UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA PRUEBAS PROTOCOLARES DE UN TRANSFORMADOR

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

BILLY JAMES PORRAS BOLAÑOS

PROMOCIÓN
2009-I

LIMA-PERÚ
2013

IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS PARA PRUEBAS PROTOCOLARES DE UN TRANSFORMADOR

A Dios por llenarme de bendiciones.

A mi familia por brindarme su apoyo y entera confianza.

A mi esposa por ser mi compañera incondicional.

A mi universidad por darme los conocimientos necesarios.

A todos mis amigos por brindarme su sincera amistad

SUMARIO

El presente trabajo desarrolla la implementación de un sistema de adquisición de datos para pruebas protocolares de un transformador.

Este diseño era necesario por cuanto se requería optimizar la realización de pruebas protocolares de un transformador, dado que el proceso previo a la solución era manual y demandaba de un operador adicional. Los datos debían se registrados manualmente por un técnico y luego los cálculos necesarios ser realizados en una hoja Excel por un ingeniero.

La solución presentada es realizada utilizando herramientas libres. Para dicho fin se aplicó Programación Orientada a Objetos (OOP), lenguajes de programación como C#, tecnología de comunicación (OPC y Modbus RTU, aplicando conversor de estándares a RS232) y elementos relacionados a bases de datos (MySQL y ODBC).

Preliminarmente se identificó el procedimiento de las pruebas protocolares y el equipo con el que la empresa contaba para realizar sus mediciones, para luego formular la arquitectura de la solución así como la codificación de la interfaz HMI.

INTRODUCCIÓN

El trabajo surge por la necesidad de una empresa de optimizar la realización de pruebas protocolares de los transformadores.

Para las pruebas de transformadores, se requiere visualizar y almacenar los valores de corriente, tensión, potencia, etc. de manera constante y automática. La finalidad es la de verificar que las pérdidas en el transformador no excedan los valores mínimos permitidos, establecidos por el estándar IEC-60076.

Para las pruebas se necesita energizar al transformador, para seguidamente medir las corrientes y tensiones y registrar los datos obtenidos de manera simultánea.

Dado esto, se debían realizar varias acciones simultaneas para estas pruebas, para ello era necesario contar con 2 operadores, uno para que se encargue de manipular la energización del transformador y monitorear las variables (corriente y tensión) y otro para que guarde los datos en un cuaderno para su registro, y se entregue al ingeiero para los cálculos respectivos.

El trabajo desarrollado se enfoca en el diseño e implementación de la aplicación HMI y del sistema de adquisición de datos, para lo cual se establecido que, desde su concepción hasta su puesta en servicio, sea realizado tres meses por una sola persona.

La solución es realizada utilizando herramientas libres: Programación Orientada a Objetos (OOP), lenguajes de programación como C#, tecnología de comunicación (OPC y Modbus RTU) y elementos relacionados a bases de datos (MySQL y ODBC). Cuyos conceptos son tratados en el informe.

El informe está organizdo de la siguiente manera

- Capítulo I "Planteamiento de Ingeniería del problema".- En donde se explica el problema de ingeniería y se precisan los objetivos del informe de suficiencia. También se hace una evaluación de la problemática y se establecen los alcances del proyecto desarrollado. Se concluye con una síntesis de la solución desarrollada.
- Capítulo II "Marco teórico conceptual".- En donde se desarrollan de manera detallada, todos los aspectos teóricos necesarios para comprender las diferentes tecnologías involucradas en el sistema: Tecnología OPC (OLE for Process Control), Protocolo Modbus, Estándares de comunicación serial, Pruebas de cortocircuito en el transformador, Plataforma .NET para desarrollo de HMI, MySQL, El medidor de potencia

PM130

- Capítulo III "Metodología para la solución del problema.- En donde se enfoca a exponer el diseño del sistema de adquisición de datos para pruebas protocolares de un transformador. Para ello preliminarmente se realiza el planteamiento de la solución, en el cual se especifica los requerimientos, alternativas de solución y dimensionamiento del proyecto. Posteriormente se describe el diseño y la implementación de la solución.

Capítulo IV "Análisis y presentación de resultados". En donde se analiza los costos y las tareas realizadas.

CAPÍTULO I
PLANTEAMIENTO DE INGENIERÍA DEL PROBLEMA

En este capítulo se explica el problema de ingeniería y se precisan los objetivos del informe de suficiencia. También se hace una evaluación de la problemática y se establecen los alcances del proyecto desarrollado. Se concluye con una síntesis de la solución desarrollada.

1.1 Descripción del problema

Necesidad de optimizar la realización de pruebas protocolares de un transformador. El proceso previo a la solución era manual y requería de un operador adicional. Los datos eran anotados manualmente por un técnico y los cálculos realizados en una hoja Excel por un ingeniero.

1.2 Objetivos del trabajo

Implementar un sistema de adquisición de datos para la realización de pruebas protocolares de un transformador.

1.3 Evaluación del problema

Los transformadores requieren de diversas pruebas para verificar sus características. Aunque son muchas las pruebas que pueden ser realizadas en un transformador, según lo especifica el estándar IEC 60076 [1], el proyecto que se explica en este informe se enfoca en dos pruebas, la de vacío y la de corto circuito.

Nota: IEC 60076 es un estándar al diseño de transformadores de potencia. El IEC 60076 posee varias partes (documentos) que van desde las generalidades hasta la profundización de cada aspecto relacionado al tema, tal como corto circuito, sobrecargas, calentamiento, ruido, ensayo de impulso, guía de aplicación, entre los más importantes. Dado su extensión solo se hará referencia puntual a lo correspondiente.

Durante estas pruebas se requiere visualizar y almacenar los valores de corriente, tensión, potencia, etc. de manera constante y automática. Esto con el fin de comprobar que las pérdidas en el transformador no sobrepasen los valores mínimos permitidos según el estándar IEC-60076.

En estas pruebas se tiene que energizar al transformador, medir las corrientes y tensiones y recolectar datos de manera simultánea. Debido a que se deben realizar varias acciones simultaneas para estas pruebas, muchas veces se debía contar con 2 operadores, un operador para que se encargue de manipular la energización del transformador y monitoree las variables (corriente y tensión) y el otro operador para que

guarde los datos en un cuaderno para su registro.

Para la medición de las corrientes y tensiones la empresa adquirió un medidor de energía PM130EH de la marca SATEC con displays el cual le permite al operador visualizar hasta tres variables simultáneamente. El problema que se presenta es que se requiere monitorear por lo menos diez variables (3 corrientes, 3 tensiones, potencia activa y frecuencia) y adicionalmente recolectar datos de las variables.

Como se describió líneas arriba, la limitante de no tener un visor en el equipo de medición y no contar con un sistema de adquisición de datos, no le permitía al operador monitorear todas sus variables y almacenarlas de manera simultánea como lo exige la prueba, es por eso que el operador solo registraba manualmente los valores de corriente, tensión, potencia y frecuencia y luego se lo entregaba al ingeniero para que realice los cálculos correspondientes.

Debido a que esto implicaba tener que consumir el tiempo de dos personas para estas pruebas, la empresa optó por invertir en una manera de optimizar este procedimiento y no tener que contar con dos personas sino con uno solo el cual se dedique a manipular la energización de los transformadores y monitorearlos sin la necesidad de registrarlos manualmente en una hoja de reporte.

Por los motivos expuestos, es que la empresa decidió el desarrollo de un sistema que permita reducir los costos de horas–hombre para estas pruebas al máximo posible. Por tal motivo es que este proyecto justificó su desarrollo.

Con el propósito de ilustrar mejor la problemática, a continuación se desarrollan dos secciones. La primera a orientar las pruebas a realizar y la segunda a explicar la situación previa a la solución y la propuesta de desarrollo.

1.3.1 Pruebas requeridas en el transformador

Como se indicó, son dos las pruebas en las que se enfoca la solución desarrollada en este informe: La prueba de cortocircuito y la prueba de vacío.

a. Prueba de cortocircuito

Orientado a la medida de la impedancia de cortocircuito y pérdidas en carga (o pérdidas en el cobre). Esta prueba se realiza para determinar las pérdidas en los bobinados y la tensión de cortocircuito.

La prueba consta en alimentar por uno de los arrollamientos (por lo general en el lado de alta) con una corriente entre el 50 al 100% de la corriente nominal estando el otro arrollamiento en cortocircuito.

Una vez llegada a dicha corriente, el operador procede a tomar nota de las corrientes medidas en las 3 fases, así como las tensiones y la potencia activa total consumida. Cabe resaltar que esta prueba se realiza a frecuencia nominal.

La Figura 1.1 ilustra dicha prueba.

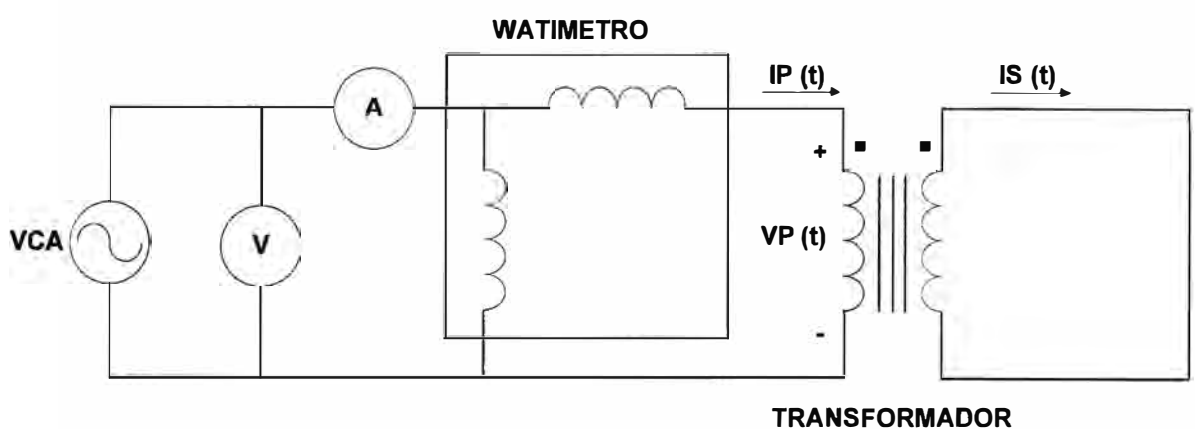

Figura 1.1 Prueba de cortocircuito (Fuente: Elab. Propia)

Con los datos obtenidos (corriente en 3 fases, tensiones y potencia) se debe realizar una serie de cálculos los cuales, dada su extensión, son detallados en el capítulo siguiente.

b. Prueba en vacío

También denominada prueba de pérdidas sin carga. Esta prueba se realiza para determinar las pérdidas en el hierro.

La prueba consta en alimentar el transformador en el lado de alta con el voltaje nominal y a frecuencia nominal mientras que el lado de baja se encuentra a circuito abierto, pudiendo medir las pérdidas en el hierro y la corriente de excitación.

Finalmente se miden las corrientes en las 3 fases, las tensiones entre fases y la potencia consumida, para luego agregarlo en el protocolo de pruebas. En esta prueba no se realiza ningún cálculo adicional puesto que los valores son los que indica el PM130EH.

$$W_{FE}=W_{PM130} \quad (1.1)$$

1.3.2 Situación inicial y propuesta de desarrollo

El esquema presentado en la Figura 1.2 ilustra la situación previa a la solución para la realización de la recopilación de datos de las pruebas en el transformador.

(1) Debido a la limitante que tiene el equipo PM130, el operador solo podía visualizar 3 variables simultáneamente y para poder visualizar las otras 7 tenía que manipular el equipo, tomándole así más tiempo en la prueba y corriendo el riesgo de poder desconfigurarlo.

(2) Con los datos de corriente (3 fases), tensión (entre fases) y potencia por fase y total; recogidos MANUALMENTE por el operador se enviaban al ingeniero para que realice los cálculos de pérdidas.

(3) El otro inconveniente de este contexto es que se tenía que invertir tiempo del ingeniero para realizar estos cálculos.

(4) El ingeniero realizaba los cálculos de pérdidas en el transformador y los guardaba en el servidor de protocolos

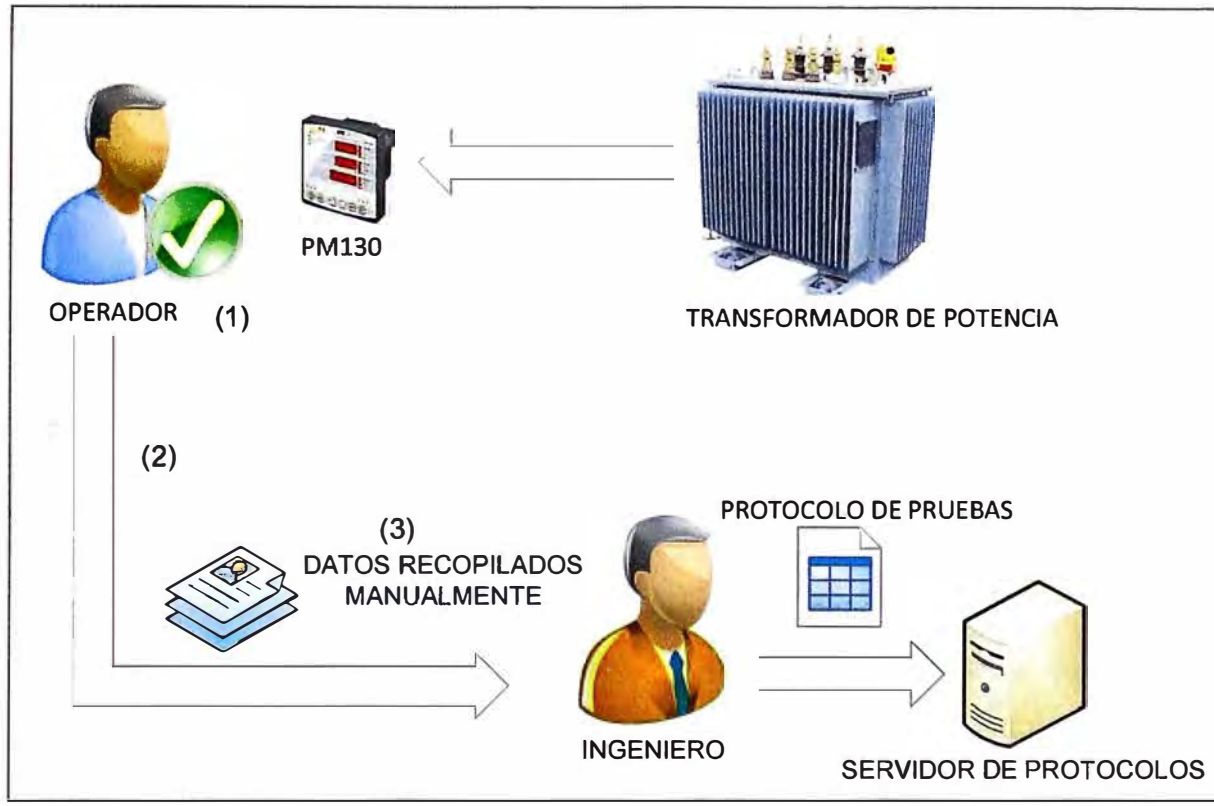

Figura 1.2 Esquema de situación previa a la solución (Fuente: Elaboración propia)

El esquema presentado en la Figura 1.3 muestra la propuesta general de la solución.

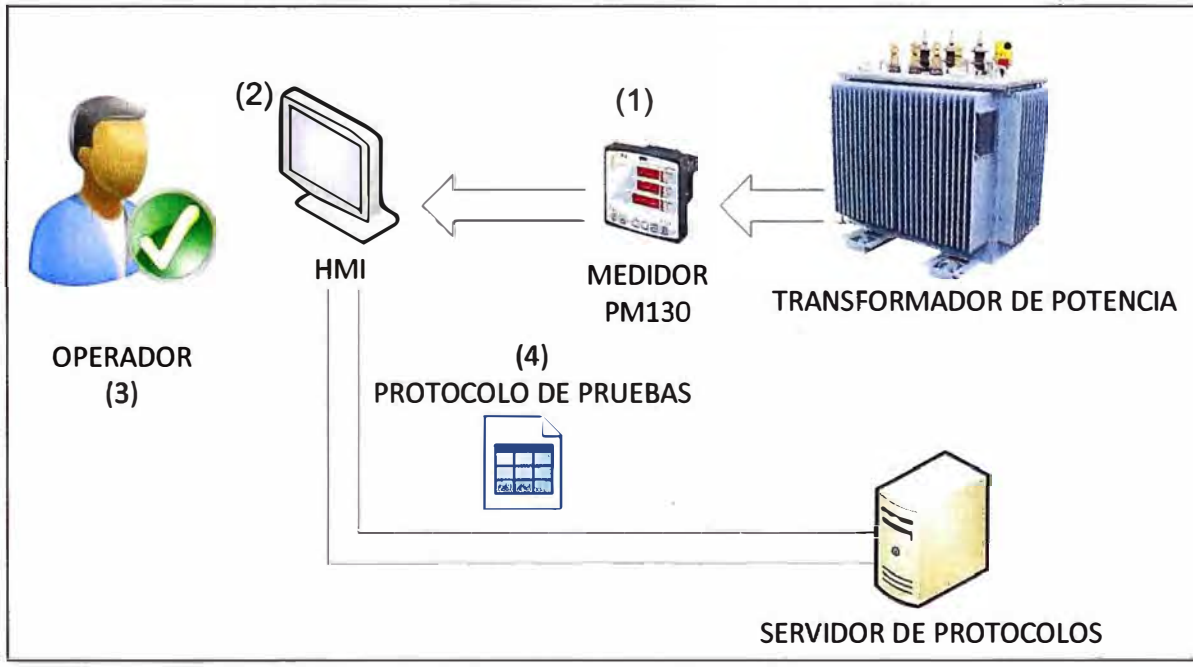

Figura 1.3 Esquema general de la solución (Fuente: Elaboración propia

(1) El PM130 posee la capacidad de enviar información (Corriente de las tres fases, voltaje entre fases, potencia y frecuencia)

(2) Se diseña un HMI para que el operador pueda visualizar todas sus variables en

tiempo real y de manera simultánea.

(3) El operador ahora puede visualizar las corrientes (3 fases), voltajes (entre fases), potencia por fase, potencia total (activa) y frecuencia simultáneamente sin la necesidad de manipular el PM130.

(4) Finalmente la aplicación diseñada genera el protocolo de pruebas de manera automática y los almacena en el servidor de protocolos.

1.4 Alcance del trabajo

El trabajo se enfoca en el diseño e implementación de la aplicación HMI y del sistema de adquisición de datos.

El presente proyecto es realizado utilizando herramientas libres. Para dicho fin se aplicó Programación Orientada a Objetos (OOP), lenguajes de programación como C#, tecnología de comunicación (OPC y Modbus RTU) y elementos relacionados a bases de datos (MySQL y ODBC). La explicación de estas herramientas se realiza en el siguiente capítulo.

El tiempo establecido para el desarrollo, desde su concepción hasta su puesta en servicio, fue de tres meses para una sola persona.

1.5 Síntesis del trabajo

Lo primero que se hizo fue identificar el procedimiento de las pruebas protocolares e identificar el equipo con el que la empresa contaba para realizar sus mediciones.

Finalmente se desarrolló la codificación del HMI implementado, efectuándose las pruebas pertinentes a fin de que se pusiera en marcha el sistema que dé solución al requerimiento de la empresa.

La Figura 1.4 ilustra la arquitectura del sistema implementado. El esquema mostrado se divide en tres áreas: Buses de campo, Sala de Pruebas, Oficina Administrativa.

- Buses de Campo.- En primer término se puede apreciar al equipo PM130 que envía la información en protocolo Modbus RTU haciendo uso del estándar serial RS485. En esta sección se considera como mejora la incorporación de un controlador de temperatura, el cual, sin embargo, no es parte de la solución desarrollada.
- Sala de Pruebas.- Como es conocido, la comunicación serial con una PC se realiza mediante el estándar RS232, por ello es imprescindible hacer uso de un conversor de estándares. Además, para que se pueda desarrollar el HMI, es imprescindible la utilización del "Servidor OPC", un software que opera sobre una PC, cuya funcionalidad es la de servir de intérprete del Modbus RTU. El OPC en sí es un estándar desarrollado para la conectividad de datos. En este caso el HMI es el Cliente que solicita datos de la Fuente, es decir del servidor OPC (OLE for Process Control).
- Oficina administrativa.- En la oficina administrativa se encuentra el servidor de

protocolos y un cliente OPC para que el ingeniero pueda revisar los protocolos realizados. Estos dispositivos se comunican utilizando el protocolo OPC sobre Ethernet.

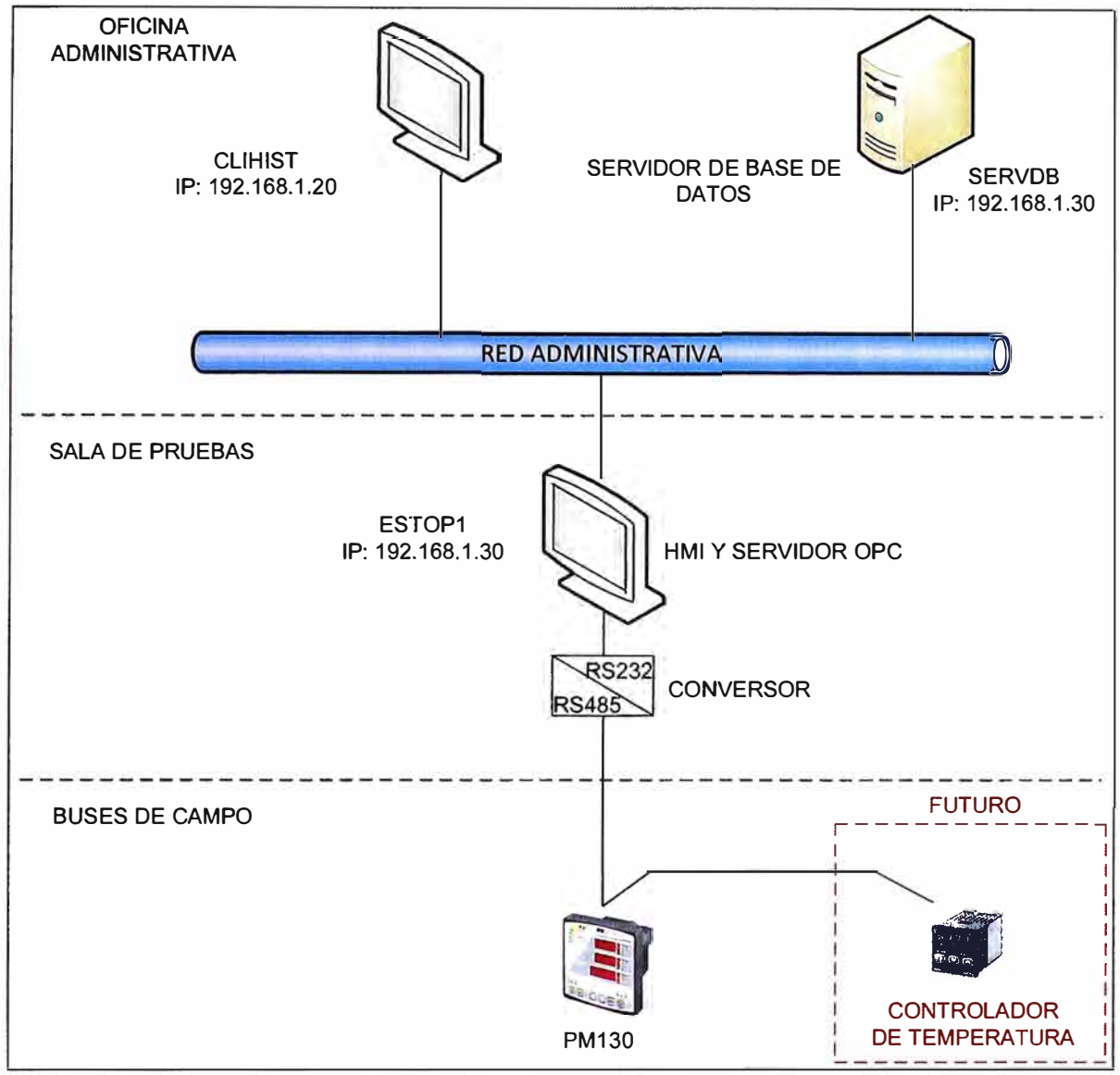

Figura 1.4 Arquitectura de la solución (Fuente: Elaboración propia)

Los aspectos relevantes a tecnología OPC, Protocolo Modbus RTU, estándares de comunicación serial, conceptos generales de herramientas/lenguaje programación usados, y las pruebas de Cortocircuito en el transformador, son desarrollados en el capítulo Marco Teórico Conceptual.

Todos los cálculos mencionados aquí y en el capítulo II, además de las mediciones de corriente, voltaje y potencia serán registrados de manera automática por el sistema y almacenados en la base de datos.

CAPÍTULO II
MARCO TEÓRICO CONCEPTUAL

Para comprender el sistema propuesto, es necesario conocer ciertos aspectos teóricos sobre las tecnologías y protocolos utilizados en este sistema y que son piezas fundamentales. Esto con el fin de tener un mejor conocimiento y entendimiento del funcionamiento del sistema y como sus componentes interactúan entre sí para lograr el objetivo.

En este capítulo se desarrollan de manera detallada, todos los aspectos teóricos necesarios para comprender las diferentes tecnologías involucradas en el sistema.

2.1 Tecnología OPC (OLE for Process Control)

OPC es el método de conectividad de datos basado en los estándares más populares del mundo. Es utilizado para responder a uno de los mayores retos de la industria de la automatización: cómo comunicar dispositivos, controladores y/o aplicaciones sin caer en los problemas habituales de las conexiones basadas en protocolos propietarios.

OPC no es un protocolo, es un estándar para la conectividad de datos que se basa en una serie de especificaciones OPC gestionadas por la OPC Foundation [2]. La fundación la define como el estándar de interoperabilidad para la automatización industrial y dominios relacionados.

Cualquier software que sea compatible con estas especificaciones OPC proporciona a usuarios e integradores conectividad abierta e independiente tanto del fabricante del dispositivo como del desarrollador de la aplicación Cliente.

Es una solución abierta y flexible al clásico problema de los drivers propietarios. Prácticamente todos los principales fabricantes de sistemas de control, instrumentación y de procesos han incluido OPC en sus productos.

La comunicación OPC se realiza a través de una arquitectura Cliente-Servidor. El servidor OPC es la fuente de datos (como un dispositivo hardware a nivel de planta), y cualquier aplicación basada en OPC puede acceder a dicho servidor para leer/escribir cualquier variable que ofrezca el servidor.

Los Servidores OPC se muestran como un nivel intermedio entre la fuente de datos y el cliente de datos, habilitando la intercomunicación sin que ningún lado conozca el protocolo nativo del otro. La Figura 2.1 ilustra ello.

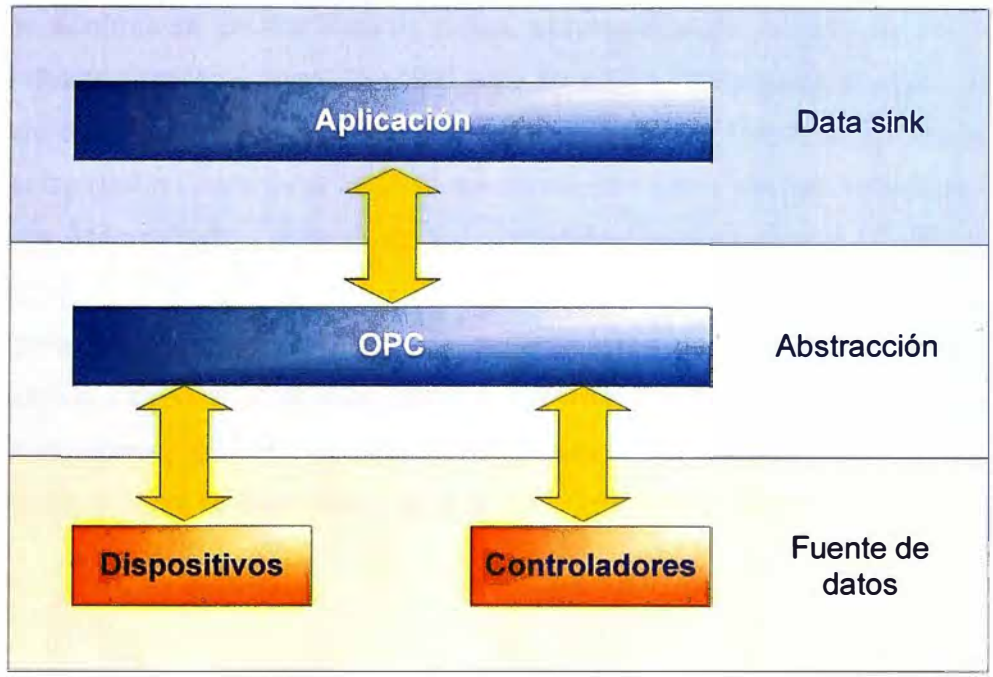

Figura 2.1 Flujo de información entre la aplicación y la fuente de datos (Fuente: Ref. [2])

Nota: Data Sink se refiere a un dispositivo capaz de aceptar señales de datos desde una dispositivo de transmisión de datos y almacenarlos para futuro uso, en este caso la fuente de datos

2.1.1 Problemática para compartir información

Las distintas industrias utilizan distintos dispositivos especializados, sistemas de control y aplicaciones, las cuales tienen como desafío común el compartir información entre todos estos componentes y el resto de la empresa.

A continuación se describen los factores que causaban los mayores problemas al compartir información, así como el impacto de OPC sobre cada uno de ellos:

a. Protocolos propietarios

Los fabricantes utilizaban frecuentemente protocolos que permitían a productos de una determinada gama comunicarse entre ellos, sin embargo requerían protocolos personalizados para comunicarse con productos de otros fabricantes, además las distintas gamas de productos del mismo fabricante frecuentemente no eran capaces de comunicarse entre sí, necesitando conectores adicionales.

OPC resuelve ese problema haciendo innecesario que el cliente de datos tenga que conocer como se comunica el servidor de datos o como organiza dichos datos.

b. Drivers de comunicación propietarios

Todas las conexiones punto a punto requerían un protocolo propietario para posibilitar la comunicación entre los extremos específicos. Por ejemplo, si un HMI necesitaba comunicarse con un PLC, se requería de un driver en el HMI escrito específicamente para el protocolo utilizado por el PLC. Si los datos de este PLC necesitaban ser

registrados además en un histórico de datos, el programa de registro de datos requería su propio driver porque el driver del HMI solo se podía utilizar para el HMI y no para el software de registro histórico de datos (que necesitaría otro Driver propietario diferente). Si el driver específico para establecer la comunicación entre los dos extremos no estaba previamente desarrollado y disponible, las comunicaciones eran muy difíciles y costosos de establecer.

La Figura 2.2 ilustra el problema de controladores personalizados, se aprecia que cada aplicación requiere un dispositivo o un controlador de protocolo específico que le permita comunicarse con los respectivos dispositivos. Los drivers de comunicación no son reutilizables entre aplicaciones, ya que cada aplicación utiliza su propio formato de datos.

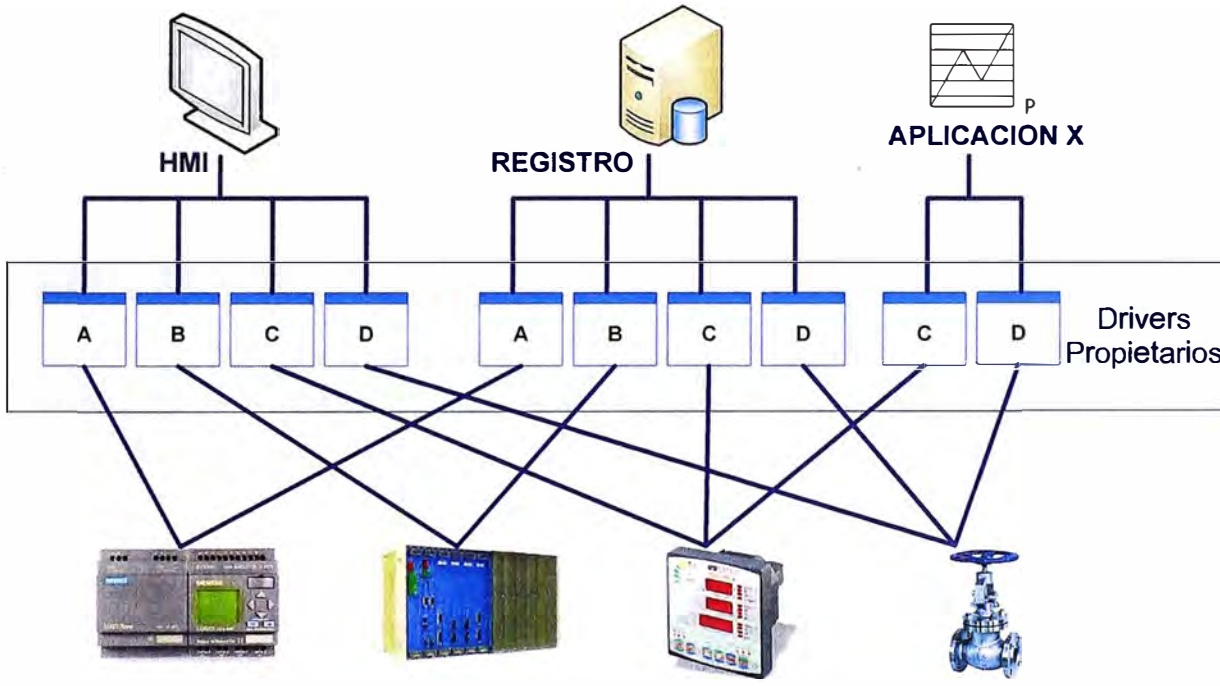

Figura 2.2 Problema del uso de Drivers propietarios (Fuente: Ref. [2])

c. Integración compleja

El uso habitual de protocolos propietarios para cada dispositivo significaba que, incluso con un pequeño número de dispositivos y aplicaciones, se requería necesariamente el uso de muchos protocolos.

