Universidad de las Ciencias Informáticas

"Facultad 5"

Título: "Capa de acceso a datos síncrona y asíncrona para dispositivos Modbus TCP"

Trabajo de Diploma para optar por el título de Ingeniero Informático

Autor(es): "Rubén Gómez Johnson"

Tutor(es): "Dr. Rafael Arturo Trujillo"

"Ing. Amado Espinosa"

Cuidad de La Habana, Julio de 2008

A mi mamá, María Elena

Y a la memoria de mi padre que siempre soñó con verme graduado.

A mis abuelos Elena y Alberto, a Danai y al resto de mi familia.

Agradecimientos

Muchas personas han contribuido de una forma u otra en la elaboración de este trabajo y en mi formación profesional, a todas ellas quiero expresar mi más sincero agradecimiento. De forma muy especial agradecerle a mi madre por el cariño, la confianza y la dedicación que siempre me brindó desde que era pequeño, junto con mi querida abuela. A Danai por la compresión y el apoyo en los momentos malos y en los buenos, a sus padres, Ricardo e Idolidia por la preocupación y la atención prestada.

Le agradezco a mi tutor, Dr. Rafael A. Trujillo el cual estuvo dispuesto en todo momento a atender mis dudas, propuestas y decisiones, que iban surgiendo en el desarrollo de la investigación, de igual manera al profesor y Co-tutor, Amado Espinosa, la profesora Iliana Perez Pupo y a mis compañeros del proyecto productivo SCADA, compañeros de brigada y a mis amigos.

Resumen

En todo proceso de automatización es necesario captar las magnitudes de la planta, para poder así, saber el estado y evolución del proceso que estamos controlando. Para ello se emplean dispositivos como autómatas programables, PLC, sensores capaces de adquirir los valores anteriormente mencionados. Estos dispositivos poseen uno o varios protocolos de comunicación mediante el cual se puede acceder a los datos recopilados del campo.

La implementación del protocolo Modbus existente en la línea de manejadores del proyecto SCADA, en la Universidad de Ciencias Informáticas, es totalmente sincrónico. Los modelos síncronos son simples y seguros ya que la lógica de programación es secuencial, pero debido a que el SCADA en condiciones normales necesita comunicarse con cientos de dispositivos de forma concurrente, este modelo exige un hilo por cada dispositivo a ser encuestado en paralelo, lo cual es ineficiente para la mayoría de los sistemas operativos.

Debido al amplio consumo de recursos del sistema cuando el número de dispositivos a atender es grande, se pretende desarrollar una capa de acceso a datos para dispositivos Modbus TCP, que pueda comunicarse con una gran variedad de dispositivos, adquiriendo de los mismos, información relacionada con los procesos automatizados, de forma concurrente y minimizando el consumo de recursos del sistema operativo utilizados para este propósito.

Con tal objetivo se realizó un estudio del protocolo de comunicación Modbus, especialmente su variante de intercambio de mensajes sobre la red Ethernet usando el protocolo de nivel de transporte TCP (*Transfer Control Protocol*) y términos relacionados, se describen librerías utilizadas para la comunicación sobre TCP en diferentes sistemas operativos, además del diseño e implementación de la capa de acceso a datos.

Índice.

