Universidad de las Ciencias Informáticas

Título: Módulo de adquisición de datos para dispositivos Elecsys2010 en el proyecto SCADA-Hospitales.

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

Autor: Humberto Llauradó Falcó

Tutor: Ing. Luís Enrique García Hernández

Cotutor: Ing. Yaima Contino Matos

Consultante: Ing. Adrián Carlos Moreno

Ciudad de La Habana, 2010.

Año del 51 aniversario de la Revolución

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____	_____
Firma del Autor	Firma del Tutor
(Humberto Llauradó Falcó)	(Ing. Luis Enrique García Hernández)

DATOS DE CONTACTO

Ing. Luis Enrique García Hernández

Ingeniero en Ciencias Informáticas en 2009, Investigador del Centro de Desarrollo de Informática Industrial. Posee cuatro años de experiencia en el desarrollo de sistemas computacionales para la supervisión y control de procesos industriales.

Ing. Yaima Contino Matos

Ingeniero en Automática en 2005 y Profesora Asistente. Posee cuatro años de experiencia en el desarrollo de sistemas computacionales para la supervisión y control de procesos industriales.

Ing. Adrian Carlos Moreno Borges

Ingeniero en Ciencias Informáticas en 2008, Investigador del Centro de Desarrollo de Informática Industrial. Posee dos años de experiencia en el desarrollo de sistemas computacionales para la supervisión y control de procesos industriales.

AGRADECIMIENTOS

A mis padres, quienes me han dado siempre mucho amor y cariño. A ellos que me han sabido educar y encaminar en la vida con su apoyo y ejemplo personal.
A mi familia que me ha apoyado de una manera u otra a superar las barreras que me ha puesto la vida, llegue a todos mis más sinceros agradecimientos.
A mi hermano que por tratar de ser un ejemplo para él, me ha hecho superarme cada día más como estudiante y como persona.
A mis amigos de la infancia por ser fieles y tolerar mi forma de ser.
A los compañeros del Centro de Desarrollo de Informática Industrial con quienes he compartido y de quienes he aprendido cada día y en especial a Pedro, Amides y a Yunaime.
A todos mis compañeros de aula con quienes he compartido en algún momento durante mi tiempo de estudiante.
A Yailín que siempre me ha ayudado incondicionalmente y de no ser por ella me hubiera sido muy difícil llegar a este día.
A todos los profesores que contribuyeron a mi formación y educación.
A mi tutor Luiso por su apoyo en la realización de esta Tesis.
A la Revolución y a Fidel por permitir que haya logrado realizar mis sueños.

DEDICATORIA

A mis padres, Humberto y Nuria.
A mis abuelos.
A mi hermano, Eduardo.
A mi familia y amigos.

RESUMEN

En toda industria automatizada es necesario obtener las magnitudes que permitan conocer el estado y evolución del proceso que estemos controlando. Para el logro de este objetivo se emplean dispositivos como pueden ser PLC, autómatas programables, sensores, etc. Estos dispositivos poseen variados protocolos de comunicación a través de los cuales se puede acceder a los datos recopilados del campo.

El presente trabajo explica el desarrollo del manejador de dispositivos que permite acceder la información almacenada en los equipos Elecsys 2010. Este manejador (en inglés, driver) está encapsulado en una biblioteca de carga dinámica que será utilizada por el sistema de Supervisión, Control y Adquisición de Datos; del Centro de Desarrollo de Informática Industrial, de la Universidad de las Ciencias Informáticas. Se abordan temas relacionados con los protocolos de comunicación, haciendo énfasis en el protocolo utilizado para la comunicación con los dispositivos Elecsys 2010. También se exponen las clases implementadas y las bibliotecas usadas para el desarrollo del manejador. Se brinda una panorámica al lector de los contenidos abordados y los principales resultados obtenidos. Se analizan las características fundamentales de las aplicaciones para la supervisión y control de los procesos industriales, y de los manejadores de dispositivos como componentes de gran valor para dichas plataformas de software.

PALABRAS CLAVE

Dispositivo, Elecsys 2010, manejador, protocolo, SCADA

Tabla de Contenidos

INTRODUCCIÓN.

En los procesos de automatización es imprescindible obtener las magnitudes de la planta, de esta forma, es posible conocer el estado y evolución del que estamos controlando. Algunos ejemplos de estas son la humedad, la temperatura, la velocidad y la presión. El control o monitoreo constante de todas estas variables más otras permite conocer en cualquier momento el estado real del proceso. Siendo esta información de gran importancia para el buen funcionamiento de la industria.

En la actualidad el mundo de la informática ha avanzado mucho y ya podemos contar con sistemas como los SCADA, acrónimo de Supervisory Control and Data Acquisition (en español, Control Supervisor y Adquisición de Datos). En estos, la señalización se realiza mediante una pantalla y un entorno gráfico que lo hace más amigable al usuario. También especifican en detalle lo que está sucediendo en cada momento con el proceso. Además se pueden visualizar alarmas y ejecutar acciones de manera automática. Con estos sistemas se controlan equipos que pueden estar a kilómetros de distancia a través distintos protocolos de comunicación.

Cuba también se encuentra inmersa en el desarrollo de estos sistemas y ejemplo de ello es el proyecto SCADA “Guardián del ALBA”. Este proyecto surge gracias a las relaciones entre Venezuela y Cuba, con el objetivo de tener desarrollo de tecnologías propias para evitar sabotajes a nuestras empresas. En la actualidad está en desarrollo también un sistema SCADA totalmente cubano, que posibilitará la soberanía tecnológica, pues hace uso del software libre. La construcción de este sistema SCADA se realiza en el Centro de Desarrollo de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas (UCI).

El intercambio de datos entre el SCADA y los dispositivos de campo o PLC se realiza a través de los manejadores. Los manejadores permiten la comunicación eficiente entre el SCADA y dispositivos. En el sistema SCADA se ha implementado un framework orientado a objetos para el desarrollo de manejadores de dispositivos, que es denominado DriversCore. Dicha plataforma proporciona una interfaz estándar de acceso a estos instrumentos y oculta al SCADA las diferencias entre los protocolos (Cedeño Pozo y otro, 2009), así como las características de hardware de los dispositivos enlazados con él. La línea de manejadores es la encargada de proveer al sistema SCADA los drivers necesarios para instalar el SCADA en las industrias cubanas. Se trabaja en la implantación de los sistemas SCADA en distintas empresas e instituciones cubanas. Por ejemplo en el Hospital CIMEQ, en el instituto de

meteorología y en ETECSA.

El CIMEQ cuenta con varios tipos de analizadores que realizan distintos tipos de pruebas, entre ellos está el analizador Elecsys 2010 totalmente automático que realiza análisis inmunológico, por ejemplo para la determinación de hormonas tiroideas y de la fecundidad, marcadores tumorales y cardiacos o parámetros para el diagnostico de enfermedades infecciosas y de osteoporosis (Hoffmann, 2007). En la actualidad los resultados que brinda este analizador sólo se pueden obtener a través de una impresora, por lo que para poderlos almacenar es necesario archivar los documentos impresos. Esto provoca que el hospital deba invertir grandes sumas de dinero en la compra de papel y tinta, además existe el inconveniente de que es difícil realizar estudios estadísticos, pues no se tiene la información en formato digital. Por estas razones el CIMEQ necesita automatizar los procesos para la gestión de la información que provee el Elecsys 2010, y para lograr este objetivo se utilizará el SCADA desarrollado en el CEDIN de la UCI. Pero dicho sistema no cuenta con un medio que permita la comunicación con este tipo de dispositivo. Por lo que dada esta situación, surge entonces como **Problema Científico:**

¿Cómo lograr la comunicación entre el SCADA desarrollado en la Universidad de las Ciencias Informáticas y los dispositivos Elecsys 2010?

Tal interrogante implica que el **objeto de estudio** de la presente investigación son los manejadores para sistemas SCADA. La misma tiene como **objetivo general:** Implementar un manejador para el SCADA desarrollado en la UCI que permita la comunicación con dispositivos Elecsys 2010.

El **campo de acción** lo constituyen los mecanismos y funcionalidades básicas para la comunicación con dispositivos autoanalizadores de laboratorios clínicos Elecsys 2010.

Para dar cumplimiento al objetivo planteado, se desarrollarán las siguientes tareas de investigación:

- Estudio de la bibliografía referente al desarrollo de manejadores de dispositivos.
- Estudio de la Interfaz Genérica creada en el proyecto SCADA para el desarrollo de manejadores.
- Análisis y selección de estructuras compatibles con la mayoría de los compiladores de C++ para su posterior utilización.
- Utilización de estructuras compatibles con la mayoría de los compiladores de C++ para lograr mayor portabilidad.
- Implementación multiplataforma del manejador.

- Implementación de la capa protocolo del manejador.
- Implementación de la capa drivers del manejador.
- Realización de pruebas al manejador.

Para el cumplimiento de estos objetivos se llevan a cabo varios métodos y técnicas en la búsqueda y procesamiento de la información como son:

1. **Métodos de Analítico-sintético:** Mediante el cual podré conocer los elementos más importantes sobre el desarrollo de manejadores.
2. **Análisis histórico-lógico:** Me permitirá conocer los fundamentos y teorías relacionados con el trabajo de investigación.
3. **Modelación**, se realizarán diagramas y modelos que permitan estructurar teóricamente el sistema.

DESARROLLO

1. FUNDAMENTACIÓN TEÓRICA.

1.1 Sistemas SCADA para laboratorios clínicos.

En las tareas automatizadas de control y visualización de datos industriales se pueden utilizar distintos sistemas. Entre los mecanismos de control de procesos industriales utilizados por el hombre en la actualidad están los SCADAs, acrónimo de Supervisory Control and Data Acquisition (en español, Control Supervisor y Adquisición de Datos). Estos sistemas juegan un papel fundamental en la industria, pues comprenden las soluciones de aplicación que necesitan de la captura de información de un proceso o planta industrial, la cual es empleada para realizar análisis con los que se pueden obtener importantes indicadores que permiten una realimentación del proceso. De forma general se consideran como funciones básicas de un sistema SCADA las enunciadas a continuación (Hernández Lugones, 1998):

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

Por sistemas SCADA se ha entendido:

“Sistemas denominados como tal no son más que aplicaciones de “software” para el control de la producción que se comunican con los dispositivos de campo y controlan los procesos desde la pantalla del ordenador; además de ello son capaces de proporcionar información de la planta a diversos usuarios

como: operadores, supervisores de control de calidad y administradores del sistema." *(Jiménez López, et al., 2007).*

"Sistema basado en computadores que permite supervisar y controlar a distancia una instalación de cualquier tipo." *(Montero, y otros, 2004).*

De manera general los sistemas SCADAs son un conjunto de software orientados a monitorear y controlar remotamente los diferentes procesos industriales que están ocurriendo diferentes tipos de dispositivos de campos que pueden encontrarse a kilómetros de distancia. Este recolecta los datos provenientes de los sensores o dispositivos de campos como los distintos tipos de **PLC** (*Programmable Logic Controller*) y procesa la información capturada ya sea con el propósito de observación o control, y los presentan a través de un ordenador de forma amigable a un operador para poder tomar decisiones operacionales apropiadas.