La misma aplicación HMI ejecutándose en múltiples ordenadores, comunicándose con el mismo dispositivo, requería instalar y configurar múltiples instancias del mismo protocolo en cada ordenador. Si las aplicaciones HMI se comunicaban a su vez con dispositivos adicionales, cada HMI requería su propio conjunto de drivers para cada uno de los dispositivos. El mantenimiento de las versiones de las aplicaciones era un problema.

OPC simplifica enormemente la integración porque, una vez que se configura un

Servidor OPC para una Fuente de Datos en particular, todas las aplicaciones que utilizan OPC pueden empezar a compartir datos con esa Fuente de Datos, eliminando la necesidad de drivers adicionales.

d. Carga de trabajo en el dispositivo

Cada protocolo establece su propia conexión al dispositivo o controlador para el que está diseñado para comunicar. Dado el gran número de protocolos que se utiliza en una instalación típica, frecuentemente el controlador se veía bombardeado por múltiples peticiones de la misma información para cada aplicación que necesitaba comunicarse con él. Además, la mayoría de dispositivos solo pueden aceptar un número limitado de conexiones simultaneas. Si el número de drivers tratando de conectarse a un dispositivo excedía el número máximo de conexiones, había que buscar soluciones adicionales para que las aplicaciones Cliente no se quedasen sin datos.

El tráfico y, por lo tanto, la carga de los dispositivos se reduce enormemente utilizando conectores OPC, porque un conector OPC específico para un dispositivo requiere una única conexión a la Fuente de Datos mientras que puede comunicarse simultáneamente con múltiples aplicaciones para enviarles los Datos obtenidos.

e. Obsolescencia de infraestructuras

A medida que los fabricantes lanzan nuevos productos, eventualmente dejan de dar soporte a los antiguos Cuando una nueva versión de HMI o SCADA sale al mercado, es posible que requiera su propio juego de protocolos que, en ocasiones, dejan de soportar comunicaciones con dispositivos con los que la anterior versión de HMI o SCADA se comunicaban.

OPC extiende la vida útil de sistemas antiguos porque, una vez que se ha configurado un servidor OPC para el sistema, permite que cualquier aplicación Cliente que utilice OPC (la mayoría) pueda comunicarse con el sistema antiguo, sin importar si la aplicación Cliente soporta o no de forma nativa la comunicación con dicho sistema antiguo. Por lo tanto, OPC permite que nuevas aplicaciones Cliente continúen comunicándose con los sistemas antiguos.

f. Conectividad corporativa

A medida que crece la necesidad de disponer de datos procedentes de la automatización en otros niveles de la empresa, los problemas de conexión se hacen más complejos, porque las aplicaciones empresariales no están diseñadas para comunicarse con dispositivos y controladores. Esto puede añadir, potencialmente, una carga extra a la infraestructura de automatización y sumar preocupaciones adicionales de seguridad.

OPC hace posible de forma real que se puedan compartir datos provenientes de la automatización a lo largo de toda la red corporativa, permitiendo que aplicaciones

validadas reciban datos con Fuentes de Datos de la red de automatización, eliminando así la necesidad de instalar nuevos drivers de comunicación. Todo lo que se requiere es un servidor OPC.

2.1.2 Arquitectura Cliente/Servidor OPC

Se puede representar como una capa de "abstracción" intermedia que se sitúa entre la Fuente de Datos y el Cliente de Datos, permitiéndoles intercambiar datos sin saber nada el uno del otro.

La Figura 2.3 muestra la arquitectura Cliente/Servidor OPC tomando como ejemplo de origen de datos al Modbus. Se puede destacar dos componentes: el Cliente OPC y el Servidor OPC. La especificación OPC define el mensaje entre estos dos componentes.

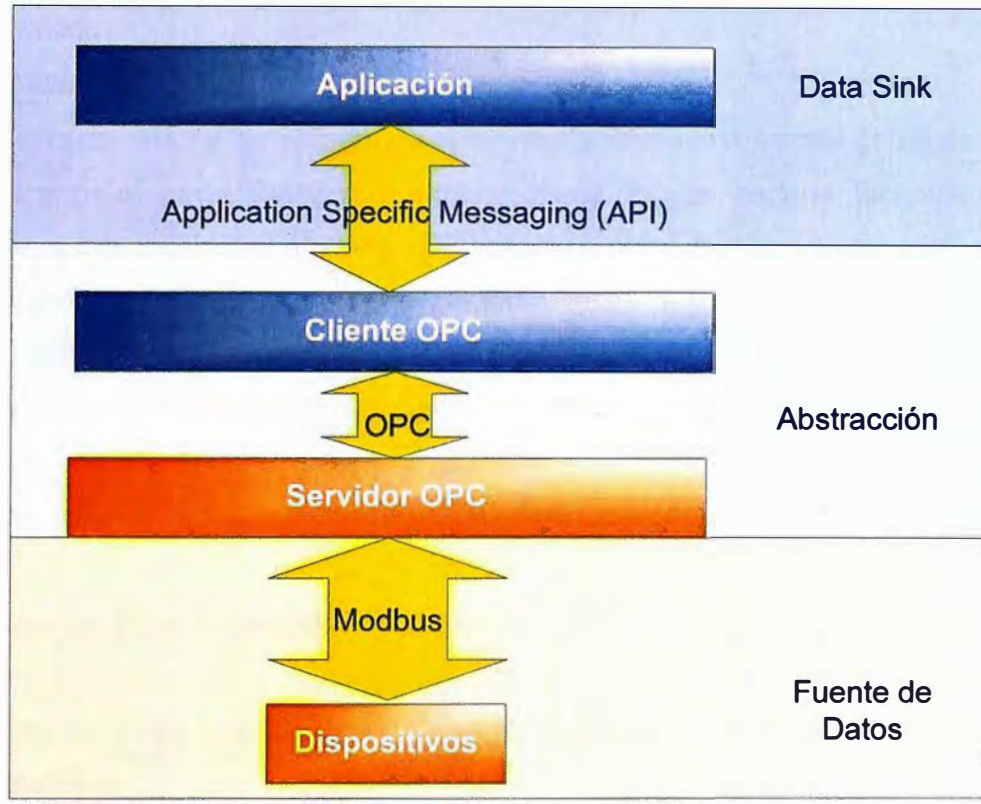

Figura 2.3 Arquitectura Cliente/Servidor OPC (Fuente: [2])

2.1.3 Beneficios de utilizar conectividad OPC

A continuación se enumeran algunos de los beneficios clave de utilizar OPC:

- Una aplicación Cliente OPC puede comunicarse libremente con cualquier Servidor OPC visible en la red sin la necesidad de utilizar un driver específico para la fuente de datos.
- Las aplicaciones cliente OPC pueden comunicarse con tantos Servidores OPC como necesiten. No hay ninguna limitación inherente a OPC en el número de conexiones que se pueden establecer.
- Hoy en día OPC está tan extendido que hay un servidor OPC disponible para prácticamente todos los dispositivos nuevos o antiguos que existen en el mercado.
- Las Fuentes de Datos (hardware o software) que utilizan OPC pueden ser

intercambiadas o actualizadas sin la necesidad de actualizar los drivers utilizados por cada aplicación que se comunique con ellas mediante OPC. Solo hay que mantener actualizado el servidor OPC para esa Fuente de Datos.

- Los usuarios pueden elegir libremente los dispositivos, controladores y aplicaciones que mejor se ajusten a sus proyectos sin preocuparse del fabricante del que provienen o de si se comunican entre sí. La intercomunicación se da por hecho.

2.1.4 Tipos de datos que soporta OPC

Los tipos de datos más comunes transferidos entre dispositivos, controladores y aplicaciones en automatización se pueden encuadrar en tres categorías:

- Datos de tiempo real.
- Datos históricos.
- Alarmas y eventos.

A su vez, cada una de las categorías anteriores soporta una amplia gama de tipos de datos. Estos tipos de datos pueden ser enteros, coma flotante, cadena, fechas y distintos tipos de arrays, por mencionar algunos. OPC asume el reto de trabajar con estas distintas categorías de datos especificando de forma independiente como se va a transmitir cada uno de ellos a través de la arquitectura Cliente OPC – Servidor OPC.

Las tres especificaciones OPC que se corresponden con las tres categorías de datos son:

a) OPC Data Access Specification (OPC DA): Utilizada para transmitir datos de tiempo real.

b) OPC Historical Data Access Specification (OPC HDA): Utilizada para transmitir datos históricos.

c) OPC Alarm & Events Specification (OPC A&E): Utilizada para transmitir información de alarmas y eventos.

No es obligatorio que todos los Servidores OPC incluyan todas las especificaciones OPC. Históricamente, la mayoría de los Servidores OPC sólo soportan datos de tiempo real (OPC DA). Es aconsejable investigar que especificaciones OPC incluye un Servidor OPC antes de elegirlo para un proyecto.

Es importante saber cuáles especificaciones OPC incluye un Cliente y Servidor OPC ya que ambos deben incluir las mismas especificaciones OPC para coordinar de forma correcta el flujo de datos entre ellos y para trabajar correctamente con la Fuente de Datos y el Cliente de Datos respectivamente.

2.1.5 Servidores OPC

Un Servidor OPC es una aplicación de software. Es un driver “estandarizado” desarrollado específicamente para cumplir con una o más especificaciones OPC. La

palabra "Server" en "OPC Server" no hace referencia en absoluto al ordenador donde este software se estará ejecutando. Hace referencia a la relación con el Cliente OPC.

Los servidores OPC son conectores que se pueden asimilar a traductores entre el mundo OPC y los protocolos nativos de una Fuente de Datos.

OPC es bidireccional, esto es, los servidores OPC pueden leer y escribir en una Fuente de Datos. La relación Servidor OPC / Cliente OPC es de tipo maestro esclavo, lo que significa que un servidor OPC solo transferirá datos de/a una Fuente de Datos si un Cliente OPC así se lo pide.

Los servidores OPC pueden comunicarse prácticamente con cualquier Fuente de Datos cuyos datos puedan ser leídos o escritos por medios electrónicos. Una breve lista de posibles Fuentes de Datos incluye: dispositivos, PLCs, DCSs, RTUs, instrumentos de medición, bases de datos, historizadores, software de cualquier tipo (ejem: Excel), páginas web e incluso archivos CSV (texto separado por comas) de actualización automática. Para comunicarse con cualquiera de estos dispositivos se requiere únicamente el uso de un servidor OPC que utilice el protocolo o interfaz nativo apropiado. Una vez que se ha configurado dicho servidor OPC, cualquier aplicación Cliente que utilice OPC (y tenga los permisos adecuados) puede empezar a comunicar con la Fuente de Datos sin que importe la forma en que esta se comunica de forma nativa.

La Figura 2.4 muestra el esquema de un Servidor OPC. Conceptualmente un servidor OPC se puede desglosar en tres módulos: módulo de comunicaciones OPC, módulo de traducción / mapeado, y un módulo de comunicaciones nativas

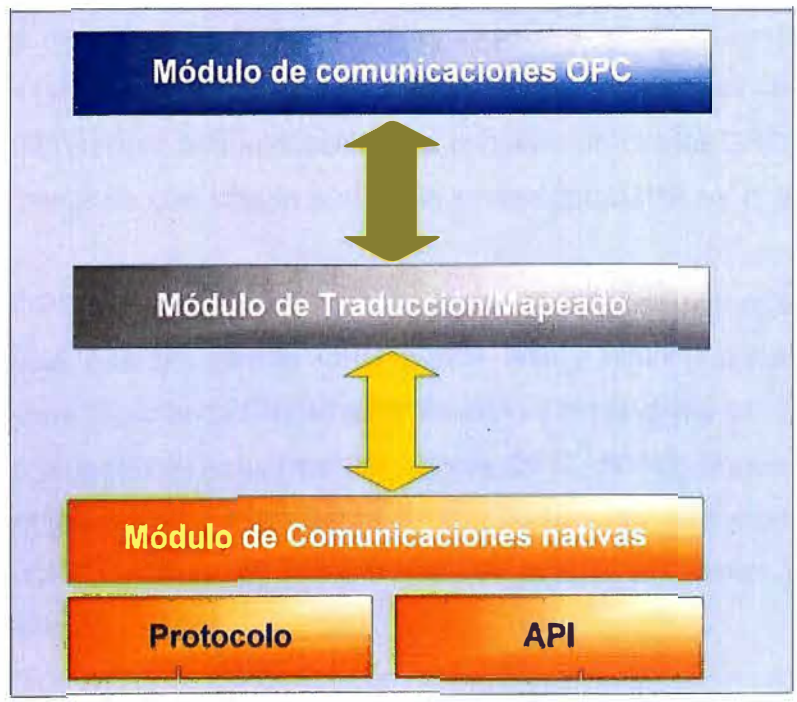

Figura 2.4 Esquema de un Servidor OPC (Fuente Ref. [2]).

a) Módulo de comunicaciones OPC: Parte del Servidor OPC responsable de comunicarse

adecuadamente con un cliente OPC. Los Servidores OPC bien diseñados deben ser plenamente compatibles con las especificaciones OPC que implementen, para asegurar la correcta comunicación con cualquier Cliente OPC.

b) Módulo de comunicaciones nativas: El servidor OPC debe emplear el método de comunicación más eficiente con la Fuente de Datos. En algunos casos, esto implica comunicarse con la fuente mediante su protocolo propietario de datos, mientras que en otros casos, esto significa comunicarse a través de una Interfaz de Programación de la Aplicación (API). Típicamente, cuanta más experiencia tenga el desarrollador del Servidor OPC con el dispositivo, mejor utilizará las posibilidades de comunicación que ofrece el dispositivo.

c) Módulo de traducción/mapeado: Aquí es donde sucede toda la "magia" de un Servidor OPC. La función de este módulo es interpretar de forma adecuada las peticiones OPC entrantes de un Cliente OPC, convirtiéndolas en peticiones nativas que se envían a la Fuente de Datos y viceversa. Si esto se hace eficientemente, se puede mantener al mínimo la carga sobre la Fuente de Datos mientras se maximiza la capacidad de transmisión de datos.

2.1.6 Clientes OPC

Un Cliente OPC es una pieza de software creada para comunicarse con Servidores OPC. Utiliza mensajería definida por una especificación concreta de la OPC Foundation.

Son módulos de software utilizados por una aplicación para permitirle comunicarse con cualquier Servidor OPC compatible visible en la red. Típicamente, los Clientes OPC están embebidos en aplicaciones como HMIs, SCADAs, graficadores, historizadores o generadores de informes, convirtiéndolos en aplicaciones compatibles OPC.

Es muy común referirse a la aplicación que contiene un cliente OPC embebido como "Cliente OPC" a pesar de que solo la parte que implementa OPC es el verdadero Cliente OPC.

Los clientes OPC pueden comunicarse de forma simultánea con múltiples Servidores OPC. Esto significa que un Cliente OPC puede leer y escribir datos desde y hacia múltiples dispositivos (Fuente de Datos) a través de sus respectivos Servidores OPC.

La Figura 2.5 muestra el esquema del Cliente OPC. Análogamente al Servidor, un Cliente OPC también puede ser considerado compuesto por tres módulos: Módulo de comunicaciones OPC, Módulo de comunicaciones con la aplicación y un Módulo de traducción/mapeado.

a) Módulo de comunicaciones OPC: Aunque no tan involucrado como en el Servidor OPC (en los Servidores OPC esta parte es más compleja) es crucial para que el Cliente OPC se comporte como debe al conectarse a un Servidor OPC, intercambiar datos con él y

desconectarse sin desestabilizar al Servidor OPC.

b) Módulo de comunicaciones con la aplicación: El Cliente OPC típicamente está diseñado para trabajar en una aplicación específica, por lo que, para permitir que la información pase de la aplicación al Servidor OPC pasando por el Cliente OPC, realiza una serie de llamadas al interfaz para la programación de la aplicación (API). También es posible que un Cliente OPC genérico se comunique con una aplicación mediante un protocolo en lugar de utilizar al API siempre que la aplicación soporte ese producto.

c) Módulo de traducción / mapeado: Una de las funciones clave del Cliente OPC es la traducir de forma bidireccional la información que su aplicación necesita leer de o escribir al dispositivo o Fuente de Datos.

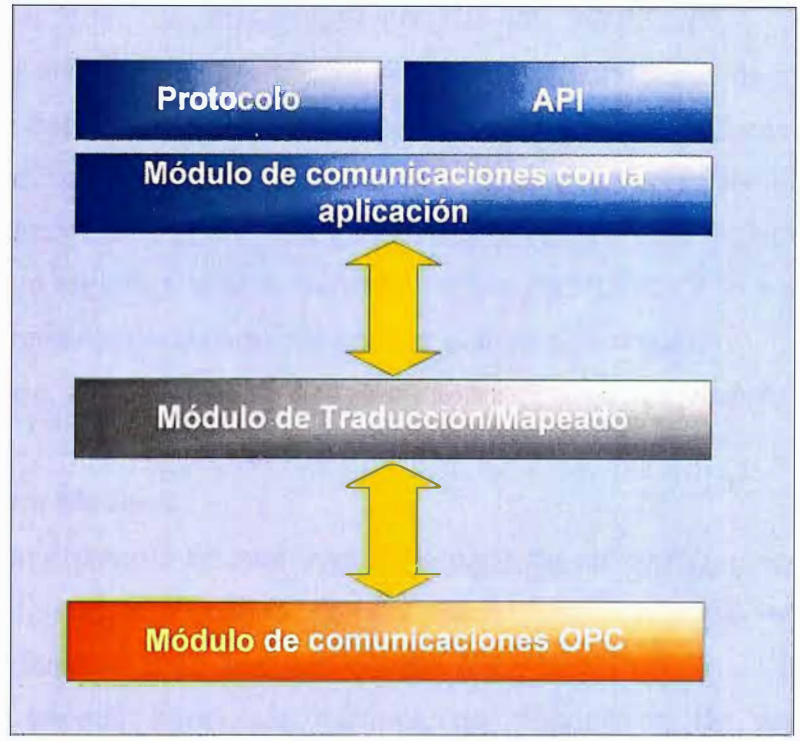

Figura 2.5 Esquema de un Cliente OPC (Fuente Ref. [2]).

En OPC las comunicaciones Cliente-Cliente no están definidas. Sólo se soporta la arquitectura Cliente/Servidor. Por ello, si una aplicación debe proveer datos OPC a otros clientes, necesita tener su propio Servidor OPC. Este Servidor OPC permitirá a otros Clientes OPC de otras aplicaciones utilizar esta aplicación como Fuente de Datos.

Los Clientes OPC, típicamente, están embebidos en la aplicación que los utiliza, como por ejemplo HMIs o historiadores. Si por alguna razón la aplicación que se tiene que utilizar no dispone de un Cliente OPC embebido, es posible que se pueda obtener uno externo del fabricante de la aplicación o de un tercero.

Un Cliente OPC externo a la aplicación típicamente se comunicará con ella a través de uno de sus protocolos nativos. En este caso, el Cliente OPC podría incluso no residir en el mismo ordenador que la aplicación.

2.2 Protocolo Modbus

El protocolo de comunicaciones industriales Modbus fue desarrollado en 1979 por la empresa norteamericana Modicon y debido a que es público, relativamente sencillo de implementar y flexible, se ha convertido en uno de los protocolos de comunicaciones más populares en sistemas de automatización y control. A parte de que muchos fabricantes utilizan este protocolo en sus dispositivos, existen también versiones con pequeñas modificaciones o adaptadas para otros entornos (como p.ej Jbus o Modbus II).

Modbus especifica el procedimiento que el controlador y el esclavo utilizan para intercambiar datos, el formato de estos datos, y como se tratan los errores. No especifica estrictamente el tipo de red de comunicaciones a utilizar, por lo que se puede implementar sobre redes basadas en Ethernet, RS-485, RS-232 etc.

En esta parte del informe se explicara la especificación Modbus de forma general, sin entrar en mucho detalle en algunas de sus particularidades, no obstante su contenido ha de ser más que suficiente para aquellos que deseen comprender el funcionamiento general de este estándar. Por otro lado, quien desee realizar una implementación precisa de Modbus deberá recurrir a la documentación oficial de MODICON o a la documentación especifica del fabricante de los equipos con los que va a comunicar.

A continuación se desarrollan dos temas principales: La Arquitectura Modbus y Tramas Modbus

2.2.1 Arquitectura Modbus

Modbus es un protocolo de mensajes de la capa de aplicación, ubicado en el nivel 7 del modelo OSI, que proporciona comunicación maestro/esclavo entre dispositivos conectados en diferentes tipos de buses o redes [3].

Modbus ha servido para que millones de dispositivos de automatización se comuniquen. Hoy en día, el soporte a la estructura simple y elegante de Modbus sigue creciendo. La comunidad de Internet puede acceder a Modbus en el puerto 502 del sistema reservado en la pila TCP/IP.

Modbus es un protocolo de solicitud/respuesta y ofrece los servicios especificados en los códigos de función.

Los códigos de función Modbus son elementos del PDU de solicitud/respuesta de Modbus.

En la actualidad se implementa mediante:

- TCP/IP sobre Ethernet.
- Transmisión serial Asíncrona sobre varios medios (cable: EIA/TIA-232-E, EIA-422, EIA/TIA-485-A, fibra, radio, etc.).

La Figura 2.6 muestra la pila de comunicación Modbus.

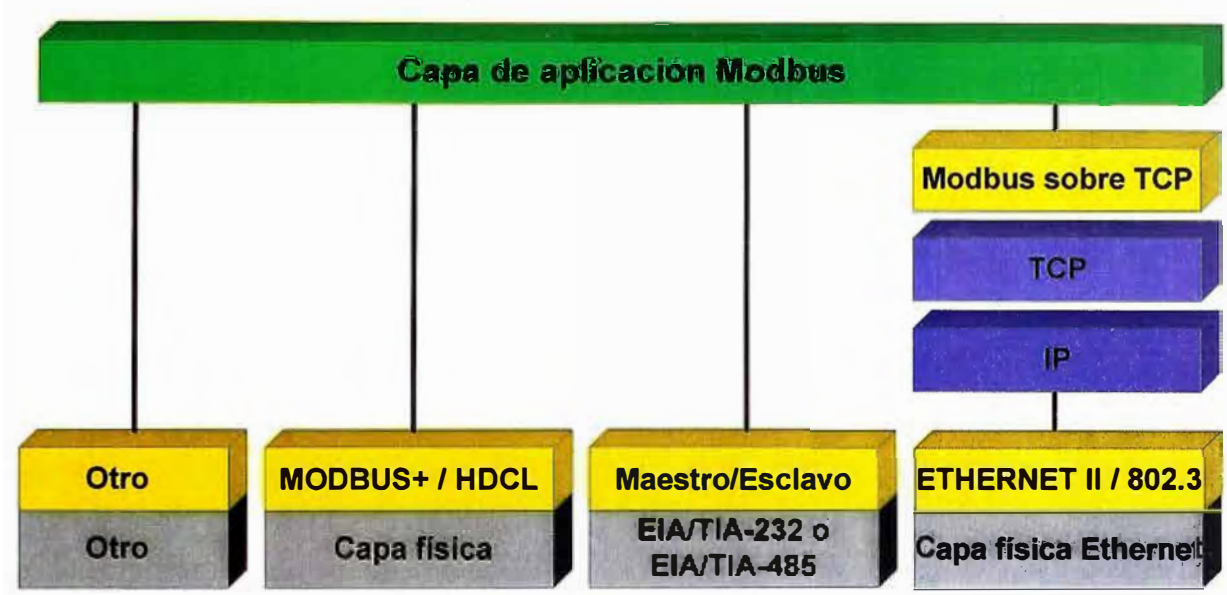

Figura 2.6 Pila de comunicación MODBUS (Fuente Ref. [3])

El protocolo Modbus permite una fácil comunicación dentro de todos los tipos de arquitectura de red.

Cualquier tipo de dispositivo de campo (PLC, HMI, Panel de Control, I/O remotos, etc.) pueden usar el protocolo Modbus para iniciar una operación remota.

La misma comunicación se puede hacer en forma serial o como una red Ethernet TCP/IP. Los Gateways permiten una comunicación entre varios tipos de buses o red, mediante el protocolo Modbus. La Figura 2.7 muestra un ejemplo de la arquitectura de red Modbus (Fuente Ref. [3])

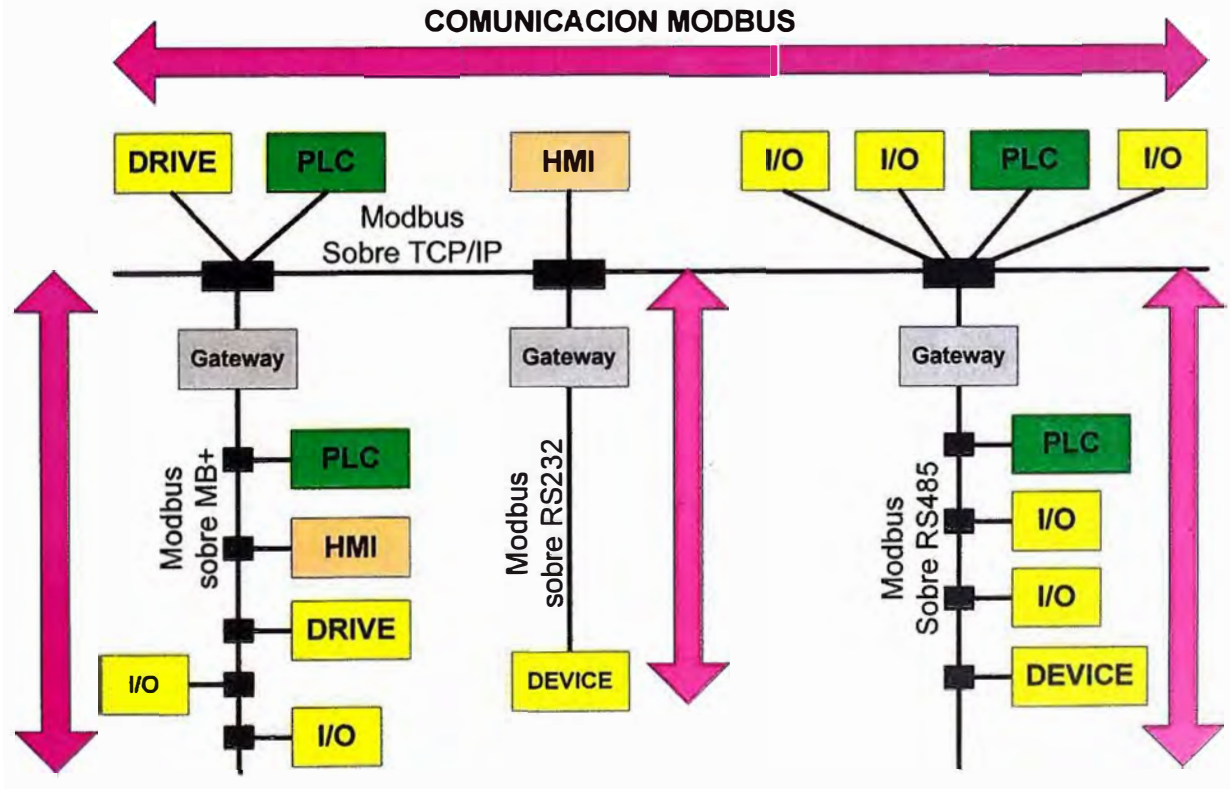

Figura 2.7 Arquitectura de red Modbus (Fuente Ref. [3])

2.2.2 Tramas Modbus

Es necesario explicar la información que contiene las tramas Modbus. En esta sección se especifican y describen los campos de las tramas Modbus

a. Campos de las tramas

El número de campos de las tramas Modbus varía ligeramente dependiendo de si se utiliza la codificación ASCII o RTU.

a.1 Codificación ASCII (formato texto):

- Inicio de trama: 2 caracteres ASCII (que representan 1 byte) codificando el caracter ":" (0x3A)
- Nº Esclavo: 2 caracteres ASCII (que representan 1 byte) codificando la dirección del esclavo destino (o origen) de la trama
- Código Operación: 2 caracteres ASCII (que representan 1 byte) con el código de operación.
- Dirección, datos y subfunciones Datos: con los parámetros necesarios para realizar la operación.
- LRC(16): H L
- Final de trama: 4 caracteres ASCII (que representan 2 bytes) con los caracteres CR (0x0D) - LF (0x0A)

a.2 Codificación RTU

En el formato binario, el inicio de trama debería ser tras 3.5 tiempo de carácter:

- Nº Esclavo: 1 byte con la dirección del esclavo destino (u origen) de la trama
- Código Operación: 1 byte con el código de operación
- Subfunciones Datos: con los parámetros necesarios para realizar la operación.
- CRC(16): H L

b. Descripción de los campos de las tramas

Se explica a continuación: Número de Esclavo; código de Operación o Función; Dirección, datos y subfunciones; y finalmente Control de errores LRC o CRC

b.1 Número de Esclavo

De 1 byte de longitud. En el caso de las tramas enviadas por el máster, el campo de número de esclavo indica la dirección del destinatario de esta trama. Permite direccionar hasta 247 esclavos, con las direcciones de 1d a 247d (0x00 a 0xF7). El 0x00 es para los mensajes de Broadcast, así el primer esclavo comienza con la dirección 1 (de 1 a 247). En el caso de las tramas enviadas por los esclavos, este byte sirve para indicar al máster a quién pertenece la respuesta. Es decir, cada vez que un esclavo responde, sitúa su propia dirección en el byte de dirección lo que permite saber al maestro a que equipo corresponde cada respuesta. Las tramas broadcast, no tienen asociada respuesta, y

algunas implementaciones de MODBUS no admiten la trama de broadcast.

b.2 Código de Operación o Función

De 1 byte de longitud. Indica el tipo de operación que se quiere realizar sobre el esclavo. Las operaciones se pueden clasificar en dos tipos:

- De lectura/escritura en memoria: para consultar o modificar el estado de los registros del mapa de memoria del esclavo.
- Órdenes de control del esclavo: para realizar alguna actuación sobre el esclavo.

El código de operación puede tomar cualquier valor comprendido entre el 0 y el 127 (el bit de más peso se reserva para indicar error). Cada código se corresponde con una determinada operación. Algunos de estos códigos se consideran estándar y son aceptados e interpretados por igual por todos los dispositivos que dicen ser compatibles con Modbus, mientras que otros códigos son implementaciones propias de cada fabricante. Es decir que algunos fabricantes realizan implementaciones propias de estos códigos "no estándar".

Es también mediante el código de función que el esclavo confirma si la operación se ha ejecutado correctamente o no. Si ha ido bien responde con el mismo código de operación que se le ha enviado, mientras que si se ha producido algún error, responde también con el mismo código de operación pero con su bit de más peso a 1 (0x80) y un byte en el campo de datos indicando el código de error que ha tenido lugar.

A continuación se describen las funciones que el Maestro ordena al Esclavo realizar error

- **Función 1 o 2 (1 Read Coil Status - 2 Read Input Status):** Permite realizar la lectura del estado de las DIs (@1XXXX el comando 2-Read input status) o DOs (@0XXXX el comando 1-Read Coil Status). Para ello el maestro solicita el número de bits que desea leer a partir de una determinada dirección. Cada dirección se corresponde con un registro de 1 bit con el estado de la entrada digital. El esclavo responde indicando el número de bits que retorna y sus valores. En la trama de respuesta se aprovechan todos los bits del byte, y puede haber hasta 256 bytes.
- **Función 3 o 4 (3 Read Holding Registers – 4 Read Input Registers)**: Permite realizar la lectura del valor de las AIs (@4XXXX el comando 3 Read Holding Registers) o AOs (@3XXXX el comando 4 Read Input Registers). El máster indica la dirección base y número de palabras a leer a partir de esta, mientras que el esclavo indica en la respuesta el número bytes retornados, seguido de estos valores. Aunque en realidad se está escribiendo en el rango de registros o valores numéricos, los registros son direccionados a partir de la dirección 0 (así el registro @40001 se direcciona 0).
- **Función 5 (Force Single Coil):** Permite modificar el estado de una DO del esclavo

(mando o relé). Es decir mediante este comando podemos modificar algún bit de alguna de las variables internas del esclavo u ordenar la ejecución o activación de un mando. Actúa sobre la zona de memoria de los DOs @0XXXX. El Maestro especifica la dirección del bit o mando que quiere modificar seguido de 0x00 para ponerlo a 0 o 0xFF para ponerlo a 1. El esclavo responde con una trama similar indicando la dirección que ha modificado y el valor que ha establecido en el bit o mando.

