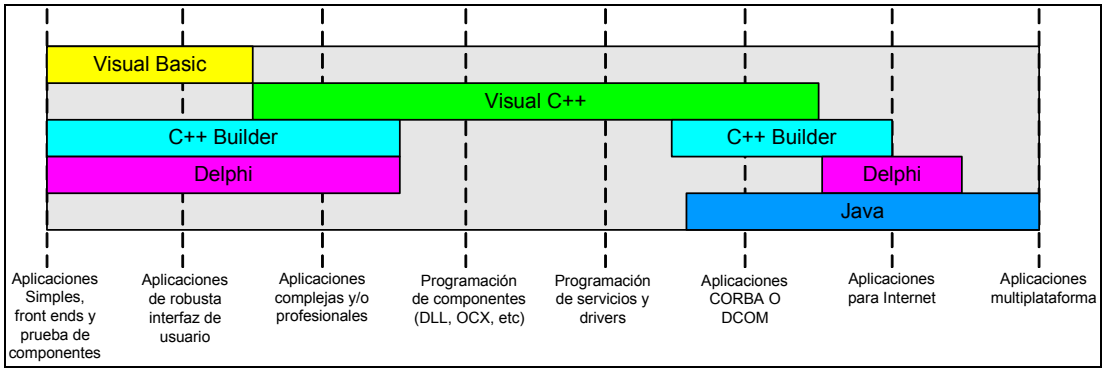


Figura 15 Cuadro comparativo de compiladores por aplicaciones.

3 DESCRIPCIÓN DE EQUIPOS UTILIZADOS (HARDWARE).

3.1 Nodo SCADA.

El departamento de Potencia de la Escuela de Ingeniería Eléctrica de la Universidad Central de Venezuela (UCV) se propuso la utilización de la red de alimentación de energía eléctrica de la Ciudad Universitaria de Caracas de la UCV (CUC) como un laboratorio nacional de supervisión y simulación de sistemas eléctricos de distribución. Con trabajos previos realizados se propuso la adquisición, instalación y puesta en marcha, de un sistema de adquisición de datos, cuya propuesta final se muestra en la Figura 16 , que describiremos a continuación [12].

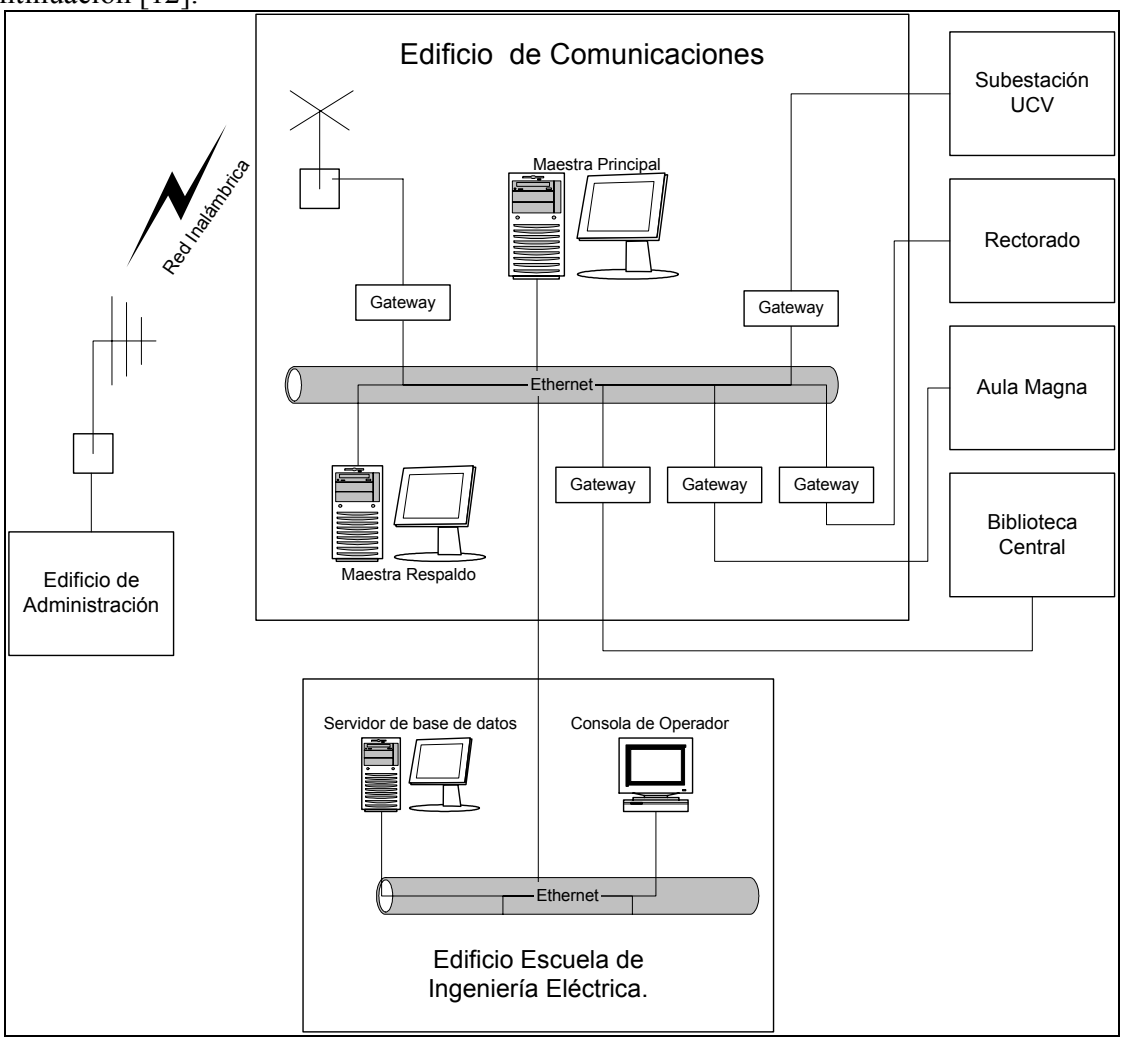


Figura 16 Diseño de la red de comunicaciones para el Laboratorio Nacional de Supervisión.

3.1.1 Descripción de equipos SCADA.

El sistema esta compuesto básicamente por una estación maestra principal, una estación maestra de respaldo, un terminal servidor de base de datos, las consolas de operador,

los dispositivos analizadores de redes, las interfaces tipo Gateway y los sistemas encargados de establecer la comunicación entre los equipos.

La estación Maestra y la de respaldo están compuestas básicamente por sistemas de computación del tipo Compaq.

Los dispositivos analizadores de redes o unidades terminales remotas son de dos tipos diferentes:

- Analizador de red MCA LIFASA.
- Analizador de red MACH 30 DUCATI Energía s.p.a..

Estos dispositivos una vez conectados a la red, almacenan temporalmente diferentes variables eléctricas, como tensión, corriente, frecuencia y otras variables derivadas (cálculos internos) como potencia, energía, etc.

Para las comunicaciones es importante señalar que los dispositivos comparten un único par de cables, por lo que coexisten en el mismo bus, debido a que sus protocolos no se interrumpen entre si.

El sistema inalámbrico esta conformado por un sistema Spread Spectrum en la banda (902-928 MHz) Frequency Hopping.

La interfase tipo Gateway esta concebida como un dispositivo interprete, para integrar protocolos de comunicaciones distintos, inicialmente basada en un PC bajo sistema operativo Windows 2000 por motivos de compatibilidad, las especificaciones y desarrollo de esta interfase son el motivo de este trabajo y será expuesto de forma detallada de manera posterior.

3.1.2 Interconexión entre equipos e interfaces de comunicación.

La Maestra principal, maestra de respaldo, servidor de base de datos, consolas e interfaces tipo Gateway, están físicamente conectadas a una red de tipo *Ethernet* y *Fast Ethernet***,** preexistente en la CUC, que permite la comunicación entre las diferentes dependencias. La comunicación entre las partes del SCADA depende básicamente de la herramienta de soporte del SCADA (iFIX, Software base para el SCADA), exceptuando la interfase tipo Gateway en desarrollo que como se muestra en la Figura 16, coexiste entre un medio físico tipo *Fast Ethernet* y una red de comunicación serial correspondiente a la norma RS-485.

La interfase tipo Gateway fue concebida para este desarrollo bajo un PC convencional, sistema operativo Windows 2000 (Sistema operativo para el SCADA) y descansando bajo el soporte que este sistema operativo provee se desarrolló un software de computadora que realizará las tareas requeridas por el SCADA en cuestión. Dentro de los equipos utilizados para la interfase se identifica una tarjeta de red para integrar el PC a la red *Ethernet*, puertos seriales del tipo RS-232 y la conexión de un convertidor de medio físico RS-232 a RS-485.

Es responsabilidad del software de SCADA (iFIX, *Intellution*) la transferencia de datos y protocolos utilizados entre los equipos del SCADA. La comunicación entre el maestro y la interfase tipo Gateway está diseñada bajo el protocolo de comunicación MODBUS-TCP pero esta contemplado un desarrollo similar bajo protocolo DNP3 sobre TCP. Desde la interfase hasta los analizadores de redes se utilizan dos protocolos de comunicación MODBUS-RTU y DUCBUS, que finalmente coexistirán en el mismo bus RS-485 y será completamente transparente para el SCADA.

Por ultimo el protocolo de comunicación usado para la red inalámbrica es MODBUS-RTU transportado bajo el sistema *Spread Spectrum* y finalmente adaptado a un puerto serial

RS-232 donde la interfase tipo Gateway realice los trabajos de interrogación cuando así lo requiera el SCADA.

La factibilidad y estudio detallado del sistema aquí descrito fue realizado en trabajos previos y los detalles del mismo no están dentro del alcance del presente trabajo.

3.2 Dispositivos a nivel de campo.

3.2.1 Analizador de redes MCA (LIFASA).

3.2.1.1 Características generales.

El analizador de redes serie MCA es un instrumento electrónico que permite la lectura de los principales parámetros de una red de alimentación en verdadero valor eficaz. Está especialmente diseñado para su fácil instalación y programación, precisa de la conexión de cuatro puntos de lectura de tensión de red y de tres transformadores de corriente. Dispone de cuatro pulsadores frontales que permiten cambiar el parámetro visualizado en pocos segundos. La serie MCA esta provista de LEDs para su fácil lectura con cualquier nivel de iluminación y ajustado a los estándares de robustez industrial.

3.2.1.2 Características técnicas.

Para la tensión de alimentación se puede escoger entre 120V/230V (-10% +15%), frecuencia 50/60 Hz, un consumo de potencia de 3 VA, máxima medición de fase – fase de 866 V y fase – neutro de 500V. Salida visual de cuatro dígitos color rojo de tipo 7 segmentos, temperaturas de operación entre -10 hasta 50 ºC, sus dimensiones físicas son 144x144 mm con un peso de 750 g, clase de precisión 0,5.

Esta versión permite obtener lecturas de tensión, corriente, potencia, energía, máxima demanda, distorsiones armónicas y otros parámetros especiales. Permite el funcionamiento en cuatro cuadrantes y la comunicación con un PC.

3.2.1.3 *Comunicaciones.*

La comunicación de esta versión del analizador de redes MCA a un dispositivo maestro es posible a través de un bus RS-485 con la posibilidad de manejar dos protocolos de comunicación configurables que son: CIRBUS (protocolo propiedad de LIFASA) y MODBUS RTU para hacerlo compatible con la mayor parte de programas SCADA y de monitorización existentes en el mercado.

3.2.1.4 *Conexiones.*

Para conectar el analizador de redes solo es preciso llevar hasta el aparato cuatro cables conectados a los puntos de lectura de tensión (tres fases y neutro), lecturas de tres transformadores de intensidad de corriente normalizados a 5A uno para cada una de las fases y los cables de alimentación para el equipo. Para el caso del bus de comunicación se recomienda un cable apantallado de tres conductores que cumpla con el estándar RS-485.

3.2.2 Analizador de redes MACH 30 (DUCATI).

3.2.2.1 *Características generales.*

Es un dispositivo electrónico destinado a la medición de variables eléctricas en baja tensión, está diseñado para instalarse de manera sencilla y programarse a través de un teclado asistido por una pantalla LCD (pantalla de cristal líquido). Es capaz de medir tensiones, corrientes, frecuencia y otros, y calcular potencia, energía y otras unidades derivadas.

3.2.2.2 Características técnicas.

Para la tensión de alimentación se puede escoger entre 120V/220V (-10% +10%), frecuencia 45/65 Hz, un consumo de potencia máximo de 3 VA, máxima medición de fase – fase de 750 V. Interfase visual de pantalla de cristal liquido, temperaturas de operación entre 0 hasta 50 ºC, sus dimensiones físicas son 157,5x90 mm con un peso de 610 g, y clase de precisión 1. Adicionalmente posee dos entradas analógicas 4-20 mA que pueden ser usadas para adquisición de datos a nivel de sensores industriales.

Esta versión permite obtener lecturas de tensión, corriente, potencia, energía, máxima demanda, distorsiones armónicas y otros parámetros especiales. Permite el funcionamiento en cuatro cuadrantes y la comunicación con un PC.

3.2.2.3 *Comunicaciones.*

La comunicación de esta versión del analizador de redes MACH 30 a un dispositivo maestro es posible a través de un bus RS-485 con la posibilidad de manejar un protocolo de comunicación que es: DUCBUS (protocolo propiedad de DUCATI) cuyas especificaciones son de dominio publico, y para hacerlo compatible con programas SCADA y de monitorización existentes en el mercado es necesario el diseño de un driver o interfase.

3.2.2.4 *Conexiones.*

Para conectar el analizador de redes se necesitan cuatro cables conectados a los puntos de lectura de tensión (tres fases y neutro), lecturas de tres transformadores de intensidad de corriente normalizados a 5A uno para cada una de las fases, cuyos cables atraviesan el dispositivo y se realiza la medición con un sistema no documentado en las especificaciones; y por ultimo los cables de alimentación para el equipo. Para el caso del bus de comunicación se recomienda un cable apantallado de dos conductores que cumpla el estándar RS-485.

3.2.3 Conexión física de los dispositivos.

Para llevar a cabo la conexión física de los dispositivos con la red, existen varias configuraciones posibles que están documentadas en el manual de usuario de dichos dispositivos, sin embargo para la etapa de desarrollo y pruebas, se conectaron un dispositivo MCA y un dispositivo MACH 30 al mismo circuito de medida, circuito de acometida del edificio de la escuela de Ingeniería Eléctrica de la CUC según lo indica la Figura 17, se implementó también una conexión entre los dispositivos de un par de cables de cobre sin apantallar, a fin de constituir un bus RS-485 que con una longitud de 20m aproximadamente se interconecto con un convertidor convencional RS-485 a RS-232, para así enlazar hasta el puerto serial del PC de desarrollo, que estuvo ubicado en una sala contigua del laboratorio de Maquinas Eléctricas del mismo edificio de la Escuela de Eléctrica con conexión física a la red *Ethernet*, que sirvió como plataforma para el desarrollo.

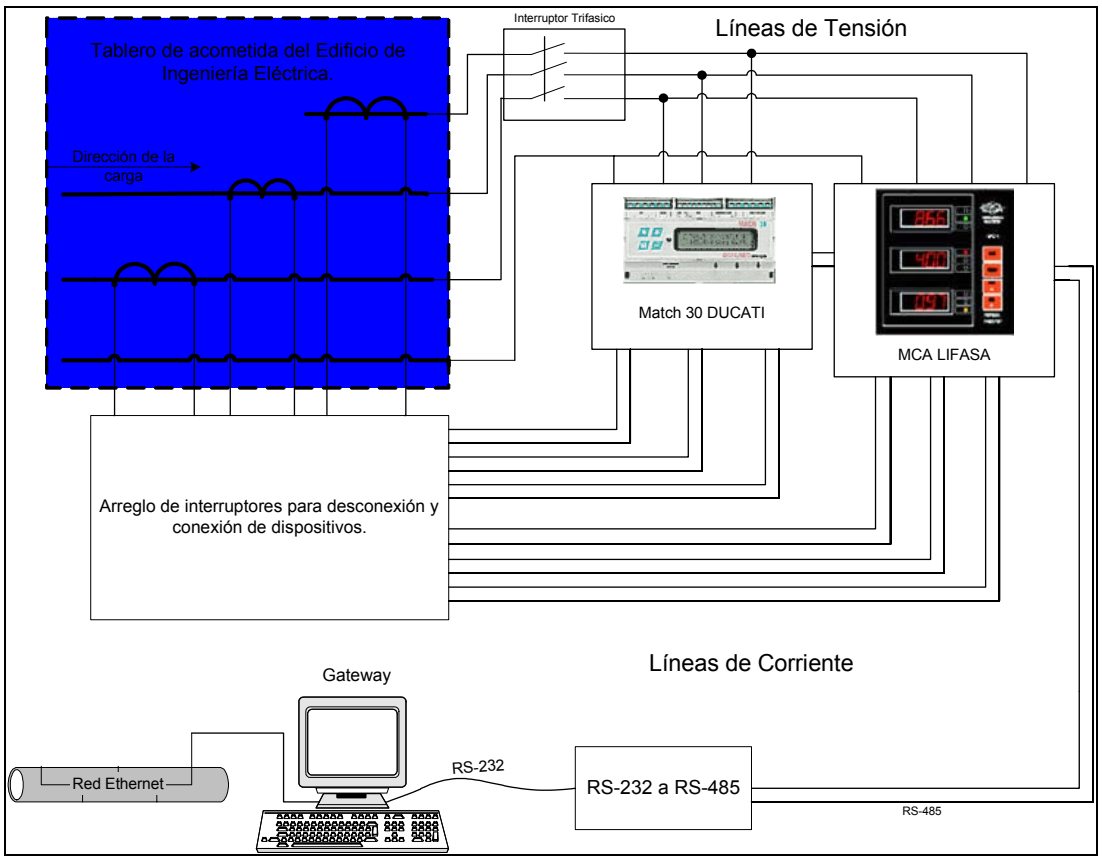


Figura 17 Esquema para desarrollo de gateway.

3.3 Interfaz Gateway (hardware).

3.3.1 Descripción de unidad para desarrollo.

La interfaz tipo Gateway desarrollada esta formada básicamente por un software de computadora, soportado por un hardware cuyas exigencias se limitan a los aspectos propios de programación (opciones del Sistema Operativo) y los periféricos necesarios para realizar las transacciones propias del sistema. El desarrollo está basado en un computador convencional capaz de soportar el sistema operativo Windows 2000 professional, aunque esto no es una limitante. La interfaz funciona bajo cualquier sistema Windows de núcleo NT (Windows NT, Windows XP, etc.). Los periféricos necesarios tampoco son excluyentes, solo se necesitas los puertos seriales (RS-232) y la tarjeta de conexión a Internet. Es necesario mencionar que cualquier sistema de conexión a Internet que sea soportado por el sistema operativo es suficiente para asegurar que el software tenga funcionalidad. Los requerimientos mínimos para asegurar la funcionalidad son los siguientes:

- PC capaz de soportar un sistema operativo Windows de núcleo NT.
- Puertos seriales UART 8250 o compatible.
- Capacidad de conexión tipo Internet (Tarjeta Ethernet, módem, etc).
- Monitor CRT o pantalla (no indispensable).
- Teclado (no indispensable).

- Mouse (no indispensable).

Los requerimientos señalados como "no indispensables", no intervienen en el funcionamiento, solo proporcionan la herramienta para configurar la interfaz tipo Gateway.

3.3.2 Propuesta de hardware alternativo.

Existen dos posibilidades bien definidas en cuanto al hardware alternativo se refiere. Por una parte tenemos la posibilidad de utilizar un PC convencional con los requerimientos mínimos ya señalados, pero cuando se toma en cuenta las condiciones de trabajo, robustez y costos, una solución alternativa consideraría reducir al máximo los componentes de hardware y hacerlos tan específicos como se pueda. En la búsqueda de hacer una propuesta alternativa se procede a este fin, aclarando que la propuesta es solo de orden referencial y no esta sujeta a desarrollo en el presente trabajo.

La posibilidad de un hardware alternativo, debe reunir características de funcionamiento similares a las herramientas usadas en este desarrollo. Dichas características se listan a continuación:

- Programación y manejo de puertos seriales.
- Soportar programación tipo servidor basado en el concepto de Sockets (Capitulo 5.2).
- Soporte de funciones para la pila de protocolos TCP/IP (Capitulo 5.2).
- Memoria de programa (tipo ROM) y de datos (tipo RAM).
- Soporte para visualización de interfaz de usuario.
- Capacidad de ejecución de programas en modo multiproceso (Capitulo 5.1.7).

Actualmente en un amplio mercado de dispositivos de hardware, encontraremos tantas soluciones como se quieran para la implementación de un hardware alternativo, por lo que nos limitaremos al estudio de un sistema que cumple con la mayor parte de las exigencias, basados en las dificultades salvadas que este propone y no en un estudio de factibilidad del mismo.

3.3.2.1 Tarjeta para desarrollos TINI (InterNet Interface).

La tarjeta Tini es una plataforma desarrollada por Dallas Semiconductor para proveer un sistema para desarrollo de aplicaciones de una manera simple, flexible y de una medida de costo efectivo, para estaciones de trabajo y conexión de dispositivos. La plataforma es una combinación de pequeños pero potentes chips y tiene el soporte de programación en lenguaje de alto nivel (Java) con su propio sistema operativo. La tarjeta fue concebida para realizar controles industriales, monitoreo y control de equipos a través de sistemas de tipo Web, conversiones de protocolos (Gateway) y otros.

3.3.2.1.1 Características de la tarjeta TINY.

La constitución básica de la tarjeta se compone de un microcontrolador, una memoria ROM de tipo flash y una memoria RAM de tipo de datos estática (ver Figura 18). Posee una memoria para la programación de 4Mbytes separando las áreas de código de proceso, código para comunicaciones y control de periféricos [13].

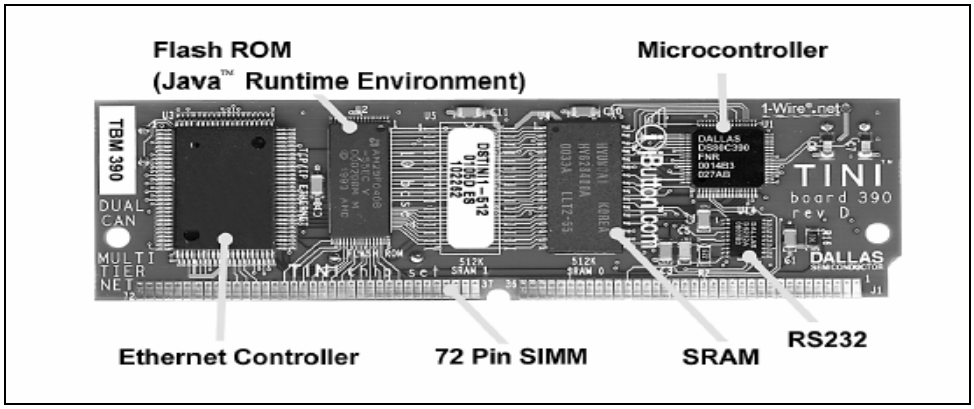


Figura 18 Tarjeta de para el desarrollo de las aplicaciones.

La tarjeta incluye 2 líneas de puertos seriales tipo UART (*Universal Asynchronous Receiver Transmitter*) con posibilidades de programación de tipo API (*Interface Programing Application*) similar al desarrollo en entornos de tipo Windows, por lo que la programación de los mismos no se aleja del concepto desarrollado en este trabajo. Tiene una conexión 10Base-T Ethernet con la que se acopla a la red, realizando labores de servidor Web (ver Figura 19).

Es importante señalar que dicho sistema posee la pila de protocolos TCP/IP y su programación se realiza con un API conocida como "API Sockets" que es también la herramienta utilizada en este desarrollo para las conexiones y desarrollo en red, para la conexión física, posee un conector de 72 pines con los que se acopla la tarjeta de desarrollo de software con los periféricos.

La tarjeta soporta la programación en modo Multiproceso por lo que se adapta a la exigencia de métodos de sondeo para las transacciones seriales, explicada en el capítulo 5.

Por ultimo es importante señalar que la programación en Java abre la posibilidad de interactuar a través de la red con controles desarrollados tipo página Web que sería el equivalente a la visualización a través de una pantalla CRT.

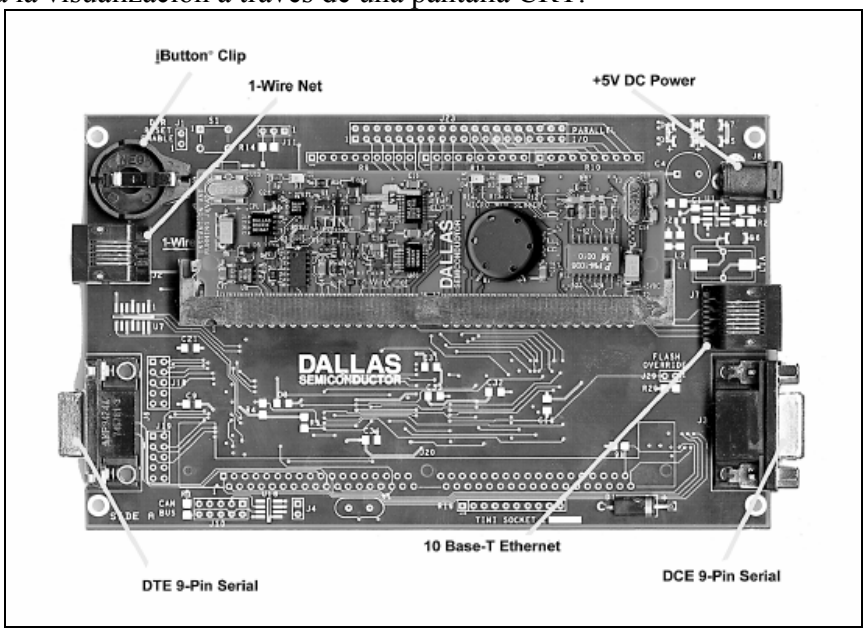


Figura 19 Vista de planta de tarjeta de desarrollo TINY.

4 DESCRIPCIÓN DE PROGRAMAS UTILIZADOS (SOFTWARE).

4.1 Software de desarrollo.

4.1.1 Descripción general de Windows 2000 Professional.

Windows 2000 Professional combina la facilidad de uso de Windows 98 en Internet, en la oficina o de forma remota, con la facilidad de administración, confiabilidad y seguridad de Windows NT. Windows 2000 Professional es un sistema operativo pensado para los equipos de escritorio y portátiles de todo tipo y tamaño de empresa.

Microsoft Windows 2000 Professional está construido sobre la sólida confiabilidad de la tecnología Windows NT, lo que lo convierte en significativamente más confiable que Windows 95 o Windows 98. En Windows NT, cada aplicación se ejecuta en su propio espacio de memoria. Este aislamiento significa que, cuando una aplicación falla durante su ejecución, no termina con el sistema completo. Diversos estudios independientes han comprobado que ésta y otras características disminuyen a la mitad la probabilidad de que los usuarios de Windows NT soliciten soporte técnico, en comparación con los usuarios de Windows 98.

Windows 2000 Professional es rápido. Más rápido que Windows 98. Con 64 MB de memoria, Windows 2000 se ejecuta un promedio de un 25% más rápido que Windows 98. Y no se ralentiza con cargas pesadas. Los usuarios pueden ejecutar más programas y hacer más tareas al mismo tiempo porque Windows 2000 está basado totalmente en una arquitectura de 32 bits. Agregándole más memoria, Windows 2000 se hace más rápido aún. Soporta hasta 4 GB de RAM y hasta dos procesadores simétricos. Por desgracia, alcanzar ese nivel de rendimiento con Windows 98, incluso con más memoria, en imposible.

A nivel de programación de aplicaciones Windows 2000, al igual que Windows NT toma por completo el control del sistema, por lo que los viejos conceptos de programación orientados a la interrogación sucesiva de los programas de aplicaciones, es sustituida por una tecnología mas eficiente orientada a los mensajes. En otras palabras se puede decir que las aplicaciones no realizan ninguna tarea sobre los periféricos, lo que sucede es que las aplicaciones solicitan los servicios de Windows para actuar sobre los periféricos, Windows a su vez corresponde a estas solicitudes enviándole mensajes que la aplicación debe entender y procesar. Esto nos haría pensar que el concepto de tiempo real se ha perdido, pero no es así, Windows ha combinado tecnologías de tipo objeto como unidades independientes que capturan peticiones y gracias a las elevadas velocidades de proceso el concepto aun es factible, mas aun, con el sistema de proceso tipo Cliente – Servidor, Windows establece prioridades que son accesibles para los programadores de aplicaciones a través de tecnologías como Windows DNA [14].

4.1.2 Descripción del entorno de desarrollo C++ Builder.

Del análisis realizado en el capitulo 2.4.4.3 (Comparación de compiladores.) se decide utilizar el compilador C++ Builder para el desarrollo en cuestión por tener propiedades de realizar aplicaciones de robusta interfaz de usuario, desarrollos posibles para Internet y control de filosofías Cliente – Servidor (COM/DCOM), todos fundamentales para realizar la interfase tipo Gateway que se pretende.

Con C++ se tiene una poderosa herramienta para la construcción de aplicaciones de diferentes naturalezas, si ahora le anexamos como parte de su estructura docenas de los objetos mas utilizados por todos los desarrolladores, encontramos un software que facilita la construcción de aplicaciones, por lo que es aún mas poderoso y efectivo; ahora el desarrollador se tiene que preocupar por que sus funciones asociadas a los objetos que seleccione para su aplicación hagan lo que se desea que realicen, y no se tiene que preocupar por como lograr la visualización en pantalla.

Con el entorno integrado de desarrollo que provee el compilador C++ Builder, la creación de botones, menús, ventanas de dialogo, barras de estado, la presentación de campos tipo memo, la visualización de directorios tipo árbol, la presentación de imágenes, secuencias de impresión, y muchas más cosas que comúnmente se realizan con demasiadas líneas de código, se podrá tener con solo elegir un componente que C++ Builder ofrece. Por lo que esta asegurada la robusta interfaz de usuario de la que finalmente se desea.

C++ Builder permite el uso de las API (*interfaces para programación de* Aplicaciones) Winsocket, necesarias para realizar las operaciones de Internet entre la Maestra de red (TCP) y funcionar como servidor de esta aplicación en particular. Por lo que cuenta con una fuerte herramienta al nivel mas bajo de programación de aplicaciones de red, ya que Winsocket es una interfase entre la capa de aplicación y la capa de transporte del modelo TCP/IP. El uso de esta herramienta supondrá un completo control de la aplicación ya que se construirá la aplicación de acuerdo a las necesidades y no se utilizarán objetos prediseñados que contemplan aspectos no útiles en nuestro caso.

Por otra parte C++ Builder también permite el uso de otra API conocida como API del puerto serie, que sirve como una interfase entre la aplicación y el completo control de los puertos seriales, pudiendo actuar y obtener datos de cualquier índole de los puertos seriales. Existe también la posibilidad de utilizar controles ActiveX para este propósito, pero recordemos que estos controles le restan eficiencia al código y por definición solo encapsulan a las API, que en fin es quien actúan sobre el periférico.

Teniendo en cuenta que el entorno permite la construcción eficiente de la aplicación en cuestión solo resta aplicar los conceptos de programación estructurada y la creación de las clases que el desarrollo requiera [15] [16].