Los sistemas SCADAs son utilizados para la automatización de los laboratorios clínicos de los hospitales. En este ambiente también se usan dispositivos autoanalizadores, que le envían los datos a los sistemas SCADA. La empresa líder a nivel mundial en la fabricación de autoanalizadores es Roche. Esta compañía está dividida en dos líneas de producción: la farmacéutica y la de diagnóstico; dispone por tanto de una amplia gama de productos dirigidos a los laboratorios clínicos entre los que se encuentran (Roche, 2000):

- HITACHI/ROCHE 902
- HITACHI/ROCHE 912
- HITACHI ROCHE 917
- cobas b121
- cobas b121 BGE
- Elecsys 2010

A nivel internacional existen empresas que desarrollan sistemas SCADA, que automatizan la adquisición de los resultados procesados por autoanalizadores Roche. Pero estos sistemas no pueden ser utilizados en Hospitales Cubanos como consecuencia de las restricciones del Bloqueo de EE.UU. Además estos sistemas SCADA son muy costosos y están desarrollados sólo para plataformas Windows, con licencias privativas.

El proceso de intercambio de información entre el nivel de supervisión y el nivel de campo se establece comúnmente a través de los protocolos de comunicación, los cuales establecen el “idioma”, y las reglas para entablar una comunicación entre el autoanalizador y el sistema SCADA.

1.2 Protocolos.

De manera general, se puede afirmar que un protocolo es: el conjunto de reglas que gobierna el intercambio de datos entre dos entidades, siendo las entidades cualquier cosa capaz de enviar y recibir información.

Los puntos clave que definen un protocolo son: (Stallings, 2003).

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

Las funciones de un protocolo se pueden agrupar en:

- **Encapsulamiento:** Añadir datos de información de control. Los datos se aceptan o generan por una entidad y se encapsulan en la unidad de datos de protocolo (PDU) junto con la información de control.
- **Segmentación y ensamblado:** Cada Protocolo puede dividir los datos de un nivel superior en bloques más pequeños. Denominaremos unidad de datos del protocolo (PDU, *Protocol Data Unit*) al bloque de datos a intercambiar entre dos entidades. El procedimiento contrario a la segmentación se denomina ensamblado. Los datos tendrán que ensamblarse recuperando el formato de los mensajes originales para ser entregados a la entidad de destino.
- **Control de la conexión:** La transferencia de datos puede ser no orientada a la conexión, cada PDU se tratará de forma independiente de los PDUs recibidos con anterioridad. La orientada a la conexión se establece una asociación lógica, o conexión, entre dos entidades.
- **Entrega en orden**:
  - Control del Flujo: El control de flujo es una operación realizada por la entidad receptora para limitar la velocidad y la cantidad de datos a enviar.
  - Control de Errores: Recuperar pérdidas o deterioro de los datos.
  - Direccionamiento: La dirección del nivel de red se utiliza para encaminar los PDUs a través de la red hasta el sistema destino.

- La multiplexación entre capas: Puede realizarse de dos formas distintas. La multiplexación ascendente o la descendente.
  - ✓ La multiplexación ascendente consiste en que varias conexiones del nivel superior compartan, o se multiplexen sobre una única conexión del nivel inferior.
  - ✓ La multiplexación descendente consiste en establecer una única conexión del nivel superior utilizando varias conexiones del nivel inferior.

- **Servicios de transmisión**:
  - Prioridad.
  - Calidad del Servicio.
  - Seguridad.

Estos protocolos son especificados en capas, dentro de las cuales la más baja se denomina capa física en la mayoría de los estándares, dentro de los que se destaca el modelo OSI. La capa física describe el medio sobre el cual se transmite la información, dentro de estos medios se pueden mencionar, par trenzado y fibra óptica. Sobre estos medio se establecen estándares de comunicación con normas de hardware que permiten una integración en redes de dispositivos que cumplan con dichas normas, dentro de los cuales podemos destacar RS-232, RS-485, Ethernet, etc. Los dispositivos de campo se conectan a estas redes y establecen la comunicación con sus homólogos a través de los protocolos de comunicación.

Entre los protocolos de comunicación más usados para la automatización industrial están Modbus (RTU, ASCII, TCP), Ethernet IP y BSAP. Sin embargo estos protocolos no son muy utilizados en la automatización de procesos relacionados con los laboratorios clínicos. En este campo el estándar más utilizado es el ASTM.

1.3 Protocolo de comunicación ASTM.

En una búsqueda estándar para la comunicación con los equipos de laboratorios clínicos se estableció el ASTM como un estándar. A diferencia de otros protocolos este estandariza la información a transmitir por lo que cada día es mayor el número de equipos que utilizan este estándar.

Entre estos equipos que han hecho suyo este estándar está el Elecsys 2010 con sus especificaciones propias para el mismo (Roche, 2000). En una búsqueda realizada se encontró referencia de trabajos que

se han estado realizando en el mundo con respecto a manejadores para este tipo de equipos, pero en esos casos se han desarrollado aplicaciones utilizando software privativo. Por lo que con este trabajo se pretende implementar un manejador de dispositivo, que permita minimizar los costos en licencia y poder lograr la comunicación con el Autoanalizador Elecsys 2010 utilizando software libre. Estos dispositivos utilizan la comunicación serial para enviar los datos hacia el sistema SCADA.

1.4 Comunicación serial.

La comunicación serial es un protocolo muy común para conexión entre dispositivos que se incluye de manera estándar en prácticamente cualquier computadora. La comunicación serial puede ser utilizada para adquisición de datos si se usa en conjunto con un dispositivo remoto de muestreo.

El puerto serial envía y recibe bytes de información un bit a la vez. Aún y cuando esto es más lento que la comunicación en paralelo, que permite la transmisión de un byte completo por vez, este método de comunicación es más sencillo y puede alcanzar mayores distancias. Por ejemplo, la especificación IEEE 488 para la comunicación en paralelo determina que el largo del cable para el equipo no puede ser mayor a 20 metros, con no más de 2 metros entre cualesquier dos dispositivos; por el otro lado, utilizando comunicación serial el largo del cable puede llegar a los 1200 metros (IEEE, 2000).

Típicamente, la comunicación serial se utiliza para transmitir datos en formato ASCII. Para realizar la comunicación se utilizan 3 líneas de transmisión: (1) Tierra (o referencia), (2) Transmitir, (3) Recibir. Debido a que la transmisión es asincrónica, es posible enviar datos por una línea mientras se reciben datos por otra.

Las características más importantes de la comunicación serial son la velocidad de transmisión, los bits de datos, los bits de parada, y la paridad. Para que dos puertos se puedan comunicar, es necesario que las características sean iguales (Instrument, 2006).

Velocidad de transmisión (baud rate): Indica el número de bits por segundo que se transfieren, y se mide en baudios (bauds).

Bits de datos: Se refiere a la cantidad de bits en la transmisión. Cuando la computadora envía un paquete de información, el tamaño de ese paquete no necesariamente será de 8 bits. Las cantidades más comunes de bits por paquete son 5, 7 y 8 bits.

Bits de parada: Usado para indicar el fin de la comunicación de un solo paquete. Los valores típicos son 1, 1.5 o 2 bits. Debido a la manera como se transfiere la información a través de las líneas de comunicación y que cada dispositivo tiene su propio reloj, es posible que los dos dispositivos no estén sincronizados. Por lo tanto, los bits de parada no sólo indican el fin de la transmisión sino además dan un margen de tolerancia para esa diferencia de los relojes. Mientras más bits de parada se usen, mayor será la tolerancia a la sincronía de los relojes, sin embargo la transmisión será más lenta.

Paridad: Es una forma sencilla de verificar si hay errores en la transmisión serial. Existen cuatro tipos de paridad: par, impar, marcada y espaciada. La opción de no usar paridad alguna también está disponible.

Los estándares de comunicación serie que por su gran versatilidad, robustez mecánica e inmunidad al ruido, han conseguido una enorme difusión y popularidad en la industria. Definen las interfaces físicas, funcionales y lógicas para comunicaciones de datos digitales punto a punto o punto-multipunto. No especifica un protocolo en particular dado que esto estará facilitado por las capas superiores. Por ejemplo el estándar RS-232 es:

RS-232 (Estándar ANSI/EIA-232) es el conector serial hallado en las PCs IBM y compatibles. Es utilizado para una gran variedad de propósitos, como conectar un ratón, impresora o modem, así como instrumentación industrial (Santos, 2008). Gracias a las mejoras que se han ido desarrollando en las líneas de transmisión y en los cables, existen aplicaciones en las que se aumenta el desempeño de RS-232 en lo que respecta a la distancia y velocidad del estándar. RS-232 está limitado a comunicaciones de punto a punto entre los dispositivos y el puerto serial de la computadora. El hardware de RS-232 se puede utilizar para comunicaciones seriales en distancias de hasta 50 pies (Santos, 2008).

1.5 Manejadores de dispositivos.

En el mundo actual existen infinidad de dispositivos hardware que podemos acoplar tanto a nuestro ordenador como a los mencionados sistemas SCADA. Estos dispositivos a su vez deben de ser reconocidos y controlados por el sistema al cual esté conectado. Por lo que resulta fácil intuir que el manejador de dispositivo será específico para un dispositivo en concreto, así como de un modelo y marca, el además del sistema operativo, actuando mismo como un puente entre ambos. Dada la ingente cantidad de combinaciones resultantes, es materialmente imposible que un SCADA pueda llevar todos los drivers existentes aportando los más estándares o de uso más habitual.

“Un driver (manejador de dispositivo) técnicamente, no es más que un software o componente de hardware que nos sirve de intermediario entre un dispositivo de hardware y el sistema operativo. Su finalidad y diseño es la de permitir extraer el máximo de las funcionalidades del dispositivo para el cual ha sido diseñado.” (Informática.Net, 2004).

2. CARACTERÍSTICAS DEL SISTEMA.

2.1 Tecnologías utilizadas.

Basado en un estudio de las tecnologías más actuales y teniendo en cuenta que estas no limiten la capacidad multiplataforma del manejador se ha seleccionado las siguientes herramientas para el desarrollo.

2.1.1 Asio C++ Library

Se selecciono la biblioteca “Asio C++ Library” para el envió y recepción de mensajes a través del medio físico. Entre las razones para escogerla esta que la biblioteca Asio es parte bibliotecas Boost, que tienen amplio soporte en la comunidad del Software Libre. Así como también que Asio es una biblioteca multiplataforma utilizada para la implementación de aplicaciones de red, brindando este un desarrollo consistente de un mecanismo asíncrono de flujos de entrada y salida, usando un moderno enfoque de C++. Entre la amplia gama de funcionalidades que brinda podemos encontrar, soporte para la resolución de direcciones, aceptación de nuevas conexiones, socket orientados a datagrama y funcionalidades de temporizador. La biblioteca en si trata de abordar los siguientes objetivos:(Kohlhoff, 2008).

Portabilidad: La biblioteca soporta los sistemas operativos más comunes, y proporciona un comportamiento consistente a través de estos sistemas operativos, dentro de los que se encuentran:

Win32 y Win64 usando Visual C++ 7.1 y Visual C++ 8.0.

Win32 usando Borland C++Builder 6 patch 4.

Win32 usando MinGW.

Win32 usando Cygwin. (__USE_W32_SOCKETS)

Linux (2.4 or 2.6 kernels) usando g++ 3.3 o superior.

Solaris usando g++ 3.3 o superior.

Mac OS X 10.4 usando g++ 3.3 o superior.

Escalabilidad: La biblioteca debe permitir, alentar y, de hecho, el desarrollo de aplicaciones de red que escala a cientos o miles de conexiones simultáneas. La biblioteca de ejecución para cada sistema operativo debe utilizar el mecanismo que mejor escalabilidad le permita a esta.

Eficiencia:

Modelo de Berkeley sockets: Es ampliamente entendido y aplicado en la biblioteca.

Facilidad de uso:

Bases para una mayor abstracción: La biblioteca debe permitir el desarrollo de otras bibliotecas que ofrecen mayores niveles de abstracción. Por ejemplo, las implementaciones de los protocolos de uso común, tales como HTTP.