- **Función 6 (Preset Single Register)**: Permite la escritura en las AOs del esclavo (ya sea una señal o valor interno del equipo), y por tanto actúa sobre la zona de memoria de las AOs (@4XXXX). Deberemos indicar la dirección del valor que queremos modificar y la magnitud que queremos asignarle. Luego el esclavo debería responder con la dirección del dato que ha modificado y el valor que le ha asignado, que debería coincidir con el enviado. Aunque en realidad se está escribiendo en el rango de AOs, los registros son direccionados a partir de la dirección 0 (así el registro @40001 se direcciona 0)

Como se mencionó, cuando se produce un error en la ejecución de un comando en el esclavo, este responde poniendo a 1 el bit de más peso del código de función (0x80). Con este bit el maestro sabe que se ha producido un error. Para obtener más detalle sobre el tipo de error, comprueba el campo de datos (código de errores en la Tabla 2.1).

Tabla 2.1 Código de errores (Fuente Ref. [3])

Código	Nombre	Significado
01	ILLEGAL FUNCTION	El código de función recibido no se corresponde a ningún comando disponible en el esclavo
02	ILEGAL DATA ADRESS	La dirección indicada en la trama no se corresponde a ninguna dirección válida del esclavo
03	ILLEGAL DATA VALUE	El valor enviado al esclavo no es válido
04	SLAVE DEVICE FAILURE	El esclavo ha recibido la trama y la ha comenzado a procesar, pero se ha producido algún error y no ha podido termina la tarea.
05	ACKNOWLEDGE	El esclavo ha recibido la trama y la está procesando pero esto le llevará un periodo un poco largo. Mediante esta respuesta se evita que el máster considere un error de timeout. El máster podrá enviar más tarde una trama una trama de tipo Poll Program Complete para verificar si ha completado el comando
06	SLAVE DEVICE BUSY	El esclavo está ocupado realizando otra tarea y no puede atender a esa petición en ese instante por lo que el máster tendrá que reintentarlo más adelante.

b.3 Dirección, datos y subfunciones

De longitud variable (n bytes). Este campo contiene la información necesaria para realizar la operación indicada en el código de operación. Cada operación necesitará de unos parámetros u otros, por lo que el número de bytes de este campo variará según la operación a realizar. En el caso del esclavo, este puede responder con tramas con o sin campo de datos dependiendo de la operación. En los casos en que se produzca algún error es posible que el esclavo responda con un byte extra para especificar el código de error.

Al establecer la dirección de una variable u otro elemento en el mapa de direcciones Modbus, se direcciona con 1 unidad menos a la del registro al que se quiere acceder, de manera que si p.ej. se deseara acceder al relé @ 127d, esto se haría situando el valor 126d en el byte del campo de dirección. Otros ejemplos:

- El relé número 1 de un controlador se direccionaría con el valor 0000 en el campo de dirección de un mensaje MODBUS.
- El relé 0x007F (127d) de un controlador se direccionaría con el valor 0x007E (126d) en el campo de dirección de un mensaje MODBUS.
- El Holding Register 40001 se accedería situando el valor 0000 en el campo de dirección del mensaje. Como se puede ver el código de función de acceso a los Holding Registers lleva implícito el acceso a la dirección '4XXXX'.
- El Holding Register 40108 es accedido leyendo de la dirección 0x006B (107d).

Generalmente en Modbus cada tipo de dato se mapea en un rango de memoria concreto:

- @1-10000 (DOs - digital outputs): 1 bit por dirección para indicar el estado de una salida, mando o relé (0 desactivado, 1 activado). Las direcciones de este rango se suelen acceder mediante las funciones 1 (lectura), 5 (escritura), 15 (escritura múltiple).
- @10001-20000 (DIs - digital inputs): 1 bit por dirección para leer el estado de una entrada digital (0 desactivada, 1 activada) también denominadas DIs (Digital Inputs). Las direcciones de este rango se suelen acceder con la función 2 (lectura) y llevan implícita la dirección 10001 como dirección base para acceder a una dirección bastará con especificar la distancia entre esta y la dirección base).
- @20001-30000: el protocolo Modbus estándar no hace uso de este rango de direcciones.
- @30001-40000 (AIs - analog inputs): 16 bits por dirección con el estado de las medidas o entradas analógicas también denominadas AIs (Analog Inputs). Dependiendo del dispositivo este puede hacer uso de más de un registro para almacenar la información de la medida, así con 2 registros consecutivos podríamos almacenar medidas de 32 bits. Las

direcciones de este rango se acceden mediante la función 4 (lectura) y llevan implícita la dirección 30001 como dirección base (para acceder a una dirección bastará con especificar la distancia entre esta y la dirección base).

- @40001-50000 (AOs - analog outputs): 16 bits con los registros de salidas analógicas o de propósito general (Output Registers – Holding Registers). Se acceden con las funciones 3 (lectura), 6 (escritura) o 16 (escritura múltiple) y llevan implícita la dirección 40001 como dirección base (para acceder a una dirección bastará con especificar la distancia entre esta y la dirección base).

Algunos fabricantes expresan la dirección de forma compuesta, separando la dirección de palabra y la dirección de bit: p.ej word 0x30 bit 1.

b.4 Control de errores LRC o CRC

Se utiliza un sistema de detección de errores diferente dependiendo del tipo de codificación utilizado (ASCII o RTU). En el caso de la codificación ASCII es el checksum (o Longitud Redundancy Check LRC) en módulo 16 expresado en ASCII (2 caracteres representan 1 byte), sin considerar el ":" ni el “CR LF” de la trama. En la codificación RTU se utiliza el método de CRC (Cyclical Redundancy Check) codificado en 2 bytes (16 bits).

Para calcular el CRC se carga un registro de 16 bits todo con '1's, se hace OR con cada uno de los caracteres de 8 bits con el contenido de cada byte y el resultado se desplaza una bit a la izquierda insertando un 0 en la posición de menos peso (la de la derecha). El de la izquierda se extrae y se examina: si es 1 se vuelve a hacer OR con un valor prefijado, si es 0 no se hace ninguna OR y el proceso se repite hasta que se han hecho los 8 shifts (desplazamientos) del byte.

2.3 Estándares de comunicación serial

RS232, RS422, RS423 y RS485 son métodos de comunicación en serie para ordenadores y dispositivos. RS232 es sin duda la mejor interfaz conocida, ya que esta interfaz serial se lleva a cabo en casi todos los ordenadores disponibles en la actualidad. Pero algunas de las otras interfaces son ciertamente interesantes, ya que se puede utilizar en situaciones en las que RS232 no es apropiado. En esta sección se desarrolla lo relacionado a los estándares utilizados en el proyecto: RS232 y RS485.

2.3.1 Estándar RS232

RS232 (Recommended Standard 232, también conocido como Electronic Industries Alliance RS-232C) es una interfaz que designa una norma para el intercambio de una serie de datos binarios entre un DTE (Equipo terminal de datos) y un DCE (Data Communication Equipment, Equipo de Comunicación de datos), aunque existen otras en las que también se utiliza.

En particular, existen ocasiones en que interesa conectar otro tipo de equipamientos,

como pueden ser computadores. Evidentemente, en el caso de interconexión entre los mismos, se requerirá la conexión de un DTE (Data Terminal Equipment) con otro DTE. Para ello se utiliza una conexión entre los dos DTE sin usar módem, por ello se llama: null módem o módem nulo.

El RS-232 consiste en un conector tipo DB-25 (de 25 pines), así como de 9 pines (DB-9). En la Tabla 2.2 se muestran las señales RS-232 más comunes según los pines asignados (desde la perspectiva del DTE):

La interfaz RS-232 está diseñada para imprimir documentos para distancias cortas, de hasta 15 metros según la norma, y para velocidades de comunicación bajas, de no más de 20 kbps.

Tabla 2.2 Señales del estándar RS232 (Fuente:[4])

SEÑAL		DB-25	DB-9
Common Ground	G	7	5
Transmitted Data	TD	2	3
Received Data	RD	3	2
Data Terminal Ready	DTR	20	4
Data Set Ready	DSR	6	6
Request To Send	RTS	4	7
Clear To Send	CTS	5	8
Carrier Detect	DCD	8	1
Ring Indicator	RI	22	9

A pesar de esto, muchas veces se utiliza a mayores velocidades con un resultado aceptable. La interfaz puede trabajar en comunicación asíncrona o síncrona y tipos de canal simplex, half duplex o full duplex. En un canal simplex los datos siempre viajarán en una dirección, por ejemplo desde DCE a DTE. En un canal half duplex, los datos pueden viajar en una u otra dirección, pero sólo durante un determinado periodo de tiempo; luego la línea debe ser conmutada antes que los datos puedan viajar en la otra dirección. En un canal full duplex, los datos pueden viajar en ambos sentidos simultáneamente. Las líneas de handshaking de la RS-232 se usan para resolver los problemas asociados con este modo de operación, tal como en qué dirección los datos deben viajar en un instante determinado.

Si un dispositivo de los que están conectados a una interfaz RS-232 procesa los datos a una velocidad menor de la que los recibe deben de conectarse las líneas handshaking que permiten realizar un control de flujo tal que al dispositivo más lento le dé tiempo de procesar la información. Las líneas de "handshaking" que permiten hacer este control de flujo son las líneas RTS y CTS. Los diseñadores del estándar no concibieron estas líneas para que funcionen de este modo, pero dada su utilidad en cada interfaz

posterior se incluye este modo de uso.

Las UART o U(S)ART (Transmisor y Receptor Asíncrono Universal) se diseñaron para convertir las señales que maneja la CPU y transmitirlas al exterior. Las UART deben resolver problemas tales como la conversión de voltajes internos del DCE con respecto al DTE, gobernar las señales de control, y realizar la transformación desde el bus de datos de señales en paralelo a serie y viceversa. Debe ser robusta y deberá tolerar circuitos abiertos, cortocircuitos y escritura simultánea sobre un mismo pin, entre otras consideraciones. Es en la UART en donde se implementa la interfaz.

Generalmente cuando se requiere conectar un microcontrolador (con señales típicamente entre 3.3 y 5 V) con un puerto RS-232 estándar se utiliza un driver de línea, típicamente un MAX232 o compatible, el cual mediante dobladores de voltaje positivos y negativos permite obtener la señal bipolar (típicamente alrededor de +/- 6V) requerida por el estándar.

Para los propósitos de la RS-232 estándar, una conexión es definida por un cable desde un dispositivo al otro. Hay 25 conexiones en la especificación completa, pero es muy probable que se encuentren menos de la mitad de éstas en una interfaz determinada. La causa es simple, una interfaz full duplex puede obtenerse con solamente 3 cables.

Sobre los circuitos, todos los voltajes están con respecto a la señal de tierra (desbalanceado). Las convenciones que se muestran en la Tabla 2.3:

Tabla 2.3 Convenciones de niveles lógicos vs eléctricos (Fuente: [4])

Voltaje	Señal	Nivel Lógico	Control
+3 a +15	Espacio	0	On
-3 a –15	Marca	1	Off

Los valores de voltaje se invierten con respecto a los valores lógicos. Por ejemplo, el valor lógico positivo corresponde al voltaje negativo. También un 0 lógico corresponde a la señal de valor verdadero o activado. Por ejemplo si la línea DTR está al valor 0 lógico, se encuentra en la gama de voltaje que va desde +3 a +15 V, entonces DTR está listo (ready).

El canal secundario a veces se usa para proveer un camino de retorno de información más lento, de unos 5 a 10 bits por segundo, para funciones como el envío de caracteres ACK o NAK, en principio sobre un canal half duplex. Si el módem usado acepta esta característica, es posible para el receptor aceptar o rechazar un mensaje sin tener que esperar el tiempo de conmutación, un proceso que usualmente toma entre 100 y 200 milisegundos.

Los siguientes criterios son los que se aplican a las características eléctricas de cada

una de las líneas:

- La magnitud de un voltaje en circuito abierto no excederá los 25 V.
- El conductor será apto para soportar un corto con cualquier otra línea en el cable sin daño a sí mismo o a otro equipamiento, y la corriente de cortocircuito no excederá los 0,5 A.
- Las señales se considerarán en el estado de MARCA, (nivel lógico "1"), cuando el voltaje sea más negativo que - 3 V con respecto a la línea de Signal Ground. Las señales se considerarán en el estado de ESPACIO, (nivel lógico "0"), cuando el voltaje sea más positivo que +3 V con respecto a la línea Signal Ground. La gama de voltajes entre -3 V y +3 V se define como la región de transición, donde la condición de señal no está definida.
- La impedancia de carga tendrá una resistencia a DC de menos de 7000 Ω al medir con un voltaje aplicado de entre 3 a 25 V pero mayor de 3000 Ω cuando se mida con un voltaje de menos de 25 V.
- Cuando la resistencia de carga del terminal encuentra los requerimientos de la regla 4 anteriormente dicha, y el voltaje del terminal de circuito abierto está a 0 V, la magnitud del potencial de ese circuito con respecto a Signal Ground estará en el rango de 5 a 15 V.
- El driver de la interfaz mantendrá un voltaje entre -5 a -15 V relativos a la Signal Ground para representar una condición de MARCA. El mismo driver mantendrá un voltaje de entre 5 V a 15 V relativos a Signal Ground para simbolizar una señal de ESPACIO. Obsérvese que esta regla junto con la Regla 3, permite 2 V de margen de ruido. En la práctica, se utilizan –12 y 12 V respectivamente.
- El driver cambiará el voltaje de salida hasta que no se excedan 30 V/µs, pero el tiempo requerido a la señal para pasar de –3 V a +3 V de la región de transición no podrá exceder 1 ms, o el 4% del tiempo de un bit.
- La desviación de capacitancia del terminal no excederá los 2500 pF, incluyendo la capacitancia del cable. Obsérvese que cuando se está usando un cable normal con una capacitancia de 40 a 50 pF/Pie de longitud, esto limita la longitud de cable a un máximo de 50 Pies, (15 m). Una capacitancia del cable inferior permitiría recorridos de cable más largos.
- La impedancia del driver del circuito estando apagado deberá ser mayor que 300 Ω.

Existen en el mercado dos circuitos integrados disponibles, (los chips 1488 y 1489) los cuales implementan dos drivers y receptores TTL, (4 por chip), para una RS-232 de forma compatible con las reglas anteriores.

2.3.2 Estándar RS485

RS485 es el estándar de comunicación más versátil de la serie estándar definido por la EIA, ya que se desempeña bien en los cuatro puntos. Esa es la razón por el que

RS485 es actualmente una interfaz de comunicación ampliamente utilizada en la adquisición de datos y aplicaciones de control donde varios nodos se comunican entre sí.

Uno de los principales problemas con RS232 es la falta de inmunidad de ruido en las líneas de señal. El transmisor y el receptor comparan los voltajes de las líneas de datos y de palabra con una línea cero común. Los cambios en el nivel de tierra pueden tener efectos desastrosos. Por lo tanto, el nivel de activación de la interfaz RS232 se ajusta relativamente alta a ± 3 voltios. El ruido es fácilmente recogido y limita tanto la distancia máxima y la velocidad de comunicación.

Con RS485, por el contrario no hay tal cosa como un cero común como una señal de referencia. Las señales RS485 son flotantes y cada señal se transmite a través de una línea Sig + y una línea de Sig-. El receptor RS485 compara la diferencia de tensión entre las dos líneas, en lugar del nivel de tensión absoluta de una línea de señal. Esto funciona bien y evita la existencia de los bucles de tierra, una fuente común de problemas de comunicación. Los mejores resultados se obtienen si el Sig+ Sig- y el cable es trenzado.

En la Figura 2.8, el ruido es generado por los campos magnéticos del medio ambiente. El dibujo muestra las líneas de campo magnético y el ruido de la corriente en las líneas RS485 de datos que es generado por dicho campo. En el cable recto, toda la corriente de ruido está fluyendo en la misma dirección, prácticamente la generación de un bucle de corriente al igual que en un transformador de corriente. Cuando el cable se retuerce (trenza), se observa que en algunas partes de las líneas de señal de la dirección de la corriente de ruido es el opuesto de la corriente en otras partes del cable.

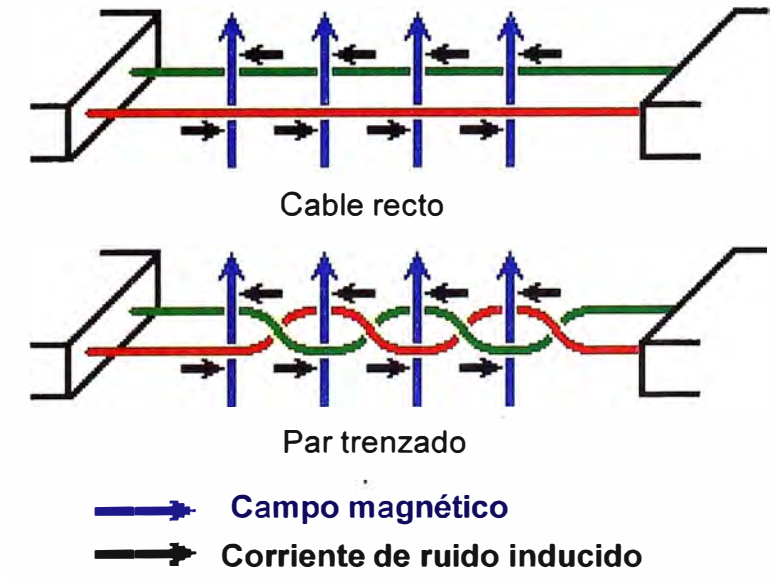

Figura 2.8 Campo magnético y ruido inducido para dos tipos de cable (Fuente: Ref. [4])

Debido a esto, la corriente de ruido resultante es muchos factores menores que con un cable recto ordinario. El blindaje (apantallamiento), que es un método común para evitar el ruido en las líneas RS232, trata de mantener los campos magnéticos lejos de las

líneas de señal. Los pares trenzados de comunicación RS485 sin embargo, añaden una forma mucho mejor para luchar contra el ruido. Los campos magnéticos pueden pasar, pero no causar daño. Es por ello que con el RS485 se logran comunicaciones de hasta 1200 m de distancia. En la actualidad se producen los conductores RS485 que puede alcanzar una velocidad de transferencia de 35 Mbps.

La Tabla 2.4 resume las características de RS232, RS422, RS423 y RS485.

Tabla 2.4 Características de RS232, RS422, RS423 y RS485 (Fuente: Ref. [4])

	RS232	**RS485**
Diferencial	No	sí
Número máximo de conductores	1	32
Máximo número de receptores	1	32
Modos de funcionamiento	half duplex full duplex	half duplex
La topología de red	punto-a-punto	multipunto
Distancia máxima (según norma)	15 m	1200 m
Velocidad máxima de 12 m	20 kbps	35 Mbps
Velocidad máxima de 1200 m	(1 kbps)	100 kbps
Max velocidad de subida	30 V / ms	n / a
Receptor resistencia de entrada	3 .. 7 kW	≧ 12 kW
Conductor impedancia de carga	3 .. 7 kW	54 Ω
Receptor de sensibilidad de entrada	± 3 V	± 200 mV
Receptor de rango de entrada	± 15 V	-7 .. 12 V
Máxima del conductor de voltaje de salida	± 25 V	-7 .. 12 V
Mínima del conductor de voltaje de salida (con carga)	± 5 V	± 1,5 V

Se puede apreciar que la velocidad de la interfaz RS485 diferencial es muy superior a la terminación única de RS232. También que hay una velocidad de subida máxima definida para RS232. Esto se ha hecho para evitar las reflexiones de señales. La velocidad de rotación máxima, también limita la velocidad máxima de comunicación en la

línea. Para la interfaz RS485 la velocidad de subida es de carácter indefinido. Para evitar reflexiones sobre los cables más largos, es necesario utilizar resistores de terminación apropiadas.

También se ve que los niveles de tensión máximos permitidos para todas las interfaces están en el mismo rango, además que el nivel de señal es inferior para las interfaces más rápidas. Debido a esto RS485 puede ser utilizado en situaciones en donde los cambios de niveles de tierra pueden ser mayores, donde al mismo tiempo son posibles altas tasas de bits debido a que la transición entre el 0 lógico y 1 lógico es sólo unos pocos cientos de milivoltios.

RS232 es la única interfaz que permite la comunicación bidireccional. Esto es, porque en las otras interfaces del canal de comunicación es compartida por múltiples receptores y en el caso de RS485 varios remitentes. RS232 tiene una línea de comunicación separada para transmitir y recibir que permite mayores velocidades de datos efectivas a la misma velocidad de bits de las otras interfaces.

La topología de red es probablemente la razón por la que RS485 es ahora el favorito en aplicaciones de adquisición de datos y control. RS485 es la única de las interfaces capaces de realizar transmisiones y recepciones múltiples de interconexión en la misma red.

Cuando se utilizan los receptores de RS485 por defecto con una resistencia de entrada de 12 kW es posible conectar hasta 32 dispositivos a la red. La alta resistencia de las entradas RS485 permite que este número se amplíe hasta 256. Existen los repetidores RS485 que hacen posible aumentar el número de nodos a varios miles, hasta abarcar varios kilómetros. Posee una interfaz que no requiere hardware de red inteligente: la aplicación en el lado del software no es mucho más difícil que con RS232. Es la razón por la que RS485 es tan popular con los ordenadores, autómatas programables, microcontroladores y sensores inteligentes en aplicaciones científicas y técnicas.

En la Figura 2.9 se muestra la topología general de red RS485.

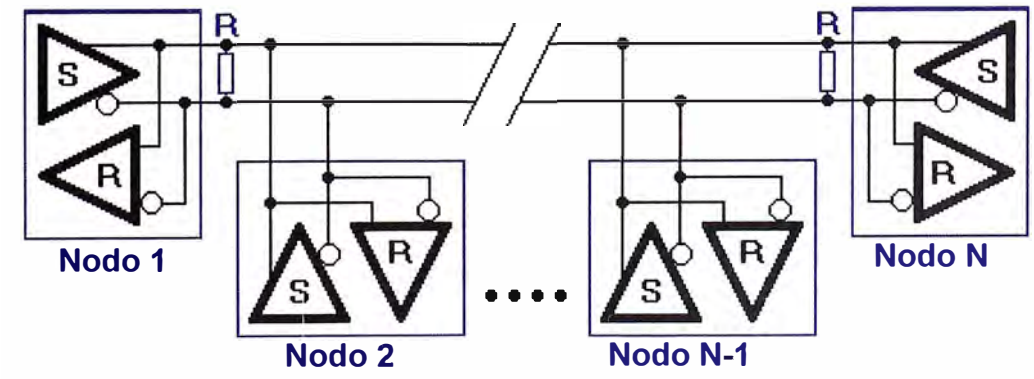

Figura 2.9 Topología de red RS485 (Fuente: Ref. [4])

Se puede ver N nodos que están conectados en una red RS485 multipunto. Para velocidades más altas y líneas más largas, las resistencias de terminación son necesarias en ambos extremos de la línea para eliminar las reflexiones.

Por defecto, todos los remitentes en el bus RS485 se encuentran en los tres estados con alta impedancia. En la mayoría de los protocolos de nivel superior, uno de los nodos se define como un maestro que envía consultas u órdenes a través del bus RS485. Todos los demás nodos reciben estos datos.

Hay otras implementaciones de redes RS485 donde cada nodo puede iniciar una sesión de datos por sí mismo. Esto es comparable con la forma de función de las redes Ethernet. Debido a que existe la posibilidad de colisión de datos con esta aplicación, la teoría dice que en este caso sólo el 37% del ancho de banda se usa en realidad.

Con este tipo de implementación de una red RS485 es necesario que haya detección de errores implementado en el protocolo de nivel superior para detectar la corrupción de datos y volver a enviar la información en un momento posterior.

2.4 Pruebas de cortocircuito en el transformador

Las pruebas de cortocircuito constan de varias pruebas según se muestra a continuación:

- Pérdidas en los arrollamientos a temperatura ambiente y corriente nominal de todos los arrollamientos.
- Pérdidas en el cobre a temperatura ambiente y corriente nominal
- Pérdidas en los arrollamientos a 75°C
- Pérdidas adicionales a temperatura ambiente y corriente nominal.
- Pérdidas en el cobre a 75°C y corriente nominal.
- Tensión de cortocircuito con la corriente nominal a la temperatura ambiente.
- Resistencia a temperatura de ambiente.
- Resistencia a 75°C.
- Tensión de Cortocircuito a 75°C.

Estas son explicadas en las secciones siguientes.

2.4.1 Pérdidas en los arrollamientos a temperatura ambiente y corriente nominal de todos los arrollamientos

Esta está determinada por la siguiente ecuación la que realza la evaluación de pérdidas I^2R en los distintos tipos de conexiones en los transformadores (estrella y delta).

$$W_R = \sum I^2 R \tag{2.1}$$

a. Conexión tipo Estrella

Esta se ve ilustrada mediante la Figura 2.10.

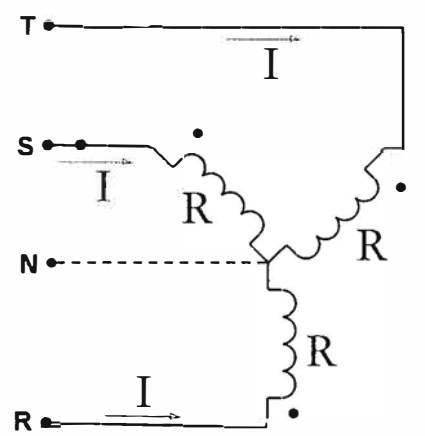

Figura 2.10 Conexión tipo estrella (Fuente: Elab. Propia)

El cálculo se realiza de acuerdo a las siguientes ecuaciones:

$$W_R=3I^2R \tag{2.2}$$

Pero:

$$R=\frac{R_L}{2} \tag{2.3}$$

Además:

$$I=I_L \tag{2.4}$$

Entonces para las tres fases:

$$W_R=3I_L{}^2\frac{R_L}{2} \tag{2.5}$$

Simplificando:

$$W_R=1.5I_L{}^2R_L \tag{2.6}$$

b. Conexión Tipo Delta

Esta se ve ilustrada mediante la Figura 2.11.

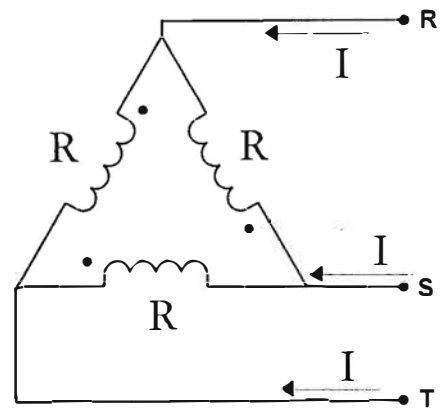

Figura 2.11 Conexión tipo delta (Fuente: Elab. propia)

De lo cual:

$$W_R=3I^2R \tag{2.7}$$

Pero:

$$R_L=\frac{2R\times R}{3R}=\frac{2}{3}R \tag{2.8}$$

Además;

$$I=\frac{I_L}{\sqrt{3}} \tag{2.9}$$

Entonces para las tres fases:

$$W_R=3\left(\frac{I_L}{\sqrt{3}}\right)^2 1.5R_L \tag{2.10}$$

Simplificando:

$$W_R=1.5I_L{}^2R_L \tag{2.11}$$

Como se puede notar, para los dos tipos de conexiones (estrella o delta) la fórmula para determinar las pérdidas son las mismas por lo que bastaría conocer la corriente nominal y la resistencia en el bobinado calculado en la primera prueba.

2.4.2 Pérdidas en el cobre a temperatura ambiente y corriente nominal

Responde a la siguiente ecuación:

$$W_{CN}=W_C\left(\frac{I_N}{I}\right)^2 \tag{2.12}$$

Donde:

- Wc: Potencia activa total medida en el equipo PM130.
- I_N: Corriente nominal en el lado de alta (dato de placa).
- I: Corriente promedio de las tres fases medidas desde el PM130.

2.4.3 Pérdidas en los arrollamientos a 75°C

Dada por la siguiente ecuación:

$$W_{R\,75°C}=W_R\frac{(235+75)}{(235+T_A)} \tag{2.13}$$

Donde:

- W_R : Pérdida en los arrollamientos a temperatura ambiente.
- T_A : Temperatura ambiente (medido con un termómetro).
- $W_{R\,75C}$: Pérdida en los arrollamientos a 75°C

2.4.4 Pérdidas adicionales a temperatura ambiente y corriente nominal

Determinada por la siguiente ecuación:

$$W_A=W_{CN}-W_R \tag{2.14}$$

Donde:

- W_A: Pérdidas adicionales.
- W_{CN}: Pérdidas en el cobre y a corriente nominal
- W_R : Pérdida en los arrollamientos a temperatura ambiente.

2.4.5 Pérdidas en el cobre a 75°C y corriente nominal

Definida por las siguientes ecuaciones:

$$W_{CN\,75°C}=W_{R\,75°C}+W_{A\,75°C} \tag{2.15}$$

$$W_{CN\,75°C}=W_R\frac{(235+75)}{(235+T_A)}+W_A\frac{(235+T_A)}{(235+75)} \tag{2.16}$$

Donde:

- $W_{CN75°C}$: Pérdidas en el cobre a 75°C y corriente nominal
- $W_{R75°C}$: Pérdidas en los arrollamientos a 75°C
- $W_{A75°C}$: Pérdidas adicionales a 75°C

2.4.6 Tensión de cortocircuito con la corriente nominal a la temperatura ambiente

Calculada por la siguiente ecuación;

$$\%Z=\frac{V}{I}\times\frac{I_N}{V_N} \tag{2.17}$$

Donde:

- V: Voltaje promedio de entre fases
- I: Corriente promedio de las tres fases.
- V_N : Voltaje nominal (dato de placa)
- I_N : Corriente nominal (dato de placa)

2.4.7 Resistencia a temperatura de ambiente

Obtenida de la siguiente relación:

$$\%R=\frac{W_R}{10KVA} \tag{2.18}$$

Donde:

W_R : Pérdida en los arrollamientos a temperatura ambiente.

2.4.8 Resistencia a 75°C

Dada por la siguiente ecuación:

$$\%R_{75°C}=\frac{W_{R\,75°C}}{10KVA} \tag{2.19}$$

Donde:

W_R: Pérdida en los arrollamientos a 75°C.

2.4.9 Tensión de Cortocircuito a 75°C

Definida por la siguiente ecuación:

$$\%Z_{75°C}=\sqrt{(\%Z)^2-(\%R)^2+(\%R_{75°C})^2} \tag{2.20}$$

Donde

- %Z : Tensión de cortocircuito con la corriente nominal y a la temperatura ambiente
- %R: Resistencia a temperatura ambiente

- $\%R_{75°C}$: Resistencia a 75°C

2.5 Plataforma .NET para desarrollo de HMI

Es un amplio conjunto de bibliotecas de desarrollo que pueden ser utilizadas por otras aplicaciones para acelerar enormemente el desarrollo y obtener de manera automática características avanzadas de seguridad, rendimiento, etc.

La plataforma ofrece un entorno gestionado de ejecución de aplicaciones, nuevos lenguajes de programación y compiladores, y permite el desarrollo de todo tipo de funcionalidades: desde programas de consola o servicios Windows hasta aplicaciones para dispositivos móviles, pasando por desarrollos de escritorio o para Internet [5].

A continuación se explican los componentes de la plataforma .NET

2.5.1 El entorno de ejecución CLR

El CLR (Common Language Runtime) es la implementación de Microsoft de un estándar llamado CLI (Common Language Infrastructure). Éste fue creado y promovido por la propia Microsoft pero desde hace años es un estándar reconocido mundialmente por el ECMA (European Computer Manufacturers Association).

El CLR/CLI esencialmente, define un entorno de ejecución virtual independiente en el que trabajan las aplicaciones escritas con cualquier lenguaje .NET. Este entorno virtual se ocupa de múltiples cosas importantes para una aplicación: desde la gestión de la memoria y la vida de los objetos, hasta la seguridad y la gestión de subprocesos.

Todos estos servicios unidos a su independencia respecto a arquitecturas computacionales, convierten al CLR en una herramienta extraordinariamente útil puesto que, en teoría, cualquier aplicación escrita para funcionar según la CLI puede ejecutarse en cualquier tipo de arquitectura de hardware.

2.5.2 El Lenguaje Intermedio y el CLS

Al contrario que otros entornos, la plataforma .NET no está atada a un determinado lenguaje de programación ni favorece a uno determinado frente a otros. En la actualidad existen implementaciones para varias decenas de lenguajes que permiten escribir aplicaciones para la plataforma .NET. Los más conocidos son Visual Basic .NET o C#. Lo mejor de todo es que cualquier componente creado con uno de estos lenguajes puede ser utilizado de forma transparente desde cualquier otro lenguaje .NET. Además, como ya se ha comentado, es posible ejecutar el código .NET en diferentes plataformas y sistemas operativos.

Dentro del CLI, existe un lenguaje llamado IL (Intermediate Language o Lenguaje Intermedio) que está pensado de forma independiente al procesador en el que se vaya a ejecutar. Es algo parecido al código ensamblador pero de más alto nivel y creado para un hipotético procesador virtual que no está atado a una arquitectura determinada.

Cuando se compila una aplicación escrita en un lenguaje .NET cualquiera, el compilador lo que genera en realidad es un nuevo código escrito en este lenguaje intermedio. Así, todos los lenguajes .NET se usan como capa de más alto nivel para producir código IL.