4.2 SCADA sobre iFIX (Intelluption Dynamics).

4.2.1 Características de iFIX.

iFIX es el software para HMI (*Human Machine Interface*) SCADA que permite la adquisición de datos, monitoreo y control de procesos en tiempo Real, basado en una arquitectura Cliente - Servidor distribuida. iFIX es un producto completamente reescrito con base en los últimos estándares tecnológicos impuestos en la industria por Microsoft (Windows versión DNA). Producto basado en tecnología por componentes incluyendo COM (*Component Object Technology*) / DCOM, Windows NT, VBA (Visual Basic para Aplicaciones), OPC (Ole for Process Control) y Controles Active.

El software está diseñado para funcionar sobre las plataformas Windows 2000, Windows NT y/o Windows XP, como software de SCADA se adapta a las necesidades de los usuarios, funcionando bajo un aspecto distribuido fue concebido como desarrollos modulares que en conjunto forman la plataforma de trabajo. Solo dependiendo de la aplicación es posible

saber que o cuales partes deben coexistir e interactuar para proveer un verdadero sistema SCADA de tipo distribuido.

4.2.2 Componentes de iFIX.

iFIX HMI es una solución para las interfaces entre los procesos y los usuarios, que ofrece seguridad y un optimo performance para monitorización y control de operaciones.

iFIX Cliente es una aplicación para una estación de tipo HMI, que utilizando redes Ethernet (TCP/IP o NetBIOS) o conexiones remotas vía Windows NT/2000 (Módem, Radio Módem, Satelite, etc) hace posible efectuar monitorización y control de operaciones en tiempo real sin requerir duplicación de bases de datos en esa estación cliente gracias a un diseño de arquitectura distribuida.

iFIX Cliente de solo lectura es una aplicación cliente en una estación que funciona de manera similar a iFIX Cliente, pero sin permitir al usuario interactuar con el proceso.

iFIX SCADA Servidor funciona como una aplicación servidora de datos con funciones SCADA asociadas que sirve datos a estaciones iFIX Clientes o iFIX HMI. Utilizando redes Ethernet (TCP/IP o NetBIOS) o conexiones remotas vía Windows NT/2000 (Módem, Radio Módem, Satélite, etc.) abre la posibilidad de efectuar monitorización y control de operaciones por una estación iFIX Cliente conectada a esta estación o directamente sobre la aplicación para la que se haya designado. Este mismo es capaz de administrar de manera distribuida las bases de datos, históricos y otros.

iClientTS es un componente de la familia de Intellution Dynamics que cuando en conjunto con una estación Servidora de iFIX sobre Windows 2000 o Windows XP, dispone datos en tiempo real incluyendo posibilidades de efectuar operaciones. Todas las estaciones clientes utilizan el navegador Explorer 5.5 para efectuar las comunicaciones con el terminal servidor, no necesitan software de Intellution y pueden funcionar bajo cualquier sistema operativo de tipo Windows.

Esta capacidad distribuida se resume en la Figura 20, se muestra la conexión de cada uno de los componentes, los sistemas operativos que se permiten y modos de comunicación.

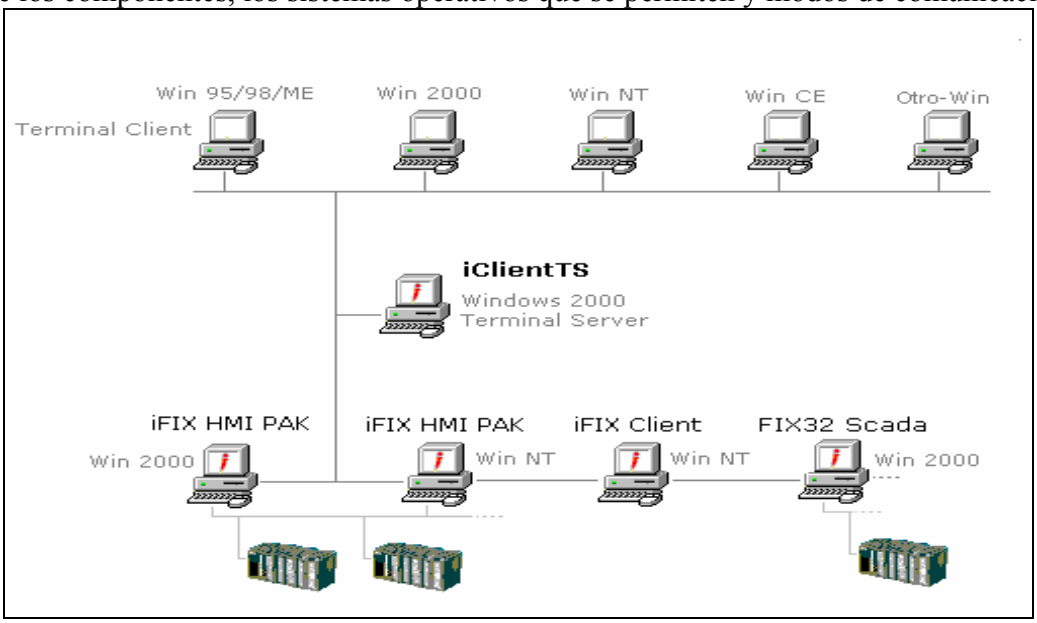


Figura 20 Arquitectura de SCADA sobre iFIX.

Para realizar la comunicación entre los diferentes componentes la filosofía esta orientada a la utilización de drivers (Convertidores de protocolo) compatibles con tecnologías de última generación para transferencia de objetos de datos y otros.

Para el caso especifico del SCADA en la Universidad Central de Venezuela, el esquema consta de todos sus componentes. La plataforma esta basada en Windows 2000, para la comunicación con dispositivos de campo se utiliza el driver MBE (MODBUS sobre Ethernet) que interactuará con el Gateway desarrollado para la adquisición de datos.

Por ultimo se acota que el desarrollo de la interfase tipo Gateway que convertirá MODBUS RTU y DUCBUS a MODBUS TCP será probado bajo este software de SCADA y funcionará como parte del mismo.

4.3 Software de prueba.

4.3.1 Comunicaciones seriales.

Antes de empezar la fase de desarrollo de la interfase tipo Gateway se hizo necesario realizar diferentes pruebas con los dispositivos de campo descritos en los capítulos 3.2.1 y 3.2.2, por una parte realizar pruebas discretas de interrogación de dispositivos enviando y recibiendo tramas bajo protocolo MODBUS RTU. Por otra parte comprender de una manera más profunda el protocolo propietario DUCBUS. Para esto se utilizó una herramienta de licencia libre llamada “LabIML”.

LabIML es un driver universal desarrollado por WindMill Software LTD para enviar o aceptar mensajes codificados en ASCII o hexadecimal sobre puertos seriales bajo la norma RS-232. Este automáticamente recoge los datos enviados desde los instrumentos y los muestra en una ventana de aplicación de Windows. Pudiendo enviar y recibir tramas de manera discreta y almacenar los datos para su posterior interpretación. El software soporta cualquier transmisión bajo RS-232 independientemente del instrumento en cuestión. Tiene la capacidad de programar todas las opciones asociadas con las comunicaciones bajo RS-232, como son velocidad, puerto, bits de dato por byte, etc. Su concepción está basada en la construcción previa de mensajes formados por cadena de caracteres codificados en ASCII o hexadecimal, luego enviar el mensaje al medio y esperar por su respuesta, utilizando un método tipo sondeo, es decir espera un tiempo por una respuesta.

Como herramienta de comunicación es muy versátil y por su aspecto bajo Windows no requiere gran experticia de parte del usuario para lograr comprender su funcionamiento, es de notar que no maneja ningún protocolo de comunicación, mas bien solo sirve de interfase entre el medio serial RS-232 y un usuario que desee enviar una trama [17].

4.3.2 Comunicación discreta bajo DNP3 (Test Harness).

La falta de una plataforma SCADA sobre el protocolo DNP3 sobre TCP, condujo a la necesidad de buscar un medio de prueba para realizar transacciones bajo este protocolo, a fin de realizar pruebas que documentaran de alguna forma el funcionamiento de la interfase tipo Gateway bajo DNP3 desarrollada. Triangle MicroWorks INC es una compañía internacional que provee servicios SCADA al igual que Intellution Dynamics, y que actualmente realiza desarrollos importantes bajo este protocolo (DNP3); para esto ha desarrollado una herramienta de prueba denominada “Communication Protocol Test Harness” [18].

Este software es una aplicación que puede ser configurada como un típico dispositivo Maestro o esclavo. Este también puede ser programado para realizar secuencias automáticas

de adquisición de datos y pruebas. El software sigue la entrada y salida de datos, haciendo puntos de verificación que el usuario considere necesarios en las pruebas realizadas, mostrando las transacciones y los errores sucedidos.

Las opciones de pruebas y programación siguen las recomendaciones publicadas por los comités técnicos de cada protocolo. Maneja diferentes protocolos de comunicación como:

- DNP3 – Distributed Network Protocol (Soportando serial y TCP/IP).
- Modbus – Serial ASCII/RTU, Modbus Plus, y modbus TCP.
- IEC 60870-5-101 – Tareas de telecontrol básicas.
- IEC 60870-5-102 – Lectura de medidores remotos.
- IEC 60870-5-103 – Equipos de protección.
- IEC 60870-5-104 – Acceso de redes (TCP/IP).

Simula las funciones básicas y especificas de los productos desarrollados bajo las especificaciones de los comités y ofrece un completo conjunto de comandos que soportan todos los aspectos funcionales de dispositivos. Además, provee una ventana para interpretación de datos transmitidos y recibidos, examinando las transacciones y guardando un documento de las mismas. El usuario debe introducir los comandos y datos pertinentes a la fabricación de la prueba, para esto se provee la documentación necesaria a fin de que el usuario comprenda y maneje el entorno.

Está herramienta esta diseñada para funcionar en Windows 98/NT/2000/XP pero su licencia no es libre y solo está disponible por un periodo limitado de prueba. Al describir las pruebas realizadas se detallará un poco mas en la interpretación de los documentos generados.

5 DESARROLLO REALIZADO.

5.1 Etapa 1: Desarrollo de comunicaciones seriales.

5.1.1 Funciones del API WIN32 para comunicaciones serie.

La utilización del las funciones del API WIN32 para trabajar con el puerto serie garantiza la portabilidad del código entre distintas versiones de WINDOWS (95/98/NT/2000/XP) y la estabilidad de los programas que se desarrollen. La lista de funciones y estructuras que permiten programar el puerto serie son más de dos docenas, se irán explicando según se necesiten. Trabajar con un puerto serie se asemeja mucho al trabajo con ficheros mediante flujos, al trabajar con un puerto serie se pueden distinguir 4 fases.

- Operación de apertura: sirve para obtener un manejador o handle y comprobar si el sistema operativo ha podido tomar el control del dispositivo para nuestra aplicación.
- Configuración: Sirve par establecer los parámetros de la comunicación: velocidad, paridad, bits de dato por byte, etc. así como el tipo de acceso: mediante sondeo o mediante eventos.
- Acceso al puerto para leer o escribir en él: Hay que tener en cuenta que el acceso al puerto siempre se realiza a través de un Buffer uno para la transmisión (escritura) y otro para la recepción (lectura).
- Cierre del puerto: Para que otros programas puedan hace uso de él.

Las funciones que se irán presentando a continuación han sido resumidas para dejar sólo aquellos parámetros que determinan el proceso de comunicación.

5.1.1.1 Apertura y cierre del puerto.

Para abrir el puerto serial se utiliza la función CreateFile(), esta función es una función de orden genérico que permite el acceso a recursos de naturaleza diversa, como pueden ser: puertos de comunicaciones seriales, puertos de comunicación paralelos, ficheros, directorios, discos, consolas etc.

CreateFile()

```
HANDLE CreateFile(
        LPCTSTR lpFileName,
        DWORD dwDesiredAccess,
        DWORD dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecurityAttributes,
        DWORD dwCreationDistribution,
        DWORD dwFlagsAndAttributes,
        HANDLE hTemplateFile);
```

ARG. ENTRADA	DESCRIPCIÓN
lpFileName	Nombre del puerto COM1, COM2 etc
dwDesiredAcces	Tipo de acceso GENERIC_READ, GENERIC_WRITE etc
dwShareMode	Modo de compartir. Debe ser 0 al no poderse compartir el puerto serie por dos aplicaciones de forma simultánea.
lpSecurityAttributes	Descriptor de seguridad. Debe ser 0
dwCreationDistribution	Especifica que acción se realizará si el fichero existe o si no existe: CREATE_NEW, OPEN_EXISTING etc. Puesto que un puerto siempre tiene que existir usaremos OPEN_EXISTING
dwFlagsAndAttributes	Especifica el tipo de atributo del fichero/recurso, debe de ser FILE_ATRIBUTE_NORMAL para los puertos.
hTemplateFile	Debe ser NULL
TIPO SALIDA	**DESCRIPCIÓN**
HANDLE	Manejador o Handle del puerto. Un puntero necesario para acceder al puerto, o NULL si la operación de apertura falló.

Si se desea conocer los tipos de datos definidos en WINDOWS para saber exactamente a que tipos básicos equivalen bastará con revisar el fichero WINDEF.H que se encuentra en el directorio INCLUDE de C++ BUILDER.

Por ejemplo el siguiente fragmento abre el puerto COM1 .

```
..
HANDLE hComPor;                                    //Manejador de Puerto.
hComPor=CreateFile("COM1",
                   GENERIC_READ|GENERIC_WRITE,
                   0,0,OPEN_EXISTING,
                   FILE_ATTRIBUTE_NORMAL,0 );      //Función de apertura.
if (hComPor==INVALID_HANDLE_VALUE)
   {
   //Error de apertura del puerto
   }
```

Una vez finalizado el trabajo con el puerto hemos de cerrar su manejador para que el sistema operativo libere los recursos que estamos ocupando, esto se hace mediante la función CloseHandle(), esta función devuelve un valor booleano (Bool), TRUE (verdadero) si el cierre del dispositivo se hizo correctamente, es importante no tratar de cerrar un manejador no válido, es decir un manejador no inicializado correctamente por CreateFile().

CloseHandle()	
`BOOL CloseHandle(HANDLE hObject);`	
ARG. ENTRADA	DESCRIPCIÓN
hObject	Manejador o handle genérico.
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

5.1.1.2 Configuración del puerto

Para leer y escribir la configuración del puerto se utilizan las funciones GetCommState() y SetCommState() ambas utilizan como argumento una estructura del tipo DCB. Esta estructura encapsula en sus campos la configuración de un puerto serie como se muestra en la tabla Tabla 3, se encuentra disponible en el fichero WINBASE.H del directorio INCLUDE\WIN32 del C++ BUILDER.

Esta extructura permite un control total sobre la configuración del puerto: tipo de conformación de byte, tipo de control de flujo, parámetros del control de flujo etc. Para trabajar con la estructura DCB disponemos de tres funciones GetCommState() para leer la configuración actual del puerto, SetCommState() para configurar el puerto y BuildCommDCB() para configurar el puerto utilizando el estilo clásico del comando MODE del DOS.

CAMPO	TIPO	DESCRIPCIÓN
DCBlength	DWORD	Tamaño de la estructura DCB
BaudRate	DWORD	Velocidad en bps: CBR_100,CBR_300,CBR_600 ...CBR_560000..
fBinary:1	BIT	Modo binario. Debe de ser TRUE
fParity: 1	BIT	Si TRUE se chequearán errores de paridad
fOutxCtsFlow:1	BIT	Si TRUE, se usa CTS para controlar el flujo entre el DTE y el DCE. Si CTS se desactiva el DTE dejará de transmitir hasta que se active de nuevo.
fOutxDsrFlow:1	BIT	Si TRUE, se usa DSRpara controlar el flujo entre el DTE y el DCE. Si DSR se desactiva el DTE dejará de transmitir hasta que se active de nuevo.
fDtrControl:2	2 BITS	Especifica como se manejara la línea DTR de salida del DTE. Valores posibles: DTR_CONTROL_ENABLE : Se activará DTR al abrir el puerto. DTR_CONTROL_DISABLE: Se desactivará DTR al abrir el puerto DTR_CONTROL_HANDSHAKE: Habilita el control de flujo hardware DTR-DSR En los dos primeros casos se puede utilizar la función **EscapeCommFunction()** para cambiar el estado de la línea DTR
fDsrSensitivity:1	BIT	Si TRUE se descartan los datos recibidos mientras la línea DSR (entrada al DTE) esta inactiva.
fTXContinueOnXoff:1	BIT	Configura el modo de operar si se llena el buffer de entrada y ya se ha transmitido el caracter XOFF. Si FALSE la transmisión se parará después de recibir XOFF, se reanudará si se recibe un XON o bien se vacía el BUFFER del receptor por debajo del límite inferior **XOnLimit**.
fOutX: 1	BIT	Si TRUE habilita el control de flujo XON/XOFF durante la transmisión .La transmisión cesará al recibirse un XOFF, se reanudará al recibirse un XON
fInX: 1	BIT	Si TRUE habilita el control de flujo XON/XOFF durante la recepción .Se enviará el caracter XOFF si el BUFFER del receptor alcanza el límite **XoffLimit**. Se enviará el caracter XON si el BUFFER bajo se vacía por debajo de **XonLimit**.
fErrorChar: 1	BIT	Si TRUE se sustituirán todos los caracteres recibido con errores por el dato **ErrorChar** definido en esta misma estructura.
fNull: 1	DWORD	Si TRUE se descartará los BYTES recibidos que sean nulos.
fRtsControl:2	2 BITS	Configura como se manejará el control de flujo mediante la línea RTS de salida del DTE. Los valores posibles son: RTS_CONTROL_DISABLE: Se mantendrá RTS inactiva RTS_CONTROL_ENABLE: Se mantendrá RTS activa RTS_CONTROL_HANDSHAKE: Habilita el control de flujo hardware RTS-CTS RTS_CONTROL_TOGGLE: Se activa mientras el BUFFER de transmisión tenga datos para transmitir. En los dos primeros casos se puede utilizar la función **EscapeCommFunction()** para cambiar el estado de la línea RTS
fAbortOnError:1	BIT	Si TRUE se abortará cualquier operación de lectura o escritura hasta que se llame explícitamente a **ClearCommError().**
fDummy2:17	17 BITS	No se usan
wReserved	WORD	No se usan
XonLim	WORD	Determina el límite inferior del BUFFER del receptor que fuerza la transmisión de XON
XoffLim	WORD	Determina el límite superior del BUFFER del receptor que fuerza la transmisión de XOFF
ByteSize	BYTE	Determina el número de bits de datos: 4,5,6,7,8
Parity	BYTE	Paridad: NOPARITY, EVENPARITY(par),MARKPARITY o ODDPARITY (impar)
StopBits	BYTE	Bits de Stop: ONESTOPBIT, ONE5STOPBITS,TWOSTOPBITS
XonChar	char	Fija el caracter que se usará como XON
XoffChar	char	Fija el caracter que se usará como XOFF
ErrorChar	char	Fija el caracter que se usará para sustituir un dato recibido erróneamente.
EofChar	char	Fija el caracter que se usará para forzar el final de la transmisión
EvtChar	char	Fija el caracter que se usará para forzar el lanzamiento de un evento
wReserved1	WORD	Reservado

Tabla 3 Miembros de la estructura DCB.

GetCommState()	
`BOOL GetCommState(HANDLE hFile,LPDCB lpDCB);`	
ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle devuelto por CreateFile()
lpDCB	Puntero a una estructura de tipo DCB donde se recibirá la configuracón actual del puerto.
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

SetCommState()	
`BOOL SetCommState(HANDLE hFile,LPDCB lpDCB)`	
ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle devuelto por CreateFile()
lpDCB	Puntero a una estructura de tipo DCB que se usará para configurar el puerto.
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

BuildCommDCB()	
`BOOL BuildCommDCB(LPCTSTR lpDef,LPDCB lpDCB)`	
ARG. ENTRADA	DESCRIPCIÓN
lpDef	Puntero a una cadena al estilo del comando MODE con la configuración a fijar, por ejemplo: "baud=9600 parity=N data=8 stop=1"
lpDCB	Puntero a una estructura de tipo DCB donde se recibirá la configuración completa del puerto.
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

La función BuildCommDCB() se utiliza en la API WIN32 solo por motivos de compatibilidad, por lo que no la utilizaremos en el desarrollo, para este fin se definirá una estructura DCB que cargaremos con los valores que solicite la aplicación, utilizaremos una estructura también DCB para guardar la configuración que el puerto tenga, a través de la función GetCommState(), por ultimo cargaremos la configuración requerida con la función SetCommState(), una vez finalizada la operación y antes de liberar el recurso cargaremos la configuración anterior del puerto para que otra aplicación pueda hacer uso de el.

5.1.2 Líneas de control de flujo.

Como se ha visto en el punto anterior, determinados miembros de la estructura DCB condicionarán el funcionamiento de las líneas de control del flujo DTR-DSR y RTS-CTS. Estas líneas de salida del DTE (DTR, RTS) pueden ser manipuladas mediante la función EscapeCommFuntion() . Del mismo modo se puede leer el estado de las líneas de entrada al DTE (DSR, CTS, RING, CD) mediante la función GetCommModemStatus().

EscapeCommFunction()	
`BOOL EscapeCommFunction(HANDLE hFile,DWORD dwFunc)`	
ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle devuelto por CreateFile()
dwFunc	Una palabra para indicar la operación a realizar: CLRDTR Pone a cero la señal DTR (data-terminal-ready). CLRRTS Pone a cero la señal RTS (request-to-send) signal. SETDTR Pone a uno la señal DTR (data-terminal-ready) SETRTS Pone a uno la señal RTS (request-to-send) SETXOFF Simula la recepción de un carácter XOFF. SETXON Simula la recepción de un carácter XON SETBREAK Suspende la transmisión CLRBREAK Reanuda la la transmisión
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

Para suspender y reanudar la transmisión se puede utilizar la función anterior con los parámetros SETBREAK y CLRBREAK o bien utilizar las funciones equivalentes

SetCommBreak() y ClearCommBreak(), ambas sólo requieren el manejador del puerto y retornan TRUE si la operación se completó. Si se desean leer las líneas de estado que proceden del DCE se puede usar la función GetCommModemStatus().

GetCommModemStatus()	
`BOOL GetCommModemStatus(HANDLE hFile,LPDWORD lpModemStat);`	
ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle que apunta al driver, devuelto por CreateFile()
lpModemStat	Un puntero a una variable DWORD donde se escribirá la configuración del registro de estado del módem, según los siguientes valores. MS_CTS_ON La señal CTS está activa MS_DSR_ON La señal DSR está activa MS_RING_ON La señal RING está activa MS_RLSD_ON La señal CD está activa
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

Es importante notar que una vez que la aplicación toma el control del puerto, otra aplicación simultánea no logrará abrir el recurso, por lo que se generará un error. La función GetLastError() sirve para obtener el código del error que se ha producido y la función ClearCommError() obtiene el código de error y limpia el estado de error, estos códigos de error se encuentran definidos en el fichero WINNT.H.

GetLastError()	
`DWORD GetLastError(VOID)`	
ARG. ENTRADA	DESCRIPCIÓN
No hay	---
TIPO SALIDA	DESCRIPCIÓN
DWORD	Código del último error producido

Se puede leer más información sobre el estado del puerto serie, comprobar si se ha producido un error y el tipo de error con ClearCommError() . Esta función además de leer el tipo de error que se ha producido, sirve para limpiar la bandera de error del puerto. En determinados modos de funcionamiento (fAbortOnError de la estructura DCB), el puerto deja de trabajar hasta que no se llama a esta función después de haberse producido un error.

ClearCommError()	

`BOOL ClearCommError(HANDLE hFile,`
`LPDWORD lpErrors,`
`LPCOMSTAT lpStat );`	

ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle devuelto por CreateFile()
lpErrors	Puntero a una variable DWORD donde se recibirá el código del error. CE_BREAK Detectado corte en la línea CE_TRAMA Detectado error de trama CE_IOE Detectado error de E/S CE_MODE Manejador hFile especificado no válido. CE_OVERRUN Se han perdido bytes al transmitir. CE_RXOVER Overflow en el buffer de entrada. CE_RXPARITY Detectado error de paridad. CE_TXFULL Buffer del transmisor está lleno.
lpStat	Puntero a una estructura de tipo COMSTAT donde se podrá leer el estado del puerto.
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

El segundo argumento de la función ClearCommError() es un puntero a una variable DWORD donde la función depositará el tipo de error que se haya producido. El tercer argumento es un puntero a una estructura de tipo COMSTAT donde la función depositará información detallada del estado del puerto serie, los miembros de esta estructura se muestran en la Tabla 4.

CAMPO	TIPO	DESCRIPCIÓN
fCtsHold : 1	BIT	Si TRUE el puerto está esperando que el DCE active CTS
fDsrHold : 1	BIT	Si TRUE el puerto está esperando que el DCE active DSR
fRlsdHold : 1	BIT	Si TRUE el puerto está esperando que el DCE active CD
fXoffHold : 1;	BIT	Si TRUE el puerto ha recibido XOFF
fXoffSent : 1;	BIT	Si TRUE el puerto ha enviado XOFF
fEof : 1;	BIT	Si TRUE el puerto ha recibdio un carácter EOF
fTxim : 1;	BIT	Si TRUE Hay un caracter esperando ser transmitido
DWORD fReserved : 25;	25 BIT	No se usan
DWORD cbInQue;	DWORD	Número de caracteres en BUFFER del receptor
DWORD cbOutQue;	DWORD	Número de caracteres en el BUFFER del transmisor

Tabla 4 Miembros de la estructura COMSTAT.

5.1.3 Buffer del transmisor y del receptor.

Todas las comunicaciones mediante el API de WINDOWS se realizan a través del BUFFER tanto para la transmisión como para la recepción. Una vez que el puerto ha sido abierto y configurado cualquier carácter que llegue al puerto será almacenado en el buffer del receptor de forma automática por el sistema operativo, no importa lo que está haciendo nuestro programa o cualquier otro que se encuentre en ejecución. Del mismo modo, cuando escribamos en el puerto se hará a través del BUFFER del transmisor, el programa escribirá en él y el sistema operativo irá enviando los datos de forma totalmente automática siempre que el control de flujo implementado lo permita (Si se habilita control de flujo RTS-CTS, será necesario que DCE mantenga activa su línea CTS para que la transmisión se produzca).

El tamaño de los BUFFERS de entrada y salida deberá de ser acorde con el tamaño de la trama que se está manejando, si se utiliza una trama de 512 Bytes el BUFFER deberá tener al menos ese tamaño.

Podemos monitorizar los BYTES pendientes de ser enviados o los que estén pendientes de ser leídos mediante los campos cbInQue y cbOutQue de la estructura COMSTAT vista anteriormente. Podemos además cambiar el tamaño de estos BUFFERS mediante la función SetupComm().

SetupComm()	
`BOOL SetupComm(HANDLE hFile,DWORD dwInQueue,DWORD dwOutQueue)`	
ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle devuelto por CreateFile()
dwInQueue	Tamaño que se desea para el BUFFER del receptor
dwOutQueue	Tamaño que se desea para el BUFFER del transmisor
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

5.1.4 Lectura y escritura en el puerto.

Después de tomar el control del puerto se procede a la comunicación en si misma, para esto es necesario utilizar funciones de lectura o escritura. Existen unas cuantas funciones previstas para esto, desde funciones que envían carácter por carácter, hasta funciones más específicas. Para lograr compatibilidad entre plataformas se utilizarán las funciones más primitivas, o sea, funciones básicas que permiten la construcción a la medida. Para leer y escribir en el puerto se utilizan las funciones genéricas ReadFile() y WriteFile(), ambas son funciones genéricas que permiten leer o escribir ficheros, puertos etc. En lo que sigue se muestra una descripción de cada una de ellas.

ReadFile()	

`BOOL ReadFile(HANDLE hFile,`
`LPVOID lpBuffer,`
`DWORD nNumberOfBytesToRead,`
`LPDWORD lpNumberOfBytesRead,`
`LPOVERLAPPED lpOverlapped);`	

ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle devuelto por CreateFile()
lpBuffer	Puntero al buffer donde la función depositará los datos leidos.
nNumberOfByteToRead	Número de bytes que se desean leer
lpNumberOfByteRead	Puntero a una DWORD donde la función indicará los datos que realmente se han leído.
lpOverlapped	Puntero a una estructura OVERLAPPED para acceso a especial a puerto.
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

La función utiliza como parámetros de entrada el manejador de dispositivo (hFile) y el número de bytes que se desean leer (nNumberOfByteRead). La función retorna a través de los argumentos pasados por referencia el número de bytes realmente leídos del dispositivo en lpNumberOfByteRead y los bytes leídos a través del BUFFER que empieza en lpBuffer. El argumento lpOverlapped es para permitir un modo de acceso sobrecargado que no se utilizará.

WriteFile()	

`BOOL WriteFile(HANDLE hFile,`
`LPCVOID lpBuffer,`
`DWORD nNumberOfBytesToWrite,`
`LPDWORD lpNumberOfBytesWritten,`
`LPOVERLAPPED lpOverlapped);`	

ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle devuelto por CreateFile()
lpBuffer	Puntero al buffer donde la función leerá los datos que escribirá en el puerto.
nNumberOfByteToWrite	Número de bytes que se escribirán en el puerto
lpNumberOfByteWritten	Puntero a una DWORD donde la función indicará los datos que realmente se han escrito.
lpOverlapped	Puntero a una estructura OVERLAPPED para acceso a especial a puerto.
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

La función WriteFile() utiliza como parámetros de entrada el manejador de dispositivo(hFile) y el número de bytes que se desean escribir (nNumberOfByteToWrite), y un BUFFER donde la función leerá los bytes que se escribirán en el dispositivo lpBuffer. La función retorna a través de los argumentos pasados por referencia el número de bytes

realmente escritos en el dispositivo en lpNumberOfByteWritten. Ambas devuelven TRUE si la operación de lectura o escritura se efectuó correctamente.

Los dispositivos sobre los que se utilizan estas funciones soportan el control de TIMEOUT o tiempo máximo de lectura y escritura. Por ejemplo, un dispositivo puede ser configurado para que la operación de lectura dure como máximo 100 ms. esto implica que si la operación se prolonga más del tiempo permitido la función finalizará normalmente, es decir retornará TRUE. Para determinar si una operación de lectura/escritura finalizó por un TIMEOUT habría que comparar los bytes que se querían leer/escribir con los bytes realmente leídos/escritos.

La funciones GetCommTimmeout() y SetCommTimeouts() permiten configurar como se realizarán las operaciones de lectura y escritura. Ambas trabajan sobre una estructura COMMTIMEOUTS (ver Tabla 5) que define el tiempo máximo (TIMEOUT) que durará una operación de lectura o escritura.

CAMPO	TIPO	DESCRIPCIÓN
ReadIntervalTimeout	DWORD	Tiempo máximo de espera (ms) entre dos bytes recibidos.
ReadTotalTimeoutMultiplier	DWORD	Multiplicador (ms) para calcular el TIMEOUT total en operaciones de lectura.
ReadTotalTimeoutConstant	DWORD	Constate (ms) para calcular el TIMEOUT total en operaciones de lectura.
WriteTotalTimeoutMultiplier	DWORD	Multiplicador (ms) para calcular el TIMEOUT total en operaciones de escritura.
WriteTotalTimeoutConstant	DWORD	Constate (ms) para calcular el TIMEOUT total en operaciones de escritura

Tabla 5 Miembros de la estructura COMMTIMEOUTS.