2.1.2 Herramientas de desarrollo.

Para el desarrollo de manejadores es necesario seleccionar el lenguaje de programación y el (IDE) que se utilizará. Un IDE es un Entorno de Desarrollo Integrado que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario (GUI). Estos deben tener un nivel de desarrollo y estabilidad alcanzada, documentación existente, flexibilidad y personalización, y como elemento esencial que sean aplicaciones desarrolladas bajo los términos de las licencias que rigen el mundo del software libre. En este caso ya estaban definidas las herramientas ya que son las que estaban definidas en el proyecto al cual está integrado este trabajo. Por lo que a continuación se hará una breve descripción de las mismas.

2.1.2.1 Eclipse.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios (eclipse, 2009).
Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent Azureus. (Kubuntu, 2008).

2.1.3 Lenguaje de programación.

2.1.3.1 C++.

C++ es un lenguaje de programación creado por Bjarne Stroustrup en los laboratorios de At&T en 1983. Stroustrup tomó como base el lenguaje de programación más popular en aquella época el cual era C.

Bjarne Stroustrup, creo lo que se conoce como C++ (Peñalvo, 2002). Necesitaba ciertas facilidades de programación, incluidas en otros lenguajes pero que C no soportaba, al menos directamente, como son las llamadas clases y objetos, conceptos muy en boga en la programación actual. Para ello rediseñó el C, ampliando sus posibilidades pero manteniendo su mayor cualidad, la de permitir al programador en todo momento tener controlado lo que está haciendo, consiguiendo así una mayor rapidez que no se conseguiría en otros lenguajes. El mismo es un lenguaje orientado a objetos derivado del C. Surgió con el objetivo de añadirle cualidades y características que carecía su ancestro. Este lenguaje es muy ligado al hardware, manteniendo una considerable potencia para programación a bajo nivel, le añade elementos que le permiten también un estilo de programación con alto nivel de abstracción.
Una de las razones de programar en C++ es su increíble versatilidad. Con el pueden programarse desde los programas más simples, a los programas más complicados como incluso sistemas operativos. Además es portable, es decir, un programa con el código escrito en C++, se podrá compilar en cualquier sistema operativo o sistema informático si necesidad de cambiar casi el código fuente. Este es por ejemplo uno de los grandes secretos de Linux, al estar el código escrito en este lenguaje(al menos en su concepción original), es más fácil portarlo a diferentes ordenadores como PC´s, Macintosh. Además otras de las grandes ventajas del C++, es que es un lenguaje multi-nivel, es decir, puedes usarlo tanto para programar directamente el hardware(dependiendo del sistema operativo, eso sí), como para crear aplicaciones tipo Windows definidas todas por poseer una misma interfaz. Permite añadir soporte de programación orientada a objeto (incluida la herencia múltiple), además de que se puede potentes bibliotecas por parte de los desarrolladores.

2.1.4 Herramienta CASE.

2.1.4.1 Visual Paradigm para UML.

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue (Pasaje, 2005). El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.
Características principales:

- Soporte de UML versión 2.1.
- Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio, Documento.
- Modelado colaborativo con CVS y Subversion (control de versiones).
- Ingeniería inversa - Código a modelo, código a diagrama.

- Generación de código - Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
- Editor de figuras.

Algunas herramientas y plugins de modelado UML:

- Plataforma Java (Windows/Linux/Mac OS X):
- SDE para Eclipse.
- SDE para Microsoft Visual Studio

2.2 Especificaciones del protocolo de comunicación para el Elecsys 2010. (Roche, 2000)

Como había mencionado un protocolo de comunicación no es más que un conjunto de reglas de programación que están definidas para la interpretación de señales que son transmitidas a través de una conexión física. Donde el protocolo para la comunicación con el Elecsys 2010 lleva:

- Un mensaje libre de errores desde un extremo al otro del mismo ya que el mismo lleva adjunto al mismo un chequeo de errores al final de cada trama que se envía, respondiendo el receptor un ACK si al verificar el mensaje es correcto, en caso contrario se envía NAK.
- Utiliza dos capas de software para posibilitar las comunicaciones, siendo estas la capa de enlace de datos (data link layer) y la capa de aplicación (aplication layer).
- Indica fallos con un código de error y hace reintentos ante errores.

2.2.1 ASTM.

El protocolo ASTM usa 4 capas jerárgicas para procesar los datos enviados en la comunicación.

- Capa de aplicación: Intercambia mensajes (message).

- Capa de presentación: Intercambia registros (records). Un mensaje consta de múltiples registros.
- Capa de enlace a datos: Intercambia marcos (frames). Un registro consiste en uno o más marcos.
- Capa física: El intercambio es en código ASCII de 8 bits a través de una interfaz serie estándar. Donde un marco consiste como máximo de 240 caracteres (en bytes) y 7 bytes de delimitador y comprobación de datos (checksum data).

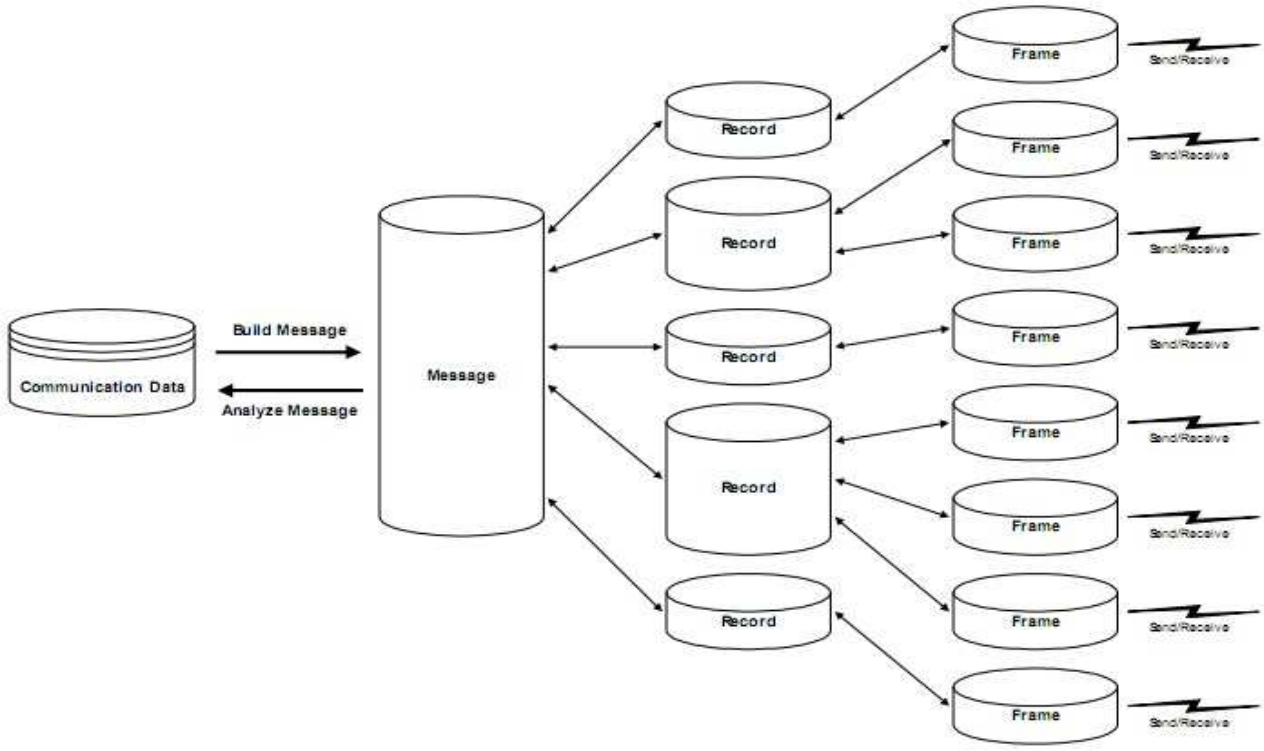

Figura 2. Correlación entre Mensaje, Registros y Marco (Roche, 2000).

2.2.1.1 Descripción de los términos específicos para la norma ASTM.

Para aclarar algunos términos de este capítulo se definen algunos términos específicos de acuerdo con las especificaciones ASTM.

- Mensaje: Cuerpo textual de la información. Por ejemplo: Los resultados de las pruebas de todos los pedidos de un grupo de pacientes y los datos correspondientes.

- Prueba: Una determinación de un análito o una combinación de valores de otras determinaciones u observaciones que constituyen una medida de un sistema único atributo. Ejemplo: Determinación de la TSH en suero.
- Registro: un conjunto de campos que describen un aspecto del mensaje completo. Ejemplo: La clase Patient Info Record en un "Mensaje de datos medidos" contiene información relacionada con el paciente cuyos resultados se reportan.
- Campo: Un atributo específico de un expediente que puede contener agregados de datos nuevos elementos de remitir el atributo básico. Ejemplo: El campo Nombre del paciente en el Registro de Información para el paciente.
- Upload: Datos transmitidos desde un instrumento clínico a un sistema informático. Ejemplo: Cuando el analizador ha terminado todas las pruebas para una muestra determinada que descarga los resultados a la acogida.
- Download: Datos transmitidos desde un sistema informático para un instrumento clínico. Ejemplo: Las órdenes de acogida en los exámenes de un grupo de pacientes mediante la subida de un pedido de prueba para el instrumento.

2.2.2 Caracteres restringidos en el mensaje.

El protocolo de enlace de datos está diseñado para enviar mensajes orientados a carácter. Los caracteres restringidos se colocan para que se pueda conocer cuándo es que comienza a llegar un mensaje, una trama o un paciente en general, así como cuando termina cada uno de ellos y así enviar la respuesta que en cada caso se espera por el que envía.

ASCII	Decimal	Hexadecimal	Descripción
<STX>	2	0x2	Separa los datos de la cabecera de comunicación.
<ETX>	3	0x3	Indica la finalización del mensaje.
<EOT>	4	0x4	Indica el fin de la transmisión.
<ENQ>	5	0x5	Indica el comienzo de la transmisión.

<ACK>	6	0x6	Respuesta que indica que el mensaje se recibió satisfactoriamente.
<LF>	10	0xA	Fin de la trama.
<NAK>	21	0x15	Respuesta que indica que el mensaje recibido contiene error.
<ETB>	23	0x17	Fin del bloque de transmisión.

Tabla 1. Caracteres de control (Roche 2000).

2.2.3 Estructura del mensaje: Registros.

El típico mensaje contiene un grupo de información. Por ejemplo, la solicitud de resultados contiene cuatro niveles (0 - 3) de información.

- Los datos relativos a los pacientes con los resultados de las pruebas se transmiten.
- Los datos relativos a las órdenes (baterías de pruebas) con los resultados del examen pertenecen.
- Los datos relativos a los resultados del examen de cada batería de pruebas para cada paciente.
- Comentario de datos con la información que pertenece a cada resultado.

ASTM define los mensajes que consiste de una jerarquía de los registros de varios tipos (ver Figura 3). Los registros en el nivel cero contienen información referente a la identificación del remitente y la finalización de la transmisión. Pueden ser vistos como una especie de definición de información en relación con el mensaje. Los registros en el nivel uno de la jerarquía contienen información sobre los pacientes individuales. Los registros en el nivel dos contienen información sobre las pruebas para peticiones y las muestras. Los registros en el nivel tres contienen información sobre resultados de las pruebas.

Los comentarios se pueden insertar en cualquier nivel en la jerarquía. Un comentario siempre se refiere a la paciente inmediatamente anterior, el orden, resultado, científicos o fabricante de registrar la información. Por lo tanto, si un registro de observaciones iban a seguir una historia clínica del paciente (nivel uno), entonces ese registro comentario sería tratada como un nivel de dos registro. Un registro comentario no puede seguir el registro de terminación de mensajes.