Un elemento fundamental del CLR es el compilador JIT (just-in-time). Su cometido es el de compilar bajo demanda y de manera transparente el código escrito en lenguaje intermedio a lenguaje nativo del procesador físico que va a ejecutar el código. Al final, lo que se ejecuta es código nativo que ofrece un elevado rendimiento.

La Figura 2.12 muestra el aspecto que tiene el código intermedio de una aplicación sencilla y se puede obtener usando el compilador que viene con la plataforma .NET.

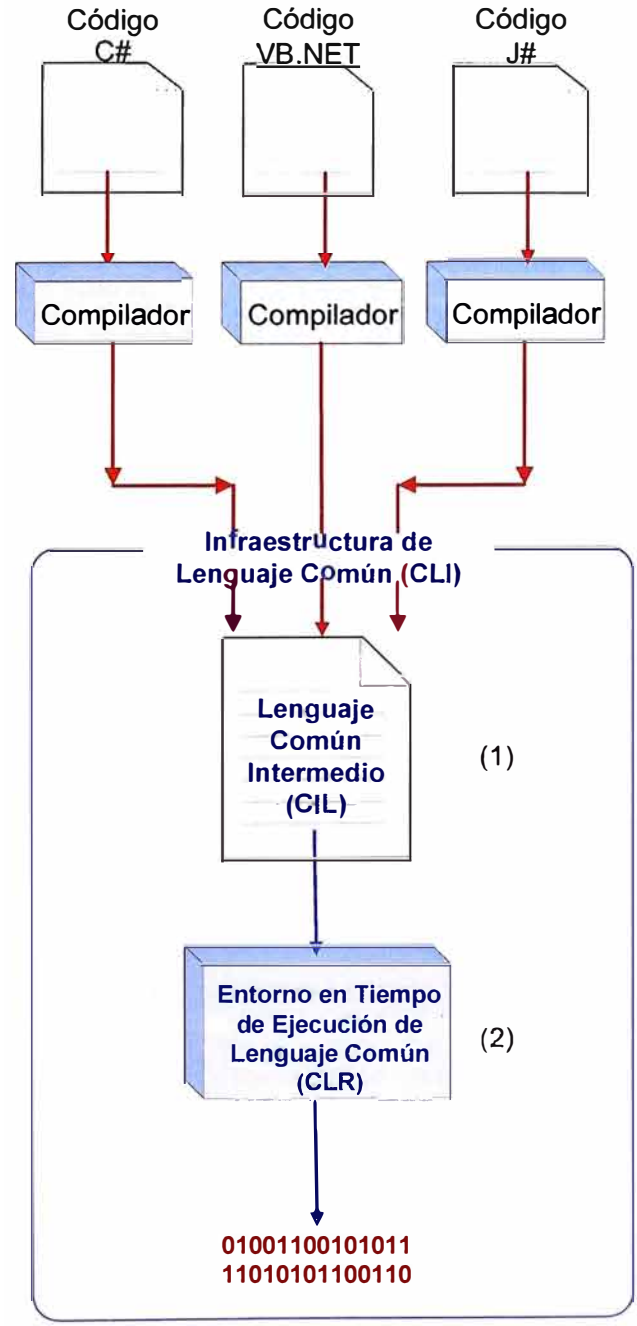

Figura 2.12 Infraestructura de Lenguaje Común (Fuente: Ref. [5])

Nota:

(1) Los lenguajes compatibles con .NET compilan a una segunda plataforma de lenguaje neutral llamada Lenguaje Común Intermedio (CIL)

(2) La plataforma especifica (CLR) compila el CIL a código de máquina que puede ser ejecutado sobre la plataforma donde se encuentra (Windows, Linux, etc)

2.5.3 La biblioteca de clases de .NET

La plataforma .NET ofrece infinidad de funcionalidades "de fábrica" que se utilizan como punto de partida para crear las aplicaciones. Existen funcionalidades básicas (por ejemplo todo lo relacionado con la E/S de datos o la seguridad) y funcionalidades avanzadas en las que se fundamentan categorías enteras de aplicaciones (acceso a datos, creación de aplicaciones Web...).

Toda esta funcionalidad está implementada en forma de bibliotecas de funciones que físicamente se encuentran en diversas DLL (bibliotecas de enlazado dinámico). A su conjunto se le denomina Base Classes Library (Biblioteca de clases base o BCL), y forman parte integral de la plataforma .NET, es decir, no se trata de añadidos que se deban obtener o adquirir aparte.

La Figura 2.13 ilustra la arquitectura conceptual general de la plataforma .NET. En ella se pueden observar los elementos que se han mencionado en apartados anteriores (lenguajes, CLR, CLS...) y en qué lugar se ubican las bibliotecas de clases base:

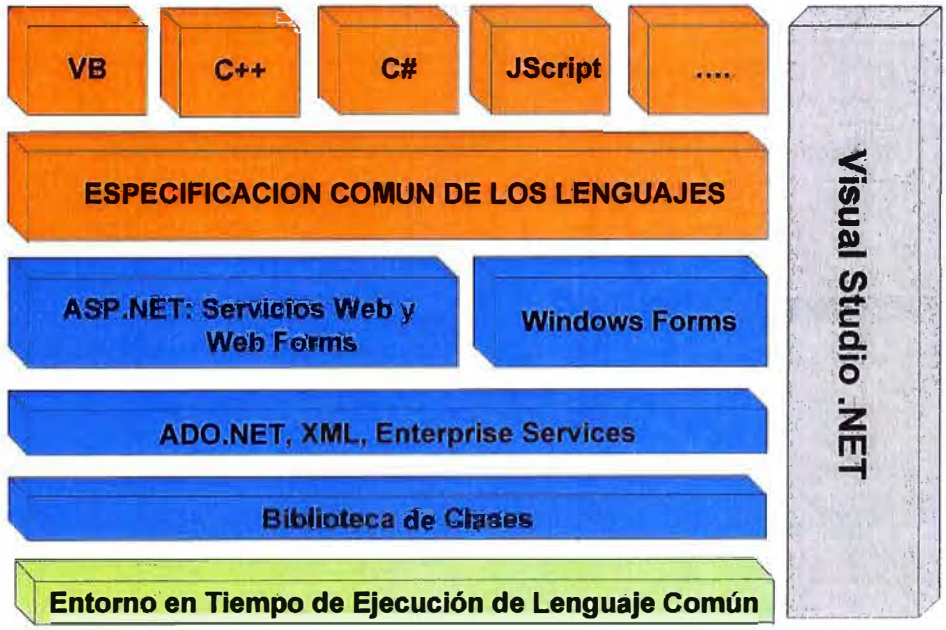

Figura 2.13 Elementos de la plataforma .NET (Fuente: Ref. [5])

Todo lo que se encuentra en la BCL forma parte de la plataforma .NET. De hecho existe tal cantidad de funcionalidad integrada dentro de estas bibliotecas (hay más de 4000 clases) que el mayor esfuerzo que todo programador que se inicia en .NET debe hacer es el aprendizaje de las más importantes, aumentando el conocimiento del resto a

base de práctica.

2.5.4 Acceso a Datos con ADO.NET

El acceso a fuentes de datos es algo indispensable en cualquier lenguaje o plataforma de desarrollo. La parte de la BCL que se especializa en el acceso a datos se denomina de forma genérica como ADO.NET.

ADO.NET es un modelo de acceso mucho más orientado al trabajo desconectado de las fuentes de datos de lo que nunca fue ADO. Si bien este último ofrecía la posibilidad de desconectar los Recordsets y ofrecía una forma de serialización de estos a través de las diferentes capas de una aplicación, el mecanismo no es ni de lejos tan potente como el que nos ofrece ADO.NET.

El objeto más importante a la hora de trabajar con el nuevo modelo de acceso a datos es el DataSet. Se podría calificar casi como un motor de datos relacionales en memoria.

La Figura 2.14 ilustra su arquitectura y en donde se destacan dos capas fundamentales: la capa conectada y la desconectada.

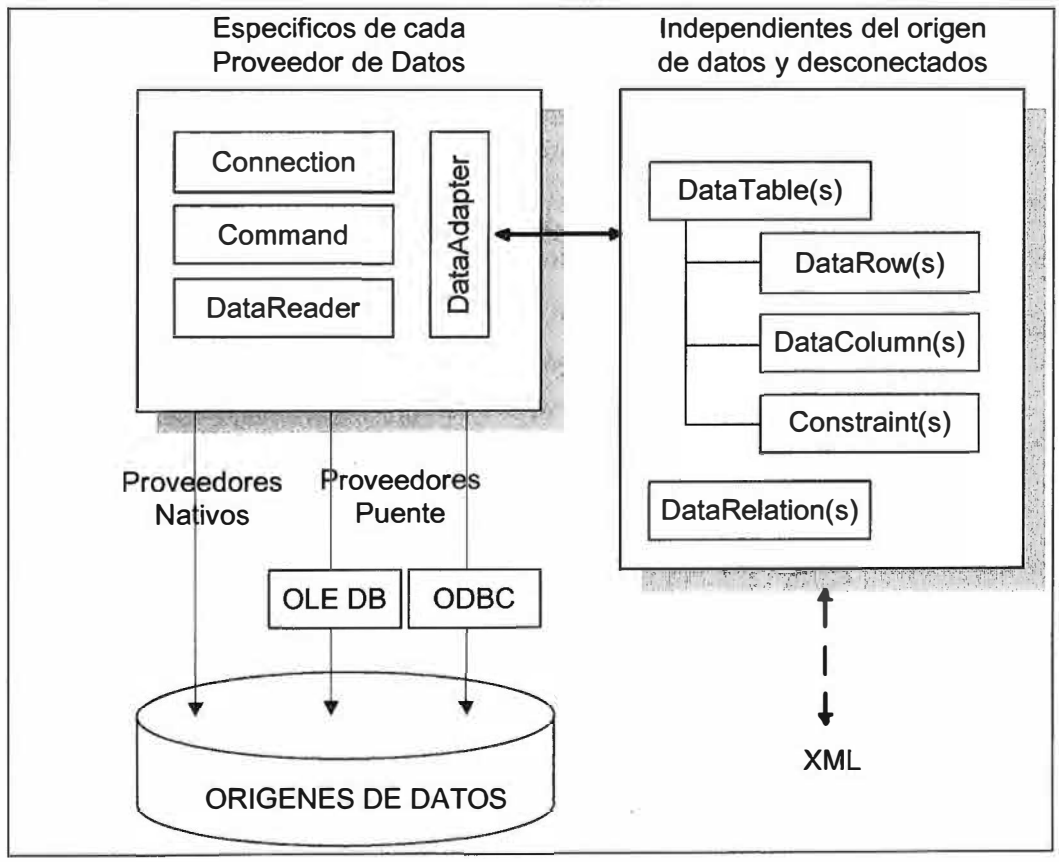

Figura 2.14 Arquitectura de ADO.NET (Fuente: [5])

a. Capa conectada

La primera de ellas contiene objetos especializados en la conexión con los orígenes de datos. Así, la clase genérica Connection se utiliza para establecer conexiones a los orígenes de datos. La clase Command se encarga de enviar comandos de toda índole al origen de datos. Por último la clase DataReader está especializada en leer los resultados

de los comandos mientras se permanece conectado al origen de datos. La clase DataAdapter hace uso de las tres anteriores para actuar de puente entre la capa conectada y la desconectada.

Estas clases son abstractas, es decir, no tienen una implementación real de la que se pueda hacer uso directamente. Es en este punto en donde entran en juego los proveedores de datos. Cada origen de datos tiene un modo especial de comunicarse con los programas que los utilizan, además de otras particularidades que se deben contemplar.

Un proveedor de datos de ADO.NET es una implementación concreta de las clases conectadas abstractas que hemos mencionado, que hereda de éstas y que tiene en cuenta ya todas las particularidades del origen de datos en cuestión. Así, por ejemplo, las clases específicas para acceder a SQL Server se llaman SqlConnection, SqlCommand, SqlDataReader y SqlDataAdapter y se encuentran bajo el espacio de nombres System.Data.SqlClient.

b. Capa desconectada

Una vez que ya se han recuperado los datos desde cualquier origen de datos que requiera una conexión ésta ya no es necesaria. Sin embargo sigue siendo necesario trabajar con los datos obtenidos de una manera flexible. Es aquí cuando la capa de datos desconectada entra en juego. Además, en muchas ocasiones es necesario tratar con datos que no han sido obtenidos desde un origen de datos relacional con el que se requiera una conexión. A veces únicamente necesitamos un almacén de datos temporal pero que ofrezca características avanzadas de gestión y acceso a la información.

Por otra parte las conexiones con las bases de datos son uno de los recursos más escasos con los que contamos al desarrollar. Su mala utilización es la causa más frecuente de cuellos de botella en las aplicaciones y de que éstas no escalen como es debido. Finalmente otro motivo por el que es importante el uso de los datos desconectado de su origen es la transferencia de información entre capas de una aplicación. Éstas pueden encontrarse distribuidas por diferentes equipos, e incluso en diferentes lugares del mundo gracias a Internet. Por ello es necesario disponer de algún modo genérico y eficiente de poder transportar los datos entre diferentes lugares, utilizarlos en cualquiera de ellos y posteriormente tener la capacidad de conciliar los cambios realizados sobre ellos con el origen de datos del que proceden.

2.5.5 Aplicaciones Windows Forms

Las aplicaciones de escritorio son aquellas basadas en ventanas y controles comunes de Windows que se ejecutan en el sistema local. Son el mismo tipo de aplicaciones que antes se construían con Visual Basic 6 u otros entornos similares.

En la plataforma .NET, el espacio de nombres que ofrece las clases necesarias para construir aplicaciones de escritorio bajo Windows se denomina Windows Forms. Este es también el nombre genérico que se le otorga ahora a este tipo de programas basados en ventanas.

Windows Forms está constituido por multitud de clases especializadas que ofrecen funcionalidades para el trabajo con ventanas, botones, rejillas, campos de texto y todo este tipo de controles habituales en las aplicaciones de escritorio.

Visual Studio ofrece todo lo necesario para crear visualmente este tipo de programas, de un modo similar aunque más rico al que ofrecía el entorno de desarrollo integrado de Visual Basic.

2.6 MySQL

Es un sistema de gestión de bases de datos relacional que fue creada por la empresa sueca MySQL AB, la cual tiene el copyright del código fuente del servidor SQL, así como también de la marca [6]. MySQL es un software de código abierto, licenciado bajo la GPL (General Public License) de la GNU, un sistema operativo similar a Unix que es software libre y respeta su libertad.

Aunque MySQL AB distribuye una versión comercial, en lo único que se diferencia de la versión libre, es en el soporte técnico que se ofrece, y la posibilidad de integrar este gestor en un software propietario, ya que de otra manera, se vulneraría la licencia GPL.

El lenguaje de programación que utiliza MySQL es Structured Query Language (SQL) que fue desarrollado por IBM en 1981 y desde entonces es utilizado de forma generalizada en las bases de datos relacionales.

MySQL surgió alrededor de la década del 90, Michael Windenis comenzó a usar mSQL para conectar tablas usando sus propias rutinas de bajo nivel (ISAM). Tras unas primeras pruebas, llegó a la conclusión de que mSQL no era lo bastante flexible ni rápido para lo que necesitaba, por lo que tuvo que desarrollar nuevas funciones. Esto resulto en una interfaz SQL a su base de datos, totalmente compatible a mSQL.

Inicialmente, MySQL carecía de algunos elementos esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. A pesar de esto, atrajo a los desarrolladores de páginas web con contenido dinámico, debido a su simplicidad, de tal manera que los elementos faltantes fueron complementados por la vía de las aplicaciones que la utilizan. Poco a poco estos elementos faltantes, están siendo incorporados tanto por desarrolladores internos, como por desarrolladores de software libre.

En las últimas versiones se pueden destacar las siguientes características principales:

- El principal objetivo de MySQL es velocidad y robustez.

- Soporta gran cantidad de tipos de datos para las columnas.
- Gran portabilidad entre sistemas, puede trabajar en distintas plataformas y sistemas operativos.
- Cada base de datos cuenta con 3 archivos: Uno de estructura, uno de datos y uno de índice y soporta hasta 32 índices por tabla.
- Aprovecha la potencia de sistemas multiproceso, gracias a su implementación multihilo.
- Flexible sistema de contraseñas (passwords) y gestión de usuarios, con un muy buen nivel de seguridad en los datos.
- El servidor soporta mensajes de error en distintas lenguas

Las ventajas del MySQL son:

- Velocidad al realizar las operaciones, lo que le hace uno de los gestores con mejor rendimiento.
- Bajo costo en requerimientos para la elaboración de bases de datos, ya que debido a su bajo consumo puede ser ejecutado en una máquina con escasos recursos sin ningún problema.
- Facilidad de configuración e instalación.
- Soporta gran variedad de Sistemas Operativos
- Baja probabilidad de corromper datos, incluso si los errores no se producen en el propio gestor, sino en el sistema en el que está.
- Conectividad y seguridad

Sus desventajas son:

- Un gran porcentaje de las utilidades de MySQL no están documentadas.
- No es intuitivo, como otros programas (ACCESS).

Un ejemplo se muestra a continuación:

```
mysql > SELECT * FROM mi_tabla WHERE nombre = "lupita" AND edad < 30;
```

En donde los comandos se escriben en mayúscula y significa: Seleccionar todos los campos de la tabla mi_tabla donde cumpla que el nombre sea igual a "lupita" y su edad sea menor a 30.

2.7 El medidor de potencia PM130

Un elemento fundamental del proyecto lo constituye el medidor de potencia PM130. Este dispositivo es el encargado de medir diversas variables eléctricas del transformador.

El PM130 posee un display limitado y botoneras que permiten seleccionar los datos que se deseen ver (Figura 2.15).

Sus rangos de entrada son [7]:

- Voltaje: ENTRADA DIRECTA (690V línea-a-línea voltaje y 400V línea-a-neutral) para una carga: <0.5 VA.

- Corriente: ENTRADA VIA CT con 5A salida secundaria Carga: 2.5 a 4 mm^2 de hilo desde el CT. Resistencia de sobrecarga:15A RMS continuas, 250A RMS por 1 segundo.
- Puerto de comunicación: EIA RS-485 standard, Sección máxima del hilo: 2 .5 mm^2 12 AWG.
- Pantallas: 3 ventanas de alta luminosidad LEDs digital de 7 segmentos. 3 gráficos de LED de barras de colores 40-110%.
- Temperatura de operación: -20°C a +60°C (-4°F a +140°F)

Figura 2.15 Panel del PM130 (Fuente: Ref. [7])

En la Figura 2.16 se pueden ver las interfaces de conexión RS-485. La Tabla 2.5 muestra el Data Transfer del PM130.

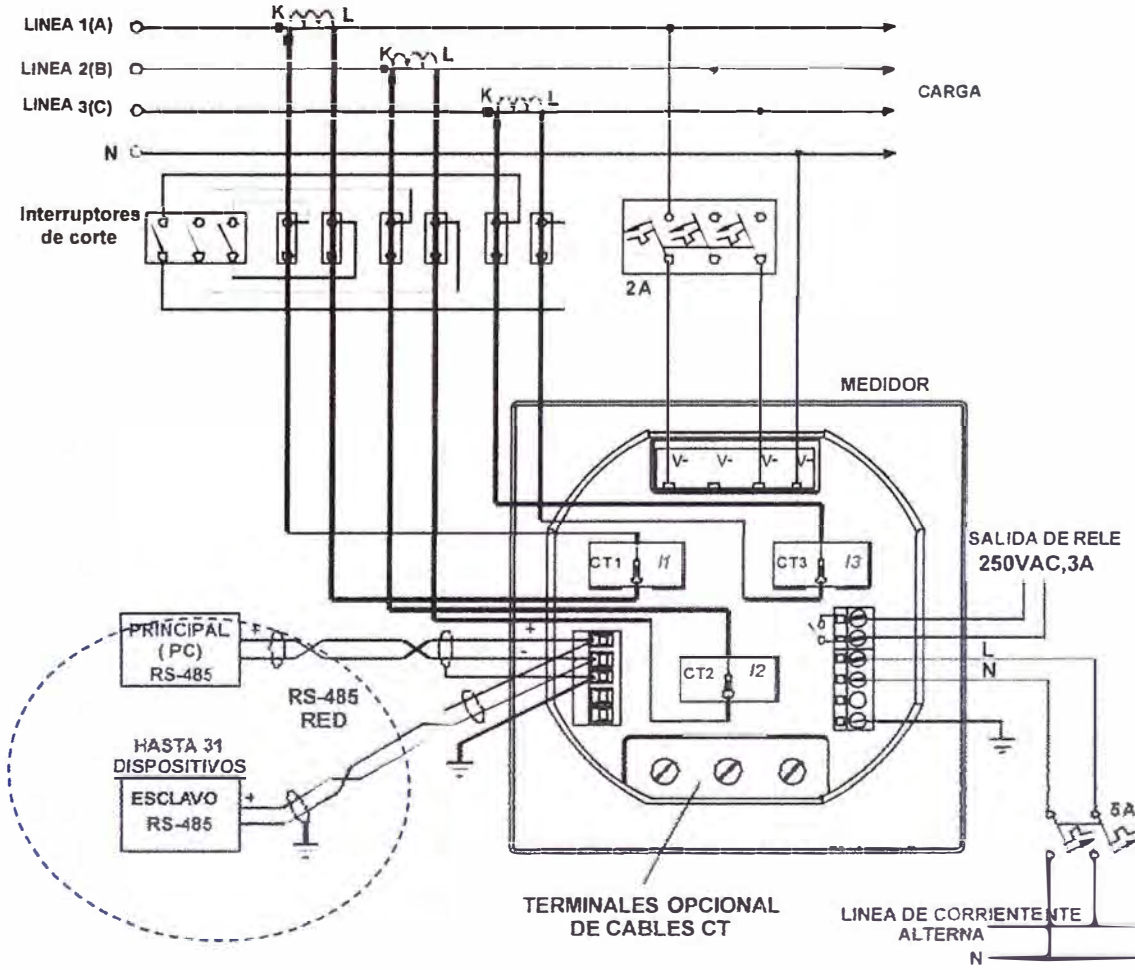

Figura 2.16 RS 485 en el esquema de conexiones del PM130 (Fuente: Ref. [7])

Tabla 2.5 Relación de parámetros de la transferencia de datos del PM130 (Fuente: Fabricante)

Parámetro	Registro (Offset)	Dirección de Memoria	Dirección Modbus	Tipo de Dato	Unidad	Valor Mínimo	Valor Máximo	Conversión
Voltaje L1/L12	0	256	40257	Lectura	V	0	Vmax	LIN3
Voltaje L2/L23	1	257	40258	Lectura	V	0	Vmax	LIN3
Voltaje L3/L31	2	258	40259	Lectura	V	0	Vmax	LIN3
Corriente L1	3	259	40260	Lectura	A	0	Imax	LIN3
Corriente L2	4	260	40261	Lectura	A	0	Imax	LIN3
Corriente L3	5	261	40262	Lectura	A	0	Imax	LIN3
Potencia Activa L1	6	262	40263	Lectura	kW	-Pmax	Pmax	LIN3
Potencia Activa L2	7	263	40264	Lectura	kW	-Pmax	Pmax	LIN3
Potencia Activa L3	8	264	40265	Lectura	kW	-Pmax	Pmax	LIN3
Potencia Reactiva L1	9	265	40266	Lectura	kvar	-Pmax	Pmax	LIN3
Potencia Reactiva L2	10	266	40267	Lectura	kvar	-Pmax	Pmax	LIN3
Potencia Reactiva L3	11	267	40268	Lectura	kvar	-Pmax	Pmax	LIN3
Potencia Aparente L1	12	268	40269	Lectura	KVA	-Pmax	Pmax	LIN3
Potencia Aparente L2	13	269	40270	Lectura	KVA	-Pmax	Pmax	LIN3
Potencia Aparente L3	14	270	40271	Lectura	KVA	-Pmax	Pmax	LIN3
Factor de Potencia L1	15	271	40272	Lectura	--	-1.00	1.00	LIN3
Factor de Potencia L2	16	272	40273	Lectura	--	-1.00	1.00	LIN3
Factor de Potencia L3	17	273	40274	Lectura	--	-1.00	1.00	LIN3
Factor de Potencia Total	18	274	40275	Lectura	--	-1.00	1.00	LIN3
Potencia Activa Total	19	275	40276	Lectura	kW	-Pmax	Pmax	LIN3
Potencia Reactiva Total	20	276	40277	Lectura	kvar	-Pmax	Pmax	LIN3
Potencia Aparente Total	21	277	40278	Lectura	KVA	-Pmax	Pmax	LIN3
Corriente de Desbalance	22	278	40279	Lectura	A (mA)	0	Imax	LIN3
Frecuencia	23	279	40280	Lectura	Hz	45.0	65.0	LIN3

Cuando se implementa el PM130 para la comunicación Modbus se debe emplear los siguientes métodos para convertir las cuentas recibidas desde el dispositivo a unidades de ingeniería.

- **NONE**: El dato se presentara exactamente como es recibido por comunicación desde el PM 130.
- **LIN3**: Esta conversión mapea los datos en cuentas recibidos por comunicación en el intervalo de 0 a 9999 en la escala definida por el usuario (Escala Baja/Escala Alta). La conversión se lleva a cabo de acuerdo con la fórmula [8]:

$$Y=X\frac{(HI-LO)}{9999}+LO \tag{2.21}$$

Donde:

Y: Valor real en unidades de ingeniería.

X: Valor real en cuentas en el rango de 0-9999

- **LO,HI:** Valor mínimo y máximo en unidades de ingeniería.

Cuando es necesaria una conversión de datos, el valor LO, HI y el método de conversión se indicara en el registro correspondiente.

Cuando el valor es escrito al PM130, la conversión se lleva a cabo a la inversa para producir el valor en el intervalo de 0 a 9999, según se muestra en la fórmula [8].

$$X=9999\frac{(Y-LO)}{(HI-LO)} \tag{2.22}$$

CAPÍTULO III
METODOLOGÍA PARA LA SOLUCIÓN DEL PROBLEMA

Este capítulo se enfoca a exponer el diseño del sistema de adquisición de datos para pruebas protocolares de un transformador. Para ello preliminarmente se realiza el planteamiento de la solución, en el cual se especifica los requerimientos, alternativas de solución y dimensionamiento del proyecto. Posteriormente se describe el diseño y la implementación de la solución

3.1 Planteamiento de la solución

Como se explicó en el primer capítulo, la limitante de contar con solo 3 displays en el medidor PM130 obligaba a realizar las pruebas con 2 personas las cuales una se dedicaba a regular las corrientes y tensiones y la otra tomaba los datos que indicaba el medidor para luego entregárselo al ingeniero para que este realice los cálculos necesarios.

En la Figura 3.1 se muestra el panel que tiene el medidor PM130, como se puede apreciar para poder visualizar las 10 variables, solo se contaba con 3 displays de 7 segmentos, cuando el operador requería visualizar las otras variables tenía que manipular el medidor.

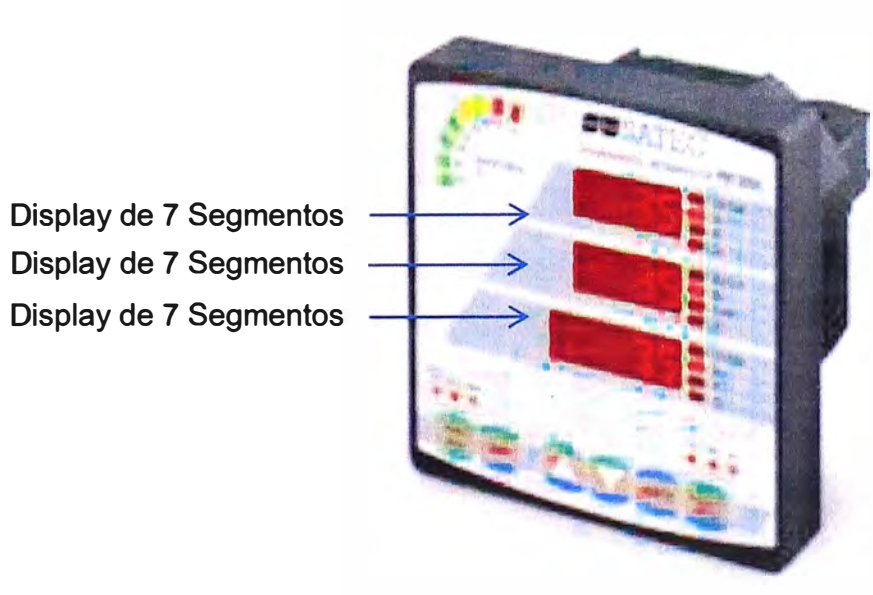

Figura 3.1 Medidor PM130 de la marca SATEC (Fuente: Fabricante)

Como se mencionó en el capítulo I el problema radica en contar con 2 personas para realizar una prueba así como el riesgo de que se pudiera desconfigurar el medidor ya que

se tenía que manipular el equipo para cambiar las variables en los displays.

La idea inicial de este proyecto fue la de implementar un HMI el cual le permita a un solo operador visualizar sus 10 variables simultáneamente y en tiempo real, pero debido a que se observó que aún se podía reducir más tiempos, la de generar reportes desde el mismo HMI para suprimir el tiempo que empleaba el ingeniero, se optó por implementar adicionalmente dicha mejora.

En conclusión, el sistema que se propone intenta reducir la cantidad de participantes en las pruebas de 3 a 1 reduciendo así los costos de horas hombre para la realización de dichas pruebas.

Para llevar a cabo este proyecto se consideraron los siguientes puntos:

1. Identificar el protocolo de comunicación y la interfaz eléctrica con la que contaba el medidor PM130.
2. Para la tecnología de comunicación para el HMI, optar por la comunicación OPC por sus múltiples ventajas expuestas en el capítulo 2. Dado que no es una herramienta libre se consideró su costo por licencia.
3. Disponer de un ordenador con interfaz eléctrica RS232.
4. Ubicar al ordenador a una distancia menor a los 120 metros del medidor (limitante de la interfaz RS485).
5. Identificación del procedimiento de las pruebas protocolares.

Esta sección se organiza en dos partes. Primero se establecen los requerimientos del sistema y luego las estrategias de diseño.

3.1.1 Requerimientos del sistema

Para el buen desempeño del sistema y para que cumpla la función para la cual fue diseñada, se logró identificar los requerimientos funcionales y no funcionales.

a. Requerimientos funcionales

Un requerimiento funcional es aquel que define el comportamiento interno de la aplicación tales como cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas. Los requerimientos funcionales para el HMI son:

- Ser amigable y que la información mostrada sea fácil de interpretar para el operador.
- Poder visualizar en tiempo real y de manera clara y precisa las variables de tensión, corriente, potencia y frecuencia simultáneamente.
- Poder ingresar datos adicionales para el reporte como: el Nombre del Cliente, Numero de Serie del Transformador, Voltajes Nominales (de Alta y de Baja), Numero de Fases del Transformador, Grupo de conexión, altura, temperatura y Fecha de realización de la prueba.
- Poder visualizar tendencias de las pruebas.

- Poder generar los reportes de pérdidas en el hierro y en el cobre.

b. Requerimientos no funcionales

Estos requerimientos se enfocan en el diseño o la implementación. Estos fueron:

- Contar con información en tiempo real.
- Realizar los cálculos de pérdidas en el hierro y en el cobre de manera automática.
- Comunicar el medidor con un ordenador.
- Recolectar los datos del medidor y guardarlos en una base de datos.

3.1.2 Estrategias de diseño

Para la implementación del sistema propuesto se estableció lo siguiente

a. Interfaz física de comunicación entre el medidor PM130 y el ordenador

Debido a que el medidor contaba con un puerto con interfaz eléctrica RS485 y el ordenador contaba con un puerto con interfaz RS232, se optó por adquirir un conversor de interfaces de RS485 a RS232 de la marca DCTRLS como se muestra en la Figura 3.2.