Si en el primer miembro usamos “MAXDWORD” y en el segundo “0” obtendremos que una operación de lectura finalizará inmediatamente incluso si no se ha leído nada del buffer. Si en el segundo y tercer miembro se usa 0 no se utilizarán el control de TIMEOUT en operaciones de lectura. Si en todos los miembros ponemos 0 las operaciones de lectura no finalizan hasta que no se lean los bytes que se piden. Para leer y escribir esta configuración se dispone de las funciones GetCommTimeout() y SetCommTimeout() respectivamente.

GetCommTimeoust()	
`BOOL GetCommTimeouts(HANDLE hFile,LPCOMMTIMEOUTS lpCommTimeouts)`	
ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle devuelto por CreateFile()
lpCommTimeouts	Puntero a una estructura de tipo COMMTIMEOUTS que se usará para leer la configurar del puerto.
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

SetCommTimeoust()	
`BOOL SetCommTimeouts(HANDLE hFile,LPCOMMTIMEOUTS lpCommTimeouts)`	
ARG. ENTRADA	DESCRIPCIÓN
hFile	Manejador del puerto o Handle devuelto por CreateFile()
lpCommTimeouts	Puntero a una estructura de tipo COMMTIMEOUTS que se usará para escribir la configurar del puerto.
TIPO SALIDA	DESCRIPCIÓN
BOOL	TRUE si la función se ejecutó correctamente.

En los dispositivos de campo es común que estos esperen un tiempo predeterminado para acceder al medio de comunicación, dependiendo de la velocidad con la que este procese la petición que se le realice y lo que establezca el protocolo utilizado, por esto estas dos ultimas funciones formarán parte de la configuración donde el usuario final decidirá el manejo de estos tiempos de espera.

5.1.5 Clase TWinSerCom para comunicaciones bajo Windows.

A fin de utilizar los potenciales de comunicación posible y desarrollar un modulo de programa capaz de proveer los servicios de comunicaciones seriales, y utilizando los conceptos de programación orientada a objetos, se define una clase que denominaremos TWinSerCom (*Servicio de comunicaciones bajo Windows*). En un módulo de proyecto nuevo se creo una nueva unidad compuesta por un fichero de programa (Comunica.CPP) y un fichero de cabecera (Comunica.H). La Figura 21 muestra de manera dinámica como funcionará dicho módulo de programa. En el fichero de programa se escriben las tareas y funciones generales propias de la clase. Este pasará por referencia los argumentos al archivo cabecera donde está la definición de las funciones y cuales API utilizará, esta entonces enviará a Windows una petición Win32 que el sistema operativo interpretará, realizará las acciones pertinentes sobre el hardware, y este a su vez responderá a Windows la solicitud hecha. Entonces Windows enviará una respuesta Win32 que será capturada por la función que invoco el servicio, guardando los resultados y pasándolos por referencia a la clase que solicitó el servicio. Finalmente se emite una respuesta a la solicitud primaria. Con la creación de la clase se pueden crear objetos de programación independientes y funcionales que presten los servicios cuando se les invoque.

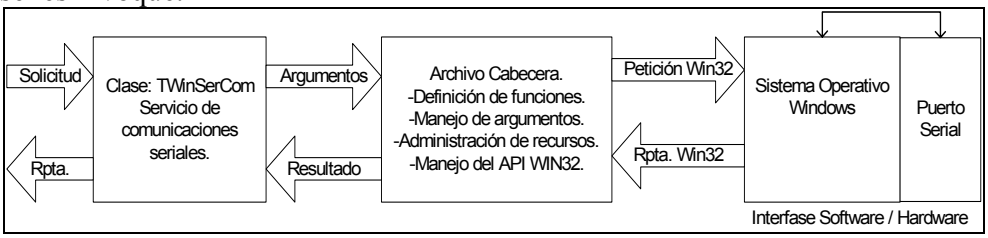


Figura 21 Funcionamiento de la clase TWinSerCom

La Tabla 6 muestra un resumen de los métodos y funciones asociados, del archivo cabecera, el código fuente se encuentra en el Anexo 4 *Código fuente estructurado* (Versión CD).

Método	Descripción
`TWinSerCom()`	Constructor, requiere un número de puerto
`~TWinSerCom()`	Destructor. Se encarga de cerrar el puerto
`Open(void)`	Abrir el puerto configurado en el constructor
`Open(`*`parametros`*`)`	Abrir cualquier otro puerto
`Close()`	Cerrar el puerto
`IsOpen()`	Revisar si el puerto está abierto
`IsDCE()`	Revisar si el DCE está conectado y listo
`TxByte()`	Transmitir un Byte
`RxByte()`	Recibir un Byte
`TxCad()`	Transmitir una cadena o trama
`RxCad()`	Recibir una cadena o trama
`LeeIde()`	Leer el manejador del puerto

Tabla 6 Interfase de la clase TWinSerCom.

5.1.6 Envío y recepción de tramas.

Una vez creada la clase esta puede ser insertada como parte de cualquier proyecto, por lo cual se tiene que crear un puntero a un objeto, que da un lugar a la clase y todos los recursos que este requiere para hacerse funcional. Finalmente tendremos un objeto al que solo invocamos pasando los parámetros por referencia y este como entidad independiente realizará las tareas para las que se habilite. Cuando se habla de parámetros por referencia se quiere señalar los diferentes punteros, valores y argumentos necesarios para que el objeto actúe de manera eficaz.

Desde cualquier módulo de programación se crea el objeto, se cargan las referencias en las estructuras previstas y se invoca los servicios del objeto. Dicho de otra manera si se quiere enviar una trama por el medio serial, primero se carga la trama en un buffer provisional, se invoca el servicio de apertura de puerto del objeto previamente creado, este informará si efectivamente resultó la apertura, entonces pasamos los parámetros necesarios invocando otro de los servicios (Envío de trama), este entonces realizará la tarea y devolverá un resultado, luego si se desea esperar una respuesta, se invoca el servicio de recepción del objeto, y este recogerá los resultados de la operación.

De esta manera queda construido un módulo independiente de servicios de comunicación serial, que realice todas las tareas requeridas para establecer un intercambio efectivo entre los dispositivos conectados. La elaboración de tramas, chequeo y corrección de errores forma ya parte del programa que invoque los servicios. La definición de funciones y código de programación se muestra en el Anexo 4. Código fuente estructurado (Versión CD).

5.1.7 Procesos y subprocesos.

El API WIN32 permite la creación de subprocesos, hilos o Threads esta es una capacidad de los sistemas operativos multitarea de que las aplicaciones en ejecución (procesos) puedan crear subprocesos que se ejecutan de forma independiente. Esta técnica es útil cuando deseamos hacer alguna tarea en segundo plano o cuando se dispone de varios procesadores y se desea asignar a cada uno de ellos una tarea específica. Por ejemplo, el WORD de Microsoft puede tener activado los subprocesos del corrector ortográfico o del corrector gramatical, de manera que los correctores se están ejecutando en “paralelo” con el editor del texto.

Del mismo modo que en el escritorio de WINDOWS podemos tener abiertos varios programas (procesos) simultáneamente, dentro de una aplicación podremos crear tantos subprocesos como necesitemos. Los procesos permiten por tanto implementar la multitarea a nivel de programa. No hay que abusar del número de subprocesos, ya que cada uno de ellos consume tiempo del procesador como si se tratase de un programa independiente. En comunicaciones los subprocesos se utilizan para recibir o transmitir en segundo plano mientras se atiende la interfase del usuario, sin esta posibilidad se correría el riesgo de que los controles propios de la aplicación, ventanas, botones y otros parecieran no responder y mostrarían comportamientos de retardo no adecuados para una aplicación profesional, por ejemplo mientras se espera la respuesta de un dispositivo el usuario no podría interactuar con el software.

El API Win32 incluye una serie de funciones para manejar los procesos (GetThreadPriority(), ResumeThread(), SetThreadPriority(), CreateThreat(), SuspendThreat() etc) estas funciones pueden ser utilizadas como las que se han vista anteriormente, no obstante

el C++ Builder encapsula y facilita el uso de subprocesos mediante la clase TThread. La Tabla 7muestra un resumen de la interfase de la clase.

Nombre	Descripción
TThread()	Es el constructor de la clase, admite un parámetro lógico para indicar si el subproceso se crea activado o no
Execute()	Código que se ejecutará como si fuese una aplicación independiente. El constructor de la clase llama a este método de forma automática.
Suspend()	Suspende temporalmente la ejecución de un proceso. En este estado el proceso no hace uso del microprocesador.
Resume()	Reanuda la ejecución de un subproceso
Termiate()	Finaliza un subproceso y libera los recursos que tuviese asignados. Cuando un proceso finaliza (al cerrar el programa) se terminan de forma automáticas todos sus subprocesos
Synchronize()	Permite ejecutar una función con acceso a cualquier objeto de la VCL.

Tabla 7 La clase TThread.

La clase no se encuentra encapsulada como un componente de la VCL (librería de controles visuales de C++ Builder), sino que para utilizarla debemos añadir una nueva unidad del tipo ThreadObject. Dentro de esta unidad se encontraría el código asignado al objeto, este puede tener acceso a los componentes visuales para pasar datos o mensajes, pero daría el aspecto de ejecución transparente a la aplicación en todo momento. El código correspondiente al manejo de tramas de comunicación y otros procesos que requieran tiempo considerable de cálculo se realizará bajo este esquema.

Los subprocesos combinados con el manejo de eventos es el mecanismo más rápido y eficaz para realizar comunicaciones a través del puerto serie, mientras que el proceso principal atiende la interfase del usuario, un proceso en segundo plano atiende al puerto: monitorización, lectura, escritura etc.

5.2 Etapa 2: Desarrollo de comunicaciones en red (Internet).

5.2.1 Introducción a la programación con Windows Sockets.

El objetivo de esta parte es ofrecer una introducción al uso del API Winsock, de forma que una vez leídos y asimilados sus conceptos y su funcionamiento, se pueda entender el objetivo básico de esta especificación: establecer una comunicación entre dos máquinas, mediante la conexión y el envío/recepción de datos entre ambos computadores.

Winsock es la base sobre la cual se construye prácticamente la totalidad de aplicaciones que utilizan Internet como forma de comunicación, ya sea e-mail, navegadores, FTP; etc. Si bien un programa puede usar controles ActiveX, funciones Wininet o cualquier otro mecanismo de alto nivel, en último extremo será Winsock quien haga efectiva la comunicación.

La especificación Winsock se basa en el concepto de "socket", "enchufe" en castellano. Un socket representa simplemente un punto de conexión, que puede ser usado para enviar o recibir datos, previa conexión con otro socket. Es aquí donde aparece el famoso principio cliente/servidor, por el cual uno de los sockets o puntos de conexión actúa como servidor, atendiendo las peticiones del otro socket, que adopta el papel de cliente, enviando peticiones al servidor y recibiendo a su vez el resultado de dichas solicitudes. No obstante, es fácil de apreciar el hecho de que, a nivel de Winsock, el papel de cliente y servidor no está claramente diferenciado, ya que un mismo socket envía y recibe datos; a nivel de aplicaciones, es más claro en sí: un cliente FTP se conecta a un servidor FTP, un cliente de correo se conecta a un servidor de correo, etc.

Es necesario aclarar que los sockets son realmente un mecanismo de comunicación entre procesos, de ahí que no sea estrictamente necesaria la existencia de dos máquinas, siendo perfectamente posible establecer una conexión entre dos aplicaciones colocadas en el mismo computador.

5.2.2 Conceptos básicos de WinSock.

Como ya se comentó, el socket es el componente básico del API Winsock, y representa un punto de conexión. Este punto de conexión, a su vez, está representado por otros dos elementos fundamentales: la dirección y el número de puerto. La dirección identifica la ubicación en la red del "enchufe", al referirnos a la dirección hablamos de la dirección IP (dirección de Internet) y el número de puerto diferencia los distintos procesos de comunicación que pueden tener lugar en la misma máquina. Es por ello que los más importantes servicios de Internet tienen asignado un número de puerto fijo, como pueda ser el 80 para HTTP, 23 para Telnet, 502 para MODBUS TCP o 20000 para DNP3 sobre TCP, entre otros muchos. Hasta este punto es importante señalar que una Intranet las direcciones IP y la asignación de puertos no tiene por que ser normada, además que todavía existen puertos libres para aplicaciones que pueden llegar hasta el 65535 (FFFF hexadecimal).

Casi todas las funciones de Winsock necesitan usar una variable de tipo SOCKET; el tipo de dato SOCKET es un entero sin signo, que actúa como handle a una estructura (interna y oculta al programador) conteniendo los distintos datos que conforman la conexión (dirección, puerto, etc) y a los que sólo se puede acceder mediante las funciones del API.

5.2.3 Los protocolos TCP y UDP.

A un nivel más bajo, en la capa de transporte, se sitúan los famosos protocolos TCP/IP: TCP y UDP, y es necesario conocer al menos sus características básicas y diferencias antes de empezar a usar sockets, ya que dependiendo del tipo de aplicación que vayamos a escribir, será más apropiado uno u otro tipo de conexión.

TCP (*Transmission Control Protocol*) es un protocolo orientado a conexión; eso significa que se ocupa de transmitir datos entre dos máquinas que previamente han establecido una conexión, y también se encarga de que todos los paquetes de datos lleguen a su destino sin alterar su contenido, y además en el orden correcto, asumiendo él mismo la corrección de los posibles errores mediante el uso de los mecanismos adecuados.

Por su parte, UDP (*User Datagram Protocol*) no está orientado a conexión, en el sentido de que no existe una conexión previa entre remitente y destinatario: el primero envía paquetes de datos al segundo, pero no se preocupa de si llegan a su destino, si lo hacen en el orden adecuado y si su integridad se mantiene por el camino.

De todo lo anterior se deduce que TCP es un protocolo perfecto para aquellas aplicaciones para las que no sea admisible la pérdida o corrupción de los datos enviados o recibidos: clientes FTP, programas de chat, etc. Por otro lado, UDP es ideal para aplicaciones que necesiten aprovechar al máximo el ancho de banda disponible, y que puedan permitirse la pérdida de algún paquete de información: juegos en red, emisión de video y audio, etc. En nuestro caso se trata de la comunicación de paquetes orientados a conexión segura, de allí que los protocolos MODBUS y DNP, se basen en TCP y no en UDP.

5.2.4 Programación con WinSock.

Para el caso de estudio se desarrolló con el compilador Borland C++ Builder una aplicación cliente/servidor. En el caso de la interfase tipo Gateway se trata de la aplicación Servidora, que abre un puerto por el que acepta la conexión entrante que proviene de la aplicación Cliente. Esta le envía las cadenas de datos que la aplicación cliente interpreta como solicitudes, y el servidor se limita a aceptar estas solicitudes, procesarlas de manera pertinente y emitir posibles respuestas, conformando una especie de comunicación bidireccional. A continuación se desarrollará el código fuente de ambos programas para ilustrar el uso del API Winsock, explicando en cada caso el objetivo de las funciones empleadas, por lo que no se comentara el código de forma exhaustiva; cualquier programador de nivel medio debe ser capaz de seguir y entender el código fuente de ambas aplicaciones. La programación con sockets es un asunto amplio y complejo, de manera que es importante señalar que se explica solo las partes que nos interesan en este desarrollo.

5.2.5 Desarrollo de sistema Cliente – Servidor.

5.2.5.1 Cliente y Servidor – Inicializar WinSock.

Antes de empezar a usar los sockets, es necesario inicializar la sesión Winsock mediante la función WSAStartup( ); el prototipo es éste:

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

El primer parámetro (wVersionRequested) determina el número de versión de Winsock más alto que nuestro programa puede manejar, en nuestro caso usamos la 1.1 por ser la mas sencilla y soportada por las mas altas, especificándola mediante la macro MAKEWORD. El segundo parámetro (lpWSAData) es un puntero a una estructura de tipo WSADATA, que recibirá información sobre la implementación de Winsock que tengamos en nuestro computador: su número de versión, una descripción y el estado actual de la misma, etc. Si la llamada tiene éxito, el valor de retorno será cero y ya podremos usar el resto de las funciones de sockets.

5.2.5.2 Cliente y Servidor – Crear un Socket.

A continuación debemos crear un punto de conexión, esto es, un socket, antes de poder establecer comunicación; lógicamente, esto es necesario tanto en el programa Cliente como en el programa Servidor. Para ello empleamos la función socket( ), que tiene el siguiente prototipo:

```
SOCKET socket(int af, int type, int protocol);
```

El parámetro af especifica el formato de direcciones que usaremos, en este caso serán de tipo Internet, así que la constante empleada será AF_INET. Por su parte, type indica el tipo de socket que vamos a crear, a elegir entre cadena de datos o datagramas, o dicho de otra forma, TCP o UDP. En nuestro ejemplo nos inclinamos por el primero, así que usamos el valor SOCK_STREAM. Por último, el parámetro protocol establece el protocolo que se usará en este socket, y hay que pasarle el valor cero para que dicho protocolo sea asignado automáticamente por Winsock a partir del anterior parámetro type.

Si todo va bien, obtendremos un valor de tipo SOCKET, que representa el nuevo punto de conexión obtenido y que tendremos que usar en las siguientes funciones para llegar a establecer una comunicación completa. En caso de error, la función devolverá la constante INVALID_SOCKET.

5.2.5.3 Servidor – Vincular el socket a una dirección local.

Además de conectarse a una dirección remota, un socket necesita enlazarse a una dirección local, ya que el protocolo IP necesita la dirección de un remitente al que identificar y, en caso necesario, remitir información sobre errores, de estado, etc. Esta operación se conoce como "vincular" y para llevarla a cabo se usa la función bind( ), con este encabezado:

```
int bind(SOCKET s, const struct sockaddr FAR* name, int namelen);
```

Aquí le pasamos primero el socket previamente creado (s), en segundo lugar un puntero a una estructura que describe la dirección local (también descrita como "nombre") a la que se vincula el socket, y para terminar un valor que especifica el tamaño de dicha estructura, cuyo nombre es sockaddr y está definida como sigue:

```
struct sockaddr
{
 u_short  sa_family;
 char      sa_data[14];
};
```

El campo sa_family indica la familia o formato de direcciones, y hay que inicializarlo con el mismo valor usado en la creación del socket, en este caso AF_INET. El campo sa_data es un arreglo de bytes que especifica el puerto y la dirección al que vincularemos el socket; como esta estructura resulta un poco críptica, a menudo se usa otra equivalente pero más clara y coherente, llamada sockaddr_in, que tiene este aspecto:

```
struct sockaddr_in
{
  short sin_family;
  u_short sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
};
```

En esta otra se distingue la famiia, el puerto y la dirección IP que a su vez se desglosa en otra estructura, de nombre in_addr:

```
struct in_addr
{
   union
   {
      struct { u_char s_b1, s_b2, s_b3, s_b4; }  S_un_b;
      struct { u_short s_w1, s_w2; }          S_un_w;
      u_long                                  S_addr;
   } S_un;
};
```

De esta forma podemos indicar de forma más cómoda cada byte de los que componen la dirección IP.

Volviendo a nuestro caso, tras crear el socket declaramos una estructura sockaddr_in, y la inicializamos a cero, tras lo cual la rellenamos con los valores adecuados. Usamos AF_INET para el tipo de direcciones, y para el puerto establecemos un valor elegido arbitrariamente (solo para el caso de prueba). Por último, para la dirección local usamos la constante INADDR_ANY, con objeto de que sea el mismo Winsock quien decida a qué dirección local vincular el socket. Sólo queda llamar a bind( ) y comprobar su valor de retorno, que será cero si la operación se ha realizado correctamente.

Debemos detenernos un momento en este punto, para explicar un concepto que hay que tener siempre presente en determinadas circunstancias, a la hora de usar sockets, y es el orden de los bytes que se envían o reciben. Se sabe que, las máquinas Intel almacenan la información en memoria usando el formato "Big-Endian", en el que para un WORD (o DWORD *Palabra*) primero se almacena en memoria el byte bajo (o WORD bajo) y a continuación el byte (o

WORD) alto, mientras que en el resto de máquinas es al contrario, primero el byte alto y luego el byte bajo.

Esta distinción se traslada al API Winsock, de manera que los datos que recibimos de Internet en las estructuras binarias usadas en la comunicación (sockaddr, etc) llegan en el llamado "Network byte order", y deben ser convertido al llamado "Host byte order" antes de ser usados. De la misma manera, los valores que se van a enviar en dichas estructuras deben traducirse del "Host byte order" al "Network byte order". Para ello disponemos de un grupo de funciones de conversión para WORDs y DWORDs (htonl, htons, ntohl, ntohs). En nuestro caso Servidor, el valor de puerto lo especificamos con la función htons (HostToNetworkShort), de esta forma traducimos el orden de los bytes para que sean manejados correctamente.

5.2.5.4 Servidor – Empezamos a "Escuchar".

Ahora que nuestro socket está vinculado a una dirección local, tenemos que "ponerlo a la escucha" para que pueda detectar una conexión entrante. La función listen hace esto mismo, estableciendo una cola para las conexiones entrantes; listen sólo necesita que le especifiquen el socket a usar (g_Socket) y el tamaño máximo de la cola, y la mejor forma de hacer esto último es indicar la constante SOMAXCONN, para que sea el sistema quien determine el mejor valor. Si listen devuelve cero, sabremos que la operación tuvo éxito.

```
int listen(g_Socket, SOMAXCONN)
```

De nuevo debemos hacer una pausa para explicar otro tema, muy importante, de la programación con Winsock: el bloqueo. Para explicarlo de forma rápida y clara, el comportamiento de los sockets es, por defecto, síncrono. Esto significa que, por ejemplo, un intento de conexión a un host remoto paraliza la ejecución del programa hasta que la conexión tiene lugar o hasta que se produce un error; de la misma forma, un intento de recibir datos paraliza la aplicación hasta que no haya datos preparados para ser recibidos; y esto puede ser aceptable o incluso útil para algunos programas, y desastrosos para otros que es nuestro caso.

Por ello existe un mecanismo que permite cambiar el estado de un socket a "no bloqueante", mediante la función ioctlsocket( ). En este nuevo modo de operación, si se llama a una de las funciones que pueden bloquear, éstas devuelven el control inmediatamente, y la manera de saber si hubiera bloqueado es comprobar si el código de error obtenido con la función WSAGetLastError( ) es WSAEWOULDBLOCK.

De nuevo tenemos que esta solución puede ser aceptable o no según el caso, así que muchos programadores pueden preferir (de hecho lo hacen) usar sockets bloqueantes pero en distintos procesos de ejecución (Thread), de manera que si un socket debe bloquear, lo haga en un proceso independiente y no en el proceso principal de la aplicación.

Hay otro mecanismo disponible en Winsock, que nos permite estar informados de los distintos sucesos que pueden tener lugar durante la comunicación, y consiste en decirle al API, mediante la función WSAAsyncSelect( ), que nos envíe un mensaje de ventana cada vez que ocurra algo relevante. Entre otras cosas, podemos ser avisados cada vez que existan datos para recibir o enviar, cada vez que se establezca una conexión, cada vez que se detecte una conexión entrante o cada vez que un socket sea cerrado. Entonces el procedimiento de ventana de la aplicación (o de la ventana especificada) recibirá un mensaje al cual podremos reaccionar de la manera más adecuada, sin preocuparnos de bloqueos ni tiempos de espera (es necesario

indicar que WSAAsyncSelect( ) cambia automáticamente el estado del socket a "no bloqueante").

5.2.5.5 Servidor – Aceptar la conexión entrante.

Después de poner a escuchar al socket, empleamos la función WSAAsyncSelect( ) para que seamos avisados, mediante mensajes, cuando llegue una petición de conexión por parte del cliente. Le pasamos el socket, el handle de la ventana que recibirá los mensajes, un valor de mensaje definido por nosotros mismos, y por último la constante FD_ACCEPT, que le indica el tipo de suceso sobre el que debe avisarnos. Sería algo como esto:

```
if(WSAAsyncSelect(g_Socket, Handle, WM_ACEPTARCONEXION, FD_ACCEPT) == SOCKET_ERROR)
        ShowMessage("No se pudo solicitar la notificación mediante mensajes");
```

El mensaje propio WM_ACEPTARCONEXION lo definimos así:

```
#define  WM_ACEPTARCONEXION   (WM_USER + 1)
```

Nuestro procedimiento de ventana (previamente hemos sub-clasificado el procedimiento de ventana de la aplicación) debe responder adecuadamente al mensaje, así que comprobamos si el mensaje recibido es WM_ACEPTARCONEXION. En ese caso, se recibirá un mensaje de Windows (wParam, lParam, etc.), wParam contiene el socket sobre el que ha ocurrido el evento, la palabra baja de lParam indica el evento que ha ocurrido y la palabra alta de lParam contiene un código de error.

Aquí respondemos al mensaje FD_ACCEPT permitiendo la conexión entrante mediante la función accept( ), cuyos parámetros son el socket usado, y dos punteros opcionales, uno a una estructura que identificará la dirección remota (esto es, el cliente) y otro a un valor que indica el tamaño de dicha estructura. El resultado, en caso de éxito, es un nuevo socket con las mismas propiedades que el socket "padre" y que será usado desde ese momento en el proceso de comunicación.

5.2.5.6 Servidor – Recibir datos del cliente.

Después de haber aceptado la conexión entrante, volvemos a usar WSAAsyncSelect( ), esta vez con el nuevo socket, para que Winsock nos avise cada vez que haya datos para leer y también cuando el cliente cierre la conexión (eventos FD_READ y FD_CLOSE). El codigo en C seria algo como sigue:

```
static SOCKET NuevoSock;

if(WSAAsyncSelect(NuevoSock, Handle, WM_FLUJODATOS, FD_READ|FD_CLOSE) == SOCKET_ERROR)
          ShowMessage("No se pudo solicitar la notificación mediante mensajes");
```

Por tanto, se añade en el procedimiento de ventana una rutina para manejar el nuevo mensaje WM_FLUJODATOS, que previamente fue definido de esta forma:

```
#define  WM_FLUJODATOS       (WM_USER + 2)
```

Cada vez que llegue un código FD_READ, usaremos la función recv( ) para recuperar los datos y posteriormente darles el debido proceso desde la aplicación Servidor. recv( ) necesita un buffer, que será rellenado con los datos enviados por el cliente, y un valor que

indica el tamaño de ese buffer. En nuestro caso se decidirá posteriormente de cuantos bytes será el buffer de recepción, por lo que la función no intentará recibir más allá de esa cantidad.

5.2.5.7 Servidor – Cortar la conexión.

Ya sólo nos queda responder al evento FD_CLOSE, que nos dice que la conexión remota (el cliente) ha sido finalizada o interrumpida, así que simplemente cerramos los dos sockets existentes y finalizamos la sesión con WSACleanup( ).

```
case FD_CLOSE:
  closesocket(NuevoSock);
  closesocket(g_Socket);
  WSACleanup();
  break;                    //Salir de la rutina.
```

Hasta aquí hemos visto el funcionamiento de la aplicación Servidora, en nuestro caso la aplicación es meramente Servidora, por lo que finalizaremos con un resumen funcional del sistema.

5.2.6 Funcionamiento del sistema Cliente – Servidor.

El sistema final para esta etapa propone lo que se observa en la Figura 22, nuestro proceso realiza la apertura de un canal creando un Socket para ello. Hace publicidad de la dirección vinculando el Socket con una dirección local. Dispone el Socket para aceptar conexiones entrantes (escuchar). En caso de recibir una solicitud de conexión es aceptada, utilizando funciones especiales para un modo que no bloquee la aplicación. Hace lectura de la petición y responde enviando los datos solicitados por la aplicación cliente.

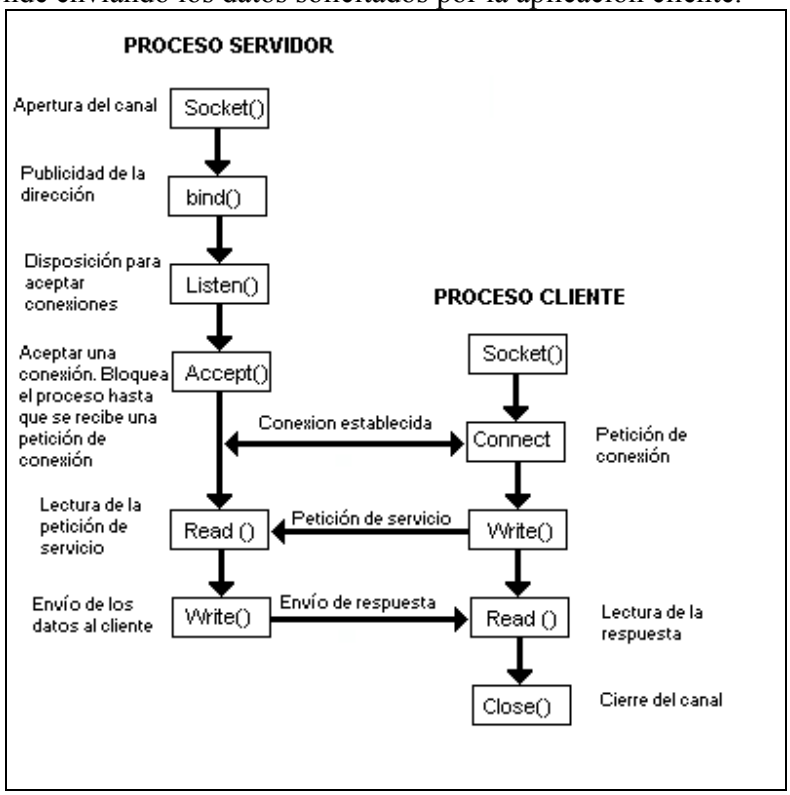


Figura 22 Sistema Cliente – Servidor a través de Sockets.

5.3 Etapa 3: Desarrollo de Gateway (Modbus TCP).

5.3.1 Descripción general del modelo.

En las etapas 1 y 2 de este desarrollo, se manejaron de manera separada dos aspectos fundamentales para la creación de la interfase tipo Gateway. Como primer aspecto se debe poder interactuar con dispositivos de campo que funcionan bajo sistemas de transmisión de datos seriales, el método de acceso empleado para este servicio se denomina "Sondeo". Este método consiste en el envío de un mensaje por una entidad al medio, y esperar por una respuesta solo por un tiempo predeterminado, por lo que si el dispositivo no contesta o tarda más del tiempo previsto de respuesta, la transacción se considera fallida, debiéndose implementar mecanismos para notificar esta falla. Visto de esta manera el sistema funcionaría como un maestro de comunicación, por lo que la tarea de interrogar dispositivos depende solo del software en sí. Por otra parte el sistema se comunicará vía Internet con otro software que funcionará como Cliente (Maestro) por lo que el software será una aplicación servidora, de allí que la tarea de atender al cliente debe ser de más alta prioridad, que la de interrogar dispositivos. Es importante mencionar que la interfase tipo Gateway no funciona como una pequeña maestra, por lo que el trabajo se concentra a labores de interpretación entre un medio bajo un protocolo, con otro medio con protocolo distinto. Bajo este esquema de servicio se explicará el modelo de la interfase tipo Gateway mostrado en la Figura 23.