Introducción. 1

1 Capítulo 1. Fundamentación Teórica. 4

Introducción.. 4

1.1 Modelo de referencia OSI. 4

1.2 Protocolo de Control de Transmisión (TCP). 6

1.3 Protocolo Modbus. 6

1.3.1 Principios de operación del protocolo. 8

1.3.2 Características de la Encuesta. 9

1.3.3 Características de la Respuesta. 9

1.3.4 Formato de la trama de encuesta/respuesta. 10

1.3.5 Modbus ASCII.. 10

1.3.6 Modbus RTU. 11

1.3.7 Modbus TCP.. 11

1.3.8 Funciones asociadas. 14

1.3.9 Clasificación de los servicios en categorías. 15

1.4 Modelo cliente/servidor en Modbus TCP.. 16

1.5 Controladores Lógicos Programables (PLC). 16

1.5.1 Campo de aplicación de los PLC. 17

1.5.2 Funciones básicas de un PLC. 18

1.6 Tendencias y desarrollo de la variante Modbus TCP. 19

1.7 Tecnologías libres más utilizadas. 20

1.7.1 Librería socket de C++.. 20

1.7.2 Qt.. 22

1.7.3 Asio C++ Library. 22
1.8 Herramientas de desarrollo empleadas. 23
1.9 Metodología de desarrollo de software. 24
2 Capitulo 2: Características del sistema. 27
Introducción. 27
2.1 Flujo actual de los procesos. 27
2.2 Objeto de automatización. 29
2.3 Propuesta del sistema. 30
2.4 Reglas del Negocio. 34
2.5 Modelo del dominio. 35
2.5.1 Glosario de términos del dominio. 35
2.6 Especificación de requerimientos del software. 36
2.6.1 Requerimientos funcionales. 36
2.6.2 Requerimientos no funcionales. 37
2.7 Modelo de casos de usos del sistema. 38
2.7.1 Actor del sistema. 38
2.7.2 Casos de uso del sistema. 38
2.7.3 Diagrama de caso de usos del sistema. 38
2.7.4 Especificación de los CU en formato expandido. 39
3 Capitulo 3: Diseño e implementación del sistema. 48
Introducción. 48
3.1 Diagrama de clases del diseño. 49
3.2 Descripción de las clases del diseño. 49
3.3 Diagramas de secuencia. 76
3.4 Estándar de codificación. 81

3.5 Diagrama de despliegue. 82

3.6 Diagrama de componentes. 83

Conclusiones 84

Recomendaciones. 85

Referencia bibliografía 86

Anexos 88

Anexo 1. Especificación del CU Obtener información de diagnóstico. 88

Anexo 2. Especificación del CU Interpretar código de error. 89

Anexo 3. Descripción de la clase AbstractVector. 90

Anexo 4. Descripción de la clase WordVector. 91

Anexo 5. Descripción de la clase BitVector. 93

Glosario de términos. 95

Índice de Figuras y Tablas. 100

Introducción.

Controlar cada detalle relacionado con los procesos que se realizan en la industria es una premisa para el hombre moderno. Tener conocimiento y la forma de poder mantener constantes algunas magnitudes que interactúan en el proceso productivo, tales como la presión, el caudal, el nivel, la temperatura, el pH, la conductividad, la velocidad, la humedad, el punto de rocío y otros, es fundamental para el buen funcionamiento de la industria.

Los primeros mecanismos de control eran muy simples, sistemas de telemetría, que proporcionaban reportes periódicos de las condiciones de campo obtenidas a través de la observación de señales que representaban medidas y/o condiciones de estado. Estos sistemas ofrecían capacidades muy simples de monitoreo y control. Los operadores interpretaban el sistema a través de una pizarra con indicadores en la cual se reflejaban luces ubicadas en la parte posterior de dicha pizarra. Con el desarrollo de la tecnología las computadoras asumen un gran papel en la recolección de datos, en el surgimiento de los comandos de control y un nuevo logro, la presentación de la información en una pantalla de video. Además permitieron poder programar los sistemas con funcionalidades un poco más complejas.

Dentro de los mecanismos de control mencionados anteriormente encontramos los sistemas SCADA acrónimo de Supervisory Control and Data Acquisition lo que significa en español Control Supervisor y Adquisición de Datos. En sus inicios un SCADA representaba o estaba desarrollado con fines específicos en dependencia de la industria a la cual pertenecía el sistema. Luego con el tiempo los desarrolladores de SCADA se dieron la tarea de diseñar un producto que pudiese satisfacer las necesidades de varias industrias o por lo menos módulos con características generales que pudiesen ser utilizados comúnmente sin importar aspectos específicos.

Hoy en día los sistemas SCADA juegan un papel fundamental en las industrias, formando parte integral de las mismas. Llevan a cabo todo el proceso de supervisión, monitoreo y control de la instalación y sus procesos, adquiriendo datos desde dispositivos encontrados a miles de kilómetros. Están concebidos para evitar, prevenir y alertar situaciones inesperadas en la industria, como desastres ecológicos en el caso de aquellas empresas que trabajan con materiales contaminantes, preservar y proteger la salud de trabajadores. Además manejan grandes cantidades de datos con los cuales se pueden realizar análisis,

estudios de comportamiento, detección de futuras irregularidades y muy importante el poder tomar decisiones ante diferentes circunstancias.

Cuba no se encuentra ajena a estos sistemas, pues se cuenta con la experiencia en la construcción de varios tales como: EROS con mas de 200 copias distribuidas por todo el país en diferentes sectores de la economía, SISPro que fue creado por la Empresa de Servicios Informáticos de Villa Clara, ahora convertida en CEDAI Villa Clara.