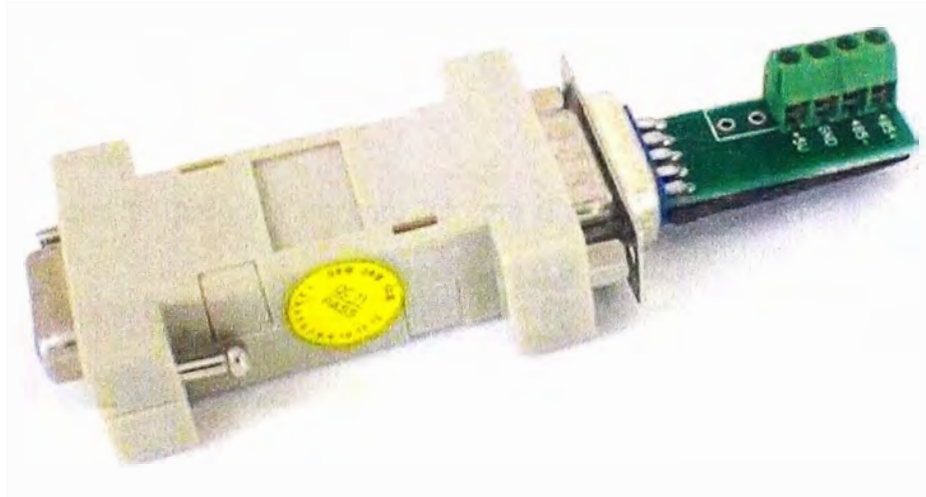

Figura 3.2 Conversor de interfaces RS232 a RS485 (Fuente: Fabricante)

b. Elección del servidor OPC para la comunicación entre el medidor PM130

El servidor OPC, el cual le permite interpretar al HMI y a la base de datos la información enviada por el medidor, debe contar con una robustez comprobada en el mercado y de bajo costo, siendo la mejor opción el servidor OPC KepServer 4.0.

c. Elección del cliente OPC

Para la comunicación entre el servidor OPC con el HMI y la base de datos se optó por adquirir cliente OPC MaycensOPC el cual trabaja con el gestor de base de datos MySQL, cabe resaltar que este software tiene muchas implementaciones en el mercado y es de bajo costo.

d. Desarrollo del HMI

Debido a que los HMIs disponibles en el mercado son muy costosos (alrededor de los $5000.00 por una licencia de 800 tags como mínimo) y que para el sistema en desarrollo

no se necesita muchos tags, no se justificaba adquirir uno, por lo que se optó por implementar un HMI utilizando herramientas de programación. Para ello se eligió la plataforma Visual Studio 2010 con el lenguaje de programación C#. Se utilizó esta plataforma por tener herramientas integradas para interactuar con Office y con MySQL.

e. Elección del gestor de Base de Datos

Como se explicó en el punto c, la elección por el cliente OPC MAYCENS OPC fue porque, aparte de ser de bajo costo, trabaja con el gestor de base de datos MySQL el cual es software libre y por lo tanto su adquisición no generaba ningún costo adicional.

f. Generación de los reportes

Para la realización de los reportes se optó por utilizar el Excel. Ya que la empresa contaba con licencias de office se aprovechó esta ventaja. Todos los cálculos para determinar las perdidas en el hierro y en el cobre se realizan utilizando plantillas predefinidas en Excel.

g. Elección del servidor de protocolos

Se optó por utilizar el mismo ordenador donde se guardaban anteriormente los reportes de las pruebas.

3.2 Descripción de la solución

El sistema que se propone en este informe presenta el siguiente esquema (Figura 3.3).

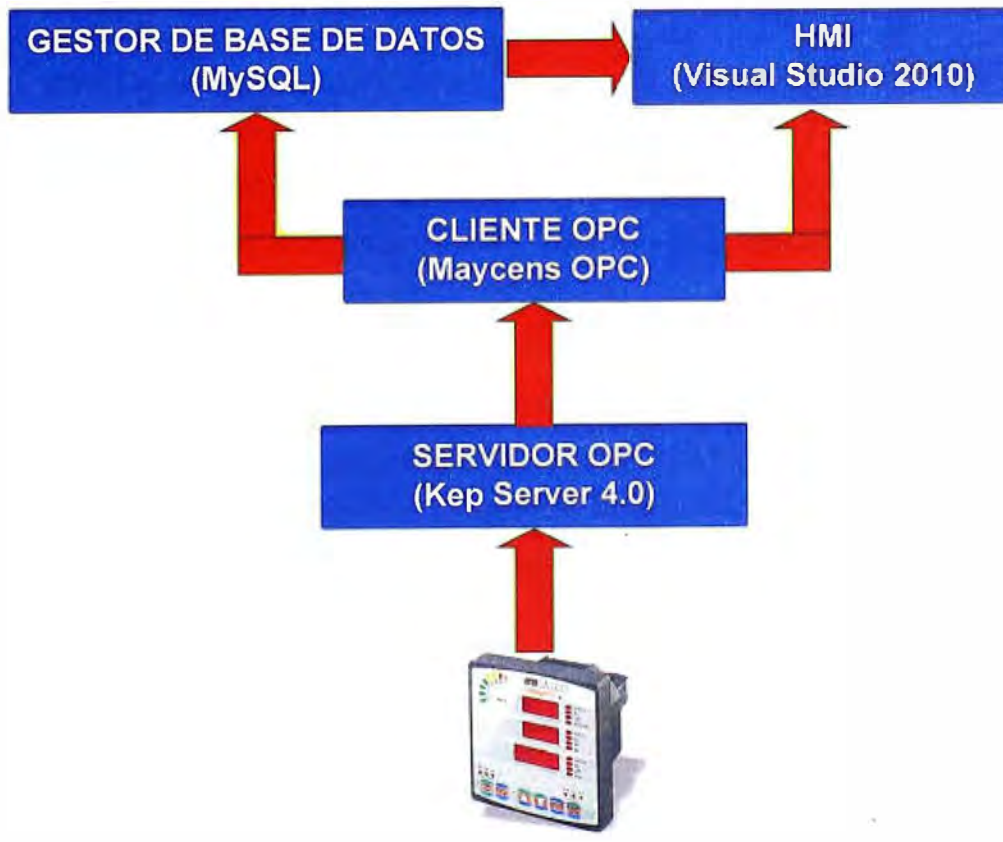

Figura 3.3 Esquema de comunicaciones del sistema de monitoreo

Como se muestra en la figura anterior, la información parte desde el PM130 vía protocolo Modbus hacia el Servidor OPC, dicho servidor brinda información al Cliente OPC quien a su vez se encarga de enviar la información hacia el HMI y finalmente el HMI envía la información hacia la base de Datos, en donde se almacenara para futuras consultas. Cabe resaltar que desde el HMI se podrán realizar futuras consultas hacia MySQL para conseguir protocolos realizados en el pasado.

Todos estos componentes del sistema son paquetes de software, a continuación se especifican los detalles de la implementación del sistema.

3.2.2 Desarrollo de la HMI

Ello se eligió la plataforma Visual Studio 2010 con el lenguaje de programación C#. Visual Studio 2010 ofrece una serie de ventajas para interactuar con el Office y el MySQL las cuales fueron aprovechadas para hacer más fácil la programación de este HMI, adicionalmente se consideraron los siguientes aspectos:

- Presentar el Logo de la empresa.
- Mostrar al operador la Hora y Fecha.
- Las pantallas deben ser de fondo negro y los textos de color verde para una mejor contrastación de las letras y los números para el operador.

La Figura 3.4 es la pantalla principal del HMI desarrollado y las partes que lo conforman. Esta pantalla muestra los principales comandos y botones de navegación del sistema.

Esta pantalla se presenta botones los cuales tienen distintas funciones que pasaremos a describir a continuación:

a. Botones de Navegación

Botones que le sirven al operador para navegar en los diferentes entornos del HMI, los entornos representan a las etapas de las pruebas y están dividas en:

a.1 Botón de Navegación "DATOS DE PLACA"

Botón que representa la primera etapa de la prueba, cuando el operador hace un click sobre este botón, se muestra el entorno (Figura 3.5). En esta pantalla el operador ingresa manualmente los datos informativos del transformador, tales como: Potencia Nominal, Tensión Nominal en el lado de Alta, Temperatura, etc., la cual es información importante para el reporte del protocolo de pruebas.

a.2 Botón de Navegación "PRUEBA EN VACIO"

Botón que representa la segunda etapa de las pruebas. Es aquí donde el operador visualiza en tiempo real los valores de corriente, tensión y potencia presentadas en las pruebas en vacío del transformador. Cuando el operador hace click sobre el botón se le mostrara el entorno que se presenta en la Figura 3.6.

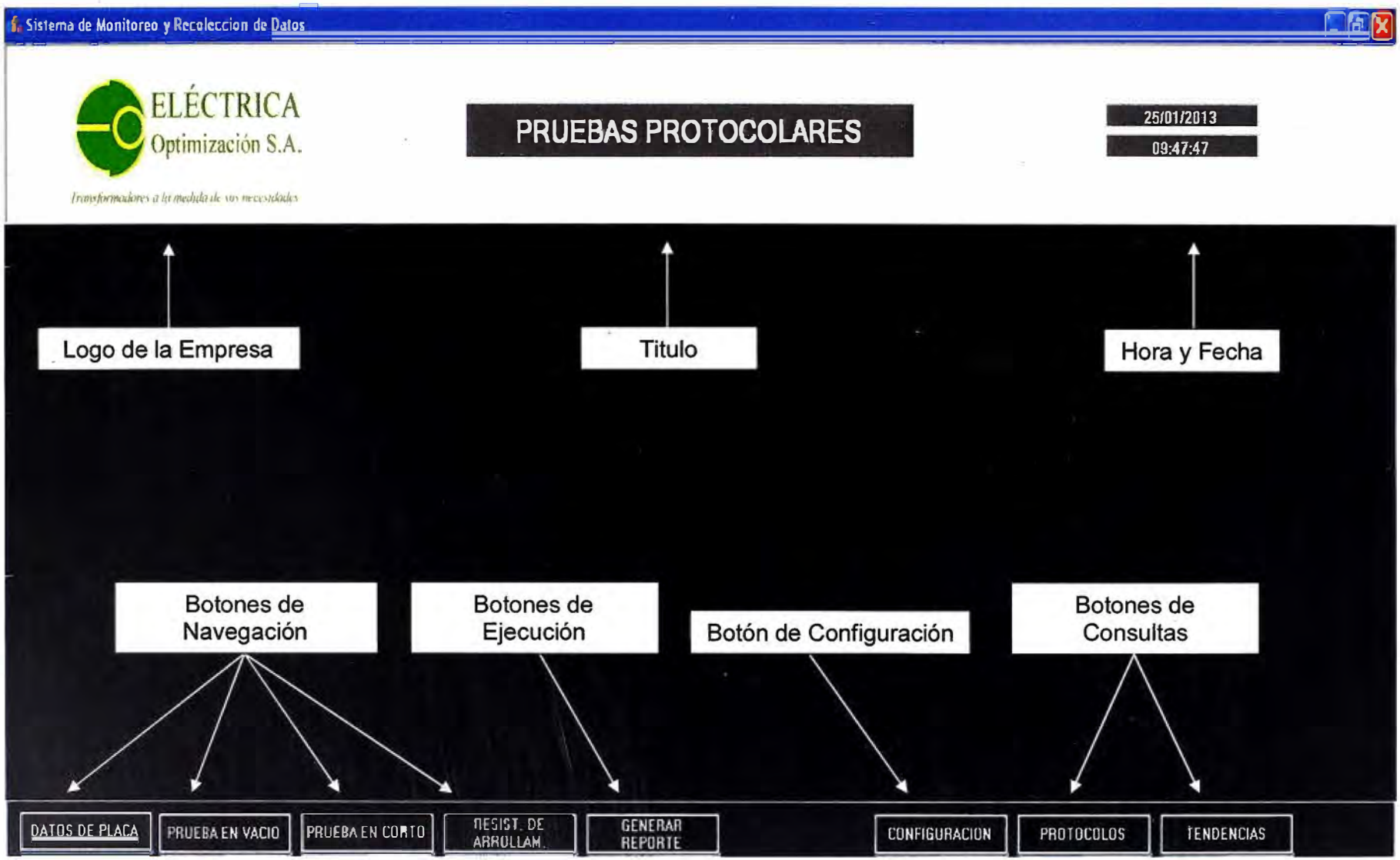

Figura 3.4 Entorno de la Pantalla Principal del HMI (Fuente: Elab. propia)

Figura 3.5 Entorno de la Pantalla para ingresar los Datos de Placa (Fuente: Elab. propia)

Figura 3.6 Entorno de la Pantalla para las pruebas en vacío del transformador (Fuente: Elab. propia)

Figura 3.7 Entorno de la Pantalla para las pruebas en cortocircuito del transformador (Fuente: Elab. propia)

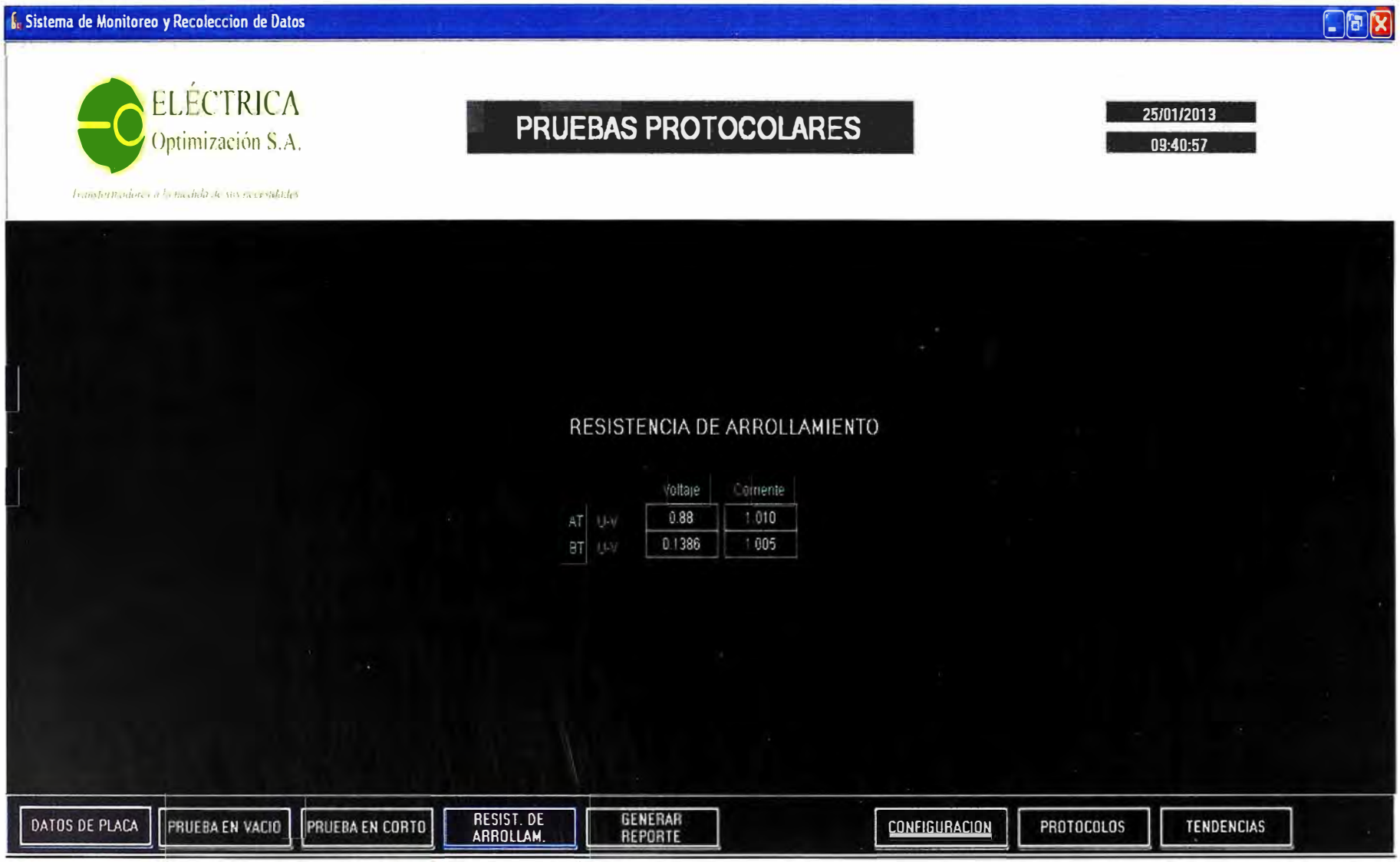

Figura 3.8 Entorno de la Pantalla para las pruebas de resistencia de arrollamiento del transformador (Fuente: Elab. propia)

El botón "CAPTURAR VALORES" sirve para que el operador congele los valores del medidor en el instante que termine sus pruebas. Estos valores finalmente son enviados el servidor para almacenarlos como resultado de las pruebas. Cabe resaltar que cuando el operador presiona este botón, los valores cambiaran de color verde (valores en tiempo real) a color blanco (valores capturados) como confirmación de que los valores se han capturado y están listos para ser enviados a la base de datos, esperando a que el operador presione el botón "GENERAR REPORTE" el cual se explicara con más detalle posteriormente.

a.3 Botón de Navegación "PRUEBA EN CORTO"

Botón que representa la tercera etapa de las pruebas, es aquí donde el operador visualiza en tiempo real los valores de corriente, tensión y potencia presentadas en las pruebas en cortocircuito del transformador. Cuando el operador hace click sobre el botón se muestra el entorno que se presenta en la Figura 3.7. Se puede apreciar que este entorno es muy similar al de las pruebas en vacío. El botón CAPTURAR VALORES cumple la misma función que el botón visto en la pantalla PRUEBA EN VACIO.

a.4 Botón de Navegación "RESISTENCIA DE ARROLLAMIENTO"

Este botón no estaba en el alcance del proyecto pero fue añadido debido a que la información que resulta de estas pruebas son parte importante en los cálculos de pérdidas mostrados en el capítulo 2. Cuando el operador hace click sobre este botón se le muestra el entorno de la Figura 3.8 que representa la cuarta etapa de las pruebas. En este entorno el operador debe ingresar de manualmente los resultados obtenidos de estas pruebas.

b. Botón de Generación de Reporte

Botón que le sirve al operador para almacenar los valores obtenidos durante las pruebas en la base de datos de MySQL y posteriormente generar el reporte de manera automática una vez que haya culminado las 4 etapas del proceso de pruebas cuando el operador hace un click sobre dicho botón. Es necesario destacar que este botón tiene la parte más importante de la programación, ya que cuando ocurra el evento click sobre el botón, este automáticamente enlaza el sistema con MySQL y guarda la información de todas las pantallas descritas anteriormente, y luego se enlaza con Excel para generar el reporte que se muestra en la Figura 3.9.

Adicionalmente, este evento guardara el reporte en Excel en la ruta mostrada en la Figura 3.10, con el nombre del número de serie del transformador ingresado por el operador en la primera pantalla.

No hay posibilidad de conflicto de nombre de los archivos en los protocolos dado que el número de serie es único en la empresa.

ELÉCTRICA Optimización S.A.

PROTOCOLO DE PRUEBAS

TRANSFORMADOR	ELECTRICA OPTIMIZACIÓN	Nº Serie	10-062-01

Cliente	IDEAS TEXTILES S.A.C.				
KVA	20	HERTZ	60	m.s.n.m.	4000
VOL.	2300 460	FASES	3	∠ Ø °C	
AMP.	5.02 25.10	GRUPO	Dyn5	T.c.c.%	6.2

1.- PRUEBA EN VACIO A 60 HZ

VOLT.	1 / 1	AMP.	50 / 5	VATIOS			
464.0164	464	0.9737	7.8	85		wfe =	850
464.0164		0.56966					
465.0125		0.78888		85 x 10 x 1 x 1			

2.- PRUEBA EN CORTOCIRCUITO

						TEMP °C=	22
VOLT.	1 / 1	AMP.	10 / 5	VATIOS		Wcu 22 °C=	406
24	24.13	0.93129	1.86	28	56	Tcc 22 °C=	2.83
24.07241		0.9288929				Wcu 75 °C=	1180
24.32143		0.9352936		28 x 2 x 1 x 1		Tcc 75 °C=	6.22

3.- MEDICION DE RESISTENCIA DE ARROLLAMIENTOS

Arrollam	Ø a °C	VOLT.	AMP.	Ohm	I2R Ta		
AT U - V	22	0.88	1.010	0.871	33	I2R 75°C	1180
BT u - v	22	1.005	1.005	1.0000	945	Ad. 75°C	0
				Suma	978	Wcu 75 °C	1180

OBSERVACIONES

FECHA	1/25/2013		
PROBADO			

Figura 3.9 Reporte Final de las pruebas protocolares (Fuente: Elab. propia)

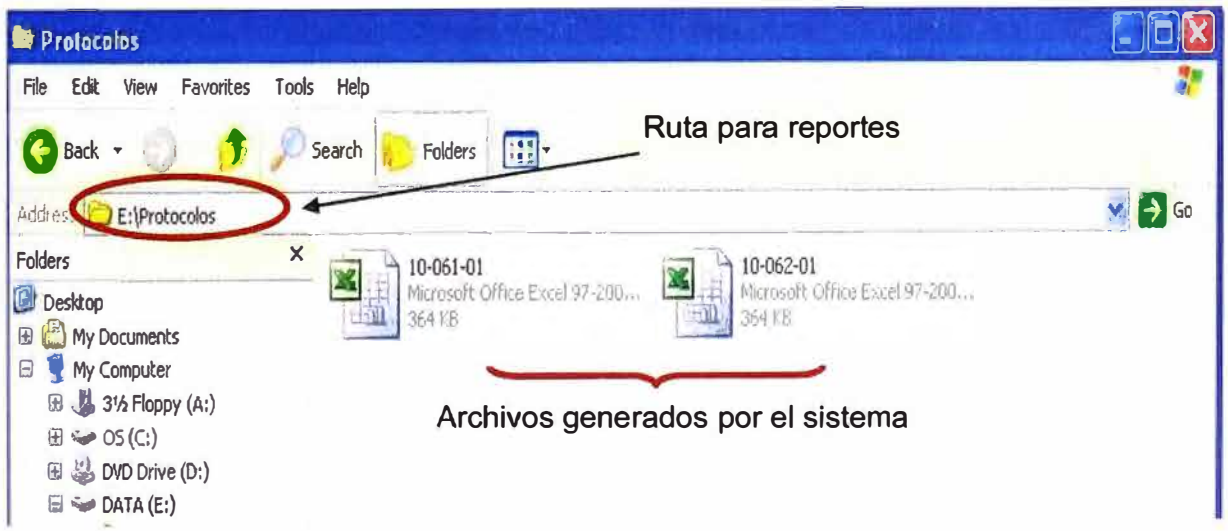

Figura 3.10 Ruta para almacenar reportes de pruebas protocolares (Fuente: Propia)

c. Botón de Configuración

Este botón es utilizado solamente por personal especializado (no autorizado para el

operador). Aquí el usuario especialista elige la base de datos donde se conectará el HMI para obtener los valores en tiempo real. Para este caso, la base de datos se llama: "RunTimeBD" (Figura 3.11)

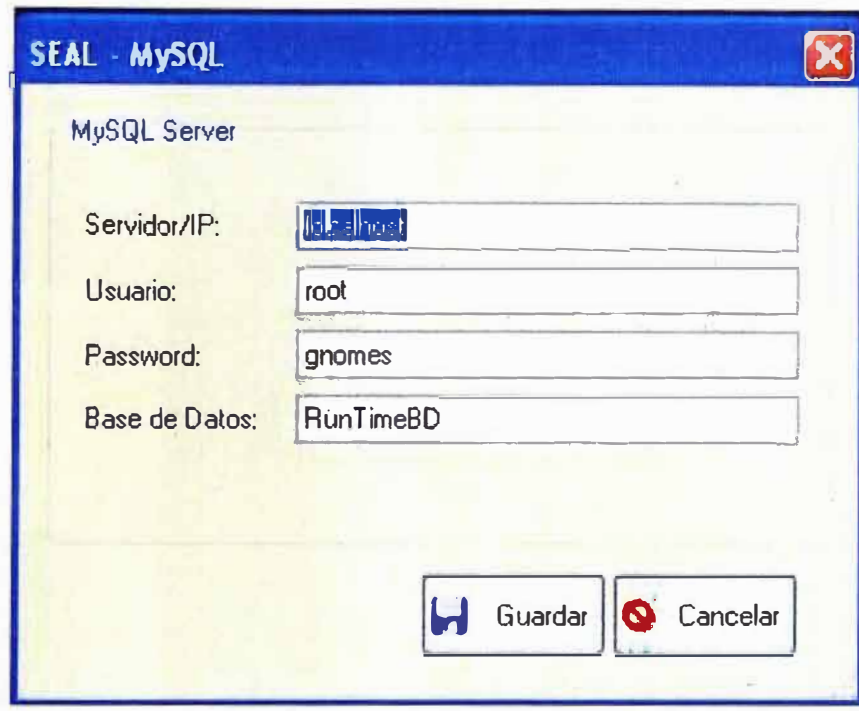

Figura 3.11 Selección de base de datos donde se conecta HMI (Fuente: Elab. propia)

d. Botones de Consultas

Está compuesta de 2 botones

d.1 Botón de Consulta "PROTOCOLOS"

En este entorno el operador podrá consultar sobre algún reporte que se haya realizado en fechas anteriores, simplemente seleccionando el nombre del cliente en la lista de clientes y filtrando por fechas tal como se muestra en la Figura 3.12.

Una vez seleccionado el nombre del cliente y las fechas tentativas de realización del reporte el operador debe presionar el botón EJECUTAR para realizar la consulta.

Una vez realizada la consulta si el operador quisiera recuperar el reporte desde la base de datos, deberá seleccionarla desde la columna NUMERO DE SERIE como se muestra en la Figura 3.13, y luego hacer click sobre el botón REPORTE, una vez hecho esto aparecerá el reporte de la prueba realizada al transformador en dicha fecha.

d.2 Botón de Consulta "TENDENCIAS"

En este entorno el operador visualizar los valores de las variables en gráficos de tendencias (variable VS tiempo). La selección de la variable y el tiempo de la consulta es similar a la de la selección de los reportes, en la Figura 3.14 se muestra el entorno donde se muestran las tendencias.

Toda la información de la programación relacionada con estas pantallas se adjunta en el apéndice A de este informe.

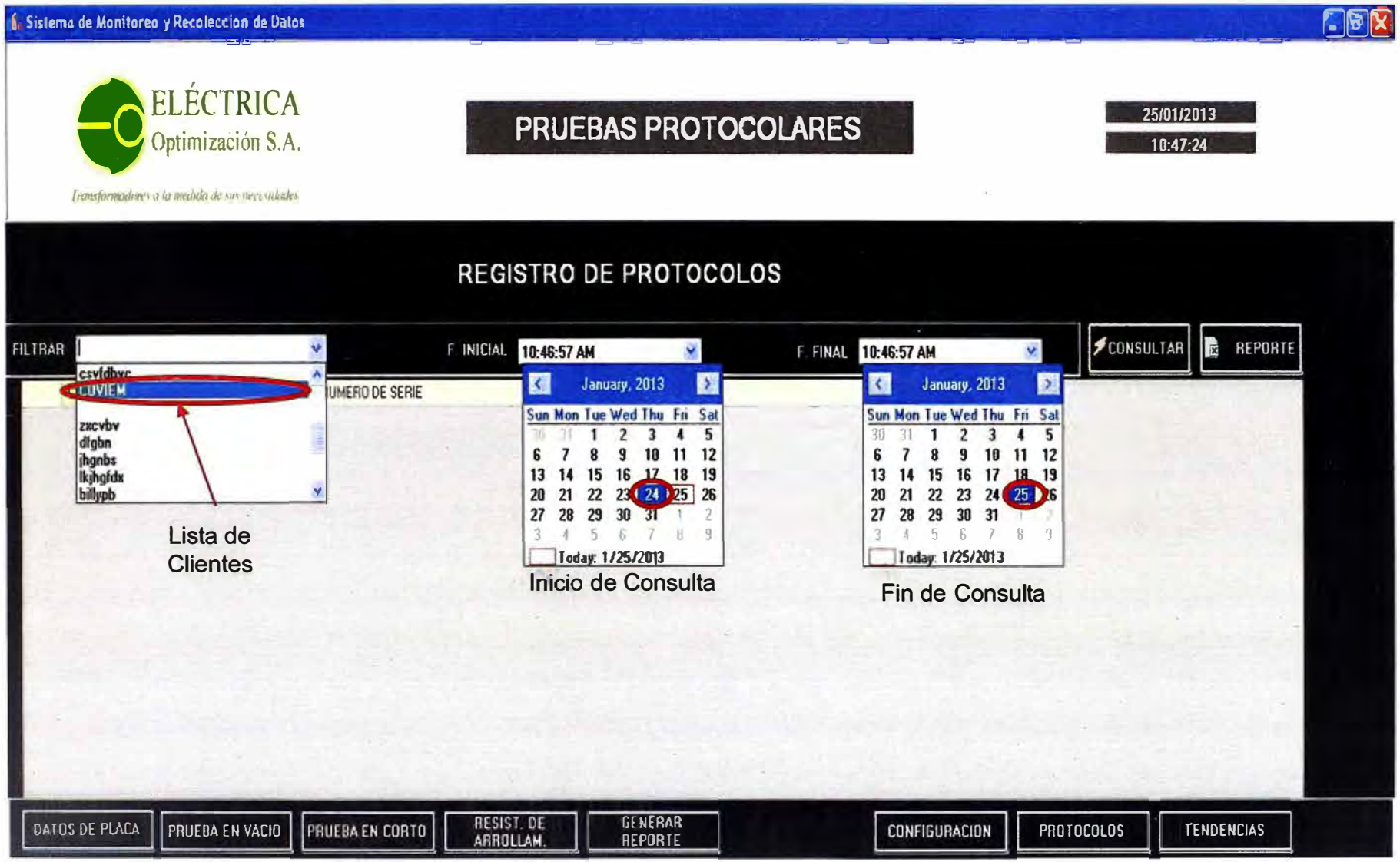

Figura 3.12 Pantalla para consulta de reportes (Fuente: Elab. propia)

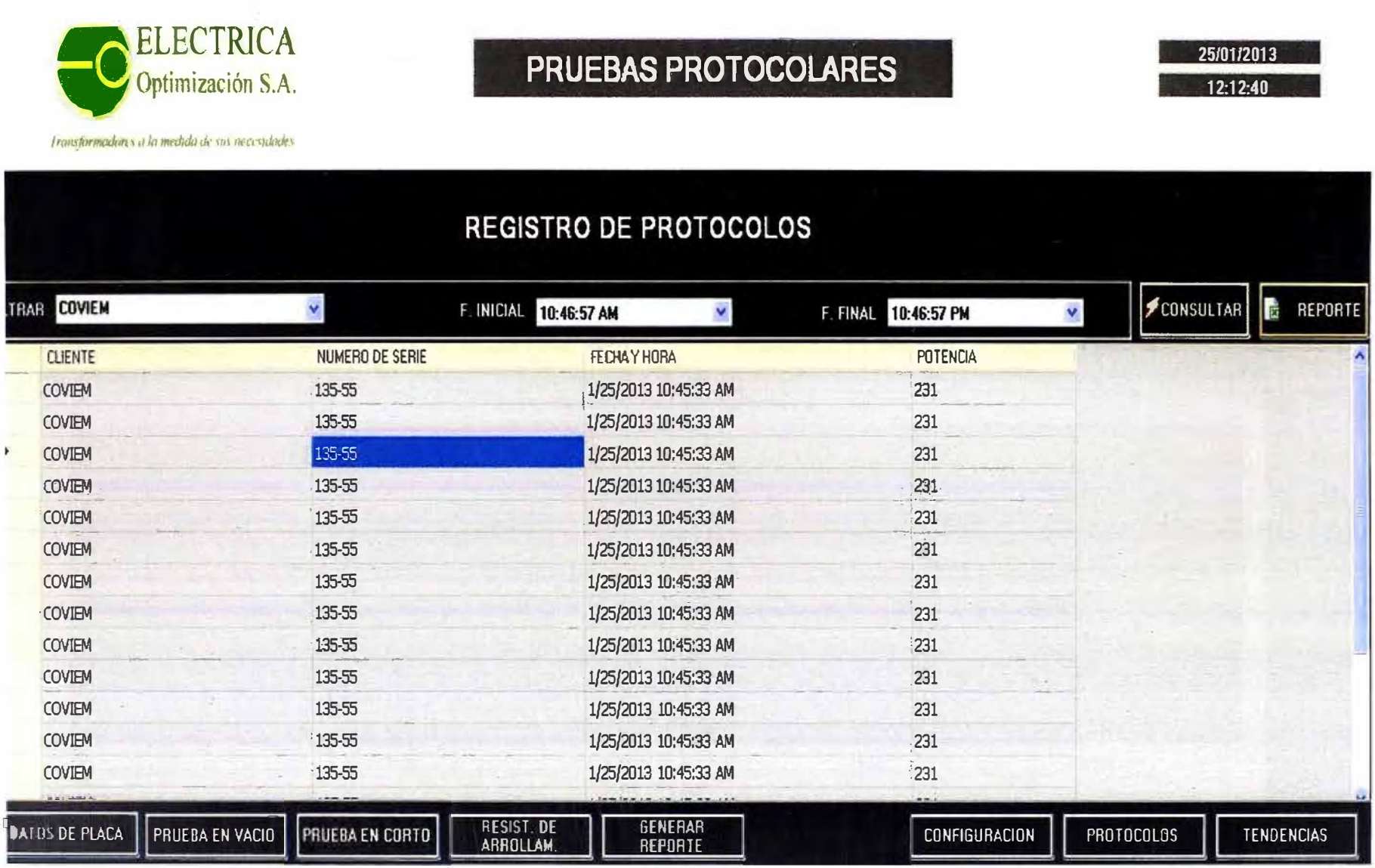

Figura 3.13 Selección de reporte por Número de Serie (Fuente: Elab. propia)

Figura 3.14 Entorno de Tendencias (Fuente: Elab. propia)

3.2.3 Implementación del Sistema de Monitoreo

Para la implementación del sistema propuesto se siguieron los siguientes pasos:

- Instalación del paquete de software del sistema
- Creación de la Base de Datos RunTimeBD
- Configuración del Servidor OPC (KepServer 4.0)
- Configuración del Cliente OPC (Maycens OPC)
- Direccionamiento en HMI

Los cuales se explican a continuación

a. Instalación del paquete de software del sistema

El sistema consta de 5 softwares que deben ser instalados en el siguiente orden:

- Instalación del Conector ODBC.
- Instalación del Gestor de Bases de Datos MySQL.
- Instalación del MySQL Tools.
- Instalación del Servidor OPC.
- Instalación del Cliente OPC.
- Instalación del HMI.

b. Creación de la Base de Datos RunTimeBD

La creación de la base de datos donde se almacenara la data histórica del sistema se hace en MYSQL. Para ello se debe ejecutar el programa MySQL Query Browser, apareciendo la ventana de consultas mostrado en la Figura 3.15.