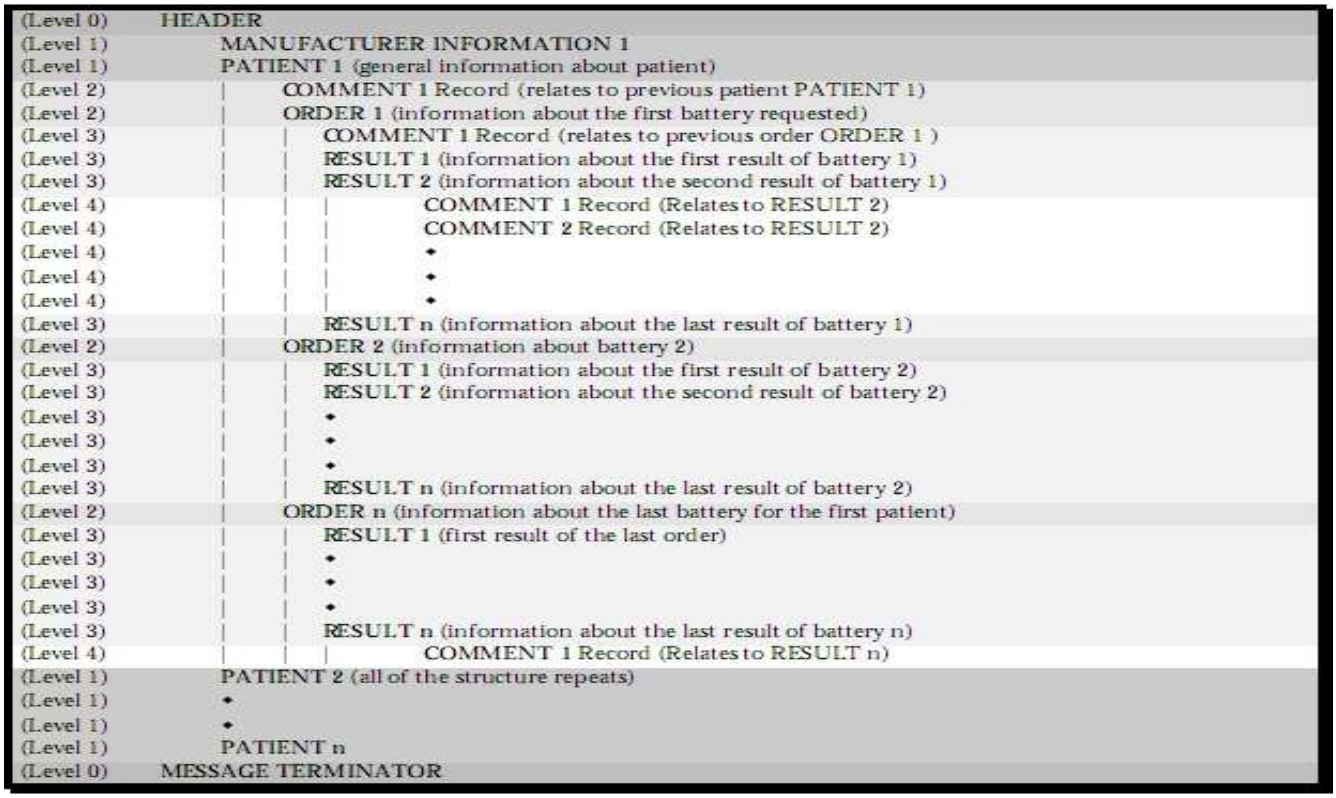

Figura 3. Jerarquía de la estructura del Mensaje. (Roche, 2000).

2.3 Capa de enlace a datos.

2.3.1 Descripción general.

La capa de enlace de datos tiene que realizar los siguientes servicios:

- Iniciación y liberación de conexión: Establece que los sistemas envían y reciben la información.
- Cortar los mensajes en pequeños marcos de texto: prevé el reconocimiento de marcos.
- Mantiene el orden secuencial de información a través de la conexión.
- La detección de errores: La transmisión de sentidos y errores de formato.
- La recuperación de errores: Los intentos por recuperarse de los errores detectados por volver a transmitir marcos defectuosos o devolver el enlace a un estado neutral de lo contrario errores son irrecuperables.

El protocolo es un simple protocolo de parada y espera. Por ejemplo información siempre fluye en una dirección a la vez. Las respuestas se producen después de la información se envía, nunca al mismo tiempo. En contradicción con otros protocolos de comunicación no hay relación maestro esclavo. Ambos instrumentos podrán iniciar la comunicación. Para establecer quien envía y quien recibe información y asegurar las acciones del emisor y el receptor estén bien coordinadas, hay tres fases distintas en la transferencia de información y ellas son:

- Fase de establecimiento. (Establishment Phase)
- Fase de transferencia. (Transfer Phase)
- Fase de terminación. (Termination Phase)

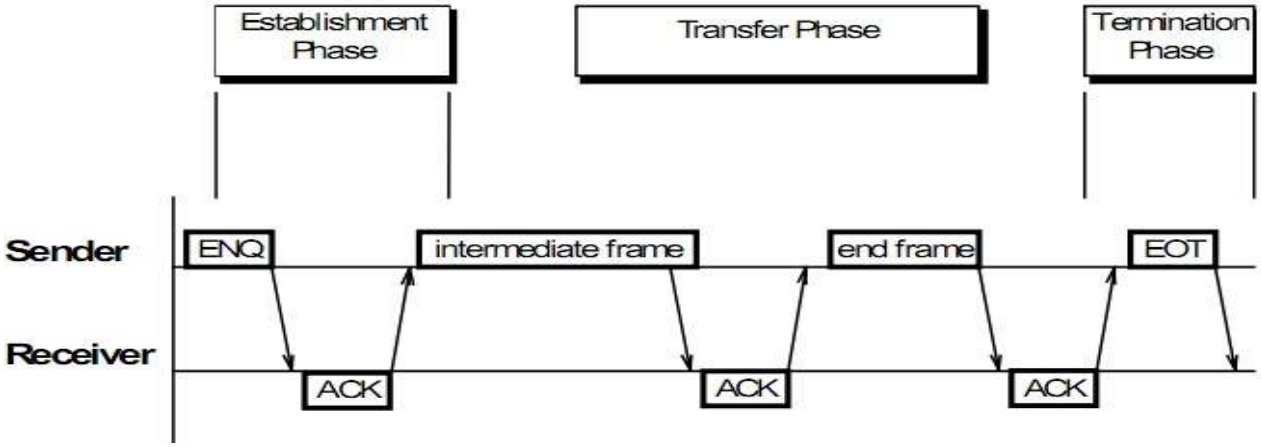

Figura 4.Fases de la capa de enlace a datos.

2.3.2 Fase de establecimiento. (ESTABLISHMENT PHASE)

En esta fase tanto el dispositivo como el sistema pueden estar esperando a convertirse en receptor, o iniciando la fase de establecimiento, en este caso mandando la información en marcos o en el último de los casos recibiendo o enviando los códigos de confirmación. Pero al mismo tiempo solo un dispositivo puede ser emisor, siendo el otro receptor o viceversa.

Ambos dispositivos pueden estar en reposo al mismo tiempo. Si la capa de presentación le pide a la capa de enlace a datos enviar algún registro el dispositivo debe de cambiar de estado de repaso a remitente. Para garantizar que un solo dispositivo trate de convertirse en emisor de una sola vez el dispositivo debe de iniciar la fase de establecimiento, por lo tanto envía un el código ASCII 5 [ENQ] y espera ser respondido con el ASCII 6 [ACK] para indicar este que ha cambiado el estado a receptor. Con la recepción de [ACK] se completa la fase que estamos tratando y comienza la fase de transferencia.

Cuando se está en estado de reposo cualquier otro carácter recibido que no sea [ENQ] será ignorado. Siempre que se reciba [ENQ] se debe contestar con [ACK].

Este sería el procedimiento normal de la fase de establecimiento. En caso de error estas son tres formas en que el receptor puede responder a un [ENQ]:

- ***Receptor envía cualquier carácter distinto de ACK.***

  Cuando el receptor envía después de haber recibido un [ENQ] un carácter distinto de [ACK] el emisor normalmente espera un tiempo que en el caso del Elecsys es 10 segundos y luego trata nuevamente con el envío de un [ENQ]. Repitiéndose esto hasta que sea respondido con un [ACK]

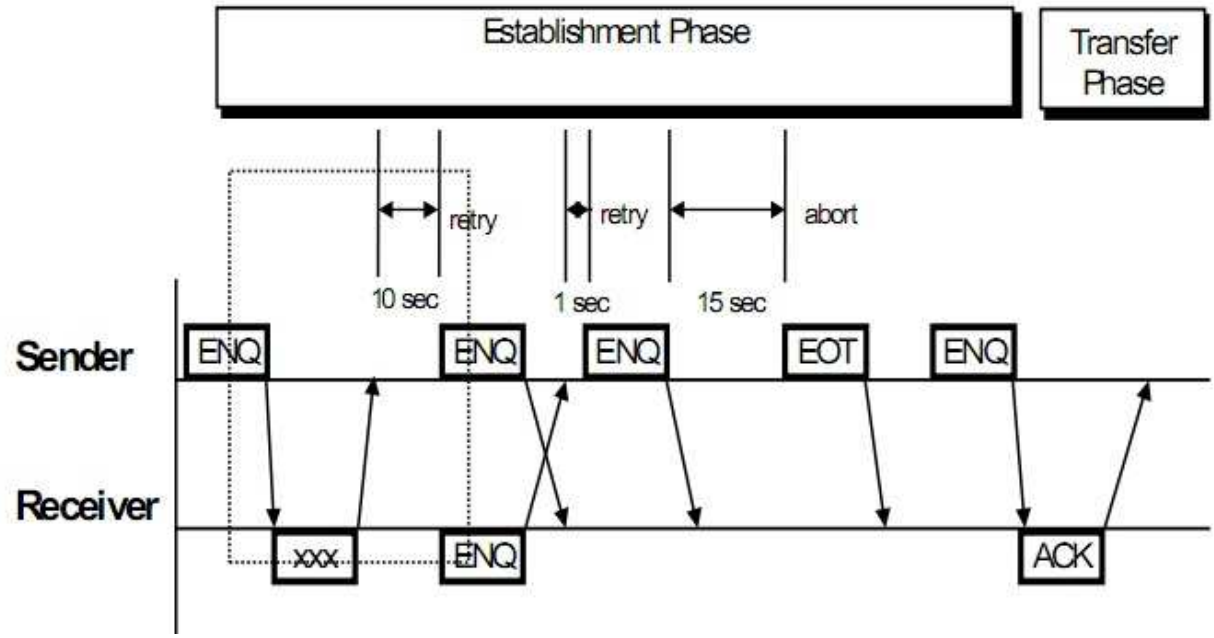

Figura 5.Respuesta de cualquier caracter.

- ***Receptor envía un ENQ.***

  Cuando los dos dispositivos tratan de convertirse en remitentes el sistema de instrumento (es decir: el Elecsys) tiene, por definición, la prioridad más alta para transmitir información. Por lo tanto el sistema de acogida tiene que parar el envío del [ENQ], ya que tiene que responder con [ACK] o [NAK]. Por otra parte el Elecsys espera al menos 1 segundo y luego regresa
  otro [ENQ]. Este ciclo se repite hasta que uno [ACK], [NAK], o cualquier otro
  caracter es recibido.