Como resultado de las relaciones de cooperación entre Venezuela y Cuba surge en la Universidad de Ciencias Informáticas (UCI), "El proyecto SCADA" o "SCADA Nacional" como así muchos le llaman. El mismo, arquitectónicamente esta dividido en varias líneas de investigación y desarrollo, donde podemos encontrar la línea de manejadores (driver en ingles) encargada de la comunicación del sistema con los dispositivos de campo y la responsable de que los datos registrados por dichos dispositivos puedan ser accedidos desde el sistema.

La línea utiliza para el acceso a los dispositivos diferentes protocolos de comunicación como Modbus [1.3] en todas sus variantes, Ethernet-IP, BSAP, ABEthernet. En la producción y desarrollo nos encontramos con la siguiente situación problemática: Los manejadores que están conveniados para el proyecto SCADA, son todos sincrónicos. Esto significa que cuando se invoca una función de lectura o escritura de alguna variable existente en algún dispositivo Modbus, por ejemplo, el hilo que realiza la solicitud queda bloqueado hasta tanto no se reciba una respuesta del dispositivo o expire el tiempo máximo predeterminado para realizar la operación (conocido como Time Out). Los modelos sincrónicos son simples y seguros ya que la lógica de programación es secuencial, sin embargo, un SCADA en condiciones normales, adquiere y modifica los valores de variables en cientos de dispositivos al mismo tiempo y para lograr la mayor concurrencia posible, este modelo exige un hilo por cada dispositivo que pueda ser encuestado en paralelo. Teniendo en cuenta que el cambio de contexto entre hilos en los sistemas de cómputo es una operación costosa en cuanto a procesamiento, nos enfrentamos a la problemática del amplio consumo de recursos del sistema cuando el número de dispositivos a atender es grande.

Dada esta situación, se analizó el siguiente **problema**:

¿Cómo implementar una capa de acceso a datos para dispositivos Modbus TCP, que permita la interacción concurrente con los mismos, minimizando los recursos utilizados del sistema operativo?

Para darle solución a dicho problema se propuso abordar como **objeto de estudio**, el protocolo de comunicación Modbus, y como **campo de acción**, el intercambio de mensajes Modbus con dispositivos de campo o PLC [1.5], sobre TCP/IP. Como **objetivo general**: implementar una capa de acceso a datos para dispositivos Modbus TCP, que permita la captura concurrente de los datos minimizando los recursos utilizados del sistema operativo.

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

1. Estudio de la bibliografía referente al protocolo de nivel de transporte TCP.
2. Estudio sobre el protocolo de comunicación Modbus.
3. Análisis y determinación de las tecnologías de transporte TCP más adecuada para el desarrollo de la capa.
4. Análisis y diseño de la capa de acceso a datos.
5. Utilización de estructuras compatibles con la mayoría de los compiladores de C++ y con los sistemas operativos más utilizados.

Se utilizarán varios métodos científicos de investigación como: ***análisis-síntesis,*** que permitirá conocer los principales fundamentos y teorías relacionadas con el protocolo de comunicación Modbus y su variante Modbus TCP. ***Modelación,*** se realizarán diagramas y modelos que me permitan estructurar teóricamente el sistema. ***Histórico lógico***, con el objetivo de conocer la evolución de las teorías y tendencias del intercambio de mensajes Modbus sobre el protocolo TCP.

El presente documento está dividido en tres capítulos, cada capitulo se divide en epígrafes y sub-epígrafes temáticos en los cuales se abordan temas particulares. El primer capítulo define los conceptos fundamentales, las tecnologías utilizadas en el objeto de estudio que se aborda, además se describe las herramientas y metodologías vinculadas con el desarrollo del software. En el segundo capítulo se analiza de forma detallada los procesos que originan la situación problemática, se describe la solución propuesta, lo que formará parte del objeto de automatización, características del sistema, requisitos funcionales y no funcionales, análisis de sistema, entre otros. El tercer capitulo aborda el diseño e implementación del sistema.

1 Capítulo 1. Fundamentación Teórica.

Introducción.