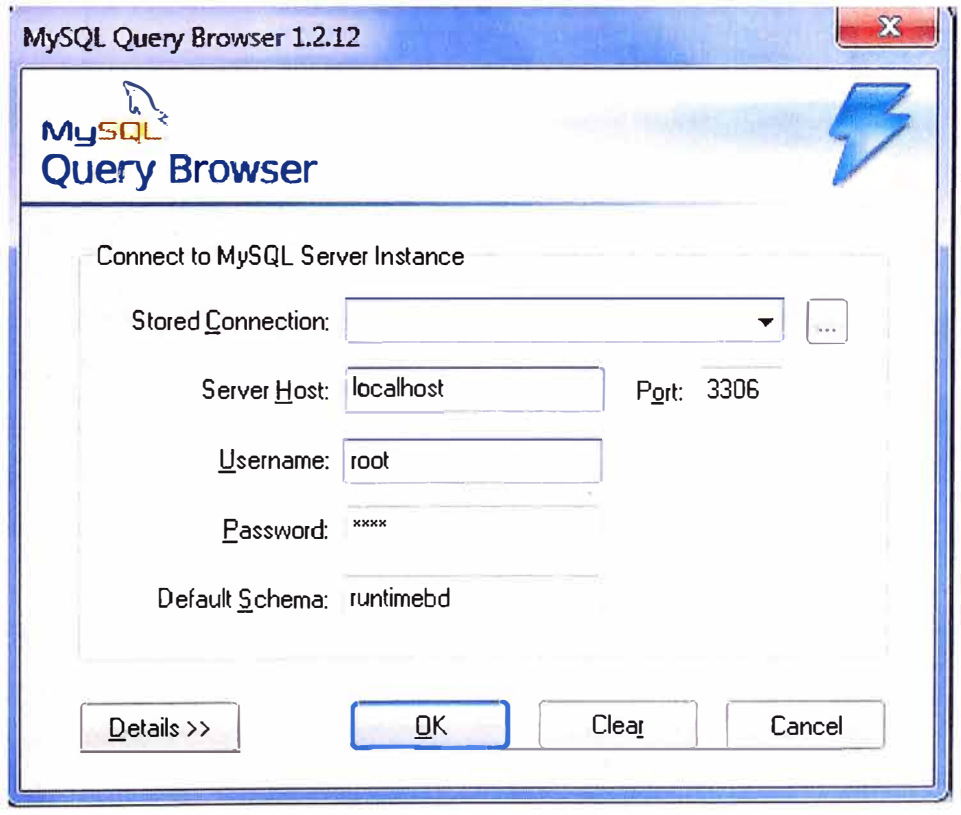

Figura 3.15 Creación de la base de datos RunTimeBD (Fuente: Elaboración propia)

En él se llena la siguiente información:

- Server Host: localhost
- Username: root
- Password: gnomes
- Default Schema: RunTimeBD

c. Configuración del Servidor OPC (KepServer 4.0)

Dentro de la configuración del servidor OPC se menciona lo más resaltante:

c.1 Especificación del protocolo de comunicación

Dentro del proceso de configuración del servidor, el primer paso importante a considerar es el protocolo, en la Figura 3.16 se muestra como:

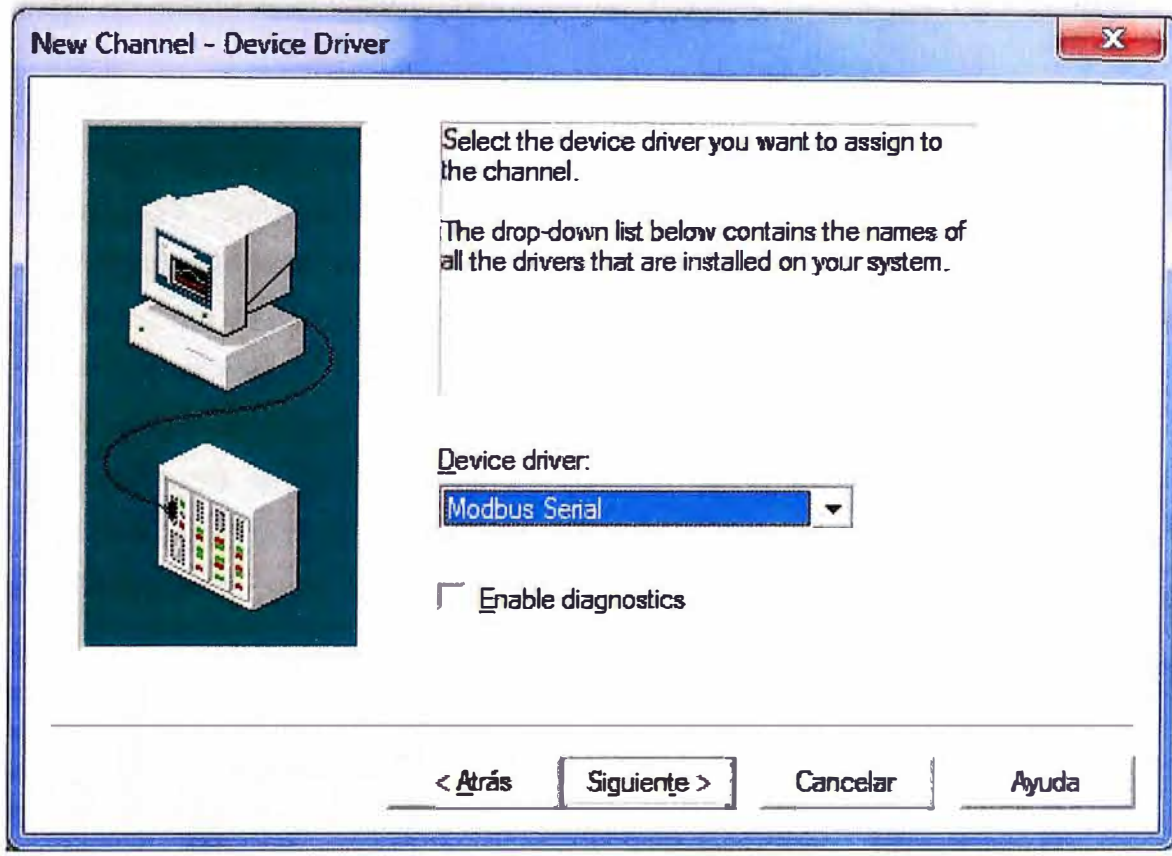

Figura 3.16 Declaración de protocolo de comunicación (Fuente: Elaboración propia)

c.2 Configuración del puerto

El segundo paso importante es la configuración del puerto. Para este caso se ingresó la siguiente información (Figura 3.17):

- ID: COM 1
- Baud Rate: 9600
- Data Bits: 8
- Parity: Even
- Stop Bits: 1
- Flow Control: None

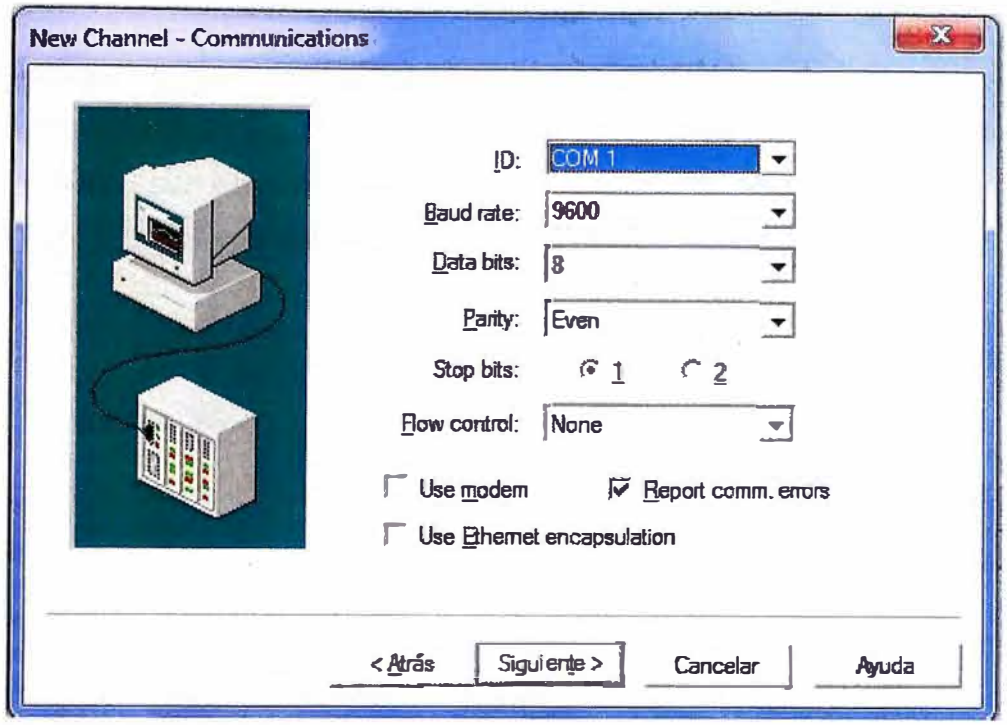

Figura 3.17 Configuración del puerto de comunicación (Fuente: Elaboración propia)

c.3 Configuración de la comunicación con el Dispositivo

En el entorno de desarrollo del KepServer hacer un click sobre: Click to add a Device, seguidamente aparecerá una ventana en donde se debe colocar el nombre del dispositivo a añadir, en este caso se colocamos Medidor1 y luego hacer click en Siguiente.

Seguidamente se debe colocar el nodo del esclavo Modbus (para este caso el esclavo Modbus es el PM130) y se le coloca el nodo 2 como fue configurado en el PM130 previamente. En la Figura 3.18 se muestra como fue configurado el nodo.

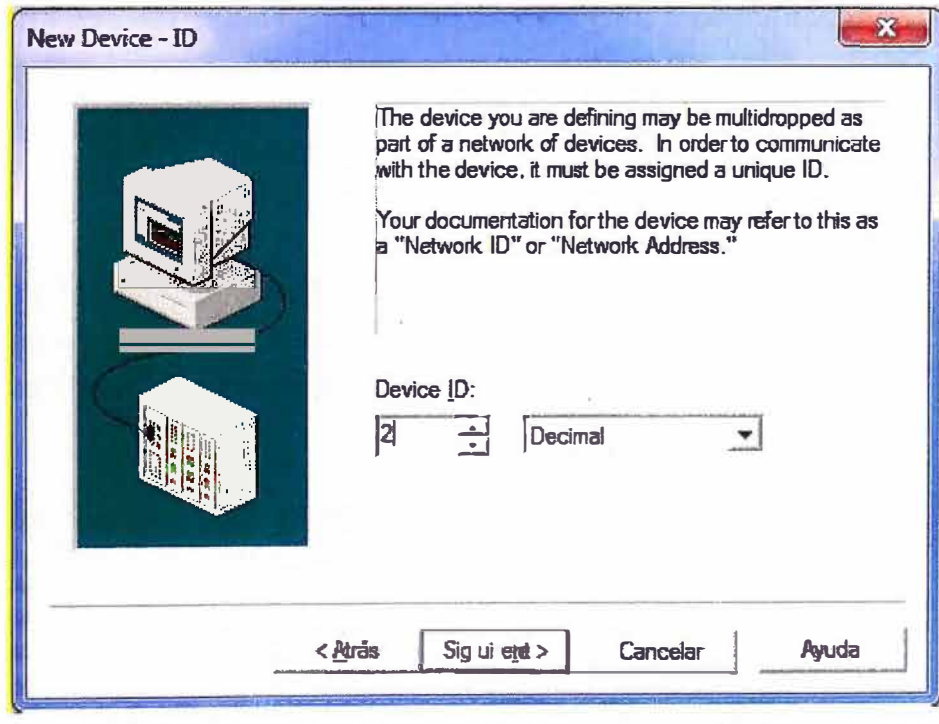

Figura 3.18 Configuración del nodo Modbus (Fuente: Elaboración propia)

c.4 Declaración de Variables en el servidor OPC

El paso siguiente es añadir manualmente las variables a leer. Para esto se hace click derecho sobre el dispositivo creado anteriormente y se escoge la opción: New Tag..., como se muestra en la siguiente Figura 3.19:

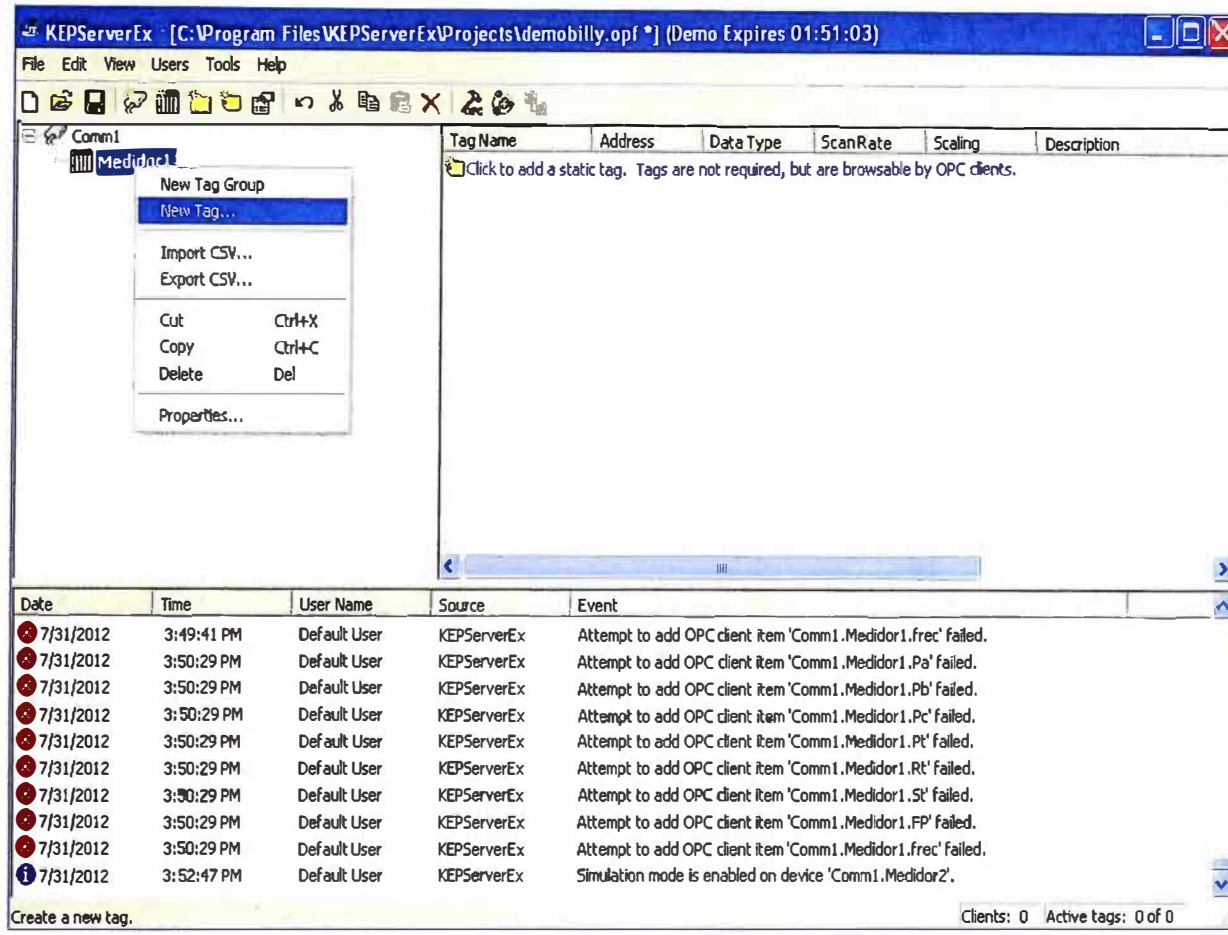

Figura 3.19 Entorno de configuración del servidor OPC (Fuente: Elaboración propia).

En la ventana que aparece a continuación, se escoge la opción General y se procede a llenar los espacios en blanco tal como se muestra en la Figura 3.20:

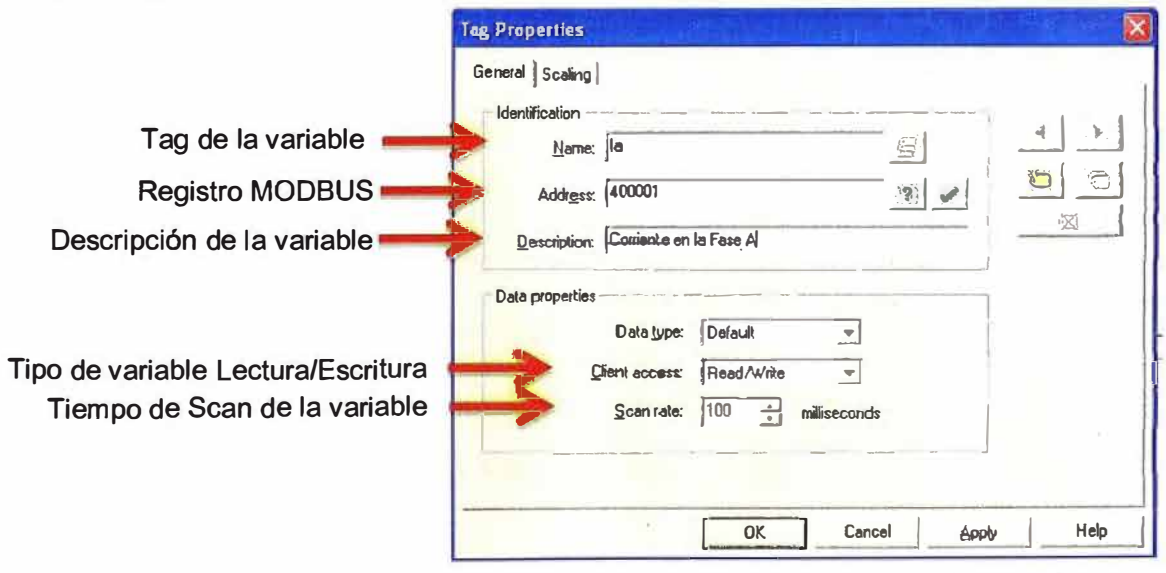

Figura 3.20 Adición manual de tags en servidor OPC (Fuente: Elaboración propia)

Luego en la misma ventana, se escoge la opción "Scaling" para darle un escalamiento a la señal. Para este caso se le da el escalamiento según lo que se indica en el manual del PM130.

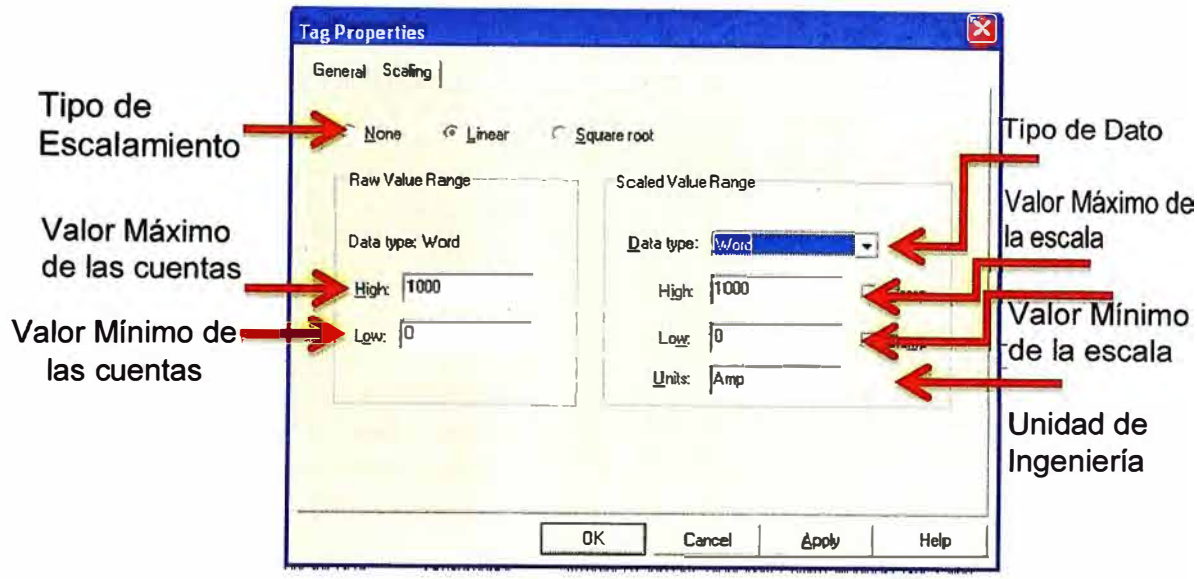

Figura 3.21 Escalamiento de tags en servidor OPC (Fuente: Elaboración propia)

De esta manera se declaran las variables que se desean leer o escribir. Para el proyecto se declaran todas las variables que se muestran en la Tabla 3.1

Tabla 3.1 Lista de tags declarados en el servidor OPC (Fuente Elaboración Propia)

Tag	Dirección	Descripción	Tipo	Escala	Tipo de Escala	Rango Alto	Rango Bajo
Ia	40004	Corriente en la Fase A	Word	Si	Linear	7.5	0
Ib	40005	Corriente en la Fase B	Word	Si	Linear	7.5	0
Ic	40006	Corriente en la Fase C	Word	Si	Linear	7.5	0
Vab	40001	Tensión entre fases A y B	Word	Si	Linear	828	0
Vbc	40002	Tensión entre fases B y C	Word	Si	Linear	828	0
Vca	40003	Tensión entre fases C y A	Word	Si	Linear	828	0
Pa	40007	Potencia Activa en la Fase A	Word	Si	Linear	2070	0
Pb	40008	Potencia Activa en la Fase B	Word	Si	Linear	2070	0
Pc	40009	Potencia Activa en la Fase C	Word	Si	Linear	2070	0
Pt	40020	Potencia Total Activa	Word	Si	Linear	2070	0
frec	40024	Frecuencia	Word	Si	Linear	1000	0

d. Configuración del Cliente OPC (Maycens OPC)

Se siguieron los siguientes pasos:

d.1 Configuración de comunicación con el servidor OPC

En la ruta Configuración->OPC Server, aparecerá la siguiente ventana en donde se debe llenar los espacios en blanco, como se muestra en la siguiente figura:

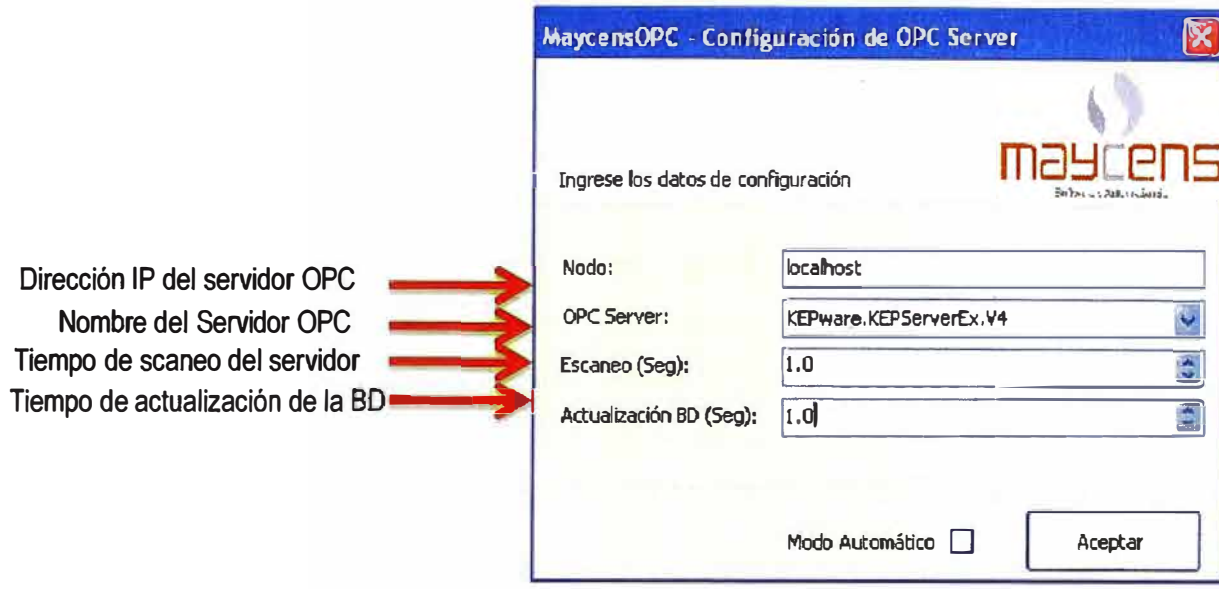

Figura 3.23 Datos de configuración del cliente OPC (Fuente: Elaboración propia)

Nota: Para el caso de que el cliente OPC no se encuentre instalado en la misma PC que el servidor OPC, en el campo: **NODO** se tiene que ingresar la IP de la PC que contenga al servidor.

d.2 Configuración con la Base de Datos MySQL

En la ruta Configuración->MySQL Server, aparecerá la siguiente ventana en donde se debe llenar los espacios en blanco, como se muestra en la Figura 3.24:

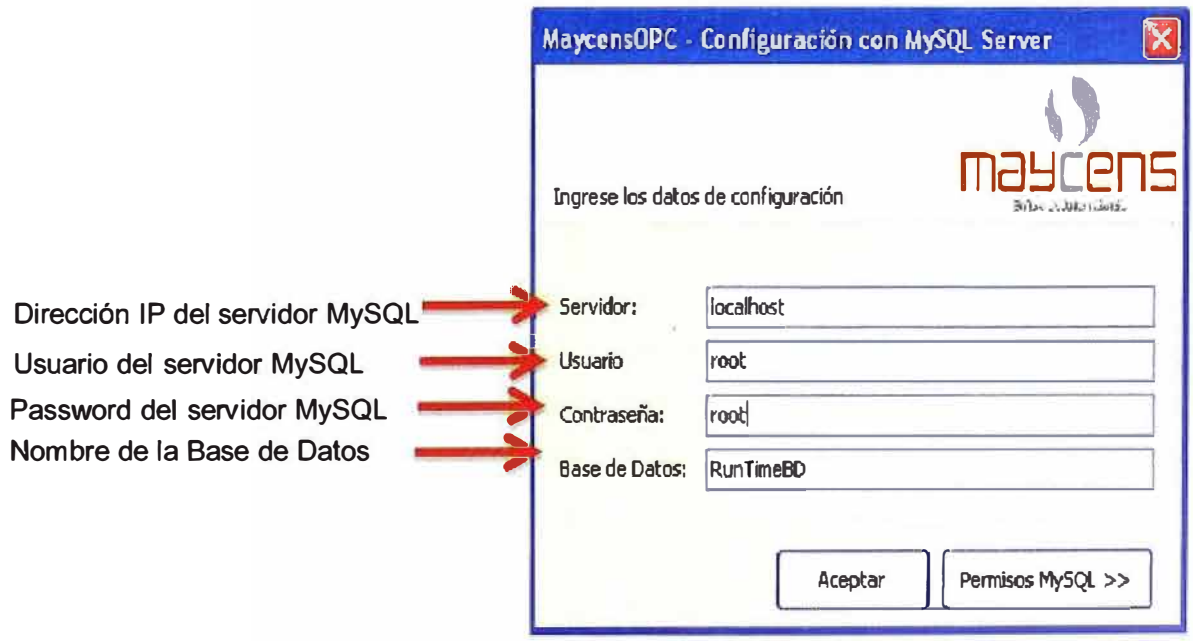

Figura 3.24 Datos de configuración para conexión con MySQL (Fuente: Elab. propia)

El paso siguiente es añadir manualmente las variables a monitorear e historizar. Para esto se hace click en Archivo -> Nuevo

- En el campo TagName, se debe colocar la dirección de la variable siguiendo el siguiente formato:

NP.ND.TV

Donde:

NP: Nombre del Puerto.

ND: Nombre del Dispositivo.

TV: Tag de la Variable.

- En el campo TagDescription se escribe la descripción de la variable.
- En el campo TagDataType se elige el tipo de señal, ya sea Analog o Discret.
- En el campo TagReadValue se visualiza el valor de la variable en tiempo real.
- En el campo TagUnit se escribe la unidad de ingeniería de la variable.
- El campo TagDateTime está referido al tiempo en que fue creada la señal.
- El campo TagQuality es referido a la calidad de la señal, es decir:

Comunicación OK=192 , Falla de Comunicación=0

- En el campo TagHistory se selecciona el tipo de historización: Delta,1, 2, 3, etc. El Delta está referido al tipo de historización por cambio de valor, los números están referidos al tiempo de muestreo para la historización; 1seg, 2seg, etc. En nuestro caso se seleccionó el tipo de historización Delta.
- El campo TagID es el identificador del tag en la Base de Datos RunTimeBD (Sirve únicamente para MySQL).

En la Figura 3.25 se muestra el entorno de configuración del cliente Maycen OPC.

e. Direccionamiento en HMI

El direccionamiento en el HMI se realiza desde el Microsoft Visual Studio, para ello se ejecuta el programa (ruta: Inicio->Todos los Programas->Microsoft Visual Studio-> Microsoft Visual Studio 2010) y a continuación se mostrara el entorno de configuración (Figura 3.26):

El paso a seguir es seleccionar el tag que se quiere direccionar (Figura 3.27, para ello se hace click en el Tab "Propiedades" ubicado en la parte derecha del entorno de configuración. A continuación se llena el campo: TagName con el mismo formato del cliente OPC.

De este modo se direccionan los demás tags según Tabla 3.2 y finalmente se guardan los cambios.