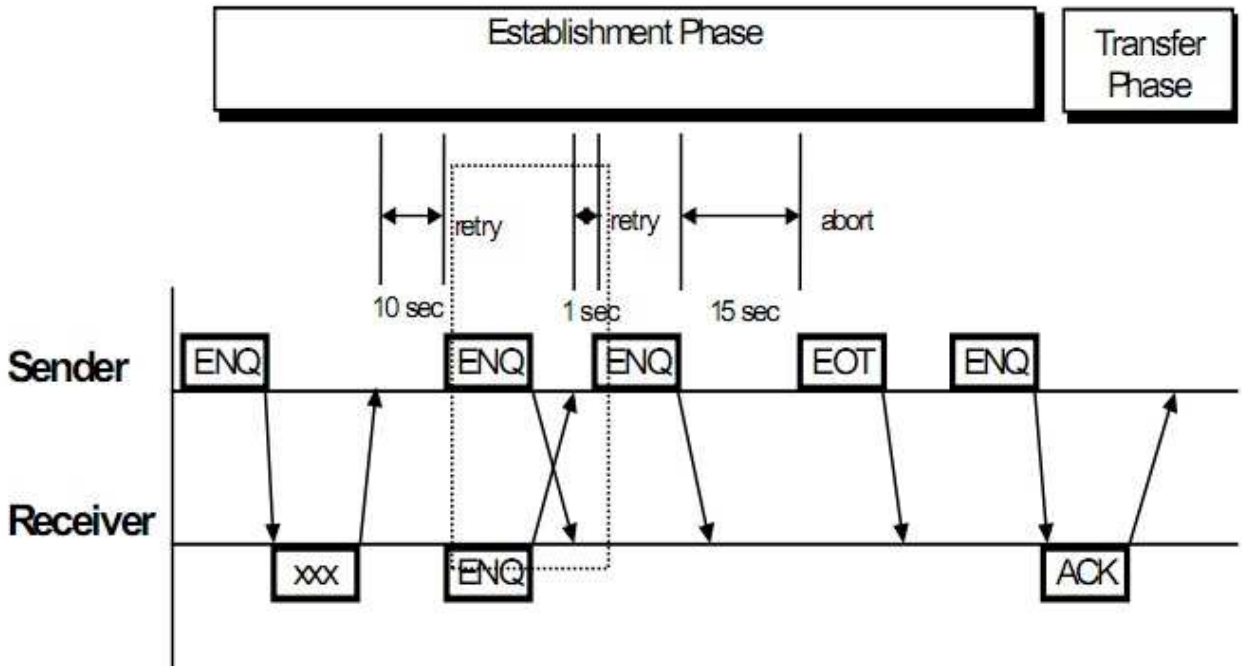

Figura 6.El receptor responde con ENQ.

- ***Receptor no responde.***

  El receptor espera 15 segundos e inicia la fase terminal mediante el envío de un [EOT] (código ASCII 4) y muestra un mensaje de error.

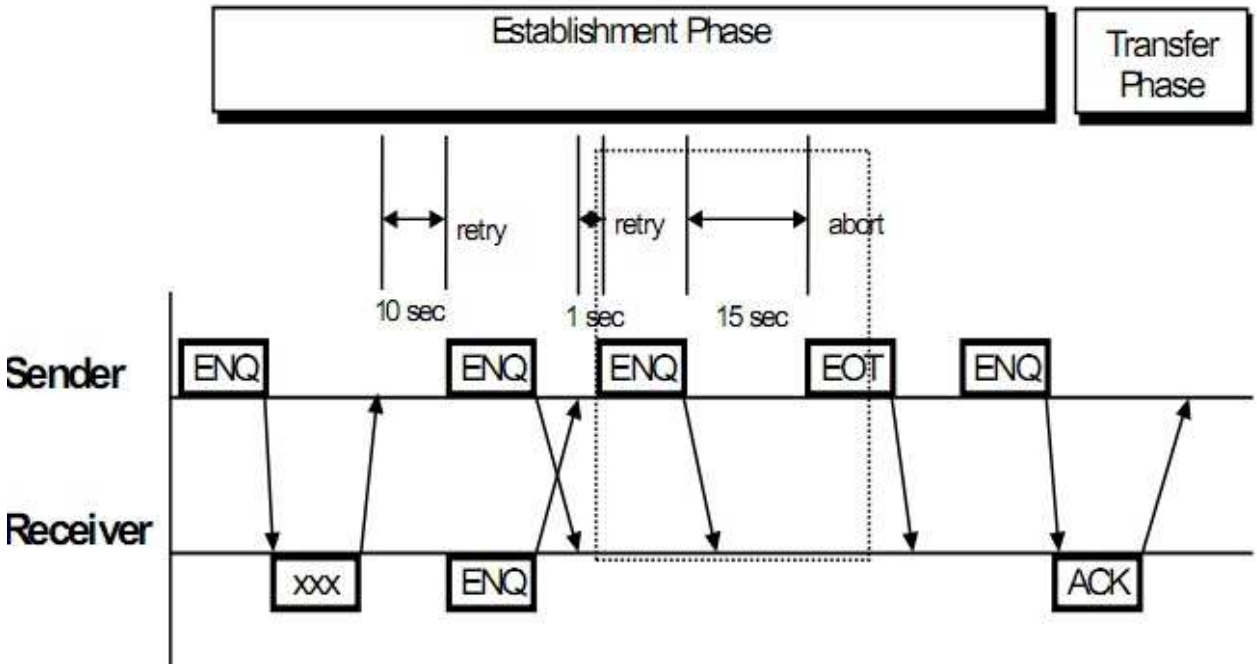

Figura 7.El receptor no responde.

- ***Receptor responde con ACK.***

El emisor cambia su estado para la fase de transferencia e inicia la transferencia de los records. Alternativamente, si el Elecsys está en modo de prueba el emisor se pone en fase terminal con el envío de un [EOT] (código ASCII 4).

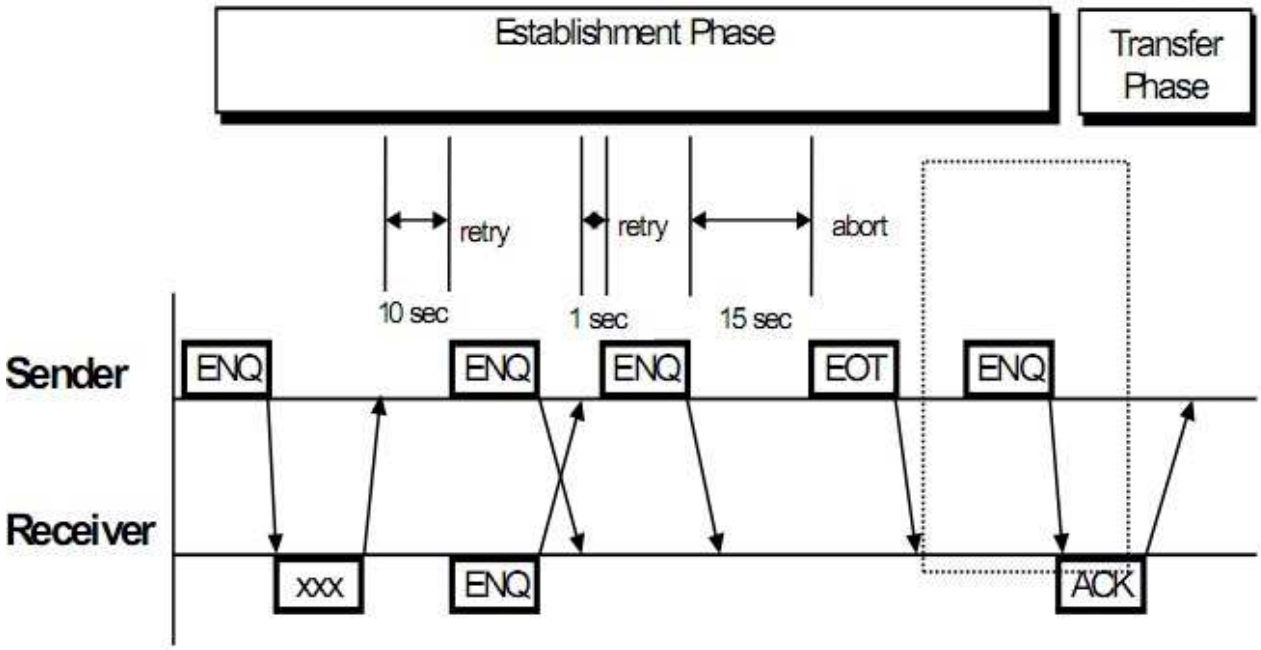

Figura 6.El receptor responde con ACK.

2.3.3 Fase de transferencia. (TRANFER PHASE)

La capa de presentación se basa en interpretar los registros que contienen la información del mensaje. Estos registros se intercambian con la capa de enlace de datos, que se espera transmitirá durante la fase de transmisión. Como los registros pueden tener una gran cantidad de bytes una tarea de la capa de enlace de datos es picar en porciones los records de no más de 240 bytes. Donde otros siete bytes son de delimitadores, checksum y los identificadores que se agregan como marco a la sobrecarga de mensajes marco. Así, el longitud de imagen puede variar entre 8 y 247 bytes.

Existen dos tipos de marcos, marcos intermedios y finales. Cuando estos marcos o mensajes sobrepasan los 240 bytes se corta en una o más porciones de 240 bytes y se ponen en marcos intermedios. Los bytes restantes para el último marco se colocan al final de este. Por ejemplo, hay dos marcos intermedios y un marco final para los registros con una longitud de 481 hasta 720 bytes. Los registros con una longitud máxima de 240 bytes dan lugar a un marco final. Dos registros diferentes nunca se ponen juntos en un solo marco.

Después que cada trama es enviada el receptor detiene la transmisión hasta que se recibe una respuesta o un tiempo que en este caso ejecuta una acción. Normalmente el receptor envía un [ACK] o un [EOT] para señalar que el último marco fue recibido con éxito y que está preparado para recibir el

siguiente marco. Cuando el receptor envía un [NAK] indica que el último marco no se ha recibido satisfactoriamente y que está preparado para la retransmisión del último marco.

2.3.4 Fase de finalización (TERMINATION PHASE).

La fase de finalización es la transición de ambos dispositivos al estado libre. Sólo puede ser iniciado por el emisor por el simple envío de un [EOT]. No hay respuesta del receptor a ese mensaje. Cada vez que un receptor detecta un EOT debe de cambiar su estado para libre.

2.4 Formato de los marcos.

Como ya se había mencionado anteriormente hay dos tipos de marcos, como son:

- Marco intermedio.

[STX]	FN	Text first char. ..... Text last char.	ETB	CH	CL	[CR]	[LF]

- Marco final.

[STX]	FN	Text first char. ..... Text last char.	ETX	CH	CL	[CR]	[LF]

2.5 Especificaciones de los requerimientos.

Para los manejadores que se desarrollan en el proyecto ya están definidos un conjunto de requerimientos tanto funcionales como no funcionales. Estos se encuentran especificados y descritos en el documento: “Especificaciones de Requerimientos de Software del Módulo de Drivers”, en su versión 1.3. De los requerimientos que se especifican en el documento antes mencionado, el manejador para el dispositivo Elecsys 2010, debe incluir en su funcionamiento los siguientes requerimientos:

Requerimientos funcionales:

- Parametrización (Casanova, 2004) del protocolo, de los dispositivos y redes.
- Configuración de dispositivos, redes y manejadores.
- Validación de las direcciones admisibles.
- Acceso a información de diagnóstico.