En el siguiente capítulo se hace alusión a teorías relacionadas con el protocolo de comunicación Modbus en su variante TCP, dentro de estas teorías se podrán encontrar el modelo de referencia OSI [1.1], el Protocolo de Control de Transmisión (TCP) [1.2], generalidades del protocolo de comunicación Modbus, aspectos específicos del intercambio de mensajes Modbus sobre TCP, características de los dispositivos de campo o PLC (Controlador Lógico Programable) [1.5] de donde se obtienen los datos, tendencias del desarrollo de librerías que implementen Modbus sobre TCP [1.2], además de tecnologías y metodologías utilizadas para el desarrollo de la capa de acceso a datos.

1.1 Modelo de referencia OSI.

En 1977 la Organización Internacional de Estandarización (*ISO* siglas en inglés) creó un subcomité para desarrollar estándares de comunicación de datos que promovieran la accesibilidad universal y una interoperabilidad entre productos de diferentes fabricantes. El resultado fue el modelo de referencia para la Interconexión de Sistemas Abiertos OSI, adoptado en los años 80, que establece unas bases que permiten conectar sistemas abiertos para procesamiento de aplicaciones distribuidas.

El Modelo OSI es un lineamiento funcional para tareas de comunicaciones y, por consiguiente, no especifica un estándar de comunicación para dichas tareas. Sin embargo, muchos estándares y protocolos cumplen con los lineamientos del Modelo OSI. (1)

El modelo define una arquitectura de comunicación estructurada en siete niveles verticales. Cada nivel soluciona una serie de problemas relacionados con la transmisión de datos y proporciona un servicio bien definido a los niveles más altos. Los niveles superiores son los más cercanos al usuario y tratan con datos más abstractos, dejando a los niveles más bajos la labor de traducir los datos de forma que sean físicamente manipulables.

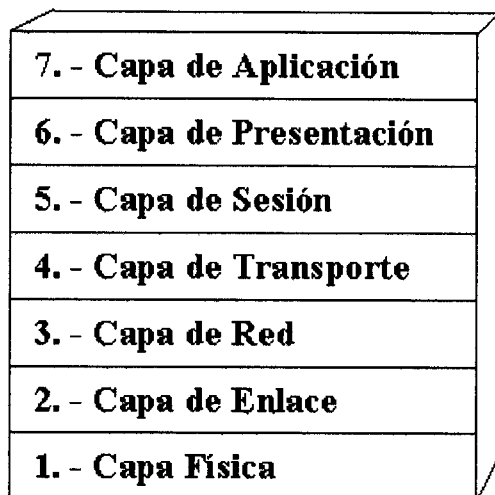

Figura 1: Capas del modelo OSI.

Capa Física: Esta capa se ocupa de la transmisión de bits .en forma continua a lo largo de un canal de comunicación.

Capa de Enlace: Realiza detección y posiblemente corrección de errores. La capa de enlace transmite los bits en grupos denominados tramas.

Capa de Red: La capa de red se ocupa del control de la subred, pues es la que tiene el conocimiento de la topología de la red, y decide porque ruta va ha ser enviada la información para evitar la congestión. En esta capa maneja los bits agrupados por paquetes.

Capa de Transporte: La capa de transporte es la encargada de fragmentar de forma adecuada los datos recibidos de la capa superior para transferirlos a la capa de red, asegurando la llegada y correcta recomposición de los fragmentos en su destino.

Capa de Sesión: Es la primera capa accesible al usuario y en un sistema multiusuario.

Capa de sesión: Se ocupa de comunicar los hosts.

Capa de Presentación: Se encarga de la preservación del significado de la información recibida y su trabajo consiste en codificar los datos de la máquina transmisora a un flujo de bits adecuados para la transmisión y luego decodificarlos, para presentarlos en el formato del destinatario.

Capa de Aplicación: La capa de aplicación contiene los programas del usuario, además que contiene los protocolos que se necesitan frecuentemente.

1.2 Protocolo de Control de Transmisión (TCP).

TCP (Transmission Control Protocol, en español **Protocolo de Control de Transmisión**) es uno de los protocolos fundamentales en Internet. Fue creado entre los años 1973 - 1974 por Vint Cerf y Robert Kahn. Muchos programas dentro de una red de datos compuesta por ordenadores pueden usar TCP para crear conexiones entre ellos a través de las cuales puede enviarse un flujo de datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. También proporciona un mecanismo para distinguir distintas aplicaciones dentro de una misma máquina, a través del concepto de puerto. TCP da soporte a muchas de las aplicaciones más populares de Internet, incluidas HTTP, SMTP y SSH.