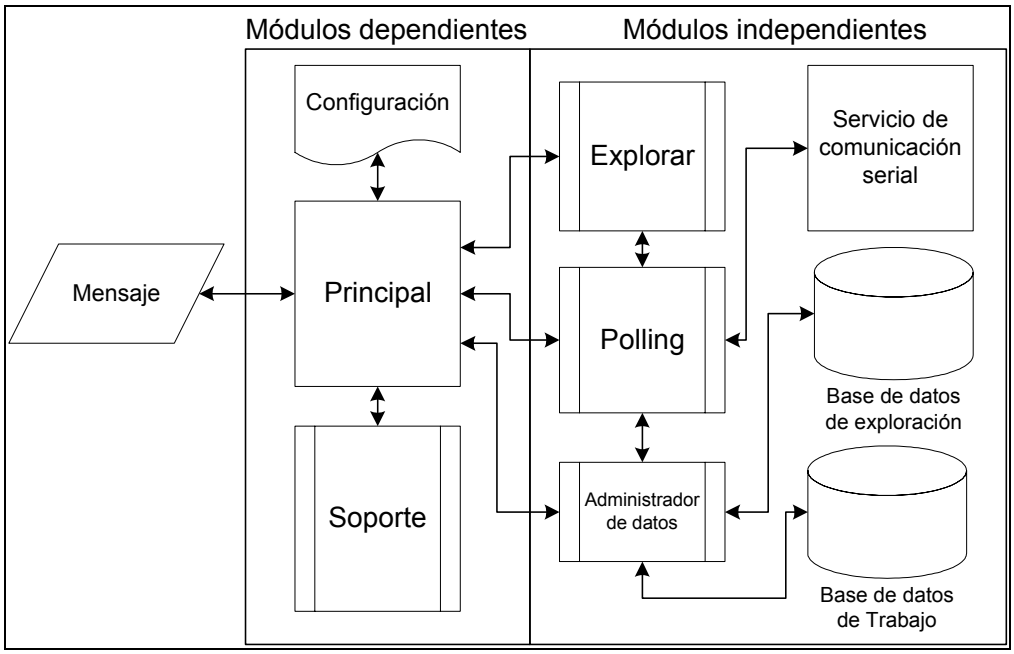


Figura 23 Modelo de Interfase tipo Gateway bajo MODBUS TCP.

Los módulos de la interfase fueron separados en dos grupos según su independencia; por módulos dependientes entendemos los procesos y rutinas que están sujetas a cambios de

manera permanente, ya sea por el usuario o vía Internet por el maestro. Los módulos independientes son procesos que si bien es cierto son accionados, monitoreados y controlados por la aplicación principal, contienen una tarea específica e independiente, y se realizan como subprocesos, siendo transparentes para la aplicación principal. La aplicación principal les asigna la tarea, estos la reconocen, la ejecutan y emiten su respuesta al terminar descargando datos y activando eventos de la aplicación principal siguiendo la filosofía de programación orientada a eventos.

La descripción de los módulos y su funcionamiento se listan a continuación:

- El mensaje es una solicitud hecha por el maestro de red, maneja el protocolo MODBUS TCP.
- La aplicación principal es la encargada de gestionar todos los procesos que forman parte del sistema. Contiene un archivo de cabecera "Soporte" donde descansan las rutinas y funciones numéricas, siendo de existencia virtual ya que forma parte del código de programa.
- Existe un archivo físico "Documento" (Remotas.dat) donde se recoge la configuración que la interfase contiene, y que la aplicación principal utiliza para recoger y descargar datos vitales de funcionamiento.
- Los módulos independientes son también de existencia virtual ya que también forman parte del código, exceptuando las dos bases de datos que son de existencia física en disco duro.
- La base de datos de trabajo contiene la información de cada uno de los dispositivos de campo que estén configurados.
- La base de datos de exploración sostiene información de solo lectura para realizar tareas de auto detección de dispositivos de campo.
- El modulo Polling (Interrogación) es accionado por el principal que le pasa datos por referencia, tiene la misión de tomar las características del dispositivo de campo, conformar tramas (MODBUS RTU y DUCBUS), administrar los servicios de comunicación serial e interrogar los dispositivos, y por último avisar al principal que ha finalizado su tarea.
- El modulo "Explorar" ejecuta una función especializada en detectar dispositivos de campo a través de sondeo, descansando su funcionamiento en las tareas de interrogación del módulo Polling.
- El administrador de bases de datos es un componente de programación conocido como BDE (*Borland Database Engine*) o sea un motor de base de datos que detallaremos posteriormente, su misión suministrar los datos que los módulos soliciten.

5.3.2 Descripción de interfaz de usuario.

La interfaz de usuario esta contenida dentro del modulo principal, es la parte visible del software, contiene los controles tipo Windows necesarios para realizar operaciones de configuración, edición de dispositivos y monitoreo de eventos. La misión principal es proveer al usuario final de información y negociar con este sus opciones de funcionamiento. El diseño de la interfaz está orientado en el modelo de los drivers que utilizará el SCADA, a manera de asegurar transparencia ante el usuario final, no cuenta con una ayuda muy elaborada, pero cuenta con ventanas de dialogo que suministran información al usuario. La Figura 24 muestra el aspecto general de la interfase.

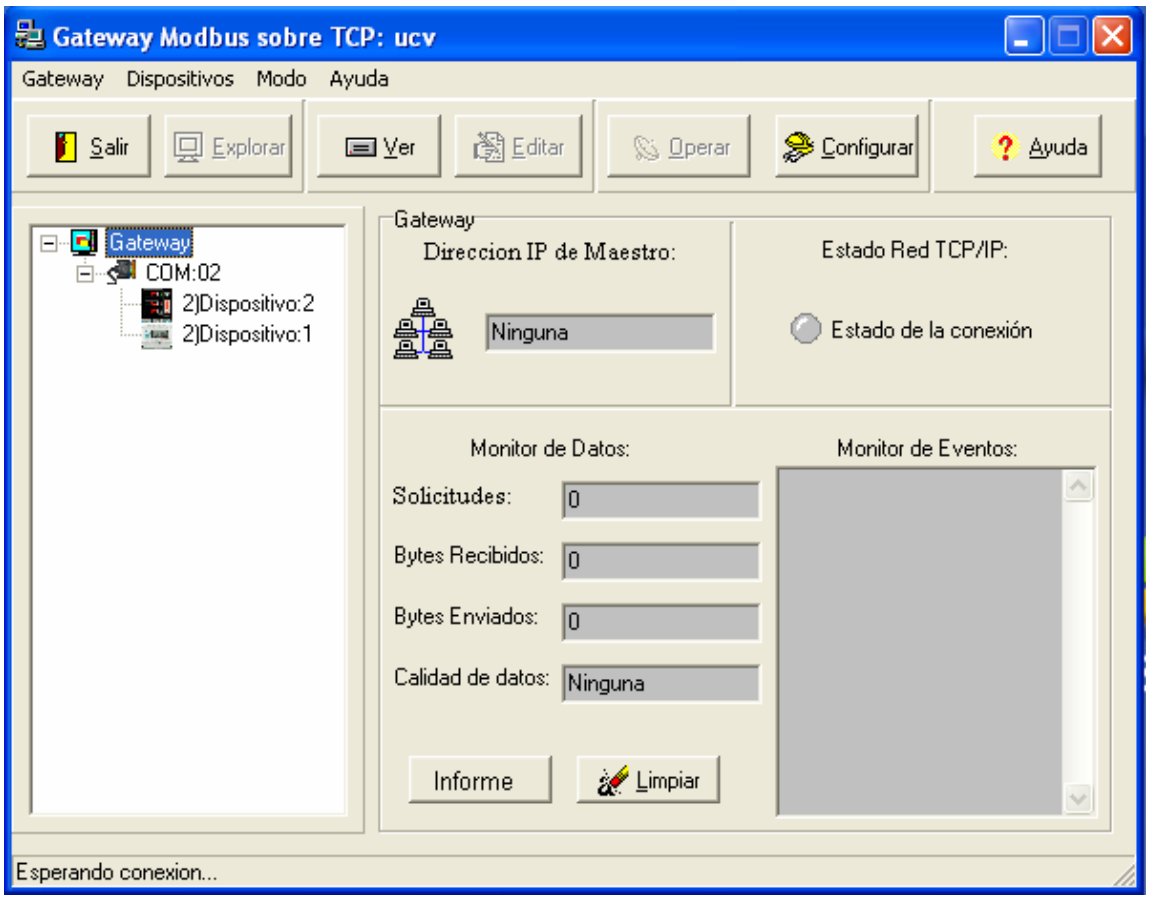


Figura 24 Interfaz de Usuario de Gateway.

Con diagramas tipo árbol, menús de barra, botones, cajas de edición y objetos de señalización se provee al usuario una interfaz de tipo Windows amigable, no se dispone de múltiples ventanas de comando, una sola concentra todos los modos de funcionamiento. Cada vez que se haga clic en una opción automáticamente se entra en la opción que se escogió. La Figura 24 muestra la ventana principal cuya finalidad es mostrar los datos del cliente de red (maestro) que accede a sus servicios, tiene un indicador tipo LED que parpadea cuando existe una conexión, un monitor de eventos donde se envían mensajes de las tareas, botones de comando y controles de edición mostrando transacciones realizadas y datos de interés. En este modo el sistema opera (nótese que el botón operar esta deshabilitado) y se puede monitorear el estado de cada uno de los dispositivos haciendo clic sobre el dispositivo en el árbol de opciones, en el botón “Ver” o a través del menú. El resultado se muestra en la Figura 25.

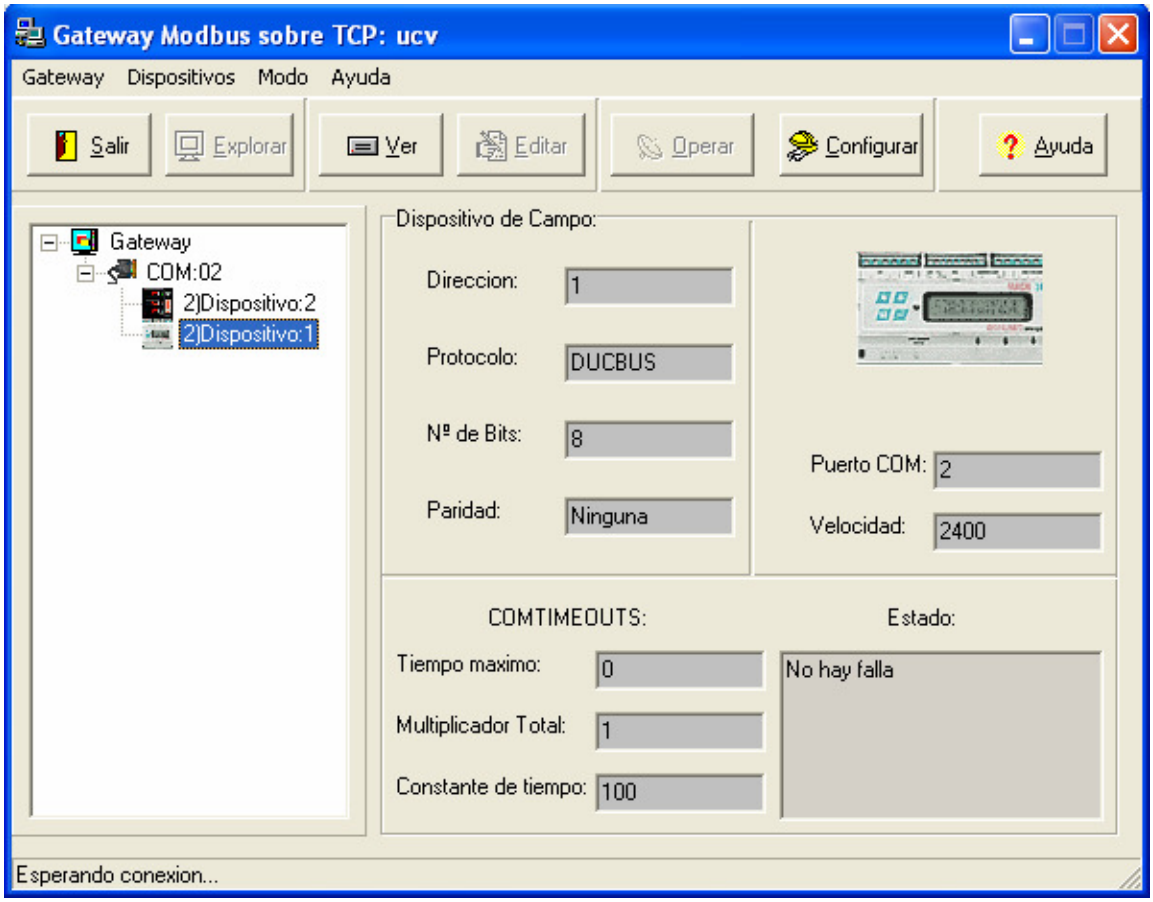


Figura 25 Interfaz para visualizar opciones de dispositivo.

Bajo esta interfaz se tiene la posibilidad de visualizar todas las características de los dispositivos de campo: dirección, protocolo, velocidad, estado del dispositivo y otros. Pero en este modo de visualización el software es capaz de ejecutar transacciones entre un maestro de red y dispositivos de campo, razón por la cual en este modo no se pueden editar las características que se muestran. Para poder configurar el software, este debe inhibirse de realizar transacciones, ya que en modo de configuración se van a editar características que forman parte del funcionamiento y son requeridas para ejecutar transacciones. Para entrar en el modo de configuración se debe escoger el correspondiente botón o escoger la opción desde la barra de menú. Es importante señalar que de la configuración hecha depende el buen funcionamiento de la interfase tipo Gateway. Si entramos en este modo y seleccionamos un dispositivo o el botón editar, pasaremos a una pantalla similar a la mostrada en la Figura 26.

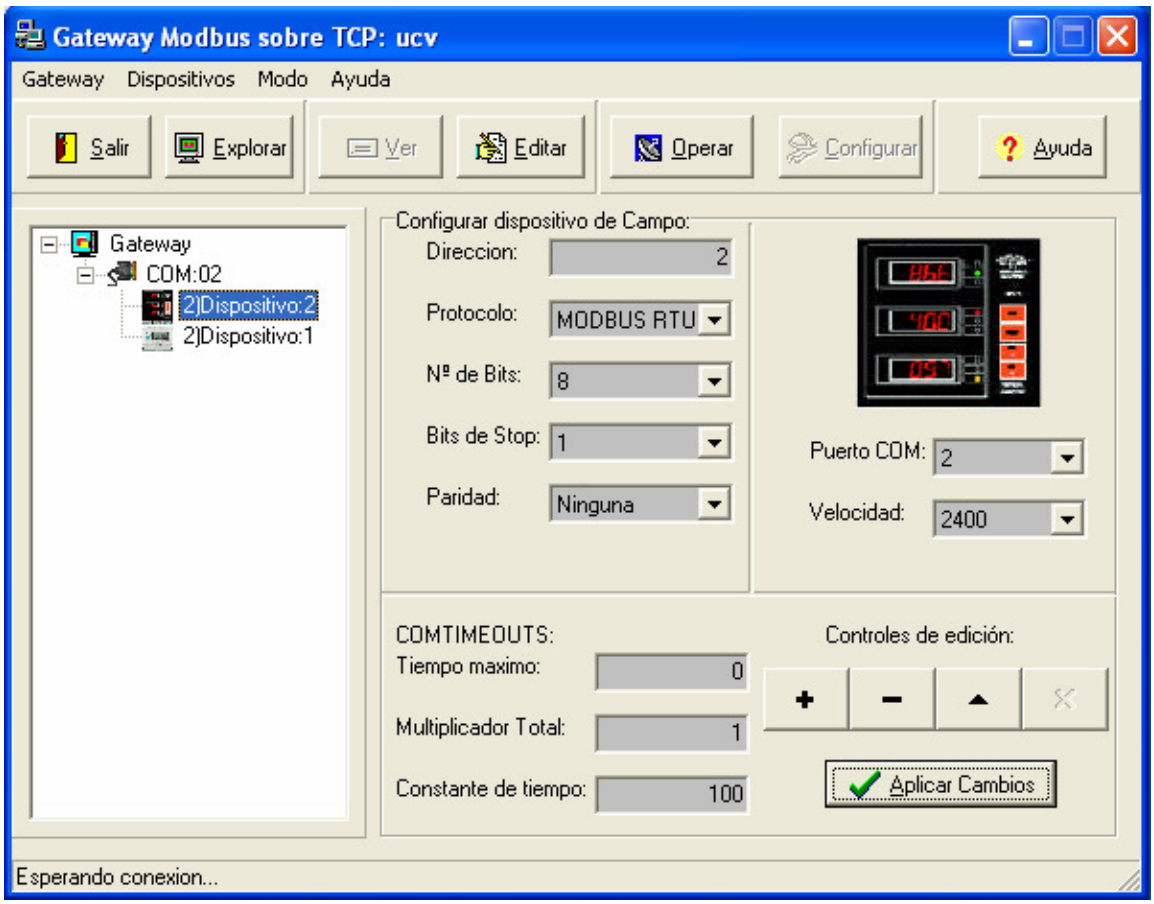


Figura 26 Interfaz para editar parámetros de dispositivo de campo.

En esta interfaz se tiene la posibilidad de modificar cualquier parámetro de configuración del dispositivo de campo, además la posibilidad de agregar, quitar o cambiar un nuevo dispositivo utilizando una simple barra de edición, para cada control y solo con colocar el ratón sobre el se desplegará una ayuda asociada al parámetro. Si escogemos en modo de configuración la opción Gateway del árbol de opciones, entramos en la configuración de la interfaz Gateway en sí, se puede escoger aquí opciones de explorar automáticamente los dispositivos de campo o cambiar el puerto en el cual esta aceptando conexiones el software. Esto con la finalidad que se puedan asignar números de puerto diferentes, así como es requerido en los sistemas SCADA.

Si se decide usar el modo de exploración el sistema explorará en busca de dispositivos de campo (ver Figura 27), explorando los 4 puertos por defecto que utilizan las computadoras convencionales. Es importante señalar que solo se exploran dispositivos MCA de LIFASA ó MACH30 de DUCATI, ya que esta diseñado solo para estos dispositivos el modo de exploración. Pero es también de interés mencionar que se puede configurar manualmente cualquier dispositivo de campo que utiliza MODBUS RTU como protocolo de comunicación de datos.

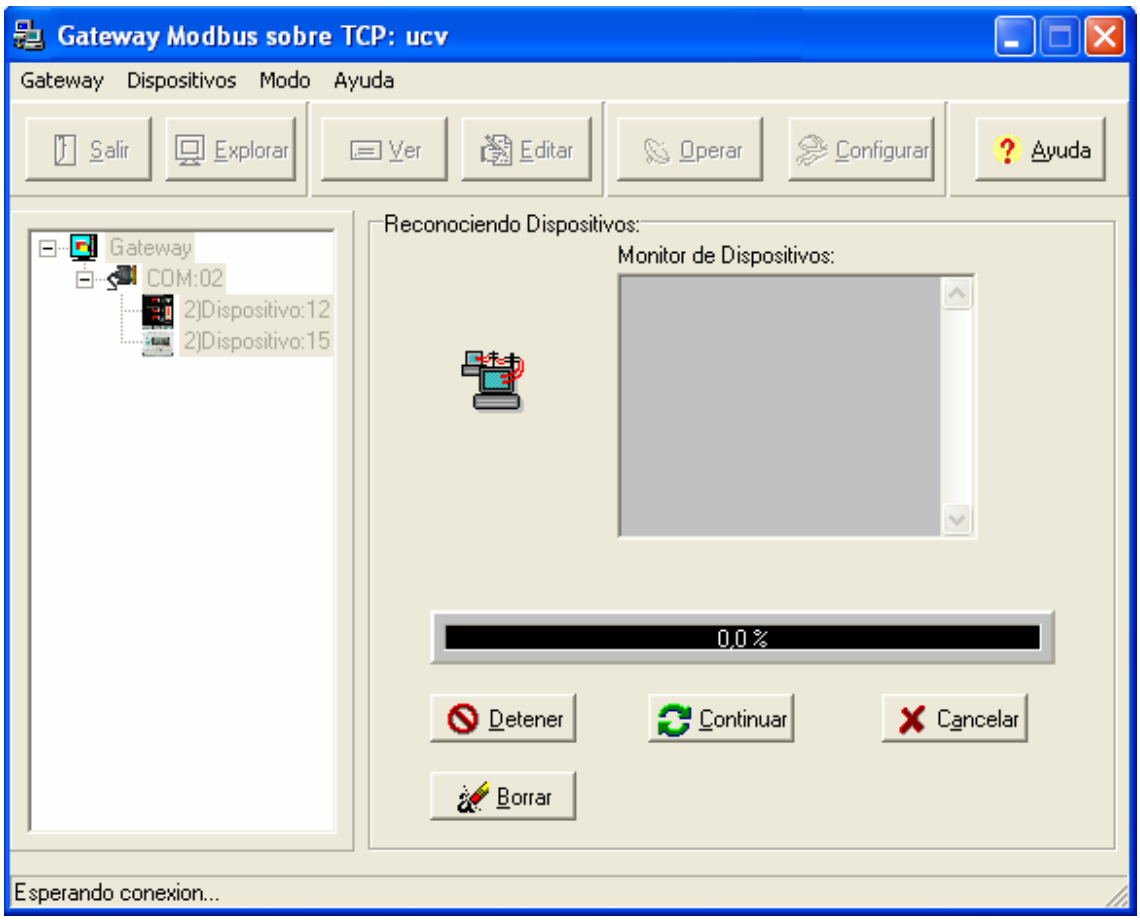


Figura 27 Interfaz para explorar dispositivos.

En este modo especial se tienen las opciones de detener y continuar el proceso o cancelarlo. A medida que se detectan dispositivos estos se anexan a la lista, por lo que si ya se han detectado todos y el proceso continúa explorando puede ser cancelado. Por último señalamos que en caso de escoger la opción explorar, todos los dispositivos de campo configurados serán borrados, por lo que se recomienda utilizar esta opción solo una vez al instalar el software y realizar después ajustes de dispositivos no detectados.

5.3.3 Acceso a datos de programa (Archivos y base de datos).

Como ya se ha mencionado los datos de configuración de la interfase tipo Gateway están contenidos en un archivo de texto llamado “Remotas.dat” un archivo donde se guardan datos en formato universal ASCII, la interfase está diseñada para operar este archivo pero no para crearlo, por lo que es indispensable que el mismo esté presente en el mismo directorio raíz donde descansa la aplicación; si por accidente se corrompiera o borrara, el software sería capaz de generarlo pero se perderían los datos y quedaría desconfigurada la interfase. Los datos de los dispositivos de campo no se guardan allí, sino en la base de datos de trabajo por lo que es parte fundamental del mismo, el archivo de datos “Dispositivos7.dbf”. Sin este último la aplicación no funcionaría, habría un error de aplicación y terminaría automáticamente. También es el caso de la base de datos de exploración “Inicio.dbf” por lo que adjunto al software de aplicación deben existir estos 3 archivos adicionales en la misma carpeta.

Para permitir el acceso a bases de datos locales y servidores SQL, C++ Builder proporciona el Borland Database Engine (BDE), que no es más que una colección de DLLs (*librerías de enlace dinámico*). El BDE es un CLI [Call-Level Interface] que hace de intermediario entre las aplicaciones y las bases de datos a las que se accede, de forma que éstas sean independientes de la base de datos que se utilice. Para que la aplicación funcione debe instalarse este motor de base de datos.

5.3.4 Compatibilidad de protocolos (MODBUS TCP, RTU y DUCBUS)

Las características particulares y específicas de cada uno de los protocolos se encuentran en el *Anexo 1. Descripción del protocolo DUCBUS de DUCATI.* y *Anexo 2. Descripción del protocolo MODBUS RTU y MODBUS TCP.*, no se realizará una descripción de cada uno de los protocolos, sino mostrar como se logra la compatibilidad entre ellos. La trama genérica de transacciones bajo protocolo MODBUS TCP tiene la estructura presentada en la Tabla 8.

Campos	Longitud	Descripción	Cliente	Servidor
Identificador de transacción.	2 Bytes	Identificación de una solicitud o respuesta.	Inicializado por el cliente	Copiado por el servidor para emitir una respuesta
Protocolo identificador	2 Bytes	0 para Modbus	Inicializado por el cliente	Copiado por el servidor para emitir una respuesta
Longitud	2 Bytes	Numero de bytes que siguen al mensaje	Inicializado por el cliente (solicitud)	Inicializado por el servidor (respuesta)
Identificación de Unidad	1 Bytes	Identificación del dispositivo conectado en la línea serial u otro bus.	Inicializado por el cliente	Copiado por el servidor para emitir la respuesta.
Datos	N Bytes	Función y datos	Inicializado por el cliente	Copiado por el servidor para realizar proceso

Tabla 8 Estructura de una trama MODBUS TCP.

Por ser la trama en protocolo MODBUS TCP la que inicia la comunicación y contiene la información de la solicitud, es a partir de ella que comienza la descripción. El protocolo MODBUS TCP es una consecuencia directa de la introducción de redes de datos de tipo Internet a los buses de campo, lo que supone que las modalidades MODBUS RTU y MODBUS ASCII se introdujeron mucho tiempo antes. MODBUS RTU obtuvo mayor éxito que su similar, demostrando mayores ventajas en los buses de campo y su difusión es muy alta. Debido a que los buses de campo más difundidos fueron sobre MODBUS RTU deciden integrar a esta misma trama un encabezado de transporte y quitar de ella el chequeo de error (CRC) para conformar la nueva trama. Recordemos que las redes TCP/IP son consideradas de alta efectividad en la transmisión confiable de paquetes, por lo que la inclusión de campos de

chequeo de error dentro de la trama era redundante. En la Figura 28 se muestra como la trama MODBUS RTU genérica esta contenida dentro de la trama MODBUS TCP genérica.

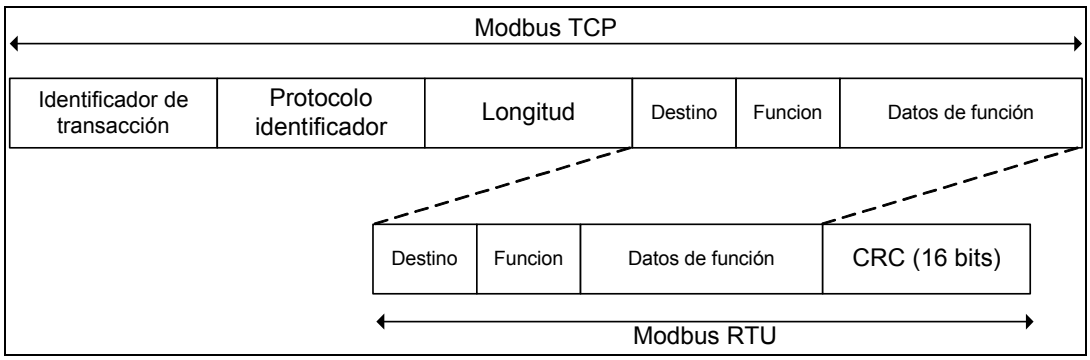


Figura 28 Comparación de tramas MODBUS TCP y RTU.

Partiendo del hecho de que la información de encabezado solo es de importancia para la transacción entre el cliente y el servidor, y que no contribuye con información necesaria para el dispositivo de campo, entonces se extrae el encabezado y se calcula el CRC del resto de la trama, luego la información ya puede ser enviada al dispositivo de campo. Al recibir respuesta sigue el mismo comportamiento, se le extrae el CRC realizando verificación de errores, y se le anexa el mismo encabezado, solo variando el campo "Longitud" que depende de los bytes transportados. De esta manera se hace posible la transacción entre Cliente – Servidor – Dispositivo. En caso de falla en la comunicación el Servidor procesará el código de error que corresponda y anexará estos datos al encabezado completando así la transacción.

Para el caso del protocolo DUCBUS tenemos que es un protocolo de tipo ASCII, la trama genérica consta de un encabezado fijo, dirección del dispositivo (ASCII), variable a solicitar, y un chequeo de error de tipo suma de verificación (*checksum*), lo que muestra que la información contenida en la trama MODBUS le es suficiente. Haciendo un análisis de funciones, MODBUS se caracteriza por realizar lecturas de grupos de datos, DUCBUS solo puede leer una variable a la vez. En consecuencia se hace necesario diseñar una rutina que partiendo de los datos MODBUS realice sucesivas solicitudes en DUCBUS y convierta los resultados ASCII obtenidos, en datos binarios para conformar la trama MODBUS RTU de respuesta, una vez realizada la conversión se le daría un tratamiento similar que a una respuesta MODBUS RTU.

La rutina diseñada esta contenida dentro del modulo "Polling" (ver Figura 23). Una vez que este modulo independiente identifica que el dispositivo de campo funciona con DUCBUS (base de datos de trabajo) pone en funcionamiento esta rutina y procesa los datos de manera completa. El detalle de conversión y funcionamiento se puede visualizar en el código fuente en lenguaje C++ del Anexo 4 Código fuente estructurado (Versión CD).

Por último es de importancia mencionar que los dispositivos de campo con protocolo MODBUS RTU responden a la solicitud en una sola transacción, en el caso de DUCBUS el número de transacciones necesarias dependerá del número de variables solicitadas, por lo que el tiempo de respuesta visto desde el cliente será mayor para estos casos. A una velocidad de 9600 baud una transacción MODBUS RTU en el peor caso (máxima solicitud de datos) tardaría, sin incluir retrasos en la línea de transmisión, aproximadamente 0,6 segundos; utilizando el mismo patrón, una transacción sobre un dispositivo DUCBUS tardaría 3,8 segundos, lo que equivale aproximadamente a 6 veces mas que MODBUS RTU. Este retardo

puede o no ser importante, dependiendo de la aplicación para las que se requiera los datos. Adicionalmente las sucesivas interrogaciones introducen una probabilidad de errores de transmisión superior, por lo que la rutina se diseño con la capacidad de detectar una transacción errónea y repetir una vez más esta única transacción a fin de asegurar una tasa menor de falla.

5.3.5 Funcionamiento del Gateway.

Para explicar el funcionamiento recordemos el modelo mostrado en la Figura 23, El mensaje es el paquete TCP que llega a la aplicación después de establecido el enlace, este contiene una solicitud en MODBUS TCP, la aplicación principal acepta esta solicitud y descansando sobre su archivo de soporte, supervisa la trama que entra a fin de corroborar que no existan errores; una vez revisado esto, la solicitud es enviada a un objeto de tipo "Polling" (Objeto para interrogar) el cual acepta los datos, quita el encabezado y opera la trama MODBUS. Primero emite una petición al administrador de base de datos, que suministra la información del dispositivo en cuestión (se busca por dirección del dispositivo), una vez reconocido los parámetros decide cual protocolo de transmisión usará, opera la trama de la manera mas conveniente y solicita los servicios seriales a través de otro objeto de datos (servicios seriales), interroga este servicio al dispositivo y suministra los datos recogidos, el objeto de tipo "Polling" recalcula el largo de la trama y agrega la información de encabezado, después de revisar que no existan errores (chequeo del CRC o Checksum). Una vez realizada la operación acciona un evento del programa principal en el que se empaqueta la respuesta en TCP y se envía al solicitante. En caso de existir un error en alguno de estos procesos se anexará a la solicitud un código de error MODBUS para notificar al cliente el estado del error.

En el modo de configuración el sistema se inhibe de realizar solicitudes al medio serial, las solicitudes se responden directamente con un código MODBUS de ocupado, los cambios realizados se almacenan tanto en la base de datos de trabajo como en el documento de configuración "Remotas.dat". Si se escoge volver a operar el sistema retomará sus funciones y las transacciones seguirán su labor.