- Soporte para la configuración de variables de diagnóstico.
- Funciones de lectura.
- Estampado del tiempo.
- Calidad de los valores.
- Mensajes de Error.

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

3. DISEÑO, IMPLEMENTACIÓN Y PRUEBA.

3.1 Arquitectura de software.

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

Entre las responsabilidades que tiene la Arquitectura de software están:

- Definir los módulos principales.
- Definir la responsabilidad que tendrán cada uno de estos módulos.
- Definir la interacción que existirá entre dichos módulos:
    - Control y flujos de datos.
    - Secuencia de la información.
    - Protocolos de interacción y comunicación.
    - Ubicación del hardware.

La Arquitectura del Software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. (Casanova, 2004).

3.1.1 Patrones de arquitectura.

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

3.1.1.1 Arquitectura en capas.

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

- Arquitectura de dos capas.
- Arquitectura de tres capas.
- Arquitectura de n capas.

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

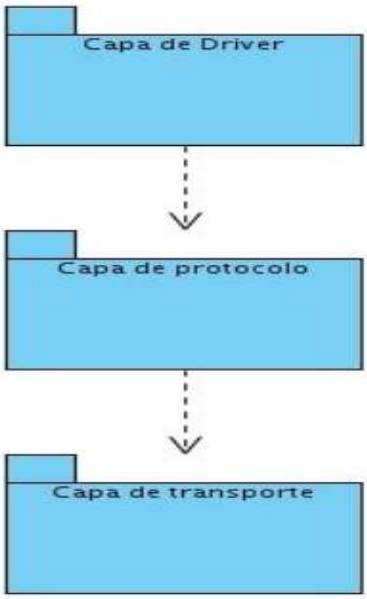

Figura 7. Arquitectura en tres capas del manejador.

Capa Driver.

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

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

La clase Block implementa el concepto de conjunto "consecutivo" de direcciones que tienen el mismo periodo de medición y que puede ser accedido a través de un solo mensaje del protocolo. Los bloques se construyen dinámicamente por los manejadores cada vez que se asocian variables al dispositivo y cada vez que se efectúan operaciones de escritura. Los bloques pueden ser de variables de diagnóstico del dispositivo, de variables de diagnóstico del manejador o de variables de protocolo, esta clasificación la regula el atributo addressType. Como las variables de un bloque son "consecutivas", para conocer las variables que integran el mismo sólo se necesita conocer el índice de la primera variable (firstVarIndex), el de la última variable (lastVarIndex) y el correspondiente

contenedor que es interno y es definido en la implementación de cada manejador. Una vez que se efectúa la lectura física el bloque tiene que retornar el valor de una dirección y de un tipo de dato. Para ello los descendientes de Block deben reimplementar los métodos para la lectura de enteros, flotantes, texto, vector de enteros y vector de flotantes, sucediendo lo mismo para la escritura.

Capa Protocolo.

La capa de protocolo tiene como objetivo garantizar la lógica de comunicación entre el manejador y los dispositivos, definiéndose para esto una clase EndPoint. En esta clase se controlan todos los pasos que define el protocolo para lograr una comunicación satisfactoria, segura y preparada para responder a los fallos que pudieran surgir, haciendo uso de una máquina de estado que facilita el diseño del comportamiento requerido por la especificación del protocolo. Cuando las tramas tienen algún nivel de complejidad, se crea una clase Mensaje que abstrae al EndPoint de la necesidad de conocer como se crean o cuáles son los elementos que constituyen una trama.

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

Capa transporte.

En la capa de transporte es utilizada la biblioteca dinámica TransportProvider. Esta biblioteca brinda una clase fábrica denominada TransportProvider para la creación de transportes asíncronos TCP, UDP y Serie, la cual hereda de una clase interfaz ITransportProvider. Para cada uno de los transportes que brinda esta biblioteca, existen dos clases: una clase interfaz que hereda de ITransport, adquiriendo sus características y una clase en la que se implementan las funcionalidades de su interfaz, correspondiente con las características específicas del transporte a que pertenece.

En el caso de TCP existen las clases ITCPTransport y TCPTransport, para UDP son la IUDPTransport y UDPTransport y por último para el transporte Serie están definidas las clases ISerialTransport y SerialTransport. La interfaz ITransport encapsula las funcionalidades asíncronas de los diferentes tipos de transporte que brinda la librería. La clase hereda de ITransportObject, define además de las funcionalidades asíncronas, otras funcionalidades comunes para las interfaces de transporte, como lo son el establecimiento de algunas propiedades y otras funcionalidades necesarias. Las funcionalidades asíncronas tienen la particularidad que retornan inmediatamente al ser invocadas, reciben como parámetro un handler, que no es más que una instancia de la clase que reimplementa la súper clase ITransportHandler. De esta manera se avisa en forma de callback cuando se culmina una lectura, escritura o conexión.

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

3.2 Diagrama de clases del diseño del manejador.

El siguiente diagrama representa el diseño de las clases más importantes del manejador para el autoanalizador Elecsys 2010.

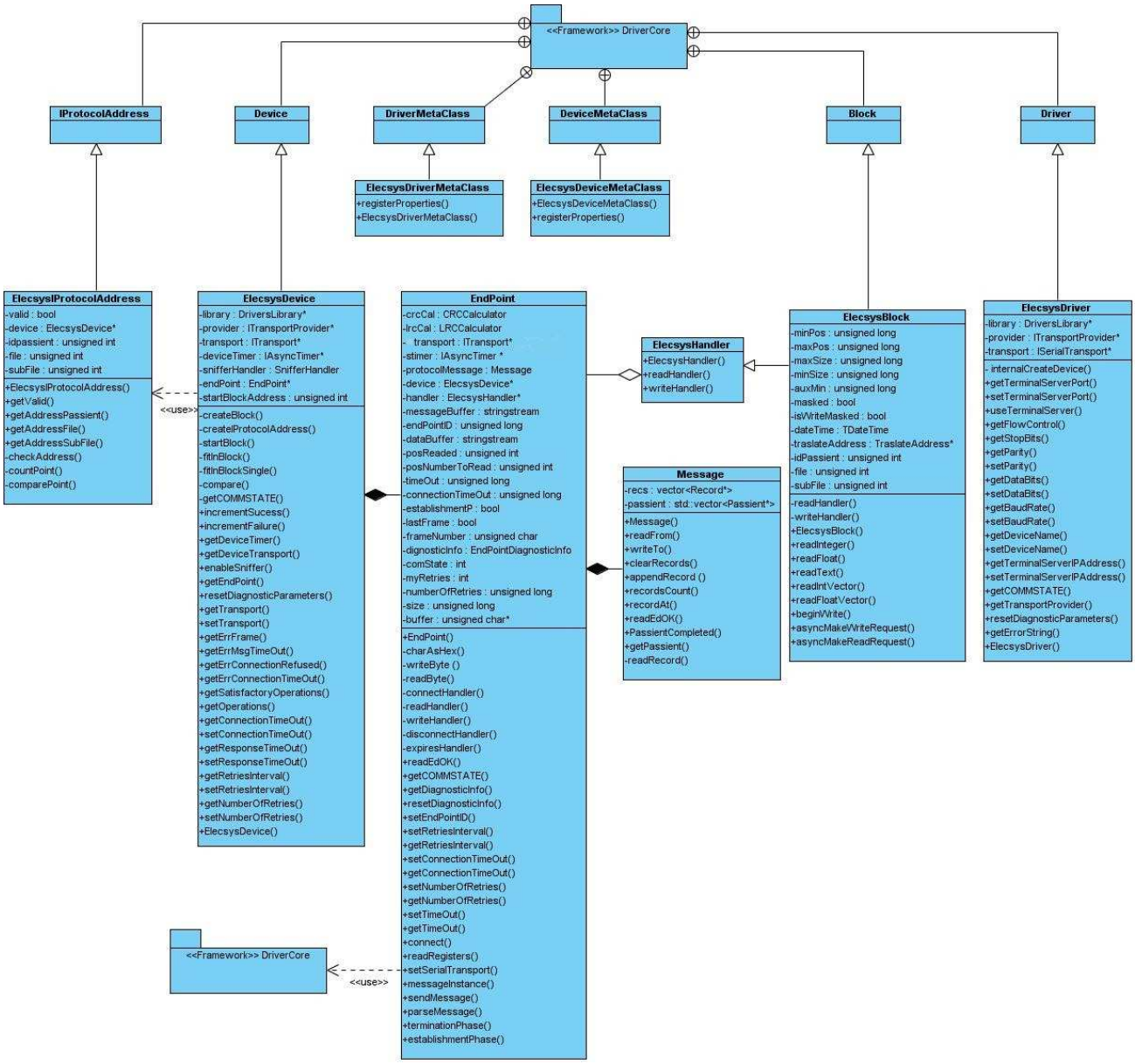

Figura 8. Diagrama de clases del Diseño Capa Driver.

3.3 Descripción de las clases del diseño.

3.3.1 Descripción de la clase EndPoint.

Esta clase encapsula la lógica de comunicación con el dispositivo. Permite obtener y configurar algunas propiedades de los dispositivos. Las respuestas a las lecturas se realizan cuando se han recibido todos los datos de una trama o del paciente del cual se requiere información. Para esto se tiene en cuenta que el Elecsys 2010 envía los datos de los pacientes de acuerdo al orden en que los va realizando.

Nombre: EndPoint	
Tipo de Clase: Controladora	
Atributo	**Tipo**
crcCal	CRCCalculator
lrcCal	LRCCalculator
transport	ITtransport*
stimer	IAsyncTimer*
protocolMessage	Message*
device	ElecsysDevice*
handler	ElecsysHandler*
messageBuffer	stringstream
posReaded	unsigned int
posNumberToRead	unsigned int
timeOut	unsigned long

connectionTimeOut	unsigned long
establishmentP	bool
lastNumber	bool
frameNumber	unsigned char
diagnosticInfo	EndPointDiagnosticInfo
comState	int
myRetries	int
numberOfRetries	unsigned long
size	unsigned long
buffer	unsigned char*

Para cada responsabilidad:	
Nombre:	EndPoint()
Descripción:	Constructor de la clase EndPoint.
Nombre:	readRegister(unsigned long posNumber, ElecsysHandler* newHandler)
Descripción:	Es llamada cuando se desea ejecutar una operación de lectura.
Nombre:	establishmentPhase()
Descripción:	Es llamada cuando se va a comenzar a leer.
Nombre:	terminationPhase()

Descripción:	Se ejecuta cuando no se quiere seguir recibiendo tramas del emisor.
Nombre:	parseMessage()
Descripción:	Envía para la clase Message un buffer con todos los datos de un paciente.
Nombre:	messageInstance()
Descripción:	Devuelve una instancia de la clase Message.
Nombre:	setSerialTransport(ISerialTransport* trans)
Descripción:	Establece el transporte que se utiliza para la conexión.
Nombre:	getTimeOut().
Descripción:	Retorna el valor definido para el timeOut.
Nombre:	setTimeOut(unsigned long _timeOut).
Descripción:	Establece el tiempo definido en el timeOut.
Nombre:	getNumberOfRetries().
Descripción:	Devuelve la cantidad de reintentos en una transacción.
Nombre:	setNumberOfRetries(unsigned long _number).
Descripción:	Establece la cantidad de reintentos en una transacción.
Nombre:	getConnectionTimeOut().
Descripción:	Retorna el tiempo que se espera por un intento de conexión.