TCP está orientado a la conexión en el sentido de que, antes de transmitir datos, los participantes deben establecer la conexión. Todos los datos viajan en segmentos TCP, en donde cada viaje se realiza a través de Internet en un datagrama IP. El conjunto de protocolos se conoce frecuentemente como TCP/IP debido a que el TCP y el IP son los dos protocolos más importantes. (2)

1.3 Protocolo Modbus.

Modbus es un protocolo de comunicaciones situado en el nivel 7 del Modelo OSI, basado en la arquitectura maestro/esclavo o cliente/servidor, diseñado en 1979 por MODICON (Modular Digital Controller) para su gama de controladores lógicos programables (PLC) [1.5]. Convertido en un protocolo de comunicaciones estándar de facto en la industria es el que goza de mayor disponibilidad para la

conexión de dispositivos electrónicos industriales. Las razones por las cuales el uso de Modbus es superior a otros protocolos de comunicaciones son:

- es público.
- su implementación es fácil y requiere poco desarrollo.
- maneja bloques de datos sin suponer restricciones.

Modbus permite el control de una red de dispositivos, por ejemplo un sistema de medida de temperatura y humedad, y comunicar los resultados a un ordenador. Modbus también se usa para la conexión de un ordenador de supervisión con una unidad remota (RTU). Existen variantes del protocolo Modbus para puerto serie y Ethernet (Modbus/TCP).

En el caso de las variantes series tenemos Modbus RTU es una representación binaria compacta de los datos y Modbus ASCII representación legible del protocolo pero menos eficiente. El formato RTU finaliza la trama con una suma de control de redundancia cíclica (CRC), mientras que el formato ASCII utiliza una suma de control de redundancia longitudinal (LRC).

La versión Modbus/TCP es muy semejante al formato RTU, pero estableciendo la transmisión mediante paquetes TCP/IP.

Cada dispositivo de la red Modbus posee una dirección única. Cualquier dispositivo puede enviar órdenes Modbus, aunque lo habitual es permitirlo sólo a un dispositivo maestro. Cada comando Modbus contiene la dirección del dispositivo destinatario de la orden. Todos los dispositivos reciben la trama pero sólo el destinatario la ejecuta (salvo un modo especial denominado "Broadcast" en español difusión). Cada uno de los mensajes incluye información redundante que asegura su integridad en la recepción. Los comandos básicos Modbus permiten controlar un dispositivo, modificando el valor de alguno de sus registros o bien solicitar el contenido de dichos registros.

Existe gran cantidad de módems que aceptan el protocolo Modbus. Algunos están específicamente diseñados para funcionar con este protocolo.

1.3.1 Principios de operación del protocolo.

En los dispositivos que se comunican usando el protocolo Modbus usan la técnica máster-esclavo, en el cual un dispositivo, en este caso el máster, inicia la negociación enviando una solicitud. El esclavo al cual se le envió la solicitud, responde al máster (Ver Figura 2). El máster también puede enviar un mensaje general para todos los esclavos en cuyo caso estos últimos no le responden al máster (Ver Figura 3).

El máster interroga a un esclavo con una única dirección en la red y espera por la respuesta de este último.

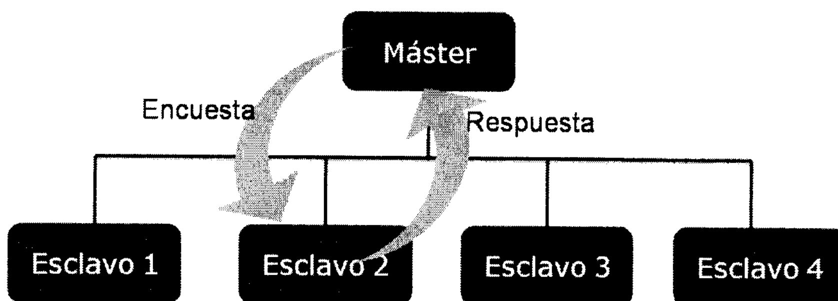

Figura 2: Encuesta respuesta.

En cuanto al segundo principio de operación del protocolo, difusión, el máster envía un mensaje a todos los esclavos presentes en la red, los cuales ejecutan el comando del mensaje sin devolver una respuesta.