En el modo de exploración de dispositivos, el proceso es similar a una solicitud genérica. El módulo principal crea un objeto de tipo "Explorar" y este a su vez crea objetos de tipo "Polling" (Interrogación) con una línea de datos, estos objetos reconocen que la llamada la hace el módulo explorar y solicita datos de la base de datos de exploración, si se detecta algún dispositivo esta información pasa a la base de datos de trabajo y se notifica al modulo principal. El objeto explorar captura al objeto "Polling" al finalizar su tarea y crea un nuevo objeto pero con una línea de datos diferente, así continua este proceso hasta que finalicen los datos. El módulo principal es notificado del avance de este proceso y puede ser suspendido, reanudado o cancelado por el principal.

Es importante señalar que el sistema no opera peticiones concurrentes, es decir el sistema acepta una solicitud y la procesa, si durante este tiempo el sistema cliente realiza otra petición el sistema responderá con un código de ocupado, este funcionamiento solo se justifica porque no existe una estandarización que indique un número fijo de concurrencia, para este protocolo (MODBUS TCP), y si se aceptará concurrencia se podrían crear colas y choques que el sistema no pudiera manejar. Recordemos en este punto que el sistema accede a un bus serial RS-485 que introduce tiempos largos con respecto a los de procesos de transmisión en TCP/IP. Si el sistema reconoce que la solicitud va hacia un puerto no ocupado esta es aceptada

y procesada, pero si el sistema detecta el puerto ocupado rechaza la solicitud con un código de ocupado sin dejar solicitudes en espera.

5.3.6 Pruebas y puesta en funcionamiento.

Para realizar pruebas se instaló una versión funcional tipo "Demostración" de iFIX, en una computadora situada geográficamente distante del Gateway (computadora donde funciona la aplicación tipo Gateway), dentro de la misma red Ethernet de la Universidad Central de Venezuela. Se instaló un nodo SCADA bajo iFIX de Intellution Dynamics y el driver MBE (MODBUS sobre Ethernet) también de iFIX, se programó dicho driver y se realizo la programación de una sencilla HMI (*Human Machine Interface*), se puso en operación de monitoreo constante y se obtuvo un funcionamiento adecuado, se realizaron pruebas de mal funcionamiento serial (desconexión de dispositivos y desconexión de los cables del bus serial) y de red (desconexión de los cables de la red), respondiendo el conjunto de manera consistente. Una vez realizada las pruebas se realizó una demostración de funcionamiento bajo supervisión de los departamentos de Potencia y Electrónica de la Escuela de Ingeniería Eléctrica de la Universidad Central de Venezuela, como parte de avance de proyecto del presente trabajo. Con base a lo anterior se comprobó que el conjunto iFIX – Gateway – Dispositivos de campo, mostraron un correcto funcionamiento.

5.4 Etapa 4: Desarrollo de Gateway (DNP3).

5.4.1 Descripción general del modelo.

Para la última etapa de este trabajo ya se contaba con el desarrollo del Gateway bajo MODBUS TCP, por lo que el trabajo se concentró en el desarrollo de transacciones bajo protocolo DNP3 para la integración en un sistema Cliente – Servidor bajo este protocolo. La etapa anterior manejó conceptos de comunicaciones en Internet y seriales; la conversión del protocolo MODBUS TCP a MODBUS RTU y DUCBUS sirve como punto de partida a fin de aprovechar los módulos desarrollados y no tener la necesidad de desarrollar nuevos. Para este fin el desarrollo se concentra en convertir el protocolo DNP3 (Protocolo Cliente de red) a MODBUS RTU, que el desarrollo anterior maneja. Recordemos que el sistema anterior es capaz de interrogar dispositivos de campo en MODBUS RTU o DUCBUS, por lo que bastará con desarrollar un modo de conversión de protocolo de DNP3 a MODBUS RTU para alcanzar nuestro objetivo.

Para este desarrollo se conservo el modelo anterior del Gateway bajo MODBUS TCP solo introduciendo algunas variantes que se pueden observar en la Figura 29.

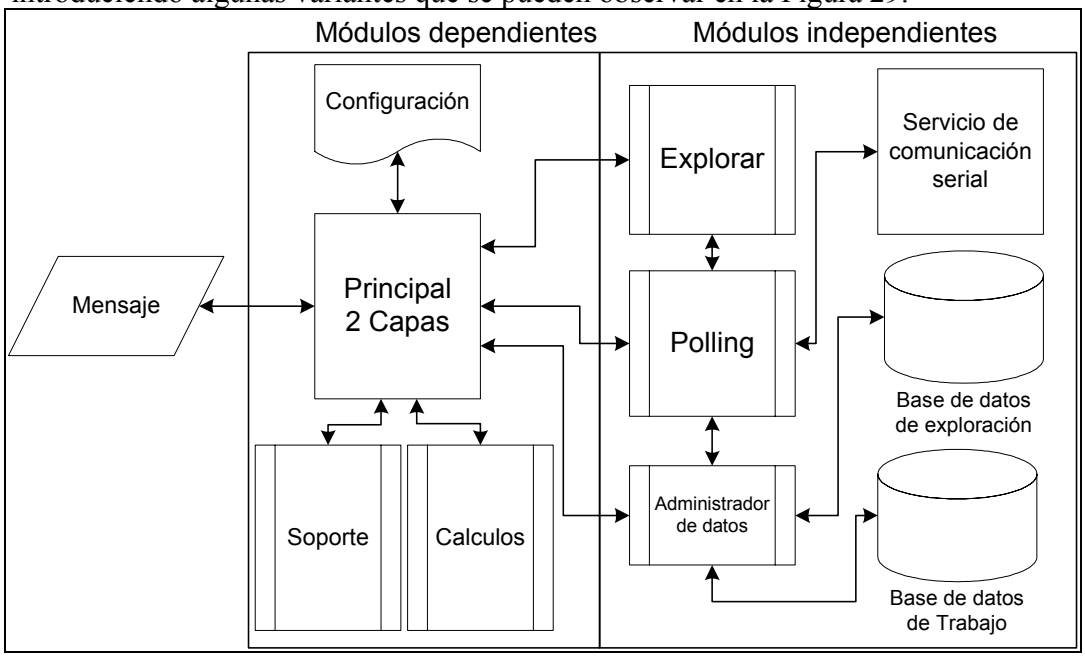


Figura 29 Modelo de interfaz tipo Gateway bajo DNP3.

Las variantes son básicamente dos; la primera consiste en la creación de un módulo nuevo llamado “Calculos”, donde se depositan nuevas funciones y procedimientos necesarios para trabajar bajo DNP3, este módulo solo prestará servicios al principal que apoya su funcionamiento en él. La segunda modificación consiste en la separación en servicios de dos capas del módulo principal. Por servicio de dos capas se entiende que el protocolo DNP3 es un protocolo encapado, es decir que está fundamentado en una capa de enlace que descansa sobre la capa de transporte de TCP/IP y una capa de aplicación que descansa sobre la capa de enlace. Dichas capas son independientes y tienen diferentes funciones. La capa de enlace se encarga de mantener los estados de transacción de solicitudes y fiabilidad de paquetes. La capa de

aplicación recibe solicitudes de la capa de enlace y es aquí donde se procesa la solicitud; una vez completada la tarea pasa la respuesta a la capa de enlace que se encargará de la transacción correspondiente.

5.4.2 Descripción de interfaz de usuario.

Para la interfaz de usuario se tomo el mismo modelo de la interfase tipo Gateway bajo MODBUS TCP; Esto debido a que las aplicaciones son similares solo cambia el protocolo de red empleado.

Como se muestra en la Figura 30 se incluyo una pantalla de programación de funciones MODBUS en los dispositivos de campo. Este sistema responde a la necesidad de que la interfase conozca con cuales de las funciones MODBUS operan dichos dispositivos. Existen varias funciones MODBUS que sirven para lectura y escritura de variables, dependiendo del dispositivo se habilita una u otra función, DNP3 realiza lecturas y escrituras dependiendo de los objetos de datos que solicite, por lo que seria imposible obtener información de la trama DNP3 acerca de la función MODBUS que corresponda.

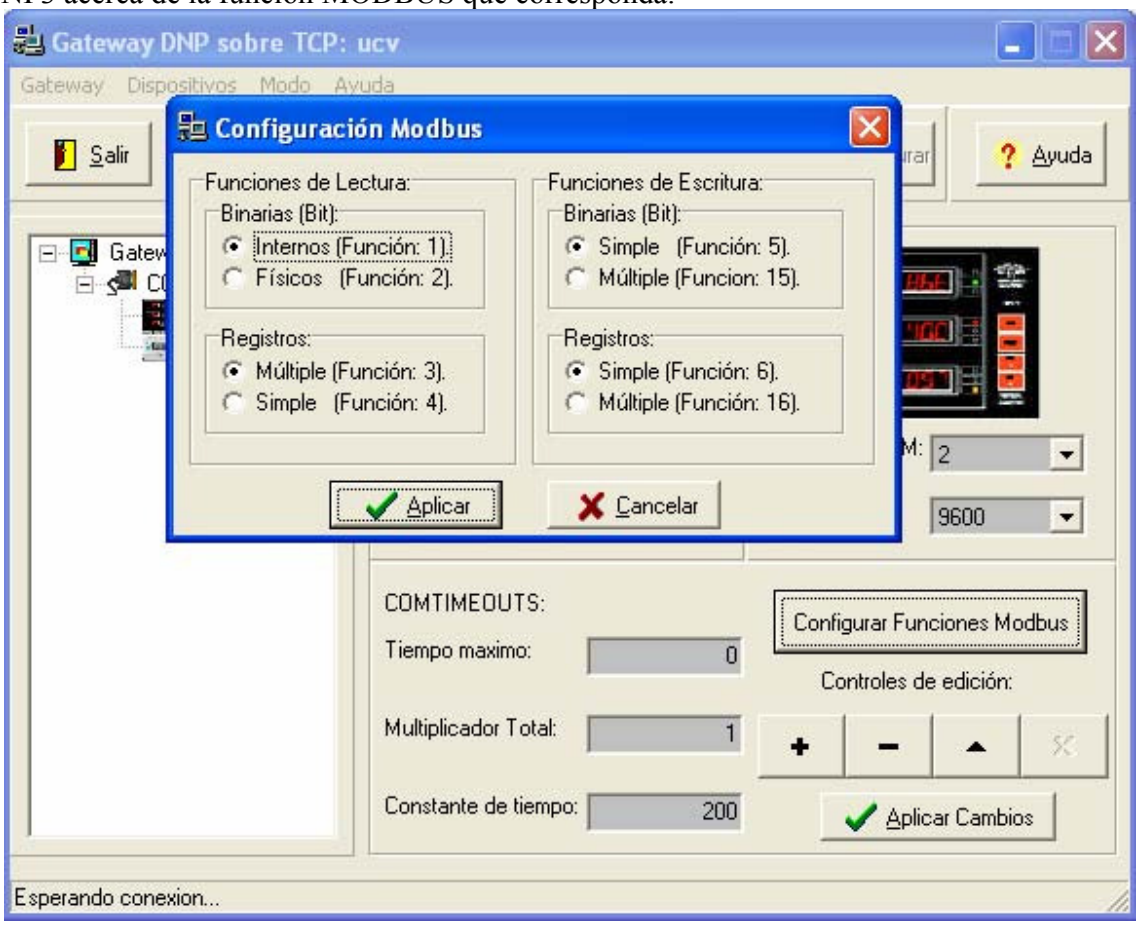


Figura 30 Interfaz de usuario para configuración de funciones MODBUS.

En DNP3 se establecen diferentes modos de comunicación a nivel de las capas de enlace y aplicación, existe un modo de transferencia que consiste en enviar un acuse de recibo por cada paquete de la capa de enlace recibido, este modo no se recomienda para la transferencia bajo TCP/IP ya que lo robusto de esta arquitectura sugiere una buena fiabilidad,

lo que recargaría el sistema de acuses de recibido inútiles, aun así, es solo una recomendación por lo que se habilita al sistema para ser capaz por si solo de responder con acuses de recibo solicitados, además es posible que la interfase se configure para solicitar acuses de recibo o no. De la misma manera funcionan unos acuses de recibido en la capa de aplicación por lo que se les da un tratamiento similar a los anteriores y la posibilidad de solicitarse o no. Esta propiedad fue incluida en el menú de configuración de la interfase como se muestra en la Figura 31. Nótese .que se colocaron casillas de verificación graficas, esto con el fin de permitir de manera rápida y sencilla habilitar o deshabilitar estos modos de comunicación. Es importante señalar, que actualmente empresas de software desarrollan aplicaciones tipo SCADA bajo DNP3; en estas soluciones se siguen las recomendaciones suministradas por el comité DNP3 (DNP.org); como es no utilizar acuses de recibo de la capa de enlace cuando se utilice TCP/IP como transporte, sin embargo, los diseños contemplan este modo de funcionamiento, por lo que se justifica la implementación de estos modos, a fin de contemplar un amplio rango de opciones de funcionamiento.

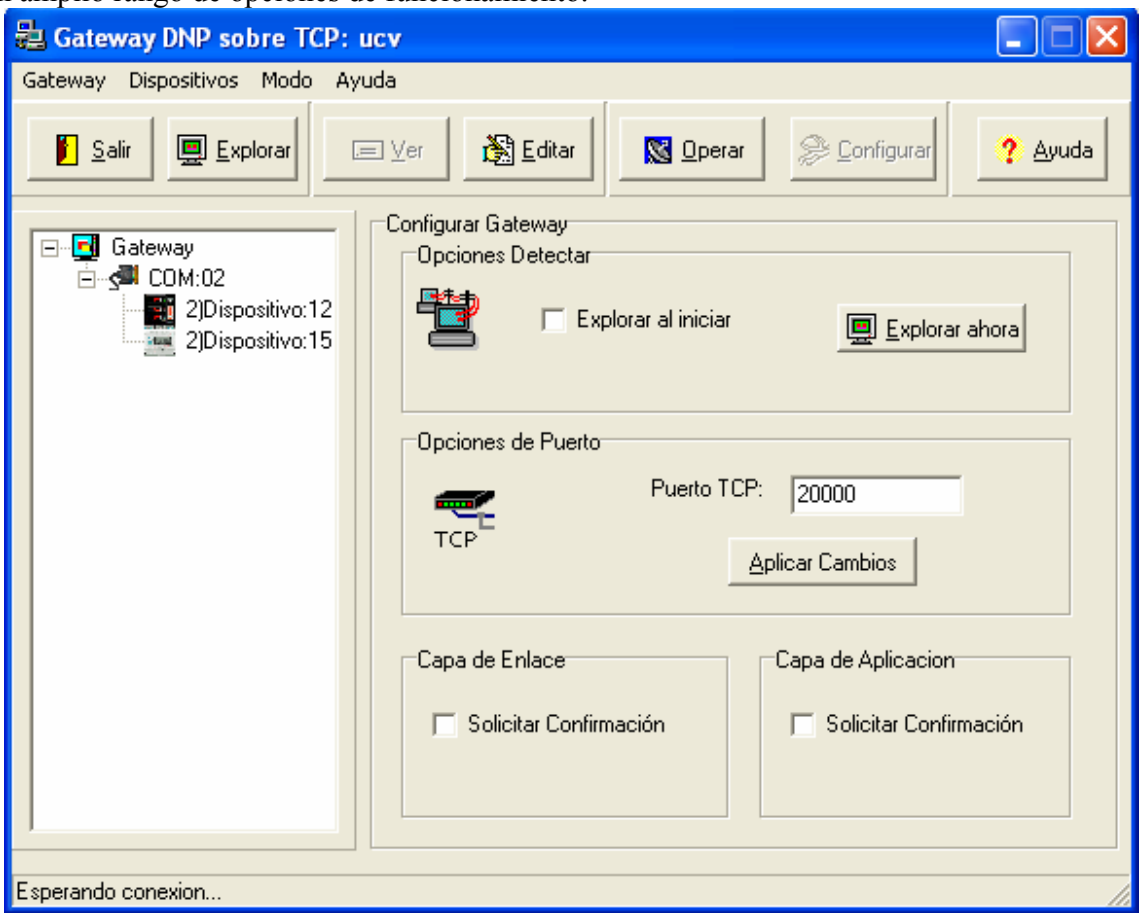


Figura 31 Interfaz de usuario para configuración de Gateway.

La interfaz de usuario para el caso DNP en los menús no señalados en esta parte son idénticos a los explicados en la sección 5.3.2 donde se especifica la interfaz bajo MODBUS TCP, por lo que bastará una lectura de esta parte para familiarizarse con el entorno.

5.4.3 Modelo de capas para DNP.

DNP 3,0 es un protocolo "encapado". Aún así, en lugar de asemejarse al modelo de 7 capas de la OSI (Open System Interconection - interconexión de sistemas abiertos), DNP3 se adhiere a un estándar simplificado de 3 capas propuesto por el IEC (International Electrotechnical Commission - Comisión internacional de Electrotecnia) para implementaciones más básicas. El IEC llama a esto Enhanced Performance Architecture, o EPA. (en realidad, DNP 3,0 agrega una cuarta capa, una capa de pseudo-transporte que permite la segmentación del mensaje).

5.4.3.1 Capa Física.

La capa física se refiere sobre todo a los medios físicos sobre los cuales se está comunicando el protocolo. Por ejemplo, maneja el estado del medio (limpio u ocupado), y la sincronización a través del medio (iniciando y parando). Más comúnmente, DNP se específica sobre una capa física serial simple tal como RS-232 o RS-485 usando medios físicos tales como fibra, radio o satélite. Los proyectos se orientan actualmente para implementar DNP sobre una capa física como Ethernet.

5.4.3.2 Capa de Transmisión de Datos.

La capa de transmisión de datos maneja la conexión lógica entre el remitente y el receptor de la información y pone a prueba las características de error del canal físico. DNP3 logra esto comenzando cada trama de transmisión de datos con una cabecera, e insertando un CRC de 16 bits cada 16 bytes de la trama. Una trama es una porción de un mensaje completo comunicado sobre la capa física. La medida máxima de una trama de transmisión de datos es 256 bytes. Cada trama tiene una dirección fuente de 16 bits y una dirección de destino también de 16 bits, las que pueden ser una dirección de difusión o broadcast (0xffff). La información del direccionamiento, junto con un código de inicio de 16 bits, la longitud del trama, y un byte de control de transmisión de datos se hallan en la cabecera (10 bytes) de transmisión de datos.
El byte de control de transmisión de datos indica el propósito de la trama de transmisión de datos, y el estado de la conexión lógica. Los valores posibles del byte de control de transmisión de datos son: ACK (*acknowledgment*), NACK (*no acknowledgment*), la conexión necesita inicializar (*reset link status*), datos de usuario sin confirmación (*unconfirmed user data*), datos de usuario con confirmación (*confirmed user data*), solicitud de estado de conexión (*link status*), y contestación de estado de conexión (*test link status*). Cuando se solicita una confirmación de transmisión de datos, el receptor debe responder con una trama ACK de transmisión de datos si el mismo es recibido y pasa los controles del CRC. Si una confirmación de la transmisión de datos no se solicita, no se requiere ninguna respuesta de la transmisión de datos.

5.4.3.3 Capa de Pseudo-Transporte.

La capa de pseudo-transporte divide mensajes de la capa de aplicación en múltiples tramas de transmisión de datos. Para cada trama, inserta un código de función de 1 byte que indica si la trama de transmisión de datos es la primera trama del mensaje, la última trama del mensaje, o ambos (para mensajes simples). El código de función también incluye un número de secuencia de la trama que se incrementa con cada uno y permite que la capa de transporte recipiente detecte tramas perdidas.

5.4.3.4 Capa de Aplicación.

La capa de aplicación responde a mensajes completos recibidos (y arribados de la capa de transporte), y construye los mensajes basados en la necesidad o la disponibilidad de los datos del usuario. Una vez que se construyan los mensajes, se pasan a la capa de pseudo-transporte donde se dividen en segmentos y se pasan a la capa de transmisión de datos y eventualmente comunicados sobre la capa física.

Cuando los datos a transmitir son demasiado grandes para un solo mensaje de la capa de aplicación, se pueden construir mensajes múltiples de la capa de aplicación y transmitirlos secuencialmente. Sin embargo, cada mensaje es un mensaje independiente de la capa de aplicación; existe una indicación de su asociación con el siguiente, en todos excepto en el último. Debido a esta posible fragmentación de los datos de aplicación, cada mensaje es referido como un fragmento, y un mensaje por ende puede ser un mensaje de un solo fragmento o un mensaje de múltiples fragmentos.

Los fragmentos de la capa de aplicación de las estaciones Master de DNP son típicamente solicitudes de operaciones sobre objetos de datos, y los fragmentos de la capa de aplicación de estaciones esclavas de DNP son típicamente respuestas a esas peticiones. Una estación esclava DNP puede también transmitir un mensaje sin una petición (una respuesta no solicitada).

Como en la capa de transmisión de datos, los fragmentos de la capa de aplicación se pueden enviar con una solicitud de confirmación. Una confirmación de la capa de aplicación indica que un mensaje no sólo ha sido recibido, sino también analizado sin error. (por otra parte, una confirmación de la capa de transmisión de datos, o ACK, indica solamente que se ha recibido la trama de la transmisión de datos y que pasó los controles de error del CRC.)

Cada fragmento de la capa de aplicación comienza con una cabecera seguida por una o más combinaciones de objetos de datos y objetos cabecera. La cabecera de la capa de aplicación contiene un código de control de la aplicación y un código de función de la aplicación. El código de control de la aplicación contiene una indicación de si el fragmento es parte de un mensaje multi-fragmento, una indicación de si una confirmación de la capa de aplicación es requerida por el fragmento, una indicación de si el fragmento fue no solicitado, y contiene un número de secuencia de la capa de aplicación. Este número de secuencia de la capa de aplicación permite que la capa de aplicación receptora detecte los fragmentos que están fuera de secuencia, o los fragmentos perdidos. El código de función de cabecera de la capa de aplicación indica el propósito, o la operación solicitada, del mensaje. A la par que DNP3 permite múltiples tipos de datos dentro de un único mensaje, permite una única operación sobre los tipos de datos dentro del mismo. Algunos ejemplos de códigos de función son: Confirmar (para las confirmaciones de la capa de aplicación), leer y escribir, seleccionar y operar, congelar y limpiar (para los contadores), reiniciar, permitir e invalidar mensajes no solicitados, y asignar la clase. El código de función de cabecera de la capa de aplicación se aplica a todas las cabeceras del objeto, y por lo tanto a todos los datos dentro del fragmento del mensaje.

5.4.4 Modelo orientado a objetos de datos de DNP3.

DNP3 es un protocolo para SCADA que fue concebido como la necesidad de interactuar con diferentes tipos de datos existentes, actualmente se discute y se anexan al protocolo nuevos tipos de datos. Cuando se tienen datos, se tiene puramente un valor numérico que carece de significado físico, después de interactuar con el valor, asignarle formato y unidades entonces toma el significado necesario para expresar algo. Con esto podemos decir

que un dato tiene significado con un marco de referencia, los protocolos de comunicación como MODBUS RTU, MODBUS TCP, DUCBUS y otros intercambian mensajes con datos que previamente tanto el maestro como el esclavo conocen, es decir, las transacciones pueden ser datos de 16 bits, enteros, sin signo y pertenecer a un contador. Esta información reside en el maestro y es su labor organizar la base de datos de modo que la información este disponible. El protocolo DNP3 trata de unificar esta información de modo que el dispositivo de campo sea capaz de servir datos con diferentes opciones de referencia. Por tanto el protocolo organiza los datos en objetos de datos, que son unidades que encapsulan los datos, le asignan su formato, tamaño y significado de manera de hacer más eficaz la tarea de organización del maestro y poder realizar transacciones de datos de diferentes tipos. Esta propiedad da como resultado que cada objeto de datos es una unidad independiente que no necesita asignación de propiedades, están listas para descargarse en la base de datos. Esta propiedad otorga a este protocolo la capacidad de transferir datos de 1, 8, 16 o 32 bits, enteros con signo o sin él, con estatus, con significado físico o no (caso de contadores) y otras propiedades que hacen del dispositivo de campo una unidad mas especializada. Para lograr este fin se encapsulan los datos con un encabezado de objeto que provee la información sobre el dato.

El encabezado de objeto de un mensaje especifica los datos del objetos(o E/S) que estarán simultáneamente contenidos en el mensaje o serán usados para responder el mensaje (ver Figura 32). El formato del encabezado de objeto es idéntico para una solicitud y su respuesta, pero la interpretación del encabezado es dependiente en cuanto si es una solicitud o una respuesta, y cual código de función acompaña al encabezado.

Objeto (2)	Calificador (1)	Rango (0,1,2,..,8)	Datos (N)

Figura 32 Formato de de objeto.

- **Objeto:** Especifica el grupo y la variación del objeto que sigue al encabezado. Este campo es de dos (2) octetos. El campo de objeto identifica únicamente el tipo o clase del objeto que sigue a la estructura del objeto de dato (*Registro de 16 bits, entero sin signo*).
- **Calificador:** Especifica la medida del campo "Rango". Esta compuesto por un (1) octeto. El calificador especifica como el campo de rango será interpretado.
- **Rango:** Indica la cantidad de objetos, arrancando y terminando con índices e identificadores para el objeto en cuestión. Este campo únicamente identifica al objeto en cuestión (*Posición de memoria, índice, etc.*). El campo de Rango puede no ser presentado si el calificador especifica que este no existe. El tamaño de este campo va desde 0 hasta 8 octetos.

5.4.5 Compatibilidad de funciones entre protocolos (DNP3 y MODBUS RTU).

El protocolo DNP3 contempla diferentes tipos de funciones a ejecutar que se dividen en 7 grupos. Los códigos de transferencia (confirmar, lectura y escritura), los códigos de control (seleccionar, operar, operación directa, etc.), códigos de congelación (congela y limpia, congela sin limpiar, etc.), códigos de control de aplicaciones (arrancar, detener, inicializar, etc.), códigos de configuración (guardar configuración, habilitar, deshabilitar, etc.), códigos de sincronización de tiempo (retardo de medidas), códigos de respuesta

(confirmación, respuesta, mensajes no solicitados, etc.) y aun existe la posibilidad de incluir grupos y funciones no definidas. Por su parte MODBUS RTU solo permite funciones del tipo transferencia, y entre ellas permite lectura y escritura. MODBUS RTU utiliza diferentes códigos de función dependiendo si la lectura o escritura es de variables o valores binarios, siendo posible solicitar operaciones de lectura y escritura solamente de variables de 16 bits o binarias. Es importante señalar que esto no significa que no se puedan obtener datos de 32 bits ya que en este caso se asigna la parte baja a una posición y la parte alta a otra (trabajo del maestro sobre la base de datos). Debido a esta limitada posibilidad la interfase solo podrá manejar objetos de datos de 16 bits y binarios. Existe la posibilidad de que el dato sea un contador, un registro analógico simple, un registro analógico congelado (detenido), una entrada binaria o una salida binaria; por lo que el maestro DNP decidirá la solicitud, la interfase hará la referencia a la posición de memoria y responderá con los datos, pero será finalmente en la base de datos donde se realizará la correcta interpretación de la variable. La interfase no reconoce el tipo de datos solicitados, como no lo hará el dispositivo de campo, la aplicación maestra recibirá el dato empaquetado de la manera que lo solicitó y es allí donde la correcta interpretación juega un papel determinante en el SCADA. La interfase no requiere la programación de las propiedades de las variables, pero es capaz de empaquetar el dato tal cual fue solicitado por el maestro, lo contrario supondría comportamiento de micro maestra a la interfase que no es el objetivo previsto. En el caso de que la solicitud sea de un objeto diferente (registro de 32 bits), no se aceptará la solicitud y se entregará un código especial del protocolo que significa “Objeto no soportado o desconocido” tal cual lo haría una unidad terminal remota DNP3 que no contemple un tipo especifico de datos. No es posible procesar las otras funciones ya que MODBUS RTU no las soporta, quedando inhabilitadas entonces, las solicitudes serán contestadas con un código del protocolo que significa “Función desconocida o no soportada”. Los códigos calificadores tienen la finalidad de especificar unívocamente una posición de memoria, por lo que las posibilidades en DNP3 son variadas, algunos modos de dirección son incompatibles para MODBUS por lo que solo se aceptan algunos tipos de direccionamiento. Existe en DNP3 la posibilidad de transferir archivos, esto con el fin de transferir informes en los RTU que tengan habilitada esta función, por su parte MODBUS RTU posee una función de lectura de archivos que en fin parte de un concepto diferente (agrupamiento de memorias de datos), por tanto son incompatibles entre si. Las siguientes consideraciones describen entonces las opciones de la interfase.

- Las únicas funciones habilitadas son las de Lectura, Escritura y de Confirmación de paquetes.
- Los objetos de datos soportados solo contemplan variables binarias o registros de 16 bits. Pudiéndose solicitar estos bajo cualquier formato compatible de DNP3 para variables de 16 bits sin estatus.
- Los calificadores posibles soportados determinan el direccionamiento adecuado eliminando la posibilidad de errores.
- Los mensajes no solicitados quedan inhabilitados ya que esto supondría tareas de micro maestra a la interfase intérprete, esto debido a que MODBUS no contempla este tipo de mensajes ni la prioridad de una u otra variable.
- En DNP3 existen cuatro tipos de clases según su prioridad, la interfase solo soporta la “Clase 0” o registros de tipo estático sin prioridad, por lo que se inhabilita la asignación de clases.

- Es necesario que la interfase pueda decidir que funciones MODBUS utilizar dependiendo del tipo de dispositivo de campo por lo que es necesario programar estas opciones.
- Queda inhabilitada la opción de transferencia de archivos.

Las opciones escogidas constituyen la mayor compatibilidad entre protocolos, lo que haría parecer que no se justifica el desarrollo de esta interfase. Esto es una visión herrada, debido a que los protocolos son fundamentados en la búsqueda de generalidad, pero los dispositivos de campo (RTU) no se diseñan buscando generalidad sino mas bien especificidad. Es decir al momento de escoger un RTU se intenta que este cumpla la tarea específica para la cual es asignado, sin importar que este no soporte funciones de su protocolo que no utilizará en sí. Es por esto que visto de otra forma, el sistema Maestro DNP3 vería al sistema Gateway–RTU (MODBUS RTU y DUCBUS), como un solo conjunto, formado por un RTU inteligente, con protocolo DNP3 sobre TCP con las posibilidades que se muestran en la Tabla 9.

Descripción	**Sistemas Operativos**	**Protocolos**	
		Maestro	**Dispositivos**
Gateway DNP	Windows 98, NT, 2000 o XP	DNP 3 Sobre TCP	MODBUS RTU y DUCBUS

Rango de direcciones de RTU	**Rango de memorias de RTU**	**Puertos disponibles(DNP)**	**Conexiones**	
			Conexión a Internet	Conexión a medio serial
0 – 255 (00h – FFh)	0 - 65535	1-65535	Cualquiera	Norma RS-232

Funciones soportadas (todas) -Capa de Enlace de datos		
Código de Función	Nombre	Descripción
0 –Solicitud	Reset link states	Petición de inicialización.
2 - Solicitud	Test link states	Estado de la conexión.
3 – Solicitud	Confirmed user data	Datos usuario con confirmación.
4 – Solicitud	Unconfirmed user data	Datos usuario sin confirmación
9 – Solicitud	Request link status	Solicitud de estado de enlace.
0 – Respuesta	Acknowledgment	Comando reconocido
1 – Respuesta	No Acknowledgment	Comando no reconocido
11 – Respuesta	Link status	Estado del enlace.
15 – Respuesta	Not supported	Función no soportada.