Nombre:	setConnectionTimeOut(int connectionTimeOut).
Descripción:	Establece el tiempo que se espera por un intento de conexión.
Nombre:	getRetriesInterval().
Descripción:	Devuelve el tiempo entre reintentos de una transacción.
Nombre:	setRetriesInterval(int newRetriesInterval).
Descripción:	Establece el tiempo entre reintentos de una transacción.
Nombre:	setEndPointID(unsigned long newEndPointID).
Descripción:	Establece el identificador del EndPoint del manejador.
Nombre:	resetDiagnosticInfo().
Descripción:	Reinicia toda la información de diagnóstico a cero.
Nombre:	getDiagnosticInfo().
Descripción:	Devuelve una estructura con la información de diagnóstico.
Nombre:	getCOMMSTATE().
Descripción:	Devuelve el estado de la comunicación con el dispositivo.
Nombre:	readEdOK().
Descripción:	Devuelve si un paciente se encuentra.

Tabla 2. Descripción de la clase EndPoint.

3.3.2 Descripción de la clase MESSAGE.

La clase Message es la encargada de manejar los mensajes de protocolo. Ensambla los mensajes que son recibidos desde el Elecsys 2010 en estructuras.

Nombre: Message.	
Tipo de Clase: Entidad.	
Atributo	**Tipo**
Recs	vector <Record*>
Passient	std::vector <Passient*>
Para cada responsabilidad:	
Nombre:	Message().
Descripción:	Constructor de la clase Message.
Nombre:	readFrom (stringstream& st)
Descripción:	Ensambla las tramas enviadas por el dispositivo.
Nombre:	clearRecords()
Descripción:	Limpia el vector de Records.
Nombre:	appendRecord (Record* rec)
Descripción:	Adiciona un nuevo record al vector.
Nombre:	recordsCount().

Descripción:	Devuelve la cantidad de records que tiene el vector.
Nombre:	recordAt (unsigned int index)
Descripción:	Dada la posición en el vector devuelve el records correspondiente.
Nombre:	readEdOK(unsigned int posNumberToRead, unsigned int *posReaded).
Descripción:	Devuelve verdadero si los datos que se quieren conocer ya han sido enviados por el equipo.
Nombre:	PassientCompleted(unsigned int *posReaded).
Descripción:	Devuelve verdadero si el paciente está completo.
Nombre:	std::vector <Passient*> getPassient().
Descripción:	Devuelve el vector de paciente.
Nombre:	readRecord (unsigned char id, stringstream& st);
Descripción:	Almacena los datos enviados por el dispositivo.

Tabla 3. Descripción de la clase Message.

3.3.3 Descripción de la clase ELECSYSBLOCK.

La clase ElecsysBlock implementa el concepto de conjunto consecutivo de direcciones que puede ser accedido a través de un solo mensaje de protocolo. Hereda de la clase Block que proporciona la biblioteca DriverCore implementando la lectura asíncrona de las variables. Los bloques son creados teniendo la posición que ocupan las muestras en el disco.

Nombre: ElecsysBlock.

Tipo de Clase: Controladora.	
Atributo	**Tipo**
idPassient	unsigned int
File	unsigned int
subFile	unsigned int
minPos	unsigned long
maxPos	unsigned long
maxSize	unsigned long
minSize	unsigned long
auxMin	unsigned long
Masked	bool
isWriteMasked	bool
dateTime	TDateTime
traslateAddress	TraslateAddress*
Para cada responsabilidad:	
Nombre:	ElecsysBlock(Device* device, unsigned long, unsigned long blockEnd,bool forread, VarLinkVector* container)
Descripción:	Constructor de la clase ElecsysBlock.

Nombre:	readInteger(IProtocolAddress* address, DeviceVarType varType, TDateTime* deviceTimeStamp, unsigned short* quality).
Descripción:	Lee un valor entero del bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	readFloat(IProtocolAddress* address, DeviceVarType varType,TDateTime* deviceTimeStamp, unsigned short* quality)
Descripción:	Lee un valor flotante del bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	readText(IProtocolAddress* address, unsigned long arraySize, char* value, TDateTime* deviceTimeStamp, unsigned short* quality)
Descripción:	Lee una cadena del bloque a partir de un IProtocolAddress.
Nombre:	asyncMakeReadRequest()
Descripción:	Realiza una solicitud de lectura asíncrona.
Nombre:	readHandler(unsigned long error, Passient* passient)
Descripción:	Callback invocado por el EndPoint cuando finaliza una lectura. Establece el valor de la variable de error de lectura del bloque y avisa al Core que finalizó la realización de la lectura a través de la función Block::asyncReadCompleted().

Tabla 4. Descripción de la clase ElecsysBlock.

3.3.4 Descripción de la clase ELECSYSDEVICE.

Nombre: ElecsysDevice.

Tipo de Clase: Controladora.	
Atributo	**Tipo**
library	DriversLibrary*
provider	ITransportProvider*
transport	ITransport*
deviceTimer	IAsyncTimer*
snifferHandler	SnifferHandler
endPoint	EndPoint*
startBlockAddress	unsigned int

Para cada responsabilidad:	
Nombre:	ElecsysDevice(unsigned long devScadaId char* transferBuffer unsigned long bufferSize, ITransportProvider* provider)
Descripción:	Constructor de la clase ElecsysDevice.
Nombre:	createBlock(int blockStart, int blockEnd, bool forRead, VarLinkVector* varsVector)
Descripción:	Crea un bloque de variables.
Nombre:	createIProtocolAddress(const char* address, bool forRead)
Descripción:	Crea una instancia de IProtocolAddress a partir de la representación textual de la dirección dada por el parámetro address. El objeto IProtocolAddress responde por almacenar los parámetros de la dirección que son específicos de cada protocolo.

Nombre:	compare(IProtocolAddress* address1, IProtocolAddress* address2).
Descripción:	Compara dos direcciones de protocolo de acuerdo a las reglas del protocolo.
Nombre:	fitInBlockSingle(IProtocolAddress* address1, IProtocolAddress* address2, DeviceVarType varType, unsigned long arraySize, bool forRead, unsigned long addressSize).
Descripción:	Debe determinar si una variable puede alojarse en un bloque.
Nombre:	fitInBlock (IProtocolAddress* address2, DeviceVarType varType, unsigned long arraySize, bool forRead, unsigned long addressSize).
Descripción:	Determina si el rango de direcciones definido entre la dirección establecida por startBlock y la dirección que se define por los parámetros del método puede ser recuperado o modificado con un solo mensaje del protocolo.

Tabla 5. Descripción de la clase ElecsysDevice.

3.3.5 Descripción de la clase ELECSYSDRIVER.

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

Nombre: ElecsysDriver.	
Tipo de Clase: Controladora.	
Atributo	**Tipo**
library	DriversLibrary*
provider	ITransportProvider*

serialTransport	ISerialTransport*
Para cada responsabilidad:	
Nombre:	ElecsysDriver(ReadHandler readHandler, WriteHandler writeHandler, SnifferHandler snifferHandler)
Descripción:	Constructor de la clase ElecsysDriver.
Nombre:	getErrorString(unsigned long errorCode, const char** ppErrorDescription)
Descripción:	Permite obtener información textual comprensible a partir de un código de error del manejador.
Nombre:	internalCreateDevice(unsigned long devScadaId, char* transferBuffer, unsigned long bufferSize)
Descripción:	Crea una instancia de ElecsysDevice.

Tabla 6. Descripción de la clase ElecsysDriver.

3.3.6 Descripción de la clase ELECSYSIPROTOCOLADDRESS.

La clase ElecsysIProtocolAddress representa las direcciones de las variables del protocolo para la comunicación con el Elecsys 2010. Hereda de la clase IProtocolAddress que brinda la biblioteca DriverCore para encapsular el concepto de dirección de protocolo. En esta se define la validez de las direcciones de lectura del protocolo.

Nombre: ElecsysIProtocolAddress.
Tipo de Clase: Controladora.

Atributo	Tipo
valid	bool
device	ElecsysDevice*
idpassient	unsigned int
file	unsigned int
subFile	unsigned int

Para cada responsabilidad:	
Nombre:	ElecsysIProtocolAddress(ElecsysDevice* device, const char* address)
Descripción:	Constructor de la clase ElecsysIProtocolAddress.
Nombre:	getValid()
Descripción:	Expresa si la dirección es valida
Nombre:	checkAddress(const char* address)
Descripción:	Comprueba si la dirección entrada es válida.

Tabla 7. Descripción de la clase ElecsysIProtocolAddress.

3.4 Estándar de codificación.

El estándar de codificación, utilizado para el desarrollo de este producto de software, permite establecer una serie de reglas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. El manejador realizado está desarrollado completamente en inglés ya que es el idioma más

utilizado internacionalmente para la construcción de componentes de software. A continuación se describen algunos de los elementos fundamentales que componen el estándar utilizado:

Nombre de los ficheros.

Todos los ficheros .h y .cpp se nombrarán comenzando con mayúscula. Si el nombre está formado por más de una palabra cada una debe comenzar con mayúscula. Todas las demás letras deben escribirse en minúscula, ejemplo:

NameOfUnit.h

NameOfUnit.cpp

Clases.

Todas las clases se nombrarán comenzando con mayúscula. Si el nombre está formado por más de una palabra cada una debe comenzar con mayúscula. Todas las demás letras deben escribirse en minúscula, ejemplo:

MyClass.

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

Métodos o función miembro.

A excepción de los constructores y destructores todos los métodos deben empezar con minúscula. Cada palabra que forme parte del nombre se escribirá comenzando con mayúscula, por ejemplo:

myFunction

Condicionales y ciclos.

Todas las expresiones condicionales y los ciclos poseerán las llaves de inicio y cierre del campo de acción de la expresión, aunque la misma posea una sola línea de código.

3.5 Diagrama de despliegue.

El siguiente diagrama representa como será desplegado el manejador del Elecsys 2010 en un entorno de ejecución en términos de nodos.

Figura 9. Diagrama de despliegue.

3.6 Diagrama de componentes.

A continuación se representa como está compuesto el manejador en términos de componentes. Donde el mismo utiliza las bibliotecas DriverCore y TransportProvider en su implementación.

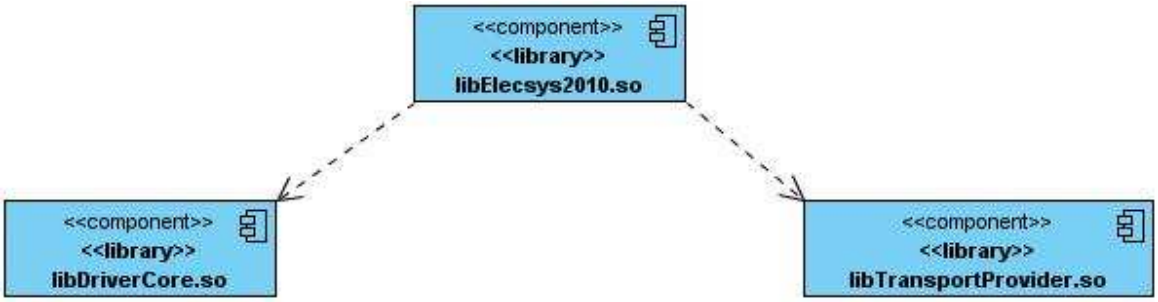

Figura 10. Diagrama de componentes.

3.7 Pruebas de software.

Las pruebas de software son los procesos que permiten verificar y revelar la calidad de un producto de software. Donde las mismas son utilizadas para identificar posibles fallos de implementación, calidad o usabilidad de un software.

Para determinar el nivel de calidad se deben efectuar unas medidas o pruebas que permitan comprobar el grado de cumplimiento respecto a las especificaciones iniciales del sistema. Por lo que al manejador Elecsys 2010 se le realizó pruebas de Caja Negra para comprobar que funcione correctamente.