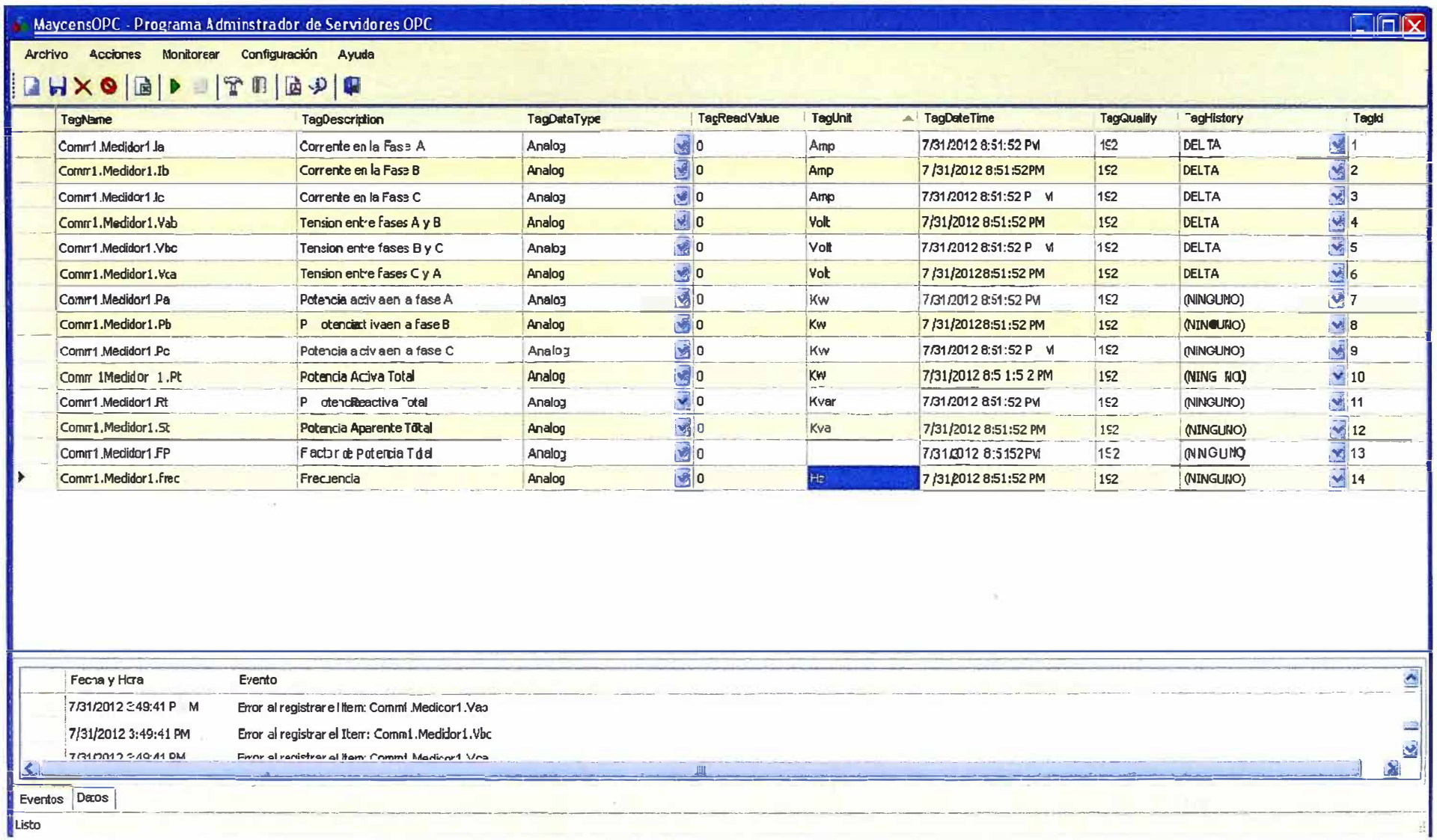

Figura 3.25 Entorno de configuración del Maycen OPC (Fuente: Elaboración propia)

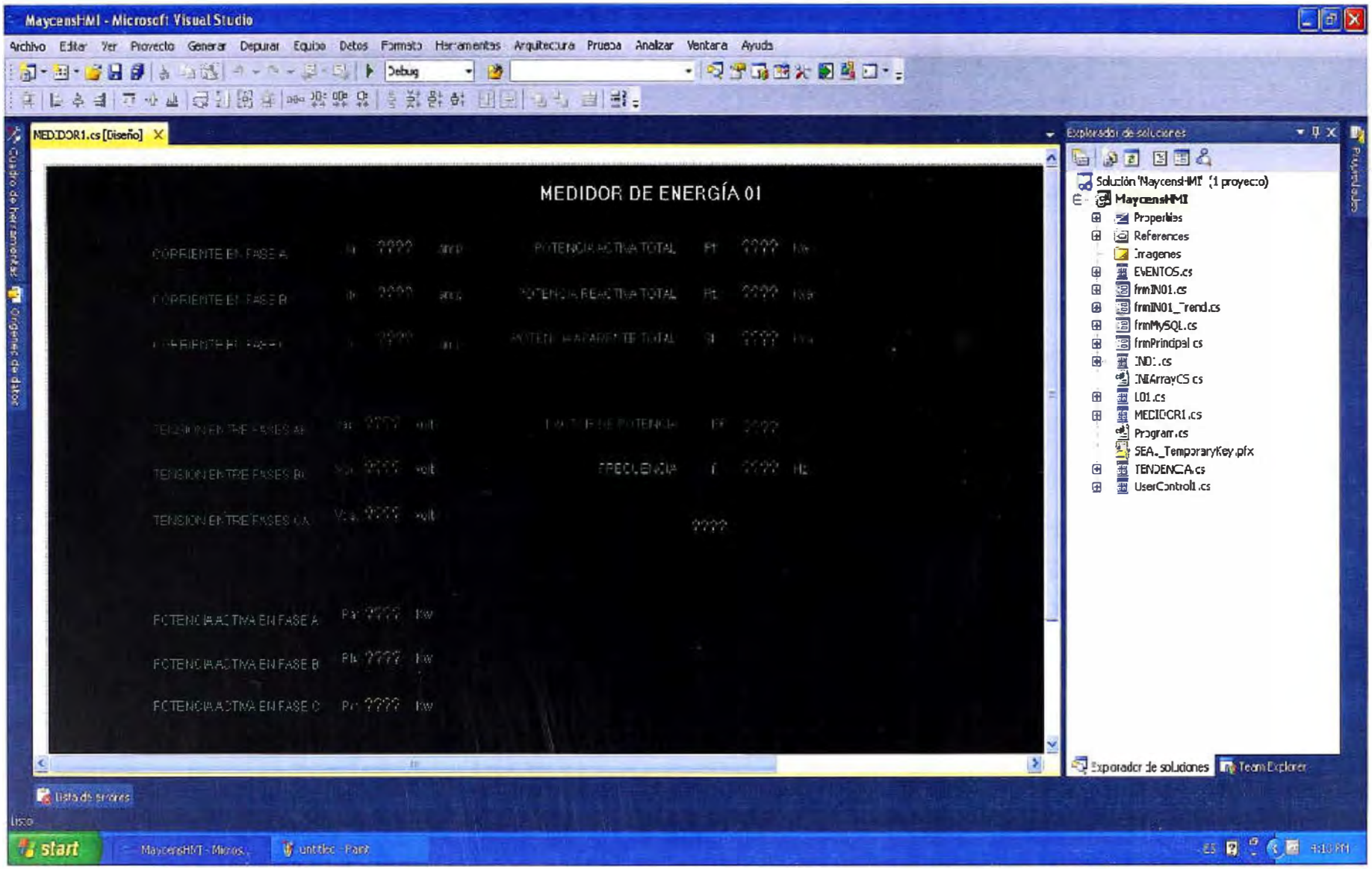

Figura 3.26 Entorno de configuración de HMI usando Visual Studio (Fuente: Elab. Propia)

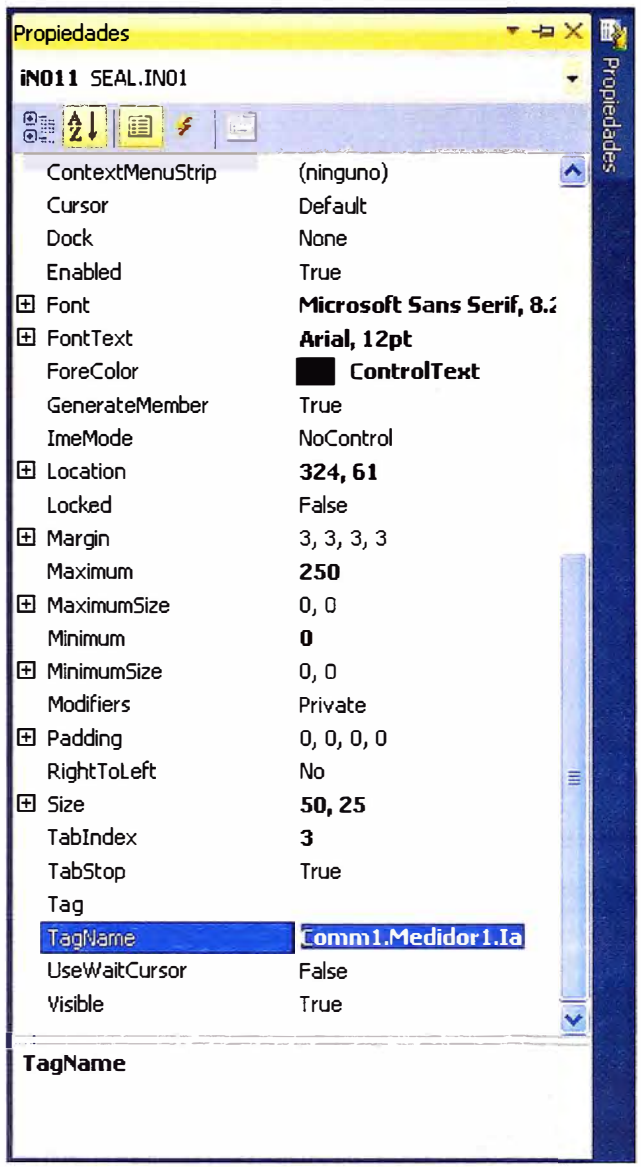

Figura 3.27 Direccionamiento en HMI de tags (Fuente: Elaboración Propia)

Tabla 3.2 Lista de direcciones en HMI para los tags (Fuente Elaboración Propia)

Tag	Dirección	Descripción	Dirección
Ia	40004	Corriente en la Fase A	Comm1.Medidor1.Ia
Ib	40005	Corriente en la Fase B	Comm1.Medidor1.Ib
Ic	40006	Corriente en la Fase C	Comm1.Medidor1.Ic
Vab	40001	Tensión entre fases A y B	Comm1.Medidor1.Vab
Vbc	40002	Tensión entre fases B y C	Comm1.Medidor1.Vbc
Vca	40003	Tensión entre fases C y A	Comm1.Medidor1.Vca
Pa	40007	Potencia Activa en la Fase A	Comm1.Medidor1.Pa
Pb	40008	Potencia Activa en la Fase B	Comm1.Medidor1.Pb
Pc	40009	Potencia Activa en la Fase C	Comm1.Medidor1.Pc
Pt	40020	Potencia Total Activa	Comm1.Medidor1.Pt
frec	40024	Frecuencia	Comm1.Medidor1.Frec

CAPÍTULO IV
ANÁLISIS Y PRESENTACIÓN DE RESULTADOS

En el presente capítulo se tocan los temas involucrados al presupuesto y las tareas realizadas.

4.1 Estimación de costos

La estimación de costos está basada en las horas-hombre del especialista-programador y de las licencias que se adquirieron para el sistema. Los días efectivos son mostrados en la tabla 4.1, los costos por licencias se detallan en la tabla 4.2 y la lista de materiales con sus precios se muestra en la Tabla 4.3.

Tabla 4.1 Relación de días efectivos de trabajo en proyecto (Fuente: Elab. Propia)

TAREAS	40 días	Horas Efectivas
Análisis del problema	1 día	8 horas
Reconocimiento del área de pruebas	1 día	2 horas
Reconocimiento de las pruebas de vacío	1 día	3 horas
Reconocimiento de las pruebas de cortocircuito	1 día	3 horas
Reconocimiento de pruebas de resistencia de arrollamiento	1 día	2 horas
Identificación de características del medidor PM130	3 días	18 horas
Diseño de Red	2 días	16 horas
Diseño de comunicaciones	2 días	16 horas
Configuración de HMI	15 días	75 horas
Estructuración de la Base de Datos	1 día	8 horas
Implementación del Sistema	5 días	40 horas
Pruebas de Comisionamiento	3 días	24 horas
Validación del Sistema	1 día	8 horas
Cierre del Proyecto	1 día	7 horas

La Figura 4.1 muestra el diagrama de Gantt el cual muestra todas las tareas realizadas y su dependencia. Las actividades fueron realizadas entre septiembre y noviembre de 2012, aproximadamente cubriendo 32 días, con 300 HH efectivas.

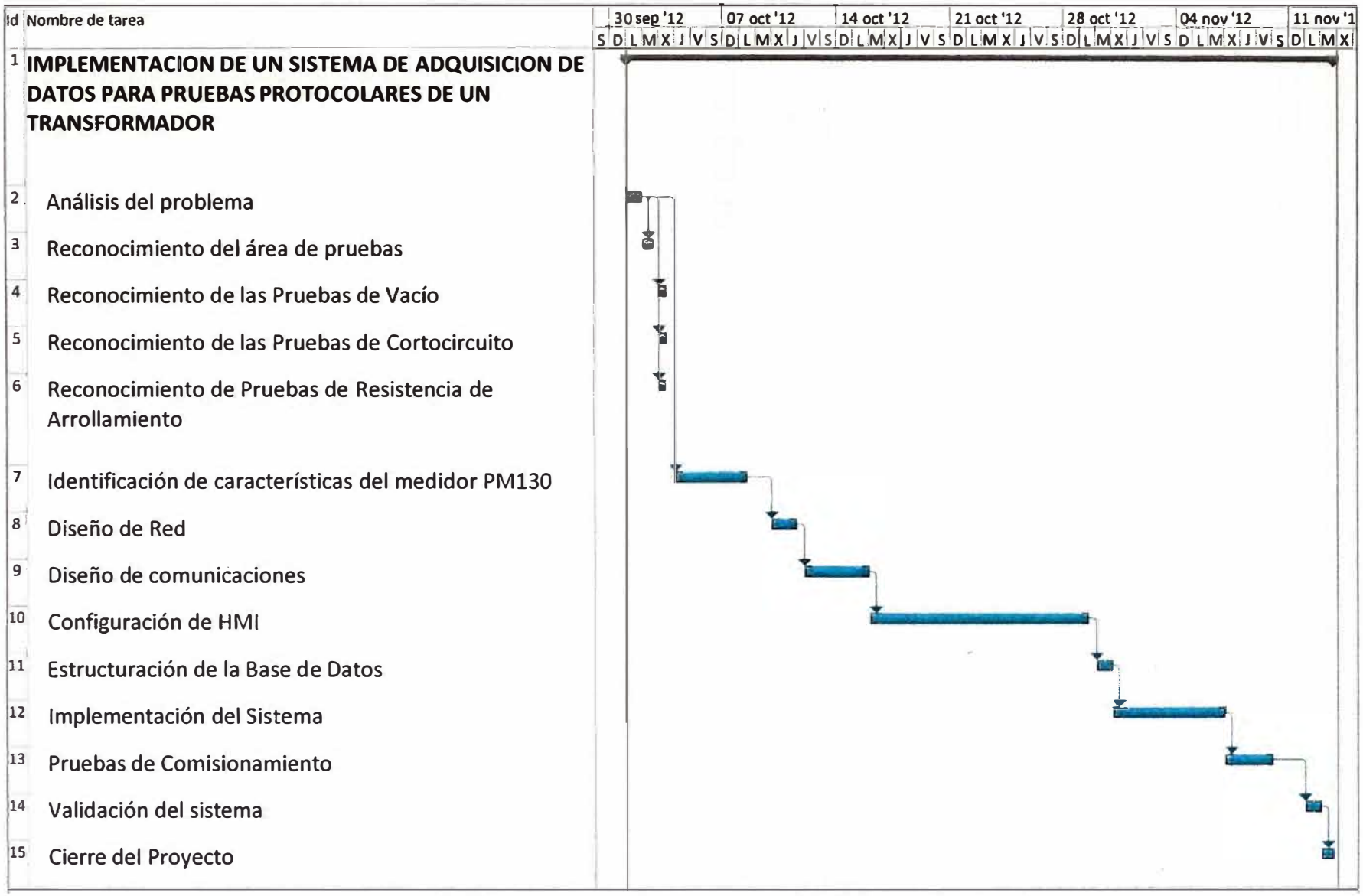

Figura 4.1 Diagrama Gantt del proyecto (Fuente: Elaboración Propia)

Los días mencionados en la tabla anterior se ajustaron de acuerdo a la disponibilidad del especialista, cabe resaltar que la disponibilidad del especialista era de 5 días cada 9 debido a que trabaja en otro lugar.

Además de esto la disponibilidad de los transformadores para el reconocimiento y comisionamiento era de 3 horas por día (1 transformador por hora).

El costo de HH del especialista fue de S/.20.00 lo que hace un total de S/.6000.00

Tabla 4.2 Relación de licencias adquiridas para el proyecto (Fuente: Elab. Propia)

Licencia	IGV (S/.)	Valor Neto (S/.)
Servidor KepServer 4.0	346.95	1927.50
Maycen OPC	144.00	800.00

El costo por licencia asciende a S/. 2727.50 incluido el IGV.

Tabla 4.3 Lista de materiales (Fuente: Elab. Propia)

Materiales	IGV (S/.)	Valor Neto (S/.)
Cable de 2 hilos con apantallamiento (10mts)	11.56	64.25
Conversor RS232 a RS485	13.87	77.1

El costo por los materiales usados asciende a S/.141.35 incluido el IGV.

En resumen el proyecto le costó a la empresa una inversión total de S/. 8868.85.

4.2 Estimación de tiempo de recuperación de la inversión:

Considerando el ahorro que se hace la empresa con el sistema implementado versus lo invertido para este proyecto se concluye lo siguiente:

- Antes la empresa invertía en las pruebas un aproximado de S/. 68.30 al día (3 personas a 4 horas diarias).
- Ahora con el sistema implementado la empresa invierte S/.13.30 al día (1 persona a 4 horas diarias).
- La empresa ahorra un total de S/. 55.00 por día.
- Haciendo cálculos la empresa recuperaria lo invertido en 161 días laborales.

La información de costos de HH se obtuvo del área de Recursos Humanos de la empresa.

4.3 Mantenimiento preventivo del sistema

Para el sistema implementado se propone un plan de mantenimiento preventivo el cual consta de lo siguiente:

- Generación mensual de backup de ordenadores.
- Generación mensual de backup mensuales de la base de datps.
- Generación mensual de backup del HMI.
- Generación mensual de backup de la configuración del cliente OPC

- Generación mensual de backup de la configuración del servidor OPC.
- Limpieza mensual de la interfaz física entre el medidor PM130 y el ordenador.
- Limpieza mensual de los conectores de comunicación del PM130.
- Actualización mensual de Antivirus.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1. El sistema implementado cumplió los objetivos para lo que fue diseñado, visualizar de manera simultánea las corrientes, tensiones, potencias y realizar los reportes de manera automática reduciendo a 1 la cantidad de participantes para las pruebas protocolares.

2. Se demostró que el diseño era amigable, puesto que el operador se familiarizó rápidamente con el HMI y aprendió rápidamente a manejar su nuevo sistema.

3. El éxito del HMI se debió a que se identificó bien el procedimiento de las pruebas y porque se trabajó en coordinación con el operador para diseñar las pantallas.

4. Conforme se avanzaba el proyecto se fueron identificando mejoras que se implementaron de tal manera que ayudo al exitoso desarrollo del sistema.

5. El operador ya no tiene la necesidad de manipular el medidor ya que toda la información la tiene en el HMI.

6. Realizando cálculos de costos por el proyecto versus el ahorro que tiene la empresa con el sistema implementado se deduce que la empresa recuperara lo invertido en 175 días laborales.

Recomendaciones

1. Tener el laboratorio de pruebas a una distancia no menor de 10 mts por temas de seguridad.

2. Tener entrenado a más de un operador que maneje el sistema.

3. Tener un plan de mantenimiento para el sistema, consistente de backups de la base de datos.

4. Adquirir un medidor de relación de transformación que cuente con protocolo Modbus para poder integrarlo al sistema y de esta manera tener integrado al sistema todas las pruebas protocolares que se realizan a los transformadores.

5. Consultar, ante cualquier cambio o mejora al sistema, con el analista que realizo la configuración e implementación del sistema.

BIBLIOGRAFÍA

[1] IEC, "Transformadores de potencia-Generalidades", IEC 60076-1. 2000

[2] Darek Kominek, "Guía para entender la Tecnología OPC", Matrikon OPC, 2009, https://www.matrikonopc.com/downloads/836/whitepapers/index.aspx

[3] Modbus Organization, "Modbus Application Protocol Specification V1.1b", 2006. www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

[4] Texas Instruments. "Interface Circuits for TIA/EIA-232-F", 2002, Curso de introduccion a .NET con Visual Basic 2010 Guillermo Som, Unai Zorrilla, Jorge Serrano.

[5] Guillermo Som, Unai Zorrilla, Jorge Serrano, "Curso de introduccion a .NET con Visual Basic", 2010.

[6] César Pérez, "MySQL para Windows y Linux", Editorial RA-MA, 2004

[7] SATEC, "PM130- Maual de Instalación y configuración", BG0279 Rev A.1

[8] SATEC, "Modbus RTU Protocol Reference Guide", BG0108 Rev C.

[9] Francisco Javier Ceballos "Enciclopedia de Microsoft Visual C#", Editorial RA-MA, 2010.

[10] Yuri Pando Fernández "SQL Server 2008", Editorial MACRO, 2009.

[11] Modicon "Modbus Protocol Reference Guide", PI–MBUS–300 Rev. J

[12] MySQL, "MySQL 5.0 Reference Manual :: 13 Sintaxis de sentencias SQL :: 13.2 Sentencias de manipulación de datos (Data Manipulation Statements)" http://dev.mysql.com/doc/refman/5.0/es/load-data.html

[13] ASPOSE, "Adding New Worksheets to Workbook and Activating a Sheet" http://www.aspose.com/docs/display/cellsnet/Adding+New+Worksheets+to+Workbook+and+Activating+a+Sheet.

[14] Centro Integrado Politécnico ETI tudela, "Sistemas de bus RS485" http://www.etitudela.com/entrenadorcomunicaciones/downloads/introduccionrs485.pdf Dr. Liew Voon Kiong "Visual Basic 2010"

[15] David Chappell "Introducing Visual Studio 2010"

[16] MSDN, "Conectar con datos y recuperarlos en ADO.NET". http://msdn.microsoft.com/es-es/library/ms254937%28v=vs.80%29.aspx

ANEXO A
CÓDIGO FUENTE DEL PROGRAMA HMI

Preliminarmente se muestra el comando para la creacion de la Tabla Protocolo_Pruebas donde se almacenara la informacion de los reportes de las pruebas protocolares. Seguidamente los programas respectivos:

- Programa de pantalla principal
- Programa para pantalla prueba en vacio
- Programa para pantalla pruebas en cortocircuito
- Programa para pantalla resistencia de arrollamientos
- Programa para pantalla protocolos
- Programa para pantalla tendencias
- Programa para pantalla de configuracion

COMANDO PARA LA CREACION DE LA TABLA PROTOCOLO_PRUEBAS

//Comando para la creacion de la Tabla Protocolo_Pruebas donde se almacenara la informacion de los reportes de las pruebas protocolares

```
create table Protocolo_Pruebas (CLIENTE varchar (50),
                                NUMEROSERIE varchar (50),
                                POTENCIA varchar (50),
                                VOLTAJEALTA varchar (50),
                                VOLTAJEBAJA varchar (50),
                                GRUPO varchar (50),
                                FASES varchar (50),
                                ALTURA varchar (50),
                                TEMPERATURA varchar (50),
                                IAVACIO double (50),
                                IBVACIO double (50),
                                ICVACIO double (50),
                                VABVACIO double (50),
                                VBCVACIO double (50),
                                VCAVACIO double (50),
                                FRECUENCIAVACIO double (50),
                                POTENCIAVACIO double (50),
                                IACORTO double (50),
                                IBCORTO double (50),
                                ICCORTO double (50),
                                VABCORTO double (50),
                                VBCCORTO double (50),
                                VCACORTO double (50),
                                FRECUENCIACORTO double (50),
                                POTENCIACORTO double (50),
                                VOLTAJEUVALTA double (50),
                                CORRIENTEUVALTA double (50),
                                VOLTAJEUVBAJA double (50),
                                CORRIENTEUVBAJA double (50),
                                FECHAREALIZACION datetime,
                                primary key(NUMEROSERIE));
```

// PROGRAMA DE PANTALLA PRINCIPAL

```
// PROGRAMA DE PANTALLA PRINCIPAL

// Declaracion de Librerias a utilzar
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
using Office = Microsoft.Office.Core;
using System.Reflection;
using System.Data.Odbc;
using System.Runtime.InteropServices;

namespace SEAL
{
    public partial class frmPrincipal : Form
    {
        private cIniArray mINI = new cIniArray();
        OdbcConnection conexionBD;
        EVENTOS EVENTOS = new EVENTOS();
        TENDENCIA TENDENCIA = new TENDENCIA();
        MEDIDOR MEDIDOR = new MEDIDOR();
        PLACA PLACA = new PLACA();
        RELACTRANSFORM RELACTRANSFORM = new RELACTRANSFORM();
        VACIO VACIO = new VACIO();
        CORTO CORTO = new CORTO();
        RESISTENCIA RESISTENCIA = new RESISTENCIA();
        AISLAMIENTO AISLAMIENTO = new AISLAMIENTO();

        public frmPrincipal()
        {
            InitializeComponent();
        }
//Actualiza Hora y Fecha
        private void timer3_Tick(object sender, EventArgs e)
        {
            lblFecha.Text = DateTime.Now.ToString("dd/MM/yyyy");
            lblHora.Text = DateTime.Now.ToString("HH:mm:ss");
        }
//Funcion que se Ejecuta cuando se carga la pantalla principal
        private void frmPrincipal_Load(object sender, EventArgs e)
        {
            BotonReporte.Visible = true;
            BotonNuevoReporte.Visible = false;
            BotonVerReporte.Visible = false;
            lblFecha.Text = DateTime.Now.ToString("dd/MM/yyyy");    // Actualiza
fecha en la etiqueta
            lblHora.Text = DateTime.Now.ToString("HH:mm:ss");       // Actualiza
hora en la etiqueta

            try // Funcion para conectarse a MySQL
            {
                string sFicINI;
```

```
                sFicINI = System.Windows.Forms.Application.StartupPath +
@"\FileINI.ini"; // Carga el archivo FileINI.ini donde se encuentra el password y el
usuario para conectarse al MySQL

                string Servidor = mINI.IniGet(sFicINI, "MySQL", "servidor", "");
                string Usuario = mINI.IniGet(sFicINI, "MySQL", "Usuario", "");
                string Password = mINI.IniGet(sFicINI, "MySQL", "Password", "");
                string BaseDatos = mINI.IniGet(sFicINI, "MySQL", "BaseDatos", "");
                //Comando para conectarse a MySQL mediante el conector ODBC
                conexionBD = new OdbcConnection("driver={Mysql odbc 5.1
driver};server=" + Servidor + ";user=" + Usuario + ";password=" + Password +
";database=" + BaseDatos);
                conexionBD.Open();

                //Habilita los Timers si la conexion es exitosa
                timer1.Enabled = true;
                timer2.Enabled = true;
            }
            catch (Exception ex) //Deshabilita los Timers si la conexion es fallida
            {
                timer1.Enabled = false;
                timer2.Enabled = false;
                // Muestra mensaje de error
                MessageBox.Show(ex.Message, "MAYCENS - Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void btnConfiguracion_Click(object sender, EventArgs e)
        {
            // Boton de Configuracion
            frmMySQL frmMySQL = new frmMySQL();
            frmMySQL.ShowDialog();
        }

        private void timer1_Tick(object sender, EventArgs e) // Timer para
actualizar la lectura de los valores del medidor
        {
            //Conexion a la base de datos para visualizar los valores en tiempo real
            string sFicINI;

            sFicINI = System.Windows.Forms.Application.StartupPath +
@"\FileINI.ini";

            try
            {
                string sql = "select TagName, TagReadValue, TagQuality,
TagDescription, TagUnit from RunTime";

                OdbcDataAdapter adaptador = new OdbcDataAdapter(sql, conexionBD);

                System.Data.DataTable dt = new System.Data.DataTable();
                adaptador.Fill(dt);

                for (int i = 0; i < dt.Rows.Count; i++)
                {
                    mINI.IniWrite(sFicINI, "TagReadValue", dt.Rows[i][0].ToString(),
dt.Rows[i][1].ToString());//Valor
                    mINI.IniWrite(sFicINI, "TagQuality", dt.Rows[i][0].ToString(),
dt.Rows[i][2].ToString());//Cualidad
```

```
                    mINI.IniWrite(sFicINI, "TagDescription",
dt.Rows[i][0].ToString(), dt.Rows[i][3].ToString());//Descripcion
                    mINI.IniWrite(sFicINI, "TagUnit", dt.Rows[i][0].ToString(),
dt.Rows[i][4].ToString());//Unidad
                }
            }
            catch { }
        }
//      Actualizar Base de datos interna del HMI
        private void ActualizaDGV1()
        {
            try
            {
                for (int i = 0; i < DGV1.Rows.Count; i++)
                {
                    string sql = "select TagDateTime, TagReadValue, TagQuality from
RunTime where TagName='" + DGV1.Rows[i].Cells[1].Value.ToString() + "'";

                    OdbcDataAdapter adaptador = new OdbcDataAdapter(sql,
conexionBD);
                    System.Data.DataTable dt = new System.Data.DataTable();
                    adaptador.Fill(dt);

                    DGV1.Rows[i].Cells[0].Value = dt.Rows[0][0].ToString();
                    DGV1.Rows[i].Cells[3].Value = dt.Rows[0][1].ToString();
                    DGV1.Rows[i].Cells[5].Value = dt.Rows[0][2].ToString();
                }
            }
            catch (Exception ex)
            {
                timer2.Enabled = false;

                MessageBox.Show(ex.Message, "MAYCENS - Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
//      Invoca la funcion de actualizacion de base de datos DGV1 interna del HMI
        private void timer2_Tick(object sender, EventArgs e)
        {
            ActualizaDGV1();
        }
//      Exportar a un archivo excel los valores de la base de datos RunTime
        private void ExportarDGV1()
        {
            Excel.Application excel = new Excel.Application();
            excel.Application.Workbooks.Add(true);
            //Ingreso de Encabezados
            for (int k = 1; k <= DGV1.Columns.Count; k++)
            {
                excel.Cells[1, k] = Convert.ToString(DGV1.Columns[k - 1].Name);
            }
            excel.get_Range("A1", "Z1").Font.Bold = true;
            excel.Columns.ColumnWidth = 20;
            //Ingreso de Datos
            for (int i = 1; i <= DGV1.Rows.Count; i++)
            {
                for (int j = 1; j <= DGV1.Columns.Count; j++)
                {
                    excel.Cells[i + 1, j] = DGV1.Rows[i -
1].Cells[Convert.ToString(DGV1.Columns[j - 1].Name)].Value;
                }
```

```
            }
            excel.Visible = true;
            Excel.Worksheet worksheet = (Excel.Worksheet)excel.ActiveSheet;
            worksheet.Activate();
        }
//      Funcion para conexion a base de datos mediante conector ODBC
        private string Ret_Cadena(string sSel)
        {
            string Valor = "";

            try
            {
                OdbcDataAdapter adaptador = new OdbcDataAdapter(sSel, conexionBD);

                System.Data.DataTable dt = new System.Data.DataTable();
                adaptador.Fill(dt);

                Valor = dt.Rows[0][0].ToString();
            }
            catch { }

            return Valor;
        }
//Animacion para abrir pantalla para ingresar los Datos de Placa del Transformador
        private void BotonPlaca_Click(object sender, EventArgs e)
        {
            this.Controls.Remove(EVENTOS);
            this.Controls.Remove(TENDENCIA);
            this.Controls.Remove(VACIO);
            this.Controls.Remove(RELACTRANSFORM);
            this.Controls.Remove(CORTO);
            this.Controls.Remove(RESISTENCIA);
            this.Controls.Remove(AISLAMIENTO);
            this.Controls.Add(PLACA);
        }
//Animacion para abrir pantalla para ingresar los datos de Relacion de
Transformacion
        private void BotonRelac_Click(object sender, EventArgs e)
        {
            EVENTOS.Location = new System.Drawing.Point(0, 50);
            this.Controls.Remove(TENDENCIA);
            this.Controls.Remove(VACIO);
            this.Controls.Remove(PLACA);
            this.Controls.Remove(EVENTOS);
            this.Controls.Remove(CORTO);
            this.Controls.Remove(RESISTENCIA);
            this.Controls.Remove(AISLAMIENTO);
            this.Controls.Add(RELACTRANSFORM);
        }
//Animacion para abrir pantalla de Pruebas en Vacio del transformador
        private void BotonVacio_Click(object sender, EventArgs e)
        {
            MEDIDOR.Location = new System.Drawing.Point(0, 50);
            this.Controls.Remove(EVENTOS);
            this.Controls.Remove(TENDENCIA);
            this.Controls.Remove(PLACA);
            this.Controls.Remove(RELACTRANSFORM);
            this.Controls.Remove(CORTO);
            this.Controls.Remove(RESISTENCIA);
            this.Controls.Remove(AISLAMIENTO);
            this.Controls.Add(VACIO);
```

```
        }
//Animacion para abrir pantalla de Pruebas en Cortocircuito del transformador
        private void BotonCorto_Click(object sender, EventArgs e)
        {
            EVENTOS.Location = new System.Drawing.Point(0, 50);
            this.Controls.Remove(TENDENCIA);
            this.Controls.Remove(VACIO);
            this.Controls.Remove(PLACA);
            this.Controls.Remove(EVENTOS);
            this.Controls.Remove(RELACTRANSFORM);
            this.Controls.Remove(RESISTENCIA);
            this.Controls.Remove(AISLAMIENTO);
            this.Controls.Add(CORTO);
        }
//Animacion para abrir pantalla de Pruebas de Resistencia de Arrollamientos
        private void BotonResistencia_Click(object sender, EventArgs e)
        {
            EVENTOS.Location = new System.Drawing.Point(0, 50);
            this.Controls.Remove(TENDENCIA);
            this.Controls.Remove(VACIO);
            this.Controls.Remove(PLACA);
            this.Controls.Remove(EVENTOS);
            this.Controls.Remove(RELACTRANSFORM);
            this.Controls.Remove(CORTO);
            this.Controls.Remove(AISLAMIENTO);
            this.Controls.Add(RESISTENCIA);
        }
//Animacion para abrir pantalla de Pruebas de Aislamiento
        private void BotonAislamiento_Click(object sender, EventArgs e)
        {
            EVENTOS.Location = new System.Drawing.Point(0, 50);
            this.Controls.Remove(TENDENCIA);
            this.Controls.Remove(VACIO);
            this.Controls.Remove(PLACA);
            this.Controls.Remove(EVENTOS);
            this.Controls.Remove(RELACTRANSFORM);
            this.Controls.Remove(CORTO);
            this.Controls.Remove(RESISTENCIA);
            this.Controls.Add(AISLAMIENTO);
        }
//Animacion para abrir pantalla de Eventos
        private void BotonEvento_Click(object sender, EventArgs e)
        {
            EVENTOS.Location = new System.Drawing.Point(0, 50);
            this.Controls.Remove(TENDENCIA);
            this.Controls.Remove(VACIO);
            this.Controls.Remove(PLACA);
            this.Controls.Remove(RELACTRANSFORM);
            this.Controls.Remove(CORTO);
            this.Controls.Remove(RESISTENCIA);
            this.Controls.Remove(AISLAMIENTO);
            this.Controls.Add(EVENTOS);
        }
//Animacion para abrir pantalla de Tendencias
        private void BotonTendencia_Click(object sender, EventArgs e)
        {
            TENDENCIA.Location = new System.Drawing.Point(0, 50);
            this.Controls.Remove(EVENTOS);
            this.Controls.Remove(VACIO);
            this.Controls.Remove(PLACA);
            this.Controls.Remove(RELACTRANSFORM);
```

```
            this.Controls.Remove(CORTO);
            this.Controls.Remove(RESISTENCIA);
            this.Controls.Remove(AISLAMIENTO);
            this.Controls.Add(TENDENCIA);
        }
//Boton para generar los reportes de las pruebas protocolares
        private void BotonReporte_Click(object sender, EventArgs e)
        {
            try
            {   // Se crea la cadena de texto para insertar valores a la tabla
Protocolo_Pruebas que son tomadas desde el HMI y la tabla RunTime.
                string MySQL = "insert into Protocolo_Pruebas values('" +
PLACA.BoxCliente.Text + "','" + PLACA.BoxNumeroSerie.Text + "','" +
PLACA.BoxPotencia.Text + "','" + PLACA.BoxVoltajeAlta.Text + "','" +
PLACA.BoxVoltajeBaja.Text + "','" + PLACA.BoxGrupo.Text + "','" +
PLACA.BoxFases.Text + "','" +

PLACA.BoxAltura.Text + "','" + PLACA.BoxTemperatura.Text + "','" +
VACIO.Ia_vacio_Reporte.Text + "','" + VACIO.Ib_vacio_Reporte.Text + "','" +
VACIO.Ic_vacio_Reporte.Text + "','" + VACIO.Vab_vacio_Reporte.Text + "','" +
VACIO.Vbc_vacio_Reporte.Text + "','" + VACIO.Vca_vacio_Reporte.Text + "','" +

VACIO.Frec_vacio_Reporte.Text + "','" + VACIO.Pt_vacio_Reporte.Text + "','" +
CORTO.Ia_corto_Reporte.Text + "','" + CORTO.Ib_corto_Reporte.Text + "','" +
CORTO.Ic_corto_Reporte.Text + "','" + CORTO.Vab_corto_Reporte.Text + "','" +
CORTO.Vbc_corto_Reporte.Text + "','" + CORTO.Vca_corto_Reporte.Text + "','" +

CORTO.Frec_corto_Reporte.Text + "','" + CORTO.Pt_corto_Reporte.Text + "','" +
RESISTENCIA.BoxVoltajeUV_Alta.Text + "','" + RESISTENCIA.BoxCorrienteUV_Alta.Text +
"','" + RESISTENCIA.BoxVoltajeUV_Baja.Text + "','" +
RESISTENCIA.BoxCorrienteUV_Baja.Text + "','" +

DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss") + "');";
//              Comando para conectarse a MySQL mediante el conector ODBC
                OdbcDataAdapter adaptador = new OdbcDataAdapter(MySQL, conexionBD);
                System.Data.DataTable dt = new System.Data.DataTable();
                adaptador.Fill(dt);
                MessageBox.Show("Datos guardados correctamente", "MAYCENS - Info",
MessageBoxButtons.OK, MessageBoxIcon.Information);
//              Animacion de visibilidad para botones de reporte
                BotonReporte.Visible = false;
                BotonNuevoReporte.Visible = true;
                BotonVerReporte.Visible = true;
//              Comandos para coectarse a Microsoft Excel
                Excel.Application excel = new Excel.Application();
//              Enlace con la plantilla Excel Predefinida
                Excel.Workbook wb = excel.Workbooks.Open(@"e:\Plantilla de
Protocolos.xls");
//              Comandos para llenar las celdas correspondientes de la plantilla
                //Datos de Placa
                excel.Cells[4, 2] = PLACA.BoxCliente.Text;
                excel.Cells[3, 13] = PLACA.BoxNumeroSerie.Text;
                excel.Cells[6, 3] = PLACA.BoxPotencia.Text;
                excel.Cells[8, 3] = PLACA.BoxVoltajeAlta.Text;
                excel.Cells[9, 3] = PLACA.BoxVoltajeBaja.Text;
                excel.Cells[11, 8] = PLACA.BoxGrupo.Text;
                excel.Cells[8, 8] = PLACA.BoxFases.Text;
                excel.Cells[6, 13] = PLACA.BoxAltura.Text;
                excel.Cells[28, 14] = PLACA.BoxTemperatura.Text;
                //Datos de Pruebas de Resistencia
                excel.Cells[35, 4] = RESISTENCIA.BoxVoltajeUV_Alta.Text;
```

```
                excel.Cells[35, 6] = RESISTENCIA.BoxCorrienteUV_Alta.Text;
                excel.Cells[36, 4] = RESISTENCIA.BoxCorrienteUV_Baja.Text;
                excel.Cells[36, 6] = RESISTENCIA.BoxCorrienteUV_Baja.Text;
                //Datos de Pruebas en Vacio
                excel.Cells[25, 1] = VACIO.Vab_vacio_Reporte.Text;
                excel.Cells[26, 1] = VACIO.Vbc_vacio_Reporte.Text;
                excel.Cells[27, 1] = VACIO.Vca_vacio_Reporte.Text;
                excel.Cells[25, 4] = VACIO.Ia_vacio_Reporte.Text;
                excel.Cells[26, 4] = VACIO.Ib_vacio_Reporte.Text;
                excel.Cells[27, 4] = VACIO.Ic_vacio_Reporte.Text;
                excel.Cells[25, 8] = VACIO.Pt_vacio_Reporte.Text;
                //Datos de pruebas en Cortocircuito
                excel.Cells[30, 1] = CORTO.Vab_corto_Reporte.Text;
                excel.Cells[31, 1] = CORTO.Vbc_corto_Reporte.Text;
                excel.Cells[32, 1] = CORTO.Vca_corto_Reporte.Text;
                excel.Cells[30, 4] = CORTO.Ia_corto_Reporte.Text;
                excel.Cells[31, 4] = CORTO.Ib_corto_Reporte.Text;
                excel.Cells[32, 4] = CORTO.Ic_corto_Reporte.Text;
                excel.Cells[30, 8] = CORTO.Pt_corto_Reporte.Text;
                excel.Cells[6, 8] = CORTO.Frec_corto_Reporte.Text;
                excel.Cells[43, 3] = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
                //Comandos para guardar el archivo con el nombre del numero de serie
                Excel.Worksheet ws = wb.ActiveSheet;
                ws.SaveAs(@"e:\Protocolos\" + PLACA.BoxNumeroSerie.Text + ".xls");
                excel.Workbooks.Close();
                Marshal.ReleaseComObject(excel);

            }
            catch (Exception ex) // Funcion en caso no se pueda realizar la accion
de Try
            {
                MessageBox.Show(ex.Message, "MAYCENS - Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }

        }
        //Funcion del Logo
        private void panel1_Paint(object sender, PaintEventArgs e)
        {

        }

        private void DGV1_CellContentClick(object sender, DataGridViewCellEventArgs
e)
        {

        }
//      Funcion del boton Nuevo Reporte
        private void BotonNuevoReporte_Click(object sender, EventArgs e)
        {
            //Animacion de visibilidad para los datos de Placa
            PLACA.BoxCliente.Text = " ";
            PLACA.BoxNumeroSerie.Text = " ";
            PLACA.BoxPotencia.Text = " ";
            PLACA.BoxVoltajeAlta.Text = " ";
            PLACA.BoxVoltajeBaja.Text = " ";
            PLACA.BoxGrupo.Text = " ";
            PLACA.BoxFases.Text = " ";
            PLACA.BoxAltura.Text = " ";
            PLACA.BoxTemperatura.Text = " ";
            RESISTENCIA.BoxVoltajeUV_Alta.Text = " ";
```

```
            RESISTENCIA.BoxCorrienteUV_Alta.Text = " ";
            RESISTENCIA.BoxVoltajeUV_Baja.Text = " ";
            RESISTENCIA.BoxCorrienteUV_Baja.Text = " ";
            //Animacion de visibilidad para los datos de Pruebas en Vacio
            VACIO.Ia_vacio_Reporte.Visible = false;
            VACIO.Ib_vacio_Reporte.Visible = false;
            VACIO.Ic_vacio_Reporte.Visible = false;
            VACIO.Vab_vacio_Reporte.Visible = false;
            VACIO.Vbc_vacio_Reporte.Visible = false;
            VACIO.Vca_vacio_Reporte.Visible = false;
            VACIO.Frec_vacio_Reporte.Visible = false;
            VACIO.Pt_vacio_Reporte.Visible = false;
            //Animacion de visibilidad para los datos de Pruebas en vacio
            VACIO.Ia_vacio.Visible = true;
            VACIO.Ib_vacio.Visible = true;
            VACIO.Ic_vacio.Visible = true;
            VACIO.Vab_vacio.Visible = true;
            VACIO.Vbc_vacio.Visible = true;
            VACIO.Vca_vacio.Visible = true;
            VACIO.Frec_vacio.Visible = true;
            VACIO.Pt_vacio.Visible = true;
            //Animacion de visibilidad para los datos de Pruebas en cortocircuito
            CORTO.Ia_corto_Reporte.Visible = false;
            CORTO.Ib_corto_Reporte.Visible = false;
            CORTO.Ic_corto_Reporte.Visible = false;
            CORTO.Vab_corto_Reporte.Visible = false;
            CORTO.Vbc_corto_Reporte.Visible = false;
            CORTO.Vca_corto_Reporte.Visible = false;
            CORTO.Frec_corto_Reporte.Visible = false;
            CORTO.Pt_corto_Reporte.Visible = false;
            //Animacion de visibilidad para los datos de Pruebas en cortocircuito
            CORTO.Ia_corto.Visible = true;
            CORTO.Ib_corto.Visible = true;
            CORTO.Ic_corto.Visible = true;
            CORTO.Vab_corto.Visible = true;
            CORTO.Vbc_corto.Visible = true;
            CORTO.Vca_corto.Visible = true;
            CORTO.Frec_corto.Visible = true;
            CORTO.Pt_corto.Visible = true;
            //Animacion de visibilidad para los botones de Reporte
            BotonReporte.Visible = true;
            BotonNuevoReporte.Visible = false;
            BotonVerReporte.Visible = false;
        }
//      Funcion para boton Ver Reporte
        private void BotonVerReporte_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(@"e:\Protocolos\" +
PLACA.BoxNumeroSerie.Text + ".xls"); // Comando para abrir el ultimo reporte creado
        }

    }
        }
```