Funciones soportadas -Capa de Aplicación		
Código de Función	Nombre	Descripción
0 - Solicitud	Confirm	Solicitud de Confirmación
1 – Solicitud	Read	Solicitud de Lectura
2 – Solicitud	Write	Solicitud de Escritura
0 – Respuesta	Confirm	Confirmación
81 – Respuesta	Response	Respuesta

Objetos soportados				
Descripción	Grupo	Variación	Calificadores	Rango
Entradas Binarias	01	01	0, 1, 3, 4, 23, 24, 25, 39, 40, 41	Todos los posibles con los calificadores
Salidas Binarias	10	01		
Contador de 16 bits	20	06		
Contador de 16 bits Congelado	21	10		
Registro de 16 bits	30	04		
Registro de 16 bits congelado	31	06		

Confirmación de enlace	Habilitada	Confirmación de aplicación	Habilitada	Transferencia de archivos	Inhabilitada	Mensajes no solicitados	Inhabilitada

Tabla 9 Opciones de Gateway para DNP 3 Sobre TCP.

5.4.6 Funcionamiento.

Para estudiar el funcionamiento recordemos la Figura 29 donde se recoge el modelo en cuestión. Como se había mencionado, la funcionalidad de la interfase tipo Gateway bajo DNP3 descansa su desarrollo sobre un modelo ya estudiado en el capitulo 5.3(Etapa 3: Desarrollo de Gateway (Modbus TCP).) por lo que solo se explicará la parte funcional del modulo principal, que es la única que cambia. Los servicios del módulo "Polling", servicios de comunicación serial y modulo "Explorar" siguen funcionando de manera similar.

Para este caso el mensaje DNP3 llega hasta el módulo principal, una vez recibido pasa a través de una rutina de chequeo de la capa de enlace, es allí donde se comprueba el CRC de 16 bits donde corresponde y se lleva a cabo la construcción del mensaje de la capa de aplicación si el mensaje recibido esta fragmentado. La comunicación comienza con una inicialización de la capa de enlace (*Reset link status*), posteriormente el maestro realiza la solicitud, esta llega a través de la capa física y es la capa de transporte (TCP) la que entrega el paquete. La aplicación reconoce la forma en que son entregados los datos de la solicitud, emitiendo automáticamente mensajes de reconocimiento si así es requerido por el código de función de enlace del maestro, una vez procesado un fragmento la aplicación decide si espera otro fragmento (caso de fragmentación) o pasa este mensaje a la capa de aplicación. Una vez completo, el mensaje pasa al modulo principal quien se inhibe de recibir mas solicitudes (modo de funcionamiento de DNP3) y entra en una rutina que secuencialmente construye los objetos de datos solicitados, esta ordena la información proveniente de los servicios de interrogación en MODBUS RTU y DUCBUS, empaquetando cada objeto de datos en el mismo orden en que fue solicitado. Si no se registran errores la solicitud esta completa y es pasada a otra rutina, dentro del modulo principal que se encargará de ir fragmentando de ser necesario y comunicando cada fragmento en el modo de comunicación seleccionado. Una vez completada la transacción el sistema esta listo para una nueva solicitud. Dentro del mensaje de respuesta existen dos octetos (2 bytes) que contienen la información de error, si un error sucede el sistema cancela la solicitud y levanta la bandera correspondiente al error en cuestión a fin de notificar la condición de error; así el maestro recibe esta condición y el sistema tomará las acciones que correspondan. Si una condición de falla ocurre en la red TCP/IP la comunicación debe ser establecida con una inicialización (reset link status) a fin de restablecer la comunicación a nivel de enlace de datos en DNP.

Adicionalmente es importante comentar que en DNP3 no se contemplan solicitudes concurrentes, por lo que el sistema responderá una solicitud a la vez. Sin olvidar que las solicitudes son los mensajes de la capa de aplicación y que la fragmentación en la capa de enlace esta contemplada, por lo que se puede observar transacciones sucesivas entre las capas de enlace de los sistemas en comunicación.

5.4.7 Pruebas y resultados.

El desarrollo de una interfaz tipo Gateway bajo DNP3 sobre TCP, quedaría suficientemente fundamentado, si se implementan pruebas de funcionamiento en un SCADA bajo este protocolo. La Universidad Central de Venezuela tiene prevista una futura implementación de un nodo SCADA bajo DNP3, razón que justifica este desarrollo. Sin embargo no existe al momento un sistema alternativo para realizar dichas pruebas. Por esta razón se realizaron las gestiones correspondientes con el foro internacional de DNP, el cual de manera receptiva otorgo una licencia gratuita provisional de 6 meses, para obtener la documentación correspondiente al protocolo de comunicación y sus definiciones. Al mismo

tiempo se comunicó la necesidad de realizar pruebas al software en desarrollo, partiendo de la carencia de recursos técnicos para estas. A esta razón el foro internacional para DNP sugirió la idea de solicitar ayuda a una empresa internacional proveedora de software para este protocolo, la cual otorgó una licencia gratuita provisional de 45 días de un software denominado "Test Harness" (ver capitulo 4.3.2). Este software funciona como maestro y esclavo para varios protocolos, incluyendo DNP3. Tiene un manual de ayuda para construir peticiones genéricas y especificas a partir de una ventana de comandos, procesa dichas solicitudes y muestra los resultados en un archivo de texto, dichos resultados se encuentran en el Anexo 5. Pruebas en DNP (Versión CD)., como sustento a la parte práctica del desarrollo. A fin de lograr la comprensión de estos archivos explicaremos de manera detallada uno de ellos, los otros archivos consisten de diferentes pruebas que con la documentación del protocolo y alguna experiencia al respecto pueden comprenderse.

El maestro DNP es el software Test Harnnes y el esclavo el conjunto Gateway – dispositivos de campo. La prueba realizada consta de una solicitud de lectura de datos de 30 registros de 16 bits cada uno. Antes de realizar la solicitud se debe habilitar un canal, donde se debe especificar que funcionará como maestro, la dirección IP del Gateway, el puerto de comunicaciones a usar, y especificar que DNP usara TCP como transporte. Después hay que habilitar la sesión para lo que debe iniciarse una sesión especificando la dirección del dispositivo a interrogar, la dirección local del dispositivo maestro y el canal habilitado para la sesión. Una vez habilitada la sesión se debe inicializar la capa de enlace (*reset link* status).

El resultado de la inicialización se muestra en el siguiente fragmento:

```
Communication Protocol Test Harness Version 2.0.19
Source Code Library Version 3.0.19
Build date Mon Aug 11 08:27:19 2003
Copyright 2000-2003

>>> mdnpopenchannel
>>> mdnpopensession channel 0
13:06:10.909: Prueba1: opened Port: 172.17.59.71:20000
>>> mdnplinkreset Session 0

13:07:40.317: <--- Prueba1    Primary Frame - Reset Link States
13:07:40.317:              LEN(5) DIR(1) PRM(1) FCV(0) FCB(0) DEST(12) SRC(1)
13:07:40.317:              05 64 05 c0 0c 00 01 00 33 b8
13:07:40.317: <... Prueba1    05 64 05 c0 0c 00 01 00 33 b8

13:07:40.448: ...> Prueba1    05
13:07:40.448: ...> Prueba1    64 05 00 01 00 0c 00 01 f1

13:07:40.448: ---> Prueba1    Secondary Frame - Acknowledge
13:07:40.448:              LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(12)
13:07:40.448:              05 64 05 00 01 00 0c 00 01 f1
```

Las primeras 4 líneas reflejan un encabezado de archivo que describe el nombre del producto, la versión, la fecha de construcción e información de la producción. Los símbolos y sus significados se muestran en la Tabla 10.

Símbolo	Descripción
>>>	Aplicación de comando.
<---	Trama saliente pasada a la capa física.
<…	Trama saliente pasada al medio físico.
…>	Trama entrante llegada del medio físico.
--->	Trama entrante a la capa de enlace.
<+++	Trama saliente pasada a la capa seudo transporte.
<= = =	Trama saliente pasada a la capa de transporte.
<~~~	Trama saliente pasada a la capa de enlace.
+++>	Trama entrante a la capa de usuario.
= = =>	Trama entrante a la capa de seudo transporte.
~~>	Trama entrante a la capa de transporte.

Tabla 10 Símbolos del software Test Harnnes.

Se puede apreciar que se aplica un comando de solicitud de inicio, este envía una solicitud de inicializar la capa de enlace (*reset link states*) y se recibe una respuesta dirigida a la capa de enlace de reconocimiento (*acknowledgment*). Hasta este punto se completa la inicialización, entonces ya es posible realizar una solicitud de datos. La dirección del maestro DNP asignada es la 01 y la dirección del esclavo es la 12 (oc hexadecimal). El siguiente fragmento de archivo muestra el intercambio en tramas DNP.

```
>>> mdnpgenrequest session 0 size 256 fc 1

>>> mdnpaddobjecthdr session 0 group 30m variation 4 qualifier 0 start 0 stop30

>>> mdnpsendrequest request 0

13:12:35.161: <+++ Prueba1    Insert request in queue: Generic Request

13:12:35.161: <=== Prueba1    Application Header, Read Request
13:12:35.161:              FIR(1) FIN(1) CON(0) UNS(0) SEQ# 1
13:12:35.161:              c1 01 1e 04 00 00 1e

13:12:35.161: <~~~ Prueba1    Transport Header
13:12:35.161:              FIR(1) FIN(1) SEQ# 0
13:12:35.161:              c0 c1 01 1e 04 00 00 1e

13:12:35.161: <--- Prueba1    Primary Frame - Unconfirmed User Data
13:12:35.161:              LEN(13) DIR(1) PRM(1) FCV(0) FCB(0) DEST(12) SRC(1)
13:12:35.161:              05 64 0d c4 0c 00 01 00 f7 ae
13:12:35.161:              c0 c1 01 1e 04 00 00 1e b9 be

13:12:35.161: <... Prueba1    05 64 0d c4 0c 00 01 00 f7 ae c0 c1 01 1e 04 00
13:12:35.161:              00 1e b9 be
```

Se genera una solicitud de lectura, grupo de objeto 30 (registro de 16 bits), calificador 0, dirección de arranque 0 y dirección de parada 30. Las transacciones muestran que la solicitud pasa desde la capa superior hasta la física y entra al medio de transmisión, las transacciones de respuesta siguen así.

```
13:12:35.392: ...> Prueba1    05

13:12:35.392: ...> Prueba1    64 05 40 01 00 0c 00 bb c1

13:12:35.392: ---> Prueba1    Primary Frame - Reset Link States
13:12:35.392:              LEN(5) DIR(0) PRM(1) FCV(0) FCB(0) DEST(1) SRC(12)
13:12:35.392:              05 64 05 40 01 00 0c 00 bb c1

13:12:35.392: <--- Prueba1    Secondary Frame - Acknowledge
13:12:35.392:              LEN(5) DIR(1) PRM(0) DFC(0) DEST(12) SRC(1)
13:12:35.392:              05 64 05 80 0c 00 01 00 89 88

13:12:35.392: <... Prueba1    05 64 05 80 0c 00 01 00 89 88

13:12:35.492: ...> Prueba1    05

13:12:35.492: ...> Prueba1    64 4d 44 01 00 0c 00 02 79

13:12:35.492: ...> Prueba1    c0 c1 81 00 00 1e 04 00 00 1e fe 71 0e 41 00 00
13:12:35.492:              14 18

13:12:35.492: ...> Prueba1    79 00 02 00 20 7d 00 00 df 49 00 00 32 15 00 00
13:12:35.492:              42 3b

13:12:35.492: ...> Prueba1    00 00 00 00 5f 00 00 00 7a 00 01 00 f4 f0 00 00
13:12:35.492:              43 24

13:12:35.492: ...> Prueba1    82 3a 00 00 49 10 00 00 00 00 00 00 60 00 00 00
13:12:35.492:              b7 8b

13:12:35.492: ...> Prueba1    7a 00 02 00 b5 31 00 00 b6 19

13:12:35.492: ---> Prueba1    Primary Frame - Unconfirmed User Data
13:12:35.492:              LEN(77) DIR(0) PRM(1) FCV(0) FCB(0) DEST(1) SRC(12)
13:12:35.492:              05 64 4d 44 01 00 0c 00 02 79
13:12:35.492:              c0 c1 81 00 00 1e 04 00 00 1e fe 71 0e 41 00 00 14 18
13:12:35.492:              79 00 02 00 20 7d 00 00 df 49 00 00 32 15 00 00 42 3b
13:12:35.492:              00 00 00 00 5f 00 00 00 7a 00 01 00 f4 f0 00 00 43 24
13:12:35.492:              82 3a 00 00 49 10 00 00 00 00 00 00 60 00 00 00 b7 8b
13:12:35.492:              7a 00 02 00 b5 31 00 00 b6 19

13:12:35.492: ~~~> Prueba1    Transport Header
13:12:35.492:              FIR(1) FIN(1) SEQ# 0
13:12:35.492:              c0 c1 81 00 00 1e 04 00 00 1e fe 71 0e 41 00 00
13:12:35.492:              79 00 02 00 20 7d 00 00 df 49 00 00 32 15 00 00
13:12:35.492:              00 00 00 00 5f 00 00 00 7a 00 01 00 f4 f0 00 00
13:12:35.492:              82 3a 00 00 49 10 00 00 00 00 00 00 60 00 00 00
13:12:35.492:              7a 00 02 00 b5 31 00 00

13:12:35.492: ===> Prueba1    Application Header, Response
13:12:35.492:              FIR(1) FIN(1) CON(0) UNS(0) SEQ# 1
13:12:35.492:              c1 81 00 00 1e 04 00 00 1e fe 71 0e 41 00 00 79
13:12:35.492:              00 02 00 20 7d 00 00 df 49 00 00 32 15 00 00 00
13:12:35.492:              00 00 00 5f 00 00 00 7a 00 01 00 f4 f0 00 00 82
13:12:35.492:              3a 00 00 49 10 00 00 00 00 00 00 60 00 00 00 7a
13:12:35.492:              00 02 00 b5 31 00 00
```

```
13:12:35.492: +++> Prueba1    Process response to request: Generic Request

13:12:35.492:          Object 30(Analog Input), variation 4, qualifier 0x00(8 Bit Start Stop)
13:12:35.502:              Analog Input 000000 = 29182, flags 0x01
13:12:35.502:              Analog Input 000001 = 16654, flags 0x01
13:12:35.502:              Analog Input 000002 = 0, flags 0x01
13:12:35.502:              Analog Input 000003 = 121, flags 0x01
13:12:35.502:              Analog Input 000004 = 2, flags 0x01
13:12:35.502:              Analog Input 000005 = 32032, flags 0x01
13:12:35.502:              Analog Input 000006 = 0, flags 0x01
13:12:35.502:              Analog Input 000007 = 18911, flags 0x01
13:12:35.502:              Analog Input 000008 = 0, flags 0x01
13:12:35.502:              Analog Input 000009 = 5426, flags 0x01
13:12:35.502:              Analog Input 000010 = 0, flags 0x01
13:12:35.502:              Analog Input 000011 = 0, flags 0x01
13:12:35.502:              Analog Input 000012 = 0, flags 0x01
13:12:35.502:              Analog Input 000013 = 95, flags 0x01
13:12:35.502:              Analog Input 000014 = 0, flags 0x01
13:12:35.502:              Analog Input 000015 = 122, flags 0x01
13:12:35.502:              Analog Input 000016 = 1, flags 0x01
13:12:35.502:              Analog Input 000017 = -3852, flags 0x01
13:12:35.502:              Analog Input 000018 = 0, flags 0x01
13:12:35.502:              Analog Input 000019 = 14978, flags 0x01
13:12:35.502:              Analog Input 000020 = 0, flags 0x01
13:12:35.502:              Analog Input 000021 = 4169, flags 0x01
13:12:35.502:              Analog Input 000022 = 0, flags 0x01
13:12:35.502:              Analog Input 000023 = 0, flags 0x01
13:12:35.502:              Analog Input 000024 = 0, flags 0x01
13:12:35.502:              Analog Input 000025 = 96, flags 0x01
13:12:35.502:              Analog Input 000026 = 0, flags 0x01
13:12:35.502:              Analog Input 000027 = 122, flags 0x01
13:12:35.502:              Analog Input 000028 = 2, flags 0x01
13:12:35.502:              Analog Input 000029 = 12725, flags 0x01
13:12:35.502:              Analog Input 000030 = 0, flags 0x01
```

El Gateway procesa la solicitud, inicializa la capa de enlace y una vez inicializada, envía la respuesta, esta pasa desde la capa física hasta la de usuario, donde finalmente se muestran los valores suministrados, en la capa física los datos suministrados entran en grupos de 16 bytes después del encabezado de 10 bytes inicial comprobando el CRC en cada grupo, como ejemplo las variables de memoria 2 y 3 contienen la parte alta y baja de una tensión de línea (121 Voltios). Se comprueba que las tramas se van desempaquetando hasta obtener el objeto de datos solicitado. En el Anexo 5. Pruebas en DNP (Versión CD). se muestran diferentes transacciones donde se comprueban modos de transmisión y fragmentación de mensajes.

6 CONCLUSIONES.

Se desarrolló e integró, a través de un computador personal, un convertidor de protocolo (GATEWAY) para la conversión hacia el protocolo MODBUS sobre TCP, de los protocolos DUCBUS – ASCII y MODBUS – RTU seriales. Ello permitirá integrar la red de campo compuesta por RTU(s) con protocolos propietarios, a un nodo de adquisición y control sobre iFIX. El buen funcionamiento del GATEWAY se realizó a través de pruebas de laboratorio empleando una estación maestra con el software iFIX versión 2.5, utilizando como enlace la red ETHERNET existente en la Escuela de Ingeniería Eléctrica.

Se desarrolló e integró, a través de un computador personal, un GATEWAY para la conversión hacia el protocolo DNP 3.0, sobre TCP, de los protocolos DUCBUS – ASCII y MODBUS – RTU seriales. Ello permitirá integrar una red de campo compuesta por varias RTU(s) con protocolo propietario, hacia un nodo de comunicación sobre DNP asociado al SCADA. El buen funcionamiento del GATEWAY se realizó a través de pruebas de laboratorio, empleando el software de prueba TEST HARNESS, versión de evaluación por tiempo limitado (trial versión)

Se delineó el circuito del GATEWAY para su implementación a través de una arquitectura integrada por componentes y/o circuitos integrados.

La tendencia de los sistemas SCADA de introducir los buses de campo en redes de datos de tipo Internet, indica hacia la división futura del hardware y del software de los dispositivos de campo. Lo que supone un desarrollo paralelo de sistemas de hardware robustos y sistemas de software eficientes como ha sucedido en las redes de datos. De allí que la ingeniería de desarrollo contemple los sistemas de manera independiente, pero sin desligar el concepto de independencia funcional de estos dispositivos cuya concepción inicial así lo dispuso.

El desarrollo de interfases tipo Gateway (pasarelas), constituye una gran herramienta para la integración de sistemas totalmente incompatibles. Es por esto que se puede decir que el estudio de factibilidad de estos sistemas, debe contemplarse inclusive en la etapa de desarrollo del SCADA, como es el caso particular de este desarrollo, a manera de considerar siempre la posibilidad de integración de sistemas y no la sustitución de equipos por tener características incompatibles.

La gran difusión y aceptación de los sistemas operativos tipo Windows de Microsoft, han provocado descontento a nivel mundial a otros desarrolladores y criticas por malfuncionamientos de estos. El presente desarrollo demuestra que las últimas versiones de estos sistemas operativos (Windows 2000, XP) han mejorado considerablemente sus prestaciones, conformando una plataforma robusta, versátil y eficiente para el desarrollo de aplicaciones de este tipo.

7 RECOMENDACIONES.

Los protocolos de comunicación poseen funciones propias, desarrolladas con fines específicos, con el fin de adaptarse a las necesidades de los sistemas que los originaron. Es por esto que en la concepción de un SCADA, es fundamental determinar las necesidades específicas del caso, para entonces decidir que protocolo de comunicaciones se adapta mejor a la aplicación. La integración de sistemas bajo protocolos diferentes es siempre posible, pero es importante señalar que aunque la integración soluciona el problema, disminuye considerablemente las capacidades generales del sistema final, que estará sujeto a funcionar ajustado a las compatibilidades y perderá la posibilidad de utilizar las opciones que queden consideradas incompatibles. Por lo que se infiere que la necesidad de integración de sistemas supone un estudio previo de los beneficios obtenidos contra las limitaciones producidas por esta.

Se recomienda que las interfaces tipo Gateway sean implementadas en las versiones del sistema operativo Windows 2000, ya que quedo demostrado que este sistema tiene características funcionales eficientes para este tipo de aplicaciones.

Para la instalación definitiva de los buses de campo bajo la norma RS-485 a través de la red telefónica preexistente en la CUC, se recomienda realizar un estudio completo de adaptación de estas líneas acorde a la especificación EIA/TIA RS-485-B (Norma oficial RS-485), ya que desde allí parte la concepción de confiabilidad del SCADA.

Para la instalación de los dispositivos de campo se recomienda utilizar los sistemas de protección recomendados por los fabricantes, haciendo énfasis en que es difícil sustituirles debido a lo cambiante de este mercado.

Se recomienda que de existir la posibilidad de adquirir nuevos dispositivos de campo, se tome en cuenta la ubicación y disponibilidad de conexión de este. Si se dispone un medio serial se recomienda adquirir equipos bajo protocolo MODBUS RTU a fin de poder incluirlos a una red bajo MODBUS TCP o DNP3 a través de los Gateway desarrollados. Y de disponerse de una conexión a la red Ethernet adquirir equipos bajo MODBUS TCP o DNP3 bajo TCP que no requieren la interfaz para interactuar con el SCADA final.

La interfase tipo GATEWAY desarrollada no se diseño para manejar solicitudes concurrentes, por lo que se recomienda no utilizar este modo de interrogación para no disminuir las prestaciones generales del sistema.

8 REFERENCIAS.

[1] Sistemas Scada. Articulo Internet.
www.geocities.com/ResearchTriangle/Campus/6495/The_Wireless_Data_Home_Page.htm Marzo 2003

[2] Departamento de automática y electrónica. Compendio de artículos de Electrónica. Internet.www.dea.icai.upco.es/ Enero 2003

[3] Wayne Tomasi, Sistemas de Comunicaciones Electrónicas.. Prentice Hall Segunda Edición. 1996. México

[4] Behrouz A. Forouzan, Transmisión de Datos y redes de comunicaciones.. Mc Graw Hill Segunda edicion. 2002. España.

[5] EIA (The Electronic Industries Association), EIA/TIA-232-E . Estándar para RS – 232. 2001

[6] EIA (The Electronic Industries Association), EIA/TIA-485-E . Estándar para RS – 485. 2001

[7] Monografías.com, Referencias de computación. Internet. www.monografias.com Marzo 2003

[8] Charles Petzold, Programación en Windows.. Mc Graw Hill. Primera Edición. 1999. México.

[9] Monografías.com, Lenguajes de Programación. Internet. www.monografias.com Marzo 2003

[10] Tutoriales.com, Programación orientada a objetos. Internet. www.topptutoriales.com Febrero 2003

[11] DeadLock++, ¿Qué compilador elijo?. Internet. http://accesobinario.neptra.com Marzo 2003

[12] Ing. Izaguirre Peggy Diseño de una red de comunicación para la supervisión de redes de distribución de la UCV. Trabajo especial de grado. Mayo de 2002. UCV

[13] Dallas Semiconductor, Tarjeta Tini http://www.ibutton.com/TINI Mayo 2003

[14] Microsoft, Sistema Operativo Windows 2000, http://www.microsoft.com Agosto 2003

[15] Charlie Calvert C++ Builder Borland. Sams Publishing. 1997. Primera edición. 1997. Estados Unidos

[16] Tutoriales de C++ builder, http://elvex.ugr.es/decsai/builder/index.html Febrero 2003

[17] Windmill Software LTD. Labial – RS-232 instrument driver. http://www.windmill.co.uk Febrero 2003

[18] Triangle Micro Works, Test Harness, http://www.TriangleMicroWorks.com Agosto 2003

[19] DUCATI Energía S.P.A. Manuales de soporte, http://www.ducatienergia.it/staging Enero 2003

[20] Foro Modbus. Especificaciones MODBUS, http://www.modbus.org Marzo 2003

[21] DNP3 Especificaciones DNP3, http://www.dnp.org

9 ANEXOS.

9.1 Anexo 1. Descripción del protocolo DUCBUS de DUCATI.

Dispositivos DUCATI:

Existen dispositivos fabricados por DUCATI Energía S.P.A. que utilizan el protocolo denominado DUCBUS para establecer la comunicación serial y extraer información de sus dispositivos. La familia de dispositivos donde se habilita este protocolo son los siguientes:

- Mach 30
- Mach 20
- Mach Smart
- Smart 96
- RMI

Aunque todos ellos utilizan el protocolo DUCBUS, para algunos no están habilitadas todas sus funciones.

Descripción:

La comunicación serial toma lugar entre el dispositivo y un terminal de computadora, acordando el modelo Maestro – Esclavo y considerando al dispositivo como el esclavo.

La comunicación se realiza bajo un interfase serial RS-485; Usando DUCBUS es posible direccionar hasta un máximo de 98 dispositivos, añadiendo un repetidor SRD cuando sea necesario.

Los dispositivos construyen una trama iniciando con el caracter <STX> (*Start Transmition*) y terminando dicha trama con el caracter <ETX> (*End Transmition*). Cuando una trama es recibida el cheksum (Chequeo de error) es verificado; el dispositivo ignora todas las tramas que no hayan sido enviadas por el terminal maestro o host.

Si el mensaje ha sido enviado por el host, el dispositivo examina la dirección destino de la trama y si esta es reconocida como su propia dirección, hace honor a cualquier solicitud contenida en ella.

Cuando un caracter <NAK> (*No Acknowledgment*) es recibido, cualquier recepción en progreso será interrumpida y el buffer serial del dispositivo conectado será limpiado.

Prototipo:

1. HOST ➔ MACH.
   <STX>H01<blank><dato><blank><checksum><ETX>

2. MACH ➔ HOST.
   <STX>M01<blank><dato><blank><checksum><ETX>

La trama 1, reporta como maestro “H” (HOST) enviando la dirección (01) del dispositivo a interrogar, cuando la trama es recibida el dispositivo “M” (MACH) contesta con su dirección (01) respondiendo la solicitud. La descripción de las etiquetas son las siguientes:

Etiqueta:	Valor:	Descripción:
<STX>	ASCII 02 hex	Comienzo de la trama.
H	ASCII 48 hex	Trama enviada por el Host.
M	ASCII 4D hex	Trama enviada por el Mach.
01	ASCII 30 31 hex	Dirección del Mach.
<blank>	ASCII 20 hex	Separador de encabezado – dato.
<data>	…	Contenido de la trama
<blank>	ASCII 20 hex	Separador de Dato – checsum
<checksum>	ASCII xx yy hex	Checksum de la trama
<ETX>	ASCII 03 hex	Final de la trama.

Los valores de al campo dato se explicaran mas adelante.

Cuando se configuran los parámetros de funcionamiento o se ejecuta un comando el dispositivo no emite ninguna trama de respuesta, pero emite un carácter de reconocimiento (Acknowledgment o no acknowledgment).

1. HOST ➔ MACH.
   <STX>H01<blank><dato><blank><checksum><ETX>

2. MACH ➔ HOST.
   <ACK> ó <NAK>

La descripción de estas etiquetas son las siguientes:

Etiqueta:	Valor:	Descripción:
<ACK>	ASCII 06 hex	Comando reconocido.
<NAK>	ASCII 15 hex	Comando no reconocido.

Ejemplo: El computador Host requiere del Mach de dirección 04 el equivalente máximo trifásico de potencia activa (código 19):

```
<0x02>H04<0x20>?19<0x20>2C<0x03> Trama enviada por el Host.
<0x02>M04       3040 19<0x03> Trama respondida por el Mach.
```

Nota: En codificación ASCII el carácter en blanco es el 0x20 hexadecimal pero por razones de facilitar la comprensión en lo sucesivo lo representaremos con el caracter "_", porque es obvio que se imposibilita contar espacios en blanco.

La respuesta enviada por el Mach esta alineada a la derecha, contiene 12 caracteres de campo, precedidos de espacios en blanco (ASCII 20 hex).

Los errores de comunicación se detectan utilizando un checksum localizado al final de la trama. El checksum se inicializa en 55 hexadecimal, y entonces es computado el OR exclusivo con cada byte de la trama, excluyendo el mismo checksum y el caracter de final <ETX>. El checksum es un byte, y esta formateado en la trama como un campo de 2 caracteres (nible alto y nible bajo): por ejemplo, si el resultado es 255, la cadena "FF" será añadida a la trama.

Una vez computado el checksum es adherido al final de la trama y se le anexa el caracter de final de trama <ETX>, si la trama es recibida se verifica que el checksum sea correcto comparando los campos correspondientes.

Formato general de una solicitud:

Para realizar una solicitud hay que completar el campo <dato> definido anteriormente, para esto se tiene un formato general que contempla la mayoría de las solicitudes.

?<nn> Formato de solicitud

La siguiente tabla muestra los posibles valores:

<nn>	**Dirección**	**Variable:**
00	00-01	Frecuencia
01	02-03	Voltaje equivalente trifásico
02	04-05	Voltaje de línea (1-2)
03	06-07	Voltaje de línea (2-3)
04	08-09	Voltaje de línea (3-1)
05	10-11	Voltaje de fase neutro de línea 1
06	12-13	Voltaje de fase neutro de línea 2
07	14-15	Voltaje de fase neutro de línea 3
08	16-17	Corriente trifásica equivalente
09	18-19	Corriente en la línea 1
10	20-21	Corriente en la línea 2
11	22-23	Corriente en la línea 3
12	24-25	Máximo promedio de canal X1.
13	26-27	Factor de potencia equivalente trifásico
14	28-29	Factor de potencia de línea 1
15	30-31	Factor de potencia de línea 2
16	32-33	Factor de potencia de línea 3
17	34-35	Potencia activa equivalente trifásico
18	36-37	Potencia activa promedio equivalente trifásico
19	38-39	Máximo potencia activa equivalente trifásico
20	40-41	Potencia activa de línea 1
21	42-43	Potencia activa de línea 2
22	44-45	Potencia activa de línea 3
23	46-47	Potencia activa promedio de línea 1
24	48-49	Potencia activa promedio de línea 2
25	50-51	Potencia activa promedio de línea 3
26	52-53	Potencia activa máxima de línea 1
27	54-55	Potencia activa máxima de línea 2
28	56-57	Potencia activa máxima de línea 3
29	58-59	Potencia aparente equivalente trifásico
30	60-61	Potencia aparente promedio equivalente trifásico
31	62-63	Máximo potencia aparente equivalente trifásico

32	64-65	Potencia aparente de línea 1
33	66-67	Potencia aparente de línea 2
34	68-69	Potencia aparente de línea 3
35	70-71	Potencia aparente promedio de línea 1
36	72-73	Potencia aparente promedio de línea 2
37	74-75	Potencia aparente promedio de línea 3
38	76-77	Potencia aparente máxima de línea 1
39	78-79	Potencia aparente máxima de línea 2
40	80-81	Potencia aparente máxima de línea 3
41	82-83	Potencia reactiva equivalente trifásico
42	84-85	Potencia reactiva promedio equivalente trifásico
43	86-87	Máximo potencia reactiva equivalente trifásico
44	88-89	Potencia reactiva de línea 1
45	90-91	Potencia reactiva de línea 2
46	92-93	Potencia reactiva de línea 3
47	94-95	Potencia reactiva promedio de línea 1
48	96-97	Potencia reactiva promedio de línea 2
49	98-99	Potencia reactiva promedio de línea 3
50	100-101	Potencia reactiva máxima de línea 1
51	102-103	Potencia reactiva máxima de línea 2
52	104-105	Potencia reactiva máxima de línea 3
53	106-107	Potencia activa promedio máxima equivalente trifásico
54	108-109	Potencia reactiva promedio máxima equivalente trifásico
55	110-111	Potencia aparente promedio máxima equivalente trifásico
56	112-113	Máximo promedio del canal 2
57	114-115	Energía trifásica activa absorbida por el sistema
58	116-117	Energía activa absorbida por la línea 1
59	118-119	Energía activa absorbida por la línea 2
60	120-121	Energía activa absorbida por la línea 3
61	122-123	Energía trifásica reactiva absorbida por el sistema
62	124-125	Energía reactiva absorbida por la línea 1
63	126-127	Energía reactiva absorbida por la línea 2
64	128-129	Energía reactiva absorbida por la línea 3
65	130-131	THDF de voltaje de línea 1
66	132-133	THDF de voltaje de línea 2
67	134-135	THDF de voltaje de línea 3
68	136-137	THDF de corriente de línea 1
69	138-139	THDF de corriente de línea 2
70	140-141	THDF de corriente de línea 3
71	142-143	X1 4-20 mA corriente de entrada.
72	144-145	X2 4-20 mA corriente de entrada.
73	146-147	X1 canal multiplicador
74	148-149	X2 canal multiplicador
75	150-151	Relación de transformación de voltaje
76	152-153	Relación de transformación de corriente

77	154-155	Tiempo promedio
78	156-157	Máximo del promedio de potencia activa de la fase 1
79	158-159	Máximo del promedio de potencia activa de la fase 2
80	160-161	Máximo del promedio de potencia activa de la fase 3
81	162-163	Máximo del promedio de potencia reactiva de la fase 1
82	164-165	Máximo del promedio de potencia reactiva de la fase 2
83	166-167	Máximo del promedio de potencia reactiva de la fase 3
84	168-169	Máximo del promedio de potencia aparente de la fase 1
85	170-171	Máximo del promedio de potencia aparente de la fase 2
86	172-173	Máximo del promedio de potencia aparente de la fase 3
87	174-175	Energía activa trifásica generada por el sistema
88	176-177	Energía activa generada por la línea 1
89	178-179	Energía activa generada por la línea 2
90	180-181	Energía activa generada por la línea 3
91	182-183	Energía reactiva trifásica generada por el sistema
92	184-185	Energía reactiva generada por la línea 1
93	186-187	Energía reactiva generada por la línea 2
94	188-189	Energía reactiva generada por la línea 3
95	190-191	X1 promedio
96	192-193	X2 promedio
97	194-195	X1 integral
98	196-197	X2 integral

Nota: La dirección indica el byte alto y bajo respectivamente.

Algunos dispositivos no responden algunas solicitudes por no estar habilitadas, para una mayor descripción del protocolo se hace la referencia a los manuales que el soporte de DUCATI Energía S.P.A. pueda prestar [19].

Nota: Los comandos de operación del mach son tan reducidos e inoperantes que no se anexan a este documento.

9.2 Anexo 2. Descripción del protocolo MODBUS RTU y MODBUS TCP.

Transacciones sobre Redes Modbus seriales.

Los puertos standard MODBUS en controladores utilizan un interfase serie compatible RS-232C 0 RS - 485. La norma EIA RS-232C define patillas del conector, cableado, niveles de señal, velocidades de transmisión y control de paridad. Los controladores pueden ser conectados en red directamente o vía módems.

Los controladores se comunican usando una técnica maestro – esclavo, en la cual sólo un dispositivo (el maestro) puede iniciar transacciones (llamadas 'peticiones'). Los otros dispositivos (los esclavos) responden suministrando al maestro el dato solicitado, o realizando la acción solicitada en la petición. Entre los dispositivos maestros típicos se incluyen los procesadores centrales y los paneles de programación. Esclavos típicos son los PLC's (controladores programables). El maestro puede direccionar esclavos individualmente o puede generar un mensaje en modo difusión a todos los esclavos. Los esclavos devuelven un mensaje (llamado 'respuesta') a las peticiones que les son direccionadas individualmente. No se devuelven respuestas a peticiones en modo difusión enviadas desde el maestro.

El protocolo MODBUS establece el formato para la petición del maestro, colocando en ella la dirección del dispositivo esclavo (0 en caso de 'difusión'), un código de función que define la acción solicitada, cualquier dato que haya de enviarse y un campo de comprobación de error. El mensaje de respuesta del esclavo está también definido por el protocolo MODBUS. Contiene campos confirmando la acción tomada, cualquier dato que haya de devolverse y un campo de comprobación de error. Si el mensaje recibido por el esclavo es defectuoso o el esclavo es incapaz de realizar la acción solicitada, construirá un mensaje de error y lo enviará como respuesta.

Transacciones en otros tipos de redes

Además de sus capacidades MODBUS standard, algunos modelos de controladores pueden comunicar sobre MODBUS Plus, utilizando puertos incorporados al efecto o adaptadores de red y sobre MAP, utilizando adaptadores de red.

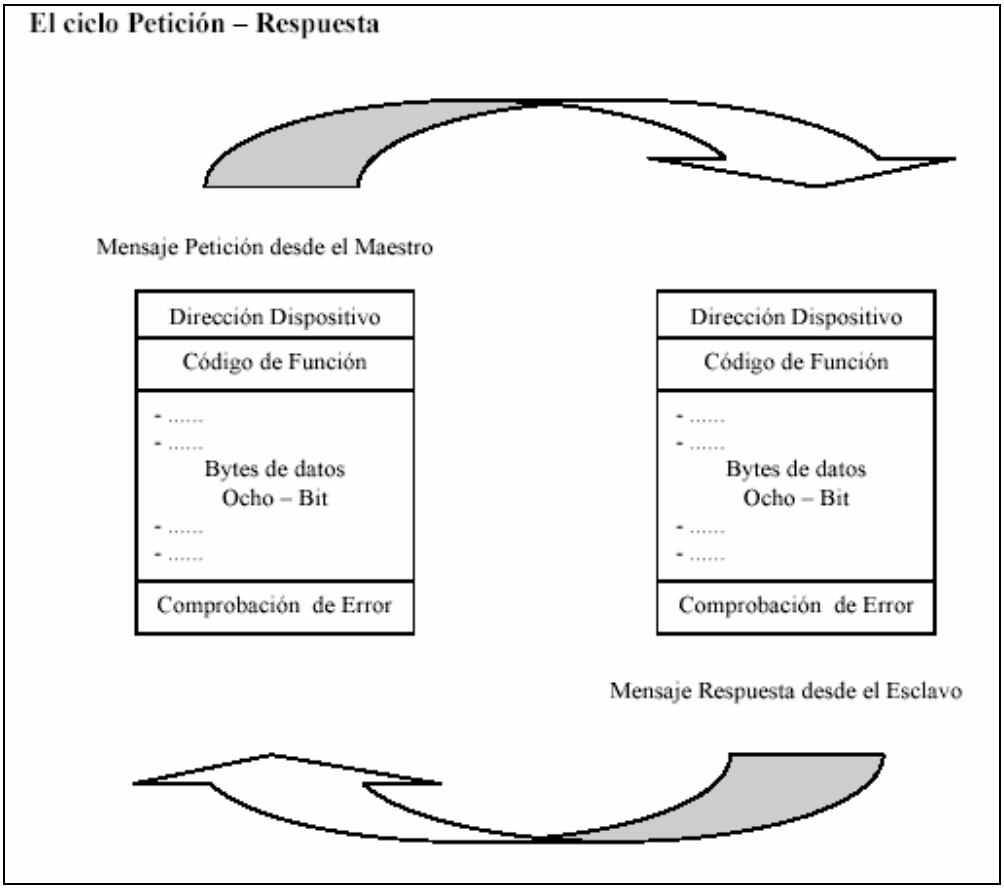
En estas redes, los controladores comunican usando una técnica 'todos contra todos' ('peer-to-peer') en la cual cualquier controlador puede iniciar transacciones con los otros controladores. Así un controlador puede operar como maestro o como esclavo en diferentes transacciones. Con frecuencia se proporcionan múltiples caminos internos para permitir procesamiento concurrente de transacciones maestro y esclavo.

A nivel de mensaje, el protocolo MODBUS todavía aplica el principio maestro-esclavo aunque el método de comunicación en red sea 'peer-to-peer'. Si un controlador origina un mensaje, lo hace como un dispositivo maestro y espera una respuesta desde un dispositivo esclavo. De la misma forma, cuando un controlador recibe un mensaje, construye una respuesta como esclavo y la envía al controlador que originó la transacción.

Otra forma del protocolo es la conocida como MODBUS sobre TCP que incluye propiedades sobre arquitecturas del tipo TCP/IP, usa un modelo maestro – esclavo y tramas similares a la variante del protocolo conocido como MODBUS RTU.

Descripción de MODBUS:

La siguiente figura muestra la manera en que el protocolo de manera general realiza las transacciones:



La Petición: El código de función en la petición indica al dispositivo esclavo diseccionado y el tipo de acción a realizar. Los bytes de datos contienen cualquier información adicional que el esclavo necesitará para llevar a cabo la función. Por ejemplo el código de función 03 pedirá al esclavo que lea registros mantenidos (holding regs.) y responda con sus contenidos. El campo de datos debe contener la información que indique al esclavo en qué registro debe comenzar y cuántos ha de leer. El campo de comprobación de error proporciona un método para que el esclavo valide la integridad del contenido del mensaje recibido.

La Respuesta: Si el esclavo elabora una respuesta normal, el código de función contenido en la respuesta es una réplica del código de función enviado en la petición. Los bytes de datos contienen los datos recolectados por el esclavo, tales como valores de registros o estados. Si ocurre un error, el código de función contenido en la respuesta es diferente al código de función enviado en la petición, para indicar que la respuesta es una respuesta de error y los bytes de datos contienen un código que describe el error. El campo de comprobación de error permite al maestro confirmar que los contenidos del mensaje son válidos.

Modo RTU:

Cuando los controladores son configurados para comunicar en una red MODBUS usando el modo RTU (Remote Terminal Unit), cada byte de 8 bits en un mensaje contiene dos

digitos hexadecimales de 4 bits. La principal ventaja de este modo es que su mayor densidad de carácter permite mejor rendimiento que el modo ASCII para la misma velocidad. Cada mensaje debe ser transmitido en un flujo continuo.

El formato para cada byte en modo RTU es:

Sistema de codificación:
Binario 8-bits, hexadecimal 0-9, A-F. Dos dígitos hexadecimales contenidos en cada campo de 8 bits del mensaje.

Bits por byte:
1 bit de arranque.
8 bits de datos, el menos significativo se envía primero.
1 bit para paridad Par o Impar; ningún bit para No paridad.
1 bit de paro si se usa paridad; 2 bits si no se usa paridad.

Campo de Comprobación de error:
Comprobación Cíclica Redundante (CRC).

Trama RTU

En modo RTU, los mensajes comienzan con un intervalo silencioso de al menos 3.5 tiempos de carácter. Esto es mas fácilmente implementado como un múltiplo de tiempos de carácter a la velocidad de transmisión configurada en la red (mostrado como T1-T2-T3-T4 en la figura que sigue).

ARRANQUE	DIRECCION	FUNCION	DATOS	COMPROB. CRC	FINAL
T1-T2-T3-T4	8 BITS	8 BITS	N x 8 BITS	16 BITS	T1-T2-T3-T4

El primer campo transmitido es entonces la dirección del dispositivo destinatario. Los caracteres a transmitir permitidos para todos los campos son 0-A, A-F hexadecimal. Los dispositivos conectados en red monitorizan el bus de red continuamente incluso durante los intervalos ‘silencioso’. Cuando el primer campo (el campo de dirección) es recibido, cada dispositivo lo decodifica para enterarse si es el dispositivo direccionado. Siguiendo al último carácter transmitido, un intervalo de al menos 3.5 tiempos de carácter señala el final del mensaje. Un nuevo mensaje puede comenzar después de este intervalo. La trama completa del mensaje debe ser transmitida como un flujo continuo. Si un intervalo silencioso de más de 1.5 tiempos de carácter tiene lugar antes de completar la trama, el dispositivo receptor desecha el mensaje incompleto y asume que el próximo byte será el campo de dirección de un nuevo mensaje.

De forma similar, si un nuevo mensaje comienza antes de que transcurran 3.5 tiempos de carácter después de un mensaje previo, el dispositivo receptor lo considerará una

continuación del mensaje previo. Esto dará lugar a un error, ya que el valor en el campo final CRC no será válido para el mensaje combinado.

Cómo es Manipulado el Campo Dirección

El campo dirección de un mensaje contiene dos caracteres (ASCII) u ocho bits (RTU). Las direcciones de esclavo válidas están en el rango de 0 – 247 decimal. Los dispositivos esclavos individuales tienen direcciones asignadas en el rango 1 – 247. Un maestro direcciona un esclavo situando la dirección del esclavo en el campo dirección del mensaje. Cuando el esclavo envía su respuesta, sitúa su propia dirección en este campo dirección de la respuesta para dar a conocer al maestro qué esclavo está respondiendo. La dirección 0 es utilizada para dirección difusión, la cual todos los dispositivos esclavos reconocen. Cuando el protocolo MODBUS es usado en redes de nivel mas alto, las difusiones pueden no estar permitidas o pueden ser reemplazada por otros métodos. Por ejemplo, Modbus Plus utiliza una base de datos global compartida que puede ser actualizada con cada rotación del testigo.

Cómo es Manipulado el Campo Función

El campo código de función de una trama de mensaje contiene dos caracteres (ASCII) u ocho bits (RTU). Los códigos válidos están en el rango de 1 – 255 decimal. De esos, algunos códigos son aplicables a todos los controladores, mientras que algunos códigos se aplican sólo en algunos modelos y otros están reservados para usos futuros. Cuando un mensaje es enviado desde un maestro a un dispositivo esclavo, el campo del código de función indica al esclavo qué tipo de acción ha de ejecutar. Por ejemplo: leer los estados ON/OFF de un grupo bobinas o entradas discretas; leer el contenido de datos de un grupo de registros; leer el status de diagnóstico de un esclavo; escribir en determinadas bobinas o registros; o permitir cargar, salvar o verificar el programa dentro del esclavo.

Cuando el esclavo responde al maestro, utiliza el campo del código de función para indicar bien una respuesta normal (libre de error) o que algún tipo de error ha tenido lugar (denominado respuesta de excepción). Para una respuesta normal, el esclavo simplemente replica el código de función original. Para una respuesta de excepción, el esclavo devuelve un código que es equivalente al código de función original con su bit más significativo puesto a valor 1.

Por ejemplo, un mensaje desde un maestro a un esclavo para leer un grupo de registros mantenidos tendría el siguiente código de función:

0000 0011 (Hexadecimal 03)

Si el dispositivo esclavo ejecuta la acción solicitada, sin error, devuelve el mismo código en su respuesta. Si ocurre una excepción. Devuelve:

1000 0011 (Hexadecimal 83)

También se acostumbra decir que se le suma 80 hexadecimal al código lo que es equivalente.

Además de la modificación del código de función para una respuesta de excepción, el esclavo sitúa un único código en el campo de datos del mensaje respuesta. Esto indica al maestro qué tipo de error ha tenido lugar, o la razón para la excepción.

El programa de aplicación del maestro tiene la responsabilidad de manejar las respuestas de excepción. Procedimiento típicos son: enviar subsiguientes reintentos de mensaje, intentar mensajes de diagnóstico al esclavo y notificar operadores.

Contenido del Campo Datos

El campo datos se construye utilizando conjuntos de 2 dígitos hexadecimales, en el rango de 00 á FF hexadecimal. Pueden formarse a partir de un par de caracteres ASCII o desde un carácter RTU, de acuerdo al modo de transmisión serie de la red.

El campo datos de los mensajes enviados desde un maestro a un esclavo, contiene información adicional que el esclavo debe usar para tomar la acción definida por el código de función. Esto puede incluir partes como direcciones discretas y de registros, la cantidad de partes que han de ser manipuladas y el cómputo de bytes de datos contenidos en el campo.

Por ejemplo, si el maestro solicita a un esclavo leer un grupo de registros mantenidos (código de función 03), el campo de datos especifica el registro de comienzo y cuántos registros han de ser leídos. Si el maestro escribe sobre un grupo de registros en el esclavo (código de función 10 hexadecimal), el campo datos especifica el registro de comienzo, cuántos registros escribir, el cómputo de bytes de datos que siguen en el campo datos y los datos que se deben escribir en los registros.

Si no ocurre error, el campo datos de una respuesta desde un esclavo al maestro contiene los datos solicitados. Si ocurre un error, el campo contiene un código de excepción que la aplicación del maestro puede utilizar para determinar la próxima acción a tomar.

El campo datos puede ser inexistente (de longitud cero) en ciertos tipos de mensajes. Por ejemplo, en una petición de un dispositivo maestro a un esclavo para que responda con su anotación de eventos de comunicación (Código de función 0B hexadecimal), el esclavo no requiere ninguna información adicional. El código de función por sí solo especifica la acción.

Contenido del Campo Comprobación de Error

Cuando el modo RTU es usado para la trama, el campo Comprobación de Error contiene un valor de 16 bits implementado como dos bytes de 8 bits. El valor de comprobación de error es el resultado de un cálculo Comprobación Cíclica Redundante (CRC) realizado sobre el contenido del mensaje.

El campo CRC es añadido al mensaje como último campo del mensaje. La forma de hacerlo es, añadir primero el byte de orden bajo del campo, seguido del byte de orden alto. El byte de orden alto del CRC es el último byte a enviar en el mensaje.

Funciones MODBUS:

Hasta el momento se utilizan aproximadamente 13 funciones que tienen básicamente la tarea de leer y escribir información. La división puede hacerse en tres grupos de funciones, divididas según el tipo de variables a obtener; el primer grupo es conformado por operaciones sobre variables binarias (bit), el segundo corresponde a operaciones sobre variables de tipo registro (variables de 16 bits) y el ultimo grupo no muy usado se refiere a funciones especiales. La siguiente tabla muestra un resumen de las funciones:

	Registro de operación:	Descripción	Código decimal:	Código hexadecimal
Acceso a Datos	Binarios (Operaciones sobre bits)	Lectura de una entradas	02	02
		Lectura de entradas	01	01
		Escritura de una entrada	05	05
		Escritura de múltiples entradas	15	0F
	Registros (Operaciones sobre registros de 16 bits)	Lectura de un registro	04	04
		Lectura de múltiples registros	03	03
		Escritura de un registro	06	06
		Escritura de múltiples registros	16	10
		Lectura y escritura de múltiples registros	23	17
		Escritura de mascara de registro	22	16
	Acceso a archivos	Leer archivo	20	14
		Escribir archivo	21	15
Otros	Identificación	Leer identificación de dispositivo	43	2B

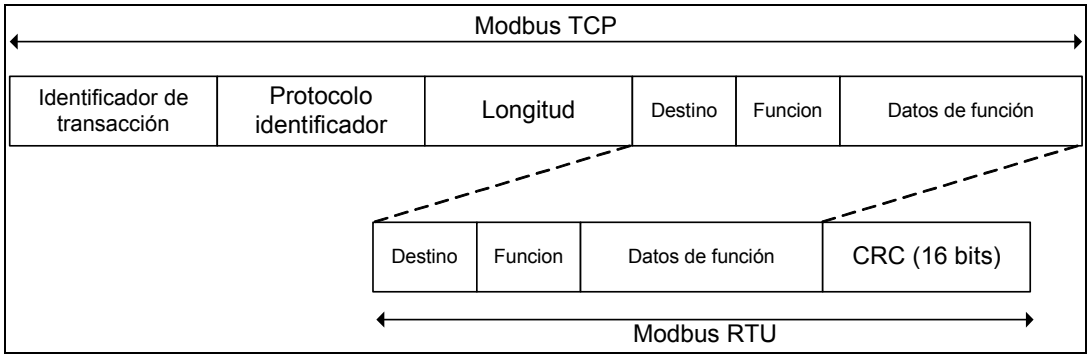
Se muestra en la tabla un resumen de las funciones y su descripción general, si se desea profundizar en el formato específico de cada función, se recomienda utilizar los manuales de usuario que de manera gratuita distribuye el foro "Modbus.Org" a través de Internet [20].

MODBUS TCP:

MODBUS TCP es una variante del protocolo MODBUS genérico para aplicaciones en comunicación de datos sobre redes de tipo TCP/IP. Este tipo de redes ha obtenido gran difusión a nivel mundial por su confiabilidad y robustez. Es por esto que los protocolos para buses de campo tienden a introducir cambios a fin de utilizar esta plataforma, que promete ser uno de los medios mas eficientes para redes de datos. El foro internacional de MODBUS decidió la creación de MODBUS TCP basado en una de sus mas potentes variantes conocida como MODBUS RTU, que ha tenido gran éxito a nivel de redes seriales. Para esta variante decidieron agregar un encabezado de longitud fija a la trama genérica de MODBUS RTU, y eliminar el chequeo de error (CRC) al final de esta; lo ultimo se debe a que el chequeo de error descansa sobre la plataforma TCP/IP. La siguiente tabla muestra como se conforma el encabezado y que acciones se toman.

Campos	Longitud	Descripción	Cliente	Servidor
Identificador de transacción.	2 Bytes	Identificación de una solicitud o respuesta.	Inicializado por el cliente	Copiado por el servidor para emitir una respuesta
Protocolo identificador	2 Bytes	0 para Modbus	Inicializado por el cliente	Copiado por el servidor para emitir una respuesta
Longitud	2 Bytes	Numero de bytes que siguen al mensaje	Inicializado por el cliente (solicitud)	Inicializado por el servidor (respuesta)
Identificación de Unidad	1 Bytes	Identificación del dispositivo conectado en la línea serial u otro bus.	Inicializado por el cliente	Copiado por el servidor para emitir la respuesta.
Datos	N Bytes	Función y datos	Inicializado por el cliente	Copiado por el servidor para realizar proceso

No sorprende la idea de introducir los conceptos de cliente y servidor, ya que esta es la filosofía seguida en redes de datos y además se asemeja al modelo Maestro – esclavo. El encabezado comienza con un identificador de transacción que indicará al proceso cual respuesta corresponde a cual pregunta, el cliente inicializa este identificador a fin de controlar el flujo. El protocolo identificador funciona para indicar el tipo de protocolo, esto con el fin de poder introducir nuevas versiones futuras. La longitud informa de cuan largo es el mensaje enviado. El identificador de unidad y datos forma parte de la trama genérica MODBUS RTU, la siguiente figura muestra la relación entre los protocolos MODBUS RTU y MODBUS TCP.



Es de hacer notar que la trama MODBUS TCP soporta toda la información que requiere la trama MODBUS RTU, solo basta calcular el CRC y anexarlo.

Finalmente es evidente que todas las especificaciones MODBUS RTU se aplican a transacciones bajo MODBUS TCP solo incluyendo la información de encabezado. Para un estudio más elaborado se recomienda visitar la Web del foro MODBUS [20] y descargar los manuales oficiales pertinentes.

9.3 Anexo 3. Descripción del protocolo DNP3.

Modelo de capas para DNP.

DNP 3,0 es un protocolo "encapado". Aún así, en lugar de asemejarse al protocolo de 7 capas de la OSI (Open System Interconection - interconexión de sistemas abiertos), DNP3 se adhiere a un estándar simplificado de 3 capas propuesto por el IEC (International Electrotechnical Commission - Comisión internacional de Electrotecnia) para implementaciones más básicas. El IEC llama a esto Enhanced Performance Architecture, o EPA. (en realidad, DNP 3,0 agrega una cuarta capa, una capa de pseudo-transporte que permite la segmentación del mensaje).

Capa Física.

La capa física se refiere sobre todo a los medios físicos sobre los cuales se está comunicando el protocolo. Por ejemplo, maneja el estado del medio (limpio u ocupado), y la sincronización a través del medio (iniciando y parando). Más comúnmente, DNP se específica sobre una capa física serial simple tal como RS-232 o RS-485 usando medios físicos tales como fibra, radio o satélite. Los proyectos se orientan actualmente para implementar DNP sobre una capa física como Ethernet.

Capa de Transmisión de Datos.

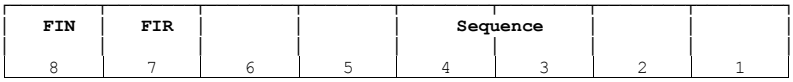
La capa de transmisión de datos maneja la conexión lógica entre el remitente y el receptor de la información y pone a prueba las características de error del canal físico. DNP3 logra esto comenzando cada trama de transmisión de datos con una cabecera, e insertando un CRC de 16 bits cada 16 bytes de la trama. Una trama es una porción de un mensaje completo comunicado sobre la capa física. La medida máxima de una trama de transmisión de datos es 256 bytes. Cada trama tiene una dirección fuente de 16 bits y una dirección de destino también de 16 bits, las que pueden ser una dirección de difusión o broadcast (0xffff). La información del direccionamiento, junto con un código de inicio de 16 bits, la longitud del trama, y un byte de control de transmisión de datos se hallan en la cabecera (10 bytes) de transmisión de datos.
El byte de control de transmisión de datos indica el propósito de la trama de transmisión de datos, y el estado de la conexión lógica. Los valores posibles del byte de control de transmisión de datos son: ACK (*acknowledgment*), NACK (*no acknowledgment*), la conexión necesita inicializar (*reset link status*), datos de usuario sin confirmación (*unconfirmed user data*), datos de usuario con confirmación (*confirmed user data*), solicitud de estado de conexión (*link status*), y contestación de estado de conexión (*test link status*). Cuando se solicita una confirmación de transmisión de datos, el receptor debe responder con una trama ACK de transmisión de datos si el mismo es recibido y pasa los controles del CRC. Si una confirmación de la transmisión de datos no se solicita, no se requiere ninguna respuesta de la transmisión de datos.

La figura siguiente muestra un descripción gráfica del formato del encabezado de la capa transmisión de datos o capa de enlace (Data link layer). Esta descripción muestra como están conformados los 10 primeros bytes de la trama completa que contienen la información de la capa de enlace. Muestra como se desglosa el campo de control y los códigos de función de manera respectiva, la idea de este texto no es hacer exhaustivas las definiciones, sino mas bien dar al lector una idea completa de las posibilidades.

Capa de Pseudo-Transporte.

La capa de pseudo-transporte divide mensajes de la capa de aplicación en múltiples tramas de transmisión de datos. Para cada trama, inserta un código de función de 1 byte que indica si la trama de transmisión de datos es la primera trama del mensaje, la última trama del mensaje, o ambos (para mensajes simples). El código de función también incluye un número de secuencia de la trama que se incrementa con cada uno y permite que la capa de transporte recipiente detecte tramas perdidas.

Una vez conformado el mensaje de la capa de aplicación, este es pasado a esta capa, aquí se le anexa un caracter de encabezado que tiene las siguientes características.



FIN: Inicializado a 1 significa que es el fragmento final de la trama, si es 0 indica que siguen fragmentos a este.
FIR: Inicializado a 1 indica que es el primer fragmento de la trama entera de aplicación, si es 0 indica que es uno de los fragmentos sucesivos.
Sequence: La secuencia es un numero entre 0 y 63 que se utiliza para resolver la organización de los fragmentos, al arribar un fragmento este numero se incrementa en la espera del proximo fragmento, al llegar a 63 comienza de nuevo en 0.

La idea consiste en fragmentar los mensajes de la capa de aplicación en grupos de hasta 249 bytes. Que al sumarle 1 byte de encabezado de transporte y 5 bytes fijos de encabezado de capa de enlace, completan un máximo de 255 que puede manejar el byte de longitud de la capa de enlace. Pareciera contradictorio el hecho de que la capa de enlace tiene 5 bytes, en vez de 10 como se dijo, lo que sucede es que no se cuentan los 2 caracteres de sincronización, ni el propio byte de longitud, ni lo dos del CRC.

Capa de aplicación:

Por ser esta capa la responsable de contener la información del proceso es aquí donde se realiza un estudio mas detallado. Es en esta capa donde se reciben y procesan las solicitudes, se manejan los tipos de objeto de datos y se construyen las respuestas, a continuación se describe cada una de sus partes de manera detallada.

Formato de los mensajes:

Esta sección define los formatos de los mensajes de la capa de aplicación (APDU). Los términos APDU y fragmento son permutables. En esta especificación se define la estación maestra como la estación que envía el mensaje de la solicitud y la estación terminal que es el dispositivo auxiliar, la unidad terminal remota (RTU) o el dispositivo inteligente del extremo (IED) al cual los mensajes solicitados son destinados. En DNP, solamente señalado las estaciones se puede enviar mensajes de solicitud desde la capa de usuario y solamente las estaciones terminales pueden enviar mensajes de respuesta a la capa de usuario. La Ilustración

1 abajo muestra la secuencia de los mensajes de la capa de usuario entre un maestro y una estación remota.

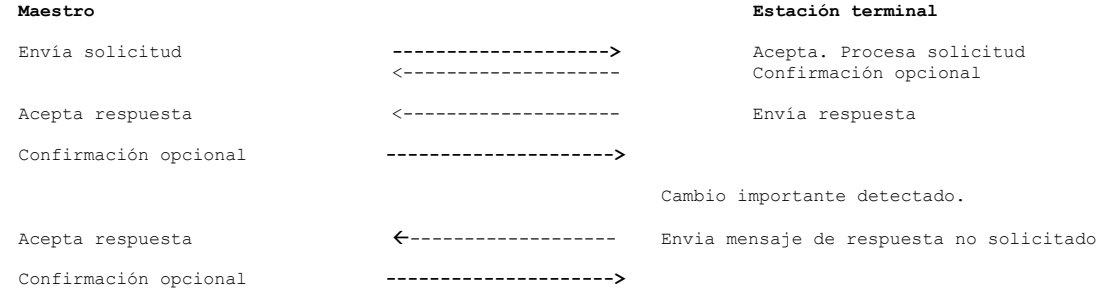


Ilustración 1 Secuencia de mensajes.

La estación maestra envía una solicitud desde la capa de usuario a la estación terminal la cuál vuelve una respuesta a la capa de usuario. La estación terminal puede decidir transmitir espontáneamente datos usando un mensaje de respuesta no solicitado desde la capa de usuario. Para un maestro, una transacción de solicitud / respuesta con una estación remota particular debe ser terminada antes de que otras peticiones se puedan enviar a esa estación terminal. Una estación maestra puede aceptar respuestas no solicitadas mientras que la transacción de la solicitud está en marcha. Para una estación terminal, una transacción de solicitud / respuesta debe ser terminada antes de que se acepte cualquier otra solicitud o enviar respuestas no solicitadas. Las respuestas no solicitadas se pueden enviar antes o después de la transacción de solicitud / respuesta pero no durante. Si una estación terminal está actualmente en el centro de una transacción no solicitada (es decir esperando una confirmación), puede aceptar condicionalmente un comando de la petición del maestro. Además, cada respuesta o solicitud puede consistir en fragmentos individuales de 1 o más.