3.7.1 Herramientas de pruebas utilizadas.

- Recolector Gráfico: Este es un componente que se desarrolló para comprobar que los manejadores cumplen sus funcionalidades.
- Simulador: Es un software desarrollado para enviar un conjunto de mensajes que están previamente definidos y almacenados en un buffer simulando al autoanalizador Elecsys 2010 hacia la computadora donde se ejecuta el manejador.
- Autoanalizador Elecsys 2010: Equipo que se encuentra en el CIMEQ al cual se le conectó una computadora para obtener los datos que envía en tiempo real y visualizarlos a través del Recolector Gráfico, luego de esto se compararon los datos mostrados por el Recolector Gráfico con los que muestra en la pantalla en Elecsys 2010 y se comprobó que los recibidos eran los correctos.
- Programa en consola de prueba, que carga la biblioteca y realiza varias lecturas llamando a la función deviceRead que proporciona la biblioteca DriverCore.

Para el caso de uso “Adquirir información de campo” especificado para todos los manejadores del SCADA se realizaron las siguientes pruebas.

Descripción general.

Este caso de uso permite recolectar los datos de los equipos que están conectados al sistema de supervisión. Las pruebas realizadas en este caso de uso son las siguientes:

1. Leer el tipo de dato entero.
2. Leer el tipo de dato real.
3. Leer el tipo de dato string.

CP1 Leer tipo de datos entero.

Descripción:

En este caso de prueba se puede obtener el valor de una variable entera y que la calidad de lectura sea 192 que es el valor que significa que es correcta la lectura. Esta prueba se realiza con el Recolector Gráfico y el Simulador del dispositivo.

Flujo central:

1. En la venta “Capturas” del Recolector Gráfico hacer click derecho y seleccionar Crear Variable.
2. En nombre se debe de establecer el nombre de la variable, en periodo poner periodo 1000, en dirección de lectura poner una dirección válida y en tipo de datos seleccionar INT y por último dar click en aceptar.
3. En el menú de Operaciones seleccionar Comenzar Recolección.

Condiciones de ejecución:

Tanto el manejador como el dispositivo deben de estar creados y configurados correctamente.

Iteraciones:

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la prueba	Observaciones
Se pide leer el número de secuencia de un paciente con la dirección 1.6		Se espera como número de secuencia 6842 y como valor de la calidad 192.	Se recibe 6842 como número de secuencia y el valor dado para la calidad es 192.	Verificar en la traza que el número enviado por el simulador es el mismo que el leído.
	Se pide leer la dirección 1.6.1.	Se debe de marcar la tabla de variables con fondo rojo y ponerla en el bloque 0.	Se marca la tabla de variables con fondo rojo y se pone en el bloque 0.	La dirección no es válida según las características del protocolo. Las variables no válidas van en el bloque 0.

CP1 Leer tipo de datos real.

Descripción:

En este caso de prueba se puede obtener el valor de una variable real y que la calidad de lectura sea 192 que es el valor que significa que es correcta la lectura. Esta prueba se realiza con el Recolector Gráfico y el Simulador del dispositivo.

Flujo central:

1. En la venta “Capturas” del Recolector Gráfico hacer click derecho y seleccionar Crear Variable.
2. En nombre se debe de establecer el nombre de la variable, en periodo poner periodo 1000, en dirección de lectura poner una dirección válida y en tipo de datos seleccionar REAL y por último dar click en aceptar.
3. En el menú de Operaciones seleccionar Comenzar Recolección.

Condiciones de ejecución:

Tanto el manejador como el dispositivo deben de estar creados y configurados correctamente.

Iteraciones:

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la prueba	Observaciones
Se pide leer el valor de la prueba TSH para el paciente 1 con dirección 1.19.1		Se espera ver el valor de la prueba con una calidad de 192.	Se visualiza un resultado correcto con una calidad de 192.	Verificar en la traza que el valor enviado por el simulador es el mismo que el visualizado por el Recolector.

CP1 Leer tipo de datos String.

Descripción:

En este caso de prueba se puede obtener el valor de una variable string y que la calidad de lectura sea 192 que es el valor que significa que es correcta la lectura. Esta prueba se realiza con el Recolector Gráfico y el Simulador del dispositivo.

Flujo central:

1. En la venta “Capturas” del Recolector Gráfico hacer click derecho y seleccionar Crear Variable.
2. En nombre se debe de establecer el nombre de la variable, en periodo poner periodo 1000, en dirección de lectura poner una dirección válida y en tipo de datos seleccionar STRING y por último dar click en aceptar.
3. En el menú de Operaciones seleccionar Comenzar Recolección.

Condiciones de ejecución:

Tanto el manejador como el dispositivo deben de estar creados y configurados correctamente.

Iteraciones:

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la prueba	Observaciones
Se pide leer la unidad de medida de la prueba TSH para el paciente 1 con dirección 1.19.1.		Se espera ver la unidad de medida de la prueba con una calidad de 192.	Se visualiza un resultado correcto con una calidad de 192.	Verificar en la traza que el valor enviado por el simulador es el mismo que el visualizado por el Recolector.

CONCLUSIONES

Al finalizar el desarrollo del manejador, que permite la comunicación con los autoanalizadores Elecsys 2010, se cumplieron los objetivos definidos para esta investigación, entre los cuales sobresalen los siguientes:

- El manejador desarrollado permite leer los datos que son enviados por el autoanalizador Elecsys 2010 de forma automática.
- Para la obtención diaria de la información de las muestras se realizará una sola configuración de las variables.
- El sistema permite que los datos se envíen de forma manual hacia la computadora en caso de falla de la comunicación.
- Almacenar los datos en formato digital permite realizar estudios sobre los pacientes atendidos y ahorrar recursos al no tener que archivar los datos impresos.
- El manejador obtenido cumple con los requisitos planteados de intercambio de información entre el SCADA y el autoanalizador Elecsys 2010.
- El proceso de desarrollo de este manejador posibilita reutilizar código ya existente y demuestra buenas prácticas de desarrollo como el uso de patrones de diseño.

RECOMENDACIONES.

Aspectos fundamentales que se recomienda tener en cuenta para versiones futuras del manejador Elecsys 2010:

- Perfeccionar los mecanismos relacionados con el tratamiento de manejadores asíncronos.
- Perfeccionar los mecanismos para la comunicación con dispositivos mediante el modelo productor-consumidor.
- Implementar las clases que permitan la lectura de datos técnicos y de calidad del equipo y reactivos.
- Implementar un modelo que reduzca el acoplamiento entre la capa de protocolo y la capa de transporte.
- Efectuar seguimiento de las actualizaciones de la biblioteca de transporte Asio C++ cuyas versiones futuras pudieran valorarse nuevamente para su utilización.

Bibliografía

Casanova, Josep. 2004. *Usabilidad y arquitectura del software.* 2004.

eclipse, Plataforma. 2009. Plataforma eclipse. [En línea] 2009. [Citado el: 25 de Marzo de 2010.] http://plataformaeclipse.com/.

IEEE, Standards Assosiations. 2000. Sitio Web de la IEEE. *Sitio Web de la IEEE.* [En línea] IEEE, 2000. [Citado el: 2 de Mayo de 2010.] http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html..

Informática.Net, Biblioteca. 2004. Biblioteca Informática.Net. *Hardware | Concepto de driver.* [En línea] 2004. [Citado el: 30 de Enero de 2010.] http://bibliotecainformatica.iespana.es/hardware/drivers.htm.

Instrument, National. 2006. National Instrument. *National Instrument.* [En línea] 6 de Junio de 2006. [Citado el: 17 de Abril de 2010.] http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1.

Jiménez López, Fernándo, y otros. 2007. [En línea] 2007. [Citado el: 14 de Enero de 2010.] http://biblioteca.reduc.edu.cu/biblioteca.virtual/cgi/CD-ROM/otros/UCIENCIA%202007%20(E)/ponencias/taia/PDF/13%20Modulo%20BDH%20para%20SCADA-2892875776/13%20Modulo%20BDH%20para%20SCADA.pdf.

Kohlhoff, Christopher M. 2008. Rationale. [En línea] 2008. [Citado el: 27 de Abril de 2010.] http://think-async.com/Asio/asio-1.4.1/doc/asio/overview/rationale.html.

Kubuntu. 2008. Kubuntu-es. *Kubuntu-es.* [En línea] 2008. [Citado el: 9 de Febrero de 2010.] http://www.kubuntu-es.org/wiki/desarrollo-programacion/programas-desarrollo-libres.

Montero, Dagoberto y Barrantes, David. 2004. *Introducción a los sistemas de control supervisor y adquisición de datos(SCADA).* Costa Rica : s.n., 2004.

Pasaje, Julio Luis Medina. 2005. *Metodología y Herramientas UML para el Modelado y Análisis de Sistemas de Tiempo Real Orientados a Objetos.* Santander : s.n., 2005.

Peñalvo, Francisco José García. 2002. Universidad de Salamanca. *Universidad de Salamanca.* [En línea] 2002. [Citado el: 3 de Febrero de 2010.] http://zarza.usal.es/~fgarcia/docencia/poo/01-02/Tema1.pdf.

Pozo, Luis Enrique García Hernámdez y Antonio Cedeño. 2009. *Manual para la implementación de Manejadores.* 2009.

Roche. 2007. *Roche de la A a la Z al servicio de la salud.* Suiza : s.n., 2007.

Roche. 2000. *Roche Diagnostics GmbH Elecsys Analyzer Host Interface Manual.* Suiza : s.n., 2000.

Santos, Leonardo Rafael Perez. 2008. *Diseño e implementación de un sistema de adquisición de datos y su interface como una red central de datos en la empresa aseria de Ecuador.* Quito : s.n., 2008.

Stallings, William. 2003. *Comunicación y redes computadores.* 2003.

GLOSARIO

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

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

Controlador Lógico Programable (PLC): dispositivos electrónicos usados en automatización industrial para realizar estrategias de control básicas. Por su robustez y características sencillas de control, están cercanas al proceso, permitiendo ejecutar las tareas básicas del control, aun cuando no tenga conexión a las capas superiores del control.

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

Protocolo Ethernet-IP: (Ethernet/Industrial Protocol) es un sistema de comunicación diseñado para entornos industriales. EtherNet-IP permite que los dispositivos de control intercambien información crítica con requerimientos estrictos de tiempo.

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

Sistemas de Adquisición de Datos y Control Supervisorio (SCADA): Aplicación de software especialmente diseñada para funcionar sobre computadores en el control de producción, proporcionando comunicación con los dispositivos de campo y controlando el proceso de forma automática desde la pantalla del operador, proveyendo toda la información asociada al proceso.

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

- Libertad 0: Libertad de ejecutar el programa como quieras.
- Libertad 1: Libertad de estudiar el código fuente y cambiarlo para realizar lo que desees.
- Libertad 2: Libertad de realizar copias y distribuirlas cuando quieras.
- Libertad 3: Libertad de distribuir o publicar versiones modificadas cuando desees.

TCP/IP: Familia de protocolos sobre los cuales funciona Internet, que se ha convertido en el estándar actual de comunicación entre computadoras. Conocida por estas siglas debido a que los dos protocolos más importantes son: el protocolo IP, que se ocupa de transferir los paquetes de datos hasta su destino adecuado y el protocolo TCP, que se ocupa de garantizar que la transferencia se lleve a cabo de forma correcta y confiable.

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

Dispositivo: Se denomina dispositivo a un elemento de hardware que alberga información de proceso o estado de sí mismo, que forma parte de un sistema automatizado. Comúnmente los dispositivos son Controladores Lógicos Programables, Computadoras Industriales, Sistemas de Control Distribuidos, sensores o actuadores inteligentes con capacidad de comunicación.