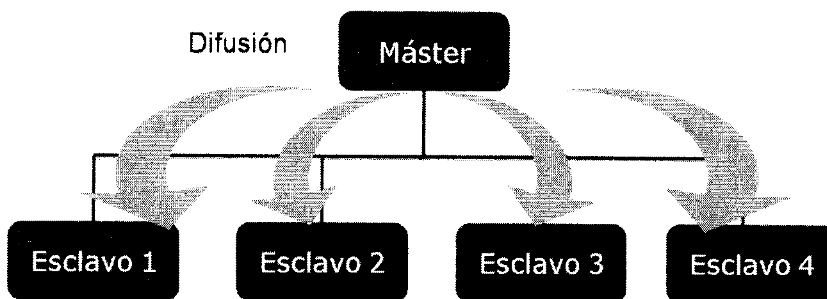

Figura 3: Difusión.

1.3.2 Características de la Encuesta.

La solicitud contiene un código que indica al esclavo adecuado que tipo de acción es solicitada. Los datos contienen información adicional la cual el esclavo necesita para ejecutar la función deseada. El campo de control de bytes le da la posibilidad al esclavo de asegurarse de la integridad del contenido de la solicitud.

1.3.3 Características de la Respuesta.

Cuando un esclavo, siguiendo una negociación normal de la comunicación, envía una respuesta, el código de la función (Función Asociada) es un eco de lo que contenía la solicitud. Los datos son coleccionados por el esclavo, por ejemplo el valor de un registro. Si ocurriera un error, el código de la función asociada es modificado para indicar que la respuesta es una respuesta de error. Los datos entonces contienen un código que permiten el reconocimiento del tipo del error. El campo de control le permite al máster la confirmación de la validez del mensaje.

Después de la recepción de una solicitud, el esclavo chequea la consistencia de la trama, si es detectado un parámetro ilegal (función asociada, dirección, valor y otros.), o si el dispositivo no es capaz de ejecutar la solicitud retorna una respuesta de error en el siguiente formato:

01: Función asociada desconocida.

02: Dirección incorrecta.

03: Valor incorrecto.

04: Estación no lista para ejecutar la solicitud.

05: Acuse de recibo, la estación ha aceptado y está procesando la solicitud.

06: La estación está procesando y no está disponible.

07: No aceptación.

Otros códigos de error pueden existir y son específicos al producto conectado a la red.

1.3.4 Formato de la trama de encuesta/respuesta.

Solicitud:			
Número del esclavo	Código de función	Información específica, asociada a la función	Palabra de control
1 byte	1 byte	n byte	2 byte
Respuesta positiva:			
Número del esclavo	Código de función	Datos recibidos	Palabra de control
1 byte	1 byte	n byte	2 byte
Respuesta negativa:			
Número del esclavo	Código de función	Código de error	Palabra de control
1 byte	1 byte (En éste caso el bit más significativo es puesto en 1)	1 byte	2 byte

Figura 4: Formato de la trama encuesta/respuesta.

1.3.5 Modbus ASCII.

En el modo ASCII (American Standard Code for Information Interchange) todos los mensajes comienzan con el carácter "dos puntos" (:), y terminan con el carácter "cambio de línea" y "retorno del carro" ("CRLF"). Los caracteres enviados en los otros campos son hexadecimales (0-9, A-F). Los dispositivos en la red continuamente monitorean el arribo del carácter ":" y cuando esto ocurre, entonces cada dispositivo decodifica el siguiente campo (Dirección) con el objetivo de discernir la dirección de destino y entonces tener en cuenta los siguientes caracteres si el esclavo es identificado. El final del mensaje es indicado por el carácter "CRLF" antecedido por los dos caracteres de control de errores que contienen el LRC (Longitudinal Reducing Check).

Inicio	Dirección	Función	Datos	LRC	Final
":" 1 caracter	2 caracteres	2 caracteres	n caracteres	2 caracteres	"CRLF" 2 caracteres

Figura 5: Representación de una trama Modbus ASCII.

1.3.6 Modbus RTU.

Este es el modo más frecuentemente usado y es más eficiente que el modo ASCII. En el modo RTU (Remote Terminal Unit) el mensaje comienza con un intervalo de silencio de al menos 3 caracteres y medio. Todos los dispositivos presentes en la red permanecen continuamente a la escucha, y decodifican el primer byte con el objetivo de determinar la dirección de destino y entonces tener en cuenta los próximos caracteres si el esclavo es identificado. Una vez que el último carácter es enviado un período de silencio de al menos tres caracteres y medio indica el final del mensaje. Entonces una nueva trama puede ser enviada.

Los caracteres son