Cada fragmento sin embargo debe ser digestible y por lo tanto ejecutable (porque el código de la función es parte de cada fragmento). Se aconseja que los dispositivos con capacidades de almacenaje limitadas, solamente sean enviados los fragmentos de peticiones del mensaje cuando la respuesta prevista (de todos los fragmentos enviados) es más grande de un fragmento. Éste debe asegurarse de que los dispositivos pueden procesar una petición y construir y enviar más rápidamente una respuesta antes de que se reciba la petición siguiente. Si no, los mensajes del multi-fragmento pueden requerir las respuestas del multi-fragmento que pueden requerir más almacenaje del que el dispositivo tiene disponible.

Formato de solicitudes en la capa de aplicación:

El formato del mensaje de la solicitud del usuario (APDU) se ilustra en la Ilustración 2. El APDU se compone de un bloque de APCI que contenga la información de control del mensaje y un ASDU que contenga la información que se procesará por la estación de recepción. El APCI a menudo se llama encabezado de la solicitud. En DNP, el ASDU es opcional y se utiliza cuando el significado del mensaje no se transporta totalmente en el encabezado de la solicitud. El encabezado de la solicitud contiene la información sobre cómo ensamblar un mensaje fragmentado y sobre el propósito del mensaje. El encabezado de la solicitud está presente en todas las solicitudes (APDUs) de la capa de usuario. Si el encabezado de la solicitud implica toda la información necesaria requerida para realizar la solicitud, el ASDU no está presente.

Cada ASDU consiste en unos o más identificadores de unidades de datos (DUI) o encabezados de objeto y los objetos asociados opcionales de la información (IO) o las zonas de información.

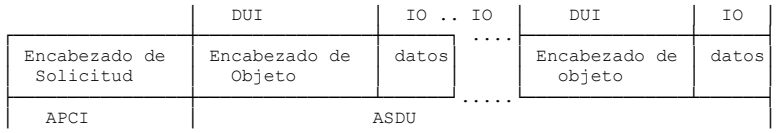


Ilustración 2 Formato de solicitud de usuario.

El encabezado de la solicitud: identifica el propósito del mensaje y consiste en APCI (información de control de protocolo de usuario).
El encabezado del objeto: identifica los objetos de los datos que siguen.
Datos: Objeto(s) de datos del tipo especificado en el encabezado del objeto.

Formato de la respuesta:

La respuesta de una estación remota a una petición APDU de la capa de usuario o la respuesta no solicitada de una estación remota tiene el formato ilustrado en la Ilustración 3. El formato es idéntico en la forma a la solicitud. El APCI a menudo se llama encabezado de la respuesta en un mensaje de respuesta del usuario. El encabezado de la respuesta contiene la misma información que el encabezado de la petición más un campo adicional que contiene las indicaciones internas de la estación remota. El encabezado de la respuesta es siempre parte de la respuesta. La respuesta ASDU tiene el mismo formato del mensaje de la solicitud con una excepción notable.

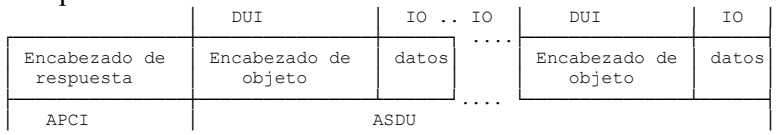


Ilustración 3 Formato de la respuesta.

Definición de los campos de DNP:

Esta sección describe las cabeceras de las solicitudes y las respuestas (**APCI –** Control de información de protocolo en la capa de aplicación) para controlar la secuencia y el flujo de mensajes de aplicación entre una estación maestra y otra estación, así como también la información a ser procesada por el receptor (**ASDU –** Servicio de unidades de datos para la capa de aplicación) cuando se incluya cabeceras y datos de objeto. Las cabeceras son usadas para ensamblar mensajes fragmentados (multi – **APDU** – Unidades de datos de protocolo para la capa de aplicación) de datos de aplicación de usuario. Los encabezados de objetos son usados para identificar únicamente la información de objetos que opcionalmente se utilizan.

Cabeceras de capa de aplicación:

Los encabezados de las solicitudes o APCI tienen 2 campos cada campo se compone de un largo de 1 octeto (8 bits) como se ilustra en la Ilustración 4:

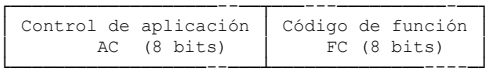


Ilustración 4 Cabecera de una solicitud.

Los encabezados de las respuestas tienen 3 campos como se ilustra en la Ilustración 5:

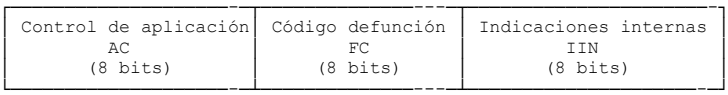


Ilustración 5 Cabecera de una respuesta.

Control de aplicación:

El campo de control de aplicación tiene un tamaño de un octeto (8 bits) y provee la información necesaria para ensamblar mensajes en la capa de aplicación que estén fragmentados.

Los mensajes de la capa de aplicación pueden ser empaquetados en fragmentos, cada uno es lo bastante pequeño como para encajar en el buffer de mensajes de aplicación. El tamaño recomendado para el tamaño de este buffer es 2048 bytes para mantener compatibilidad con dispositivos corrientes que usan DNP. Cada fragmento tiene una cabecera de aplicación y una cabecera de objeto tal que cada fragmento puede ser procesado como un mensaje individual que puede entonces ser descartado al construir el próximo fragmento. La Ilustración 6 muestra la distribución de este octeto.

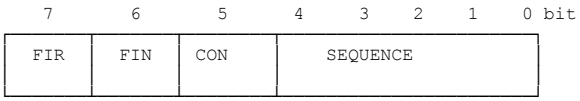


Ilustración 6 Control de aplicación.

FIR: Si esta inicializado (FIR=1), indica que el fragmento es el primero de un mensaje completo de aplicación.
FIN: Si esta inicializado (FIN=1), indica que el fragmento del mensaje es el ultimo de un mensaje completo de aplicación.
CON: Si esta inicializado (CON=1), en un mensaje recibido, indica que el que envió el mensaje espera por una confirmación de que ha sido recibido el fragmento de mensaje; Para un mensaje sin confirmación se utiliza CON=0.
SÉQUENCE: Indica el numero del fragmento. Desde 0 hasta 15 son reservados para solicitudes y respuestas de la estación maestra y otras estaciones. Desde la 16 hasta la 31 son reservados para mensajes no solicitados de las estaciones. Para mensajes no solicitados, cada fragmento consecutivo desde una estación puede incrementar el numero de secuencia (el

numero entre 16 y 31). Para respuestas de una estación, cada fragmento consecutivo recibido desde o transmitido para otra estación puede tener un incremento en el numero de secuencia.

NOTA: Una respuesta no solicitada es un mensaje generado por una estación esclava, que usualmente contiene datos de eventos que son enviados automáticamente a maestro, el maestro no necesita interrogar a la estación por estos datos.

NOTA: Es recomendado que cualquier cambio de datos que se reporte desde una estación sea enviado solicitando una respuesta o confirmación.

Control de flujo:

El flujo de las solicitudes y las respuestas entre el maestro y las estaciones, es controlado por este campo en las cabeceras de las solicitudes y respuestas, con aplicación de tiempos y parámetros. El flujo de mensajes es controlado asi:

- Campo de bit CON: Inicializando / limpiando este bit habilita / deshabilita el mensaje de confirmación de respuesta. Una respuesta de confirmación es una aplicación de reconocimiento de una solicitud previa o mensaje de respuesta.
- Campos de bit FIR y FIN.
- Campo de secuencia (SÉQUENCE): Este numero se usa para ensamblar mensajes fragmentados e identificar cual respuesta corresponde a una solicitud particular.
- Estación maestra u otra estación reporta tiempo expirado (**time – out**):Este especifica cuanto tiempo puede esperar por una respuesta o confirmación antes de retransmitir o abortar la transacción en la capa de aplicación.
- Estación maestra u otra estación reintenta un numero de veces (**retries**): Las aplicaciones pueden tener o no soporte para reintentos; Un contador de reintentos especifica cuantas veces una solicitud o respuesta ha fallado, o cuantas respuestas han sido retransmitidas sin que una respuesta de confirmación se haya recibido.

Cualquier estación puede procesar completamente una respuesta y responder para después iniciar el proceso de una segunda solicitud. **No se permiten solicitudes simultaneas.**

La secuencia de números de todas las solicitudes desde la estación maestra para otra estación es en el rango de o hasta 15 inclusive. La secuencia de números para todas las respuestas no solicitadas desde las estaciones están en el rango de 16 hasta 31 inclusive.

Las siguientes reglas dictan como es la secuencia de números de trabajo:

- La secuencia de números se restablece desde 15 a 0 o de 31 a 16. Cada fragmento de solicitud sucesiva desde la estación maestra DNP incrementará la secuencia de números, la excepción es para reintentos de solicitudes. Para un simple fragmento de reintento de solicitud, la secuencia de números no será incrementada. Para múltiple fragmentación en una solicitud, la secuencia de números del primer fragmento de la solicitud será igual a la secuencia de números del ultimo fragmento que justamente falló.

- Un simple fragmento de repuesta para un simple fragmento de solicitud, tendrá el mismo numero de secuencia que la solicitud.
- La respuesta de confirmación previa solicitud o respuesta tendrá la misma secuencia de números que la solicitud o respuesta.
- El primer fragmento de una respuesta multi - fragmentada para una solicitud multi – fragmentada tendrá la misma secuencia de números como el ultimo fragmento de la solicitud multi – fragmentada. El uso de este esquema de secuencia de números le asegura a las estaciones y a la estación maestra poder hacer frente a todo lo que pueda ocurrir entre lotes de mensaje o retrasos en una red de comunicación.

Las siguientes reglas son obligadas para ambos, estaciones maestras y esclavas:

- Si el sistema usa a nivel de aplicación reintentos, cuando una respuesta no se recibe después del time – out, la solicitud será retransmitida con la misma secuencia de números.
- Si dos mensaje son recibidos con la misma secuencia de números, esto usualmente significará que la respuesta para el mensaje no fue recibida por la estación maestra. En este caso se retransmite la respuesta (reprocesamiento del mensaje si es necesario).
- Si dos respuestas de confirmación son recibidas con la misma secuencia de números se ignora la segunda respuesta.

Solicitudes del maestro, respuestas no solicitadas y colisiones:

Cuando un mensaje no solicitado es generado por una estación esclava, entonces existe la posibilidad de que la estación maestra envíe una solicitud al mismo tiempo que la estación esclava envía un mensaje no solicitado. La estación esclava puede recibir la solicitud cuando espera recibir una confirmación del mensaje no solicitado. El maestro recibe el mensaje no solicitado cuando espera recibir la respuesta a su solicitud.

El procesamiento de este y situaciones similares dependerá del tipo de solicitud usada por la estación maestra.
La estación maestra siempre procesará un mensaje no solicitado de manera inmediata, siempre que el mismo llegue cuando la estación maestra espere una respuesta de una solicitud previa. Una confirmación del mensaje no solicitado es emitida inmediatamente si es solicitado por la estación esclava.

La estación esclava generalmente procesará una solicitud inmediatamente siempre, aunque esta espere por una confirmación de un mensaje no solicitado previo. Todas las solicitudes excepto solicitudes de READ (código de función leer) para datos de sistema serán procesados en este modo. Este modo de operación esta referido como INMEDIATE_PROCESS.

La estación esclava no procesará solicitudes de lectura (READ) si esta esperando por una confirmación de un mensaje no solicitado previo. Esta esperará la confirmación antes de procesar la solicitud. La razón para esta diferencia de funcionamiento es prevenir la perdida o duplicación de los eventos de datos. Este modo de operación es referido como modo PROCESS_AFTER_CONFIRM.

Recuperación de error:

La capa de aplicación de DNP descansa sobre la capa de enlace de datos para la transmisión, recepción y chequeo de error de mensajes. La capa de aplicación no es responsable de recuperaciones de problemas en la comunicación. Si una transacción falla es reportada por la capa de enlace de datos, la capa de aplicación iría fallando las transacciones y reportando el error al usuario. Adicionalmente la capa de aplicación del maestro iría indicando el valor de las indicaciones internas en todas las respuestas de estaciones esclavas. La capa de usuario es responsable para inicializar cualquier clase de procedimiento para recuperación de errores. En particular, el usuario iría construyendo la estrategia a seguir, de el campo IIN (Indicaciones internas) retornados en las respuestas de las estaciones esclavas.

Códigos de función:

Los códigos de función identifican el propósito del mensaje. El tamaño de este campo es un octeto (8 bits). Estos códigos se dividen en dos grupos; uno para solicitudes y otro para respuestas.

CÓDIGO	FUNCIÓN	DESCRIPCION
Códigos de transferencia:		
0	(Confirm) Confirmación	Fragmento de mensaje usado en solicitudes y respuestas. Ninguna respuesta es requerida para este mensaje.
1	(Read) Lectura	Solicitud específica de objetos de estación remota. Responde con objetos requeridos que estén disponibles.
2	(Write) Escritura	Escribe objetos específicos en la estación remota, responde con el estatus de la operación.
Códigos de control:		
3	(Select) Seleccionar	Selecciona o prepara puntos de salida pero no actua (set o clear) ni produce ninguna otra acción (Controlar, inicializar punteros, salidas analógicas), responde con el estatus del punto de control seleccionado. La función OPERATE es requerida para activar estas salidas.
4	(Operate) Operar	Inicializa o produce acciones en las salidas que previamente hayan sido seleccionadas con SELECT. Responde con el estatus de la operación.
5	(Direct Operate) Operación directa	Selecciona y opera la salida específica, responde con el estatus de la operación.
6	(Direct Operate) No Acknowledgment Operación directa sin estatus	Selecciona y opera la salida específica pero no envía una solicitud de respuesta asociada al punto de control.

CÓDIGO	FUNCIÓN	DESCRIPCION
Códigos de congelación:		
7	(Immediate Freeze) Congelar de inmediato	Copia el objeto específico en un buffer congelado y responde con el estatus de la operación.
8	(Immediate Freeze) No Acknowledgment Congelar de inmediato sin estatus	Copia el objeto específico en un buffer congelado; no responde con un mensaje.
Códigos de transferencia:		
9	(Freeze and Clear) Congela y limpia	Copia el objeto especifico en un buffer, luego limpia el objeto, responde con el estatus de la operación.
10	(Freeze and Clear) No Acknowledgment Congela y limpia sin estatus	Copia el objeto especifico en un buffer, luego limpia el objeto, no emite mensaje de respuesta.
11	(Freeze with Time) Congela con tiempo	Copia el objeto específico en un buffer a intervalos de tiempo específico. Responde con el estatus de la operación.
12	(Freeze with Time) No Acknowledgment Congela con tiempo sin estatus	Copia el objeto específico en un buffer a intervalos de tiempo específico. No emite mensaje de respuesta
Códigos de control de aplicaciones:		
13	(Cold Restart) Reiniciar en frio	Efectua la secuencia de reset deseada. Responde con un objeto de tiempo indicando cuando estará habilitada la estación remota.
14	(Warm Restart) Reiniciar en caliente	Efectúa parcialmente la secuencia de reset deseada. Responde con un objeto de tiempo indicando cuando estará lista la estación remota
15	(Initialize Data to Defaults) Inicializar datos por defecto	Inicializa los datos específicos por los valores inicialmente configurados. Responde con el estatus de la operación.
16	(Initialize Application) Inicializar aplicación	Activa la aplicación específica a correr. Responde con el estatus de la operación.
17	(Start Application) Arrancar aplicación	Arranca la corrida de la aplicación específica. Responde con el estatus de la aplicación.
18	(Stop Application) Detener aplicación	Detiene una aplicación específica. Responde con el estatus de la operación.
Códigos de configuración:		
19	(Save Configuration)	Guarda la configuración específica en una memoria no

CÓDIGO	FUNCIÓN	DESCRIPCION
	Guarda configuración	volátil. Responde con un objeto de tiempo indicando el tiempo hasta que este habilitada la estación remota.
20	(Enable Unsolicited Messages) Habilitar mensajes no solicitados	Habilita el reporte instantáneo de los objetos de datos específicos. Responde con el estatus de la operación.
21	(Disable Unsolicited Messages) Deshabilitar mensajes no solicitados	Deshabilita el reporte instantáneo del objeto de datos especifico. Responde con el estatus de la operación.
22	(Assign Class) Asignar clase	Asigna un objeto de datos específico a una clase particular.
Códigs de sincronización de tiempo:		
23	(Delay Measurement) Retardo de medidas	Permite la aplicación para calcular el retardo (retardo de propagación) para una estación remota en particular. El valor calculado por esta función será usado para ajustar el tiempo; desde que se inicializó la estación remota.
Reservados		
24 – 120		Reservados para uso futuro.
121 – 128		Reservado para testear solamente.
Códigos de respuesta		
0	(Confirm) Confirmación	Fragmento de mensaje de confirmación usado entre solicitudes y respuestas. Ninguna respuesta es requerida para este mensaje.
129	(Response) Respuesta	Respuesta a una solicitud.
130	(Unsolicited Message) Mensaje no solicitado	Mensaje no solicitado que no fue requerido por una solicitud.

Indicaciones internas:

El campo de indicaciones internas (IIN) es un campo de dos (2) octetos como se muestra en la Ilustración 7que siguen al código de la función en una respuesta. Cuando una solicitud no puede ser procesada debido a errores de formato o los datos de la solicitud no son validos, el campo de las indicaciones internas siempre devolverá este estado con el conjunto de banderas apropiado.

Ilustración 7 Indicaciones internas.

Primer octeto:

7	6	5	4	3	2	1	0	numero de bit

Un uno (1) en la posición de bit indicada describe los siguientes estados:

Bit “0”

- Todas las estaciones recibieron el mensaje.
- Se inicializa a uno (1) cuando es recibida una solicitud cuya dirección de destino son todas las estaciones (ffff hexadecimal – broadcast).
- Limpiando después en la próxima respuesta (solo si una respuesta global es requerida).
- Usado para permitir a la estación maestra saber que un mensaje difundido fue recibido por esta estación.

Bit “1”

- Datos de clase 1 disponibles.
- Se inicializa a uno (1) cuando los datos que serán configurados como datos de clase 1 son leídos para ser enviados al maestro.
- La estación maestra iría solicitando esta clase de datos desde cada estación esclava cuando éste bit este inicializado en la respuesta

Bit “2”

- Datos de clase 2 disponibles.
- Se inicializa a uno (1) cuando los datos que serán configurados como datos de clase 2 son leídos para ser enviados al maestro.
- La estación maestra iría solicitando esta clase de datos desde cada estación esclava cuando éste bit este inicializado en la respuesta

Bit “3”

- Datos de clase 3 disponibles.
- Se inicializa a uno (1) cuando los datos que serán configurados como datos de clase 3 son leídos para ser enviados al maestro.
- La estación maestra iría solicitando esta clase de datos desde cada estación esclava cuando éste bit este inicializado en la respuesta

Bit “4”

- Sincronización de tiempo requerida al maestro. El maestro sincroniza el tiempo escribiendo el objeto “TIME AND DATE” de la estación remota.
- Limpio cuando la hora es fijada por el maestro. Este bit también esta limpio cuando el maestro escribe explícitamente un 0 en este bit del objeto indicaciones internas de la estación remota.

Bit “5”

- Inicializado significa que algunos o todos los puntos de la salida digital del la estación remota están en el estado local. Es decir, las salidas del control de la estación remota no son accesibles con el protocolo DNP.
- Limpio significa que la estación remota está en el estado remoto. Es decir, las salidas del control de la estación remota son accesibles con el protocolo de DNP..

Bit “6”

- Dispositivo averiado.
- Inicializado cuando una condición anormal existe en la estación remota. El perfil del dispositivo ofrece las condiciones de estado que afectan este bit.
- Este seria utilizado solamente cuando el estado no se puede describir por una combinación de uno o más de los otros bits de IIN.

Bit “7”

- Reinicializar dispositivo.
- Inicializado cuando el usuario de la aplicación desea reinicializar el dispositivo.
- Limpio cuando el maestro explícitamente escribe un 0 en este bit del objeto indicaciones internas en la estación remota.

Segundo Octeto

7	6	5	4	3	2	1	0	numero de bit

Bit “0”

- Bit de función no implementado.

Bit “1”

- Solicitud de objeto desconocido. La estación remota no tiene los objetos especificados o no hay objetos asignados a la clase solicitada.
- Esta indicación se debe utilizar para propósitos de depuración e indica generalmente una unión mal hecha en perfiles del dispositivo o problemas de la configuración.

Bit “2”

- Calificador de parámetros, rango o campos de datos no validos o fuera de rango. Éste es captado por la aplicación que hace la solicitud para detectar errores.
- Esta indicación seria utilizada para propósitos de depuración e indica generalmente problemas de la configuración.

Bit “3”

- Evento de buffer, u otro almacenador intermediario se ha desbordado.
- El maestro debe procurar recuperar tantos datos como sea posible e indicarle al usuario los datos perdidos. Los procedimientos de recuperación apropiados de error se deben iniciar por el usuario.

Bit “4”

- .Solicitud reconocida pero la operación solicitada se esta procesando.

Bit “5”

- Inicializado para indicar que la configuración actual en la estación remota es corrupta y que la capa de aplicación debe informar al usuario esta excepción. El maestro puede descargar otra configuración a la estación remota. Observe que una configuración corrupta inhabilitará a veces a una estación remota, siéndole imposible comunicar esta condición a una estación maestra.

Bit “6”

- Reservado para uso agregado, corrientemente se retorna cero (0).

Bit “7”

- Reservado para uso agregado, corrientemente se retorna cero (0).

Encabezado de objeto:

El encabezado de objeto de un mensaje especifica los datos del objetos(o E/S) que estarán simultáneamente contenidos en el mensaje o serán usados para responder el mensaje (ver Ilustración 8). El formato del encabezado de objeto es idéntico para una solicitud y su respuesta, pero la interpretación del encabezado es dependiente en cuanto si es una solicitud o una respuesta, y cual código de función acompaña al encabezado.

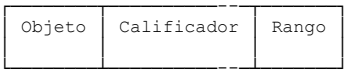


Ilustración 8 Encabezado de objeto.

Objeto: Especifica el grupo y la variación del objeto que sigue al encabezado. Este campo es de dos (2) octetos. El campo de objeto identifica únicamente el tipo o clase del objeto que sigue a la estructura del objeto de dato.

Calificador: Especifica la medida del campo "Rango". Esta compuesto por un (1) octeto. El calificador especifica como el campo de rango será interpretado.

Rango: Indica la cantidad de objetos, arrancando y terminando con índices e identificadores para el objeto en cuestión. Este campo únicamente identifica al objeto en cuestión.

Nota: El campo de Rango puede no ser presentado si el calificador especifica que este no existe. El tamaño de este campo va desde 0 hasta 8 octetos.

Campo de objeto:

El campo de objeto especifica el grupo del objeto y la variación del objeto dentro del grupo (ver Ilustración 9). La combinación de grupo de objetos diferentes, especifica únicamente el objeto al cual se hace referencia. La definición de la estructura de objetos se describe en la librería de datos.

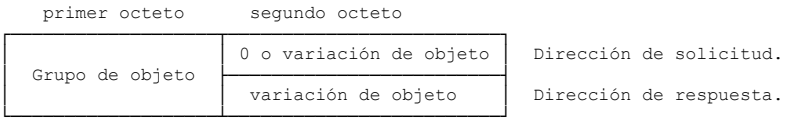


Ilustración 9 Campo de Objeto.

Los objetos pueden ser asignados en una de cuatro clases. Cuando el campo de objeto especifica una clase de datos instanciada de un tipo de objetos específico, el campo de objeto, referencia indirectamente a todos los datos asignados al objeto, para que clase de datos y no para algún tipo específico de objeto.

El campo de objeto es de una longitud de dos (2) octetos, el primer octeto especifica el tipo general de datos (e.j. Entradas analógicas) y el segundo octeto especifica la variación del tipo de datos (e.j. 16 bits entradas analógicas, 32 bits entradas analógicas). En la dirección solicitada, si la variación es especificada como cero (0), esto indica todas las variaciones que un grupo tiene (e.j. Todos o algunos de los tipos de entrada analógicos) En la dirección de respuesta, variación cero (0) no puede ser usada para especificar los objetos, la variación tiene que ser especificada.

El maestro puede inicializar la variación en cero (0) y luego interpretar y reconocer la variación soportada por una estación remota.

Campo calificador:

El campo de calificador especifica la medida del campo de rango, su estructura se muestra en la Ilustración 10.

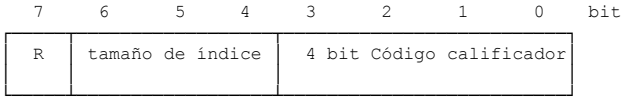


Ilustración 10 Campo calificador.

R: Bit reservado siempre se coloca a acero (0).

El campo de "rango" es usado como índice de datos o como un identificador. La estructura y el uso del campo "rango" es dependiente del valor en el campo de "tamaño de índice" y el "código calificador". Cuando el campo "rango" es usado como índice de datos, este consiste de un valor de arranque y un valor de parada. Ellos juntos definen un rango de objetos siguientes a encabezado de objeto.

Tamaño de índice (3 bits):

En un encabezado de objeto de solicitud cuando el código calificador es igual a "11":

Este formato de tamaño de índice es valido solo cuando el código calificador es 11. Estos bits indican el tamaño, en octetos, de cada entrada en el campo "rango". El campo "rango" sigue al campo calificador. El campo "rango" consiste de índices (para especificar un rango de objetos) o lista de identificadores de objeto.

0 = Invalido con código calificador 11.

1 = Tamaño de identificador 1 octeto.

2 = Tamaño de identificador 2 octetos.

3 = Tamaño de identificador 4 octetos.

4, 5, 6 y 7 = Reservados.

En un encabezado de objetos de una respuesta o solicitud esta es parte de un mensaje que contiene objetos de datos.

Los tres (3) bits de campo de "tamaño de índice" especifica el tamaño de los índices o tamaño prefijado de cada objeto.

0 = Objeto empacado sin índice prefijado.

1 = Objeto prefijado con 1 octeto de índice.

2 = Objeto prefijado con 2 octetos de índice.

3 = Objeto prefijado con 4 octetos de índice.

4 = Objeto prefijado con 1 octeto de tamaño de objeto.

5 = Objeto prefijado con 2 octetos de tamaño de objeto.

6 = Objeto prefijado con 4 octetos de tamaño de objeto.

7 = Reservado.

Código calificador (4 bits):

El código calificador es usado para especificar la medida del campo “rango”.

Sub-campos de arranque y parada en el campo “rango”

El campo “rango” sigue al campo calificador frecuentemente contiene sub-campos (rango de arranque y rango de parada) que designa un rango de valores numéricos de enteros para arrancar desde un rango de arranque (incluyendo el número de rango de arranque) hasta un rango de parada (incluyendo el número de rango de parada). Para códigos calificadores 0, 1 y 2, el rango de arranque y el rango de parada es interpretado como índice de datos. Para códigos calificadores 3, 4 y 5, el rango de arranque y el rango de parada es interpretado como direcciones virtuales de memoria. El código calificador puede ser usado en ambos, tanto en mensajes de solicitudes como en respuestas, como este puede únicamente identificar objetos de datos que existan o no en el mensaje.

El campo de tamaño de índice puede ser 4, 5 o 6 en un mensaje con objetos de datos para indicar que cada objeto de datos (con índice especificado por el campo “rango”) tiene un tamaño prefijado (con este tamaño determinado por el tamaño del índice) un mensaje con datos puede también utilizar tamaño de índice de cero (0) para indicar que no se indexaron mas.

Para un calificador se especifica índices de arranque y parada en el rango, valores de tamaño de índice de 1, 2 y 3 no se pueden usar.

Algunos códigos calificadores se definen:

0 = 8 bits índices de arranque y parada en el campo “rango”.

1 = 16 bits índices de arranque y parada en el campo “rango”.

2 = 32 bits índices de arranque y parada en el campo “rango”.

3 = 8 bits dirección absoluta identificadora del “rango”.

4 = 16 bits dirección absoluta identificadora del “rango”.

5 = 32 bits dirección absoluta identificadora del “rango”.

Todos los objetos ofrecen un tipo de objetos:

Cuando el campo calificador es igual a seis (6), el largo del campo “rango” es cero (0) (no hay campo de rango) porque todos los objetos de datos del tipo específico están siendo referenciados. Este calificador puede ser usado en mensajes con encabezados de objetos solo porque estos pueden identificar un único objeto de datos, si ellos están presentes en el mensaje. El tamaño de índice seria inicializado en cero (0) cuando este código calificador sea usado.

Código calificador = 6 sin campo de rango (implica todos los objetos específicos)

Simple campo de cantidad:

Código calificador 7, 8 y 9 son usados para indicar que el campo “rango” consiste de un simple indicador de cuenta del número de objetos de datos en cuestión. El campo “rango” que sigue designa el numero de objetos referenciados.

Si el tamaño de índice es cero (0), el campo “rango” especifica el número de objetos referenciados arrancando numéricamente desde cero (0) hasta el valor en el campo rango menos uno.

Si el tamaño de índice es 1, 2 o 3 entonces el campo rango especifica el número de índices y objetos que siguen al campo “rango”.

Códigos calificadores 7, 8 y 9, pueden ser usados en las solicitudes y respuestas. En un mensaje con o sin objetos de datos, el valor en el campo “rango” especifica el número de objetos de datos que serán referenciados así. El campo tamaño de índice seria inicializado para el tamaño de los índices que serian prefijados con objetos de datos (para mensajes con objetos de datos) o que forman una lista secuencial de identificadores.

El tamaño del índice no seria indicado en el identificador de tamaño de objeto como este no especificaría únicamente un objeto de datos en cuestión y seria inicializado a cero (0) si no identifica o indica al siguiente.

El orden de identificadores (y objetos de datos opcionales) es arbitrario pero no seria consistente duplicar identificadores.

Algunos códigos calificadores se definen así:

7 = 8 bits simple campo de cantidad.

8 = 16 bits simple campo de cantidad.

9 = 32 bits simple campo de cantidad.

Libre formato de calificador (código calificador = 11)

Este código calificador es usado para objetos específicos para los que los otros códigos calificadores son inadecuados o no proveen bastante información.

Calificador 11 es usado solo cuando el campo “rango” (índice) no puede especificar de manera única el objeto de datos en cuestión. En este caso, el código calificador define un arreglo de variables de octetos (cadenas) que contienen el identificador de objeto. Este identificador tiene un formato libre y no será interpretado en cualquier modo por la capa de aplicación a ese tiempo.

El campo “rango” es siempre un valor de un (1) octeto(contador) que especifica el número de identificadores de objeto, siguiendo al campo rango están los campos de objeto. El tamaño de los identificadores (en octetos) es determinado por el campo tamaño de objeto el cual prefija cada identificador.

El tamaño del campo de “tamaño de objeto” es determinado por el “tamaño de índice”. Tamaño de índice de 4, 5 y 6 serian usados para especificar el tamaño del campo “tamaño de objeto” en octetos.

Códigos calificadores reservados:

Los siguientes códigos calificadores son reservados y no deben usarse:

10, 12, 13, 14 y 15 = Reservados.

Campo “Rango”:

La medida del campo rango es especificada por el campo calificador, para un código calificador cero (0) hasta cinco (5) el campo de divide en dos sub-campos específicos un índice o dirección de arranque y uno de parada. Incluyendo los valores en estos campos. El rango no esta presente para un código calificador de 6. El rango es un simple campo que indica una cantidad para códigos calificadores 7,8,9 y 11.

La siguiente figura muestra un esquemático de cada una de las partes de la trama de aplicación así como las posibles opciones.