PROGRAMA PARA PANTALLA PRUEBA EN VACIO

```
//PROGRAMA PARA PANTALLA PRUEBA EN VACIO
//Declaracion de Librerias
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.Odbc;

namespace SEAL
{
    public partial class VACIO : UserControl
    {
        private cIniArray mINI = new cIniArray();
        OdbcConnection conexionBD;

        public VACIO()
        {
            InitializeComponent();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
        }
        private void PROTOCOLO_Load(object sender, EventArgs e)
        {
            Ia_vacio.Visible = true;
            Ib_vacio.Visible = true;
            Ic_vacio.Visible = true;
            Vab_vacio.Visible = true;
            Vbc_vacio.Visible = true;
            Vca_vacio.Visible = true;
            Frec_vacio.Visible = true;
            Pt_vacio.Visible = true;

            Ia_vacio_Reporte.Visible = false;
            Ib_vacio_Reporte.Visible = false;
            Ic_vacio_Reporte.Visible = false;
            Vab_vacio_Reporte.Visible = false;
            Vbc_vacio_Reporte.Visible = false;
            Vca_vacio_Reporte.Visible = false;
            Frec_vacio_Reporte.Visible = false;
            Pt_vacio_Reporte.Visible = false;

            try // Funcion para conectarse a MySQL
            {
                string sFicINI;

                sFicINI = System.Windows.Forms.Application.StartupPath +
@"\FileINI.ini"; // Carga el archivo FileINI.ini donde se encuentra el password y el
usuario para conectarse al MySQL

                string Servidor = mINI.IniGet(sFicINI, "MySQL", "servidor", "");
                string Usuario = mINI.IniGet(sFicINI, "MySQL", "Usuario", "");
                string Password = mINI.IniGet(sFicINI, "MySQL", "Password", "");
                string BaseDatos = mINI.IniGet(sFicINI, "MySQL", "BaseDatos", "");
```

```
                //Comando para conectarse a MySQL mediante el conector ODBC
                conexionBD = new OdbcConnection("driver={Mysql odbc 5.1
driver};server=" + Servidor + ";user=" + Usuario + ";password=" + Password +
";database=" + BaseDatos);
                conexionBD.Open();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "MAYCENS - Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
//      Funcion del Boton Capturar Valores
        private void Boton_Valores_Vacio_Click(object sender, EventArgs e)
        {
            Ia_vacio_Reporte.Visible = true;
            Ia_vacio.Visible = false;
            Ia_vacio_Reporte.Text = Ia_vacio.lblIndicador.Text;
//            Convert.ToDouble(Ia_vacio_Reporte.Text);

            Ib_vacio_Reporte.Visible = true;
            Ib_vacio.Visible = false;
            Ib_vacio_Reporte.Text = Ib_vacio.lblIndicador.Text;
//            Convert.ToDouble(Ib_vacio_Reporte.Text);

            Ic_vacio_Reporte.Visible = true;
            Ic_vacio.Visible = false;
            Ic_vacio_Reporte.Text = Ic_vacio.lblIndicador.Text;
//            Convert.ToDouble(Ic_vacio_Reporte.Text);

            Vab_vacio_Reporte.Visible = true;
            Vab_vacio.Visible = false;
            Vab_vacio_Reporte.Text = Vab_vacio.lblIndicador.Text;
//            Convert.ToDouble(Vab_vacio_Reporte.Text);

            Vbc_vacio_Reporte.Visible = true;
            Vbc_vacio.Visible = false;
            Vbc_vacio_Reporte.Text = Vbc_vacio.lblIndicador.Text;
//            Convert.ToDouble(Vbc_vacio_Reporte.Text);

            Vca_vacio_Reporte.Visible = true;
            Vca_vacio.Visible = false;
            Vca_vacio_Reporte.Text = Vca_vacio.lblIndicador.Text;
//            Convert.ToDouble(Vca_vacio_Reporte.Text);
            Frec_vacio_Reporte.Visible = true;
            Frec_vacio.Visible = false;
            Frec_vacio_Reporte.Text = Frec_vacio.lblIndicador.Text;
//            Convert.ToDouble(Frec_vacio_Reporte.Text);
            Pt_vacio_Reporte.Visible = true;
            Pt_vacio.Visible = false;
            Pt_vacio_Reporte.Text = Pt_vacio.lblIndicador.Text;
//            Convert.ToDouble(Frec_vacio_Reporte.Text);

        }

        private void label4_Click(object sender, EventArgs e)
        {

        }
    }
}
```

PROGRAMA PARA PANTALLA PRUEBAS EN CORTOCIRCUITO

```
//PROGRAMA PARA PANTALLA PRUEBAS EN CORTOCIRCUITO

//DEclaracion de Librerias
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SEAL
{
    public partial class CORTO : UserControl
    {
        private cIniArray mINI = new cIniArray();

        public CORTO()
        {
            InitializeComponent();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {

        }

        private void PROTOCOLO_Load(object sender, EventArgs e)
        {
            Ia_corto.Visible = true;
            Ib_corto.Visible = true;
            Ic_corto.Visible = true;
            Vab_corto.Visible = true;
            Vbc_corto.Visible = true;
            Vca_corto.Visible = true;
            Frec_corto.Visible = true;
            Pt_corto.Visible = true;

            Ia_corto_Reporte.Visible = false;
            Ib_corto_Reporte.Visible = false;
            Ic_corto_Reporte.Visible = false;
            Vab_corto_Reporte.Visible = false;
            Vbc_corto_Reporte.Visible = false;
            Vca_corto_Reporte.Visible = false;
            Frec_corto_Reporte.Visible = false;
            Pt_corto_Reporte.Visible = false;
        }
//      Funcion para boton Capturar Valores
        private void Boton_Valores_Corto_Click(object sender, EventArgs e)
        {
            Ia_corto_Reporte.Visible = true;
            Ia_corto.Visible = false;
            Ia_corto_Reporte.Text = Ia_corto.lblIndicador.Text;
            //            Convert.ToDouble(Ia_vacio_Reporte.Text);

            Ib_corto_Reporte.Visible = true;
            Ib_corto.Visible = false;
```

```
            Ib_corto_Reporte.Text = Ib_corto.lblIndicador.Text;
            //            Convert.ToDouble(Ib_vacio_Reporte.Text);

            Ic_corto_Reporte.Visible = true;
            Ic_corto.Visible = false;
            Ic_corto_Reporte.Text = Ic_corto.lblIndicador.Text;
            //            Convert.ToDouble(Ic_vacio_Reporte.Text);

            Vab_corto_Reporte.Visible = true;
            Vab_corto.Visible = false;
            Vab_corto_Reporte.Text = Vab_corto.lblIndicador.Text;
            //            Convert.ToDouble(Vab_vacio_Reporte.Text);

            Vbc_corto_Reporte.Visible = true;
            Vbc_corto.Visible = false;
            Vbc_corto_Reporte.Text = Vbc_corto.lblIndicador.Text;
            //            Convert.ToDouble(Vbc_vacio_Reporte.Text);

            Vca_corto_Reporte.Visible = true;
            Vca_corto.Visible = false;
            Vca_corto_Reporte.Text = Vca_corto.lblIndicador.Text;
            //            Convert.ToDouble(Vca_vacio_Reporte.Text);

            Frec_corto_Reporte.Visible = true;
            Frec_corto.Visible = false;
            Frec_corto_Reporte.Text = Frec_corto.lblIndicador.Text;
            //            Convert.ToDouble(Frec_vacio_Reporte.Text);

            Pt_corto_Reporte.Visible = true;
            Pt_corto.Visible = false;
            Pt_corto_Reporte.Text = Pt_corto.lblIndicador.Text;
            //
        }

    }
}
```

PROGRAMA PARA PANTALLA DATOS DE PLACA

```
//PROGRAMA PARA PANTALLA DATOS DE PLACA

//Declaracion de Librerias
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SEAL
{
    public partial class PLACA   UserControl
    {
        private cIniArray mINI = new cIniArray();

        public PLACA()
        {
            InitializeComponent();
        }
```

```
        private void timer1_Tick(object sender, EventArgs e)
        {

        }
        private void PROTOCOLO_Load(object sender, EventArgs e)
        {

        }
    }
}
```

PROGRAMA PARA PANTALLA RESISTENCIA DE ARROLLAMIENTOS

```
//PROGRAMA PARA PANTALLA RESISTENCIA DE ARROLLAMIENTOS

//Declaracion de Librerias
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SEAL
{
    public partial class RESISTENCIA : UserControl
    {
        private cIniArray mINI = new cIniArray();

        public RESISTENCIA()
        {
            InitializeComponent();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {

        }

        private void PROTOCOLO_Load(object sender, EventArgs e)
        {

        }

    }
}
```

PROGRAMA PARA PANTALLA PROTOCOLOS

```
//PROGRAMA PARA PANTALLA PROTOCOLOS

//Declaracion de Librerias
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
using System.Data.Odbc;

namespace SEAL
{
    public partial class EVENTOS : UserControl
    {
        private cIniArray mINI = new cIniArray();
        OdbcConnection conexionBD;

        public EVENTOS()
        {
            InitializeComponent();
        }

        private void BotonEjecutar_Click(object sender, EventArgs e)
        {
            LlenaDGV1();
        }

        private string Ret_Cadena(string sSel)
        {
            string Valor = "";

            try
            {
                OdbcDataAdapter adaptador = new OdbcDataAdapter(sSel, conexionBD);

                System.Data.DataTable dt = new System.Data.DataTable();
                adaptador.Fill(dt);

                Valor = dt.Rows[0][0].ToString();
            }
            catch { }

            return Valor;
        }
//      Funcion para realizar consulta a MySQL filtrando por nombre de Cliente
        private void LlenaDGV1()
        {
            try
            {
                string Cliente = Ret_Cadena("select CLIENTE from Protocolo_Pruebas
where CLIENTE='" + cboCliente.Text + "'");
                string Numero_de_Serie = Ret_Cadena("select NUMEROSERIE from
Protocolo_Pruebas where CLIENTE='" + cboCliente.Text + "'");
                string Fecha = Ret_Cadena("select FECHAREALIZACION from
Protocolo_Pruebas where CLIENTE='" + cboCliente.Text + "'");
                string Potencia = Ret_Cadena("select POTENCIA from Protocolo_Pruebas
where CLIENTE='" + cboCliente.Text + "'");

                string sql = "select CLIENTE, NUMEROSERIE , FECHAREALIZACION,
POTENCIA from Protocolo_Pruebas where FECHAREALIZACION >= '" +
dtpFecha_Inicial.Value.ToString("yyyy-MM-dd HH:mm:ss") + "' and FECHAREALIZACION <=
   + dtpFecha_Final.Value.ToString("yyyy-MM-dd HH:mm:ss") + "'";

                OdbcDataAdapter adaptador = new OdbcDataAdapter(sql, conexionBD);

                System.Data.DataTable dt = new System.Data.DataTable();
```

```
                adaptador.Fill(dt);

                DGV1.Rows.Clear();

                for (int i = 0; i < dt.Rows.Count; i++)
                {
                    DGV1.Rows.Add();
                    DGV1.Rows[i].Cells[0].Value = Cliente;
                    DGV1.Rows[i].Cells[1].Value = Numero_de_Serie;
                    DGV1.Rows[i].Cells[2].Value = Fecha.ToString();
                    DGV1.Rows[i].Cells[3].Value = Potencia.ToString();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "MAYCENS - Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
//      Conexion a la base de datos cuando carga la pantalla Protocolo
        private void EVENTOS_Load(object sender, EventArgs e)
        {
            try
            {
                string sFicINI;

                sFicINI = System.Windows.Forms.Application.StartupPath +
@"\FileINI.ini";

                string Servidor = mINI.IniGet(sFicINI, "MySQL", "servidor", "");
                string Usuario = mINI.IniGet(sFicINI, "MySQL", "Usuario", "");
                string Password = mINI.IniGet(sFicINI, "MySQL", "Password", "");
                string BaseDatos = mINI.IniGet(sFicINI, "MySQL", "BaseDatos", "");

                conexionBD = new OdbcConnection("driver={Mysql odbc 5.1
driver};server=" + Servidor + ";user=" + Usuario + ";password=" + Password +
";database=" + BaseDatos);
                conexionBD.Open();

                Llenar_cboCliente();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "MAYCENS - Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
//Funcion para hacer la consulta a MySQL sobre la lista de clientes y mostrar en la
lista de busqueda
        private void Llenar_cboCliente()
        {

            try
            {
                string sql = "select CLIENTE from Protocolo_Pruebas;";

                OdbcDataAdapter adaptador = new OdbcDataAdapter(sql, conexionBD);

                System.Data.DataTable dt = new System.Data.DataTable();
                adaptador.Fill(dt);
```

```
                cboCliente.Items.Clear();

                for (int i = 0; i < dt.Rows.Count; i++)
                {
                    cboCliente.Items.Add("");
                    cboCliente.Items[i] = Convert.ToString(dt.Rows[i][0]);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "MAYCENS - Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
// Funcion para exportar a Excel los valores de la consulta por Numero de Serie
        private void ExportarDGV1()
        {
            Excel.Application excel = new Excel.Application();
            excel.Application.Workbooks.Add(true);
            //Ingreso de Encabezados
            for (int k = 1; k <= DGV1.Columns.Count; k++)
            {
                excel.Cells[1, k] = Convert.ToString(DGV1.Columns[k - 1].Name);
            }
            excel.get_Range("A1", "Z1").Font.Bold = true;
            excel.Columns.ColumnWidth = 20;
            //Ingreso de Datos
            for (int i = 1; i <= DGV1.Rows.Count; i++)
            {
                for (int j = 1; j <= DGV1.Columns.Count; j++)
                {
                    excel.Cells[i + 1, j] = DGV1.Rows[i -
1].Cells[Convert.ToString(DGV1.Columns[j - 1].Name)].Value;
                }
            }
            excel.Visible = true;
            Excel.Worksheet worksheet = (Excel.Worksheet)excel.ActiveSheet;
            worksheet.Activate();
        }

        private void button3_Click(object sender, EventArgs e)
        {
            ExportarDGV1();
        }

        private void DGV1_CellContentClick(object sender, DataGridViewCellEventArgs
e)
        {
        }

        private void cboCliente_SelectedIndexChanged(object sender, EventArgs e)
        {
        }

        private void dtpFecha_Inicial_ValueChanged(object sender, EventArgs e)
        {
        }

    }
}
```

PROGRAMA PARA PANTALLA TENDENCIAS

```
//PROGRAMA PARA PANTALLA TENDENCIAS

//Declaracion de Librerias
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
using System.Data.Odbc;
using ZedGraph;

namespace SEAL
{
    public partial class TENDENCIA : UserControl
    {
        private cIniArray mINI = new cIniArray();
        OdbcConnection conexionBD;

        public TENDENCIA()
        {
            InitializeComponent();
        }

        private void TENDENCIA_Load(object sender, EventArgs e)
        {
            try
            {
                string sFicINI;

                sFicINI = System.Windows.Forms.Application.StartupPath +
@"\FileINI.ini";

                string Servidor = mINI.IniGet(sFicINI, "MySQL", "servidor", "");
                string Usuario = mINI.IniGet(sFicINI, "MySQL", "Usuario", "");
                string Password = mINI.IniGet(sFicINI, "MySQL", "Password", "");
                string BaseDatos = mINI.IniGet(sFicINI, "MySQL", "BaseDatos", "");

                conexionBD = new OdbcConnection("driver={Mysql odbc 5.1
driver};server=" + Servidor + ";user=" + Usuario + ";password=" + Password +
";database=" + BaseDatos);
                conexionBD.Open();

                Llenar_cboTagName();

                ConfiguraCurvas();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "MAYCENS - Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void Llenar_cboTagName()
        {
```

```
            try
            {
                string sql = "select TagName from RunTime order by TagId";

                OdbcDataAdapter adaptador = new OdbcDataAdapter(sql, conexionBD);

                System.Data.DataTable dt = new System.Data.DataTable();
                adaptador.Fill(dt);

                cboTagName.Items.Clear();

                for (int i = 0; i < dt.Rows.Count; i++)
                {
                    cboTagName.Items.Add("");
                    cboTagName.Items[i] = Convert.ToString(dt.Rows[i][0]);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "MAYCENS - Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void ConfiguraCurvas()
        {
            //Agrega letras
            zedGraphControl1.GraphPane.Title.Text = "GRÁFICA DE TENDENCIA";
            zedGraphControl1.GraphPane.XAxis.Title.Text = "";
            zedGraphControl1.GraphPane.YAxis.Title.Text = "";

            // Agrega lineas al ploteo
            zedGraphControl1.GraphPane.XAxis.MajorGrid.IsVisible = true;
            zedGraphControl1.GraphPane.YAxis.MajorGrid.IsVisible = true;
            zedGraphControl1.GraphPane.XAxis.MajorGrid.Color = Color.Yellow;
            zedGraphControl1.GraphPane.YAxis.MajorGrid.Color = Color.Yellow;

            // Llena los ejes y el panel con colores
            zedGraphControl1.GraphPane.Chart.Fill = new Fill(Color.Black,
Color.Black, 45.0F);
            zedGraphControl1.GraphPane.Fill = new Fill(Color.LightGray, Color.White,
45.0F);

            //Configura las listas
            RollingPointPairList list1 = new RollingPointPairList(2000);

            //Configura las curvas
            LineItem curve1 = zedGraphControl1.GraphPane.AddCurve("", list1,
Color.Yellow, SymbolType.None);

            // Configura el eje X para el tipo de dato
            zedGraphControl1.GraphPane.XAxis.Type = AxisType.Date;

            // Escala los Ejes
            zedGraphControl1.AxisChange();
        }

        private string Ret_Cadena(string sSel)
        {
            string Valor = "";
```

```
            try
            {
                OdbcDataAdapter adaptador = new OdbcDataAdapter(sSel, conexionBD);

                System.Data.DataTable dt = new System.Data.DataTable();
                adaptador.Fill(dt);

                Valor = dt.Rows[0][0].ToString();
            }
            catch { }

            return Valor;
        }

        private void Graficar()
        {
            try
            {
                DateTime FechaHora;
                double TagDateTime = 0;
                double TagReadValue = 0;

                string Tabla = "ID" + Ret_Cadena("select TagId from RunTime where
TagName='" + cboTagName.Text + "'");
                string Descripcion = Ret_Cadena("select TagDescription from RunTime
where TagName='" + cboTagName.Text + "'");
                string Unidad = Ret_Cadena("select TagUnit from RunTime where
TagName='" + cboTagName.Text + "'");

                string sql = "select TagDateTime, TagReadValue from " + Tabla + "
where TagDateTime >= '" + dtpFecha_Inicial.Value.ToString("yyyy-MM-dd HH:mm:ss") +
"' and TagDateTime <= '" + dtpFecha_Final.Value.ToString("yyyy-MM-dd HH:mm:ss") +
"'";

                OdbcDataAdapter adaptador = new OdbcDataAdapter(sql, conexionBD);

                System.Data.DataTable dt = new System.Data.DataTable();
                adaptador.Fill(dt);

                //Agrega letras
                zedGraphControl1.GraphPane.Title.Text = cboTagName.Text + " : " +
Descripcion;
                zedGraphControl1.GraphPane.XAxis.Title.Text = "Tiempo - HH:mm:ss";
                zedGraphControl1.GraphPane.YAxis.Title.Text = "Unidad - " + Unidad;

                // Make sure that the curvelist has at least one curve
                if (zedGraphControl1.GraphPane.CurveList.Count <= 0)
                    return;

                // Get the first CurveItem in the graph
                LineItem curve1 = zedGraphControl1.GraphPane.CurveList[0] as
LineItem;

                if (curve1 == null)
                    return;

                // Get the PointPairList
                IPointListEdit list1 = curve1.Points as IPointListEdit;

                //Limpia las listas
```

```
                list1.Clear();

                // If this is null, it means the reference at curve.Points does not
                // support IPointListEdit, so we won't be able to modify it
                if (list1 == null)
                    return;

                // Agrega a la lista los valores de X e Y FALTA HACER LAS CONSULTAS
Y AGREAGAR LOS VALORES DE Y
                for (int i = 0; i < dt.Rows.Count; i++)
                {
                    FechaHora = Convert.ToDateTime(dt.Rows[i][0]);
                    TagDateTime = (double)new XDate(FechaHora.Year, FechaHora.Month,
FechaHora.Day, FechaHora.Hour, FechaHora.Minute, FechaHora.Second);
                    TagReadValue = Convert.ToDouble(dt.Rows[i][1]);

                    list1.Add(TagDateTime, TagReadValue);
                }

                // Asegura que que el eje Y es reescalado
                zedGraphControl1.GraphPane.AxisChange();
                // Fuerza el grafico
                zedGraphControl1.Invalidate();
                //Refresca
                zedGraphControl1.Refresh();
            }
            catch { }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Graficar();
        }
    }
}
```

PROGRAMA PARA PANTALLA DE CONFIGURACION

```
//PROGRAMA PARA PANTALLA DE CONFIGURACION

//Declaracion de Librerias
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SEAL
{
    public partial class frmMySQL : Form
    {
        private cIniArray mINI = new cIniArray();


        public frmMySQL()
        {
            InitializeComponent();
        }
```

```
// Carga la informacion del archivo FileINI.ini para contrastar si el usuario ha
ingresado correctamente el nombre de la base de datos y el password
        private void frmMySQL_Load(object sender, EventArgs e)
        {
            try
            {
                string sFicINI;

                sFicINI = System.Windows.Forms.Application.StartupPath +
@"\FileINI.ini";

                txtServidor.Text = mINI.IniGet(sFicINI, "MySQL", "servidor", "");
                txtUsuario.Text = mINI.IniGet(sFicINI, "MySQL", "Usuario", "");
                txtPassword.Text = mINI.IniGet(sFicINI, "MySQL", "Password", "");
                txtBaseDatos.Text = mINI.IniGet(sFicINI, "MySQL", "BaseDatos", "");
            }
            catch { }
        }

        private void btnAceptar_Click(object sender, EventArgs e)
        {
            try
            {
                string sFicINI;

                sFicINI = System.Windows.Forms.Application.StartupPath +
@"\FileINI.ini";

                mINI.IniWrite(sFicINI, "MySQL", "Servidor",
txtServidor.Text.Trim());
                mINI.IniWrite(sFicINI, "MySQL", "Usuario", txtUsuario.Text.Trim());
                mINI.IniWrite(sFicINI, "MySQL", "Password",
txtPassword.Text.Trim());
                mINI.IniWrite(sFicINI, "MySQL", "BaseDatos",
txtBaseDatos.Text.Trim());

                System.Windows.Forms.MessageBox.Show("Datos Guardados
Correctamente", "SEAL");
            }
            catch { }
        }

        private void btnCancelar1_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

ANEXO B
GLOSARIO DE TÉRMINOS

API	Interfaz de Programación de la Aplicación
BCL	Base Classes Library (Biblioteca de clases base o BCL)
CIL	Lenguaje Común Intermedio
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CSV	Archivo de texto separado por comas
DLL	Bibliotecas de enlazado dinámico.
DCE	Equipo de Comunicación de datos),
DTE	Equipo terminal de datos
ECMA	European Computer Manufacturers Association.
GPL	General Public License
JIT	just-in-time.
OOP	Programación Orientada a Objetos
OPC	OLE for Process Control.
OPC A&E	OPC Alarm & Events Specification
OPC DA	OPC Data Access Specification
OPC HDA	OPC Historical Data Access Specification
RTU	Remote Transmission Unit
SQL	Structured Query Language
UART	Transmisor y Receptor Asíncrono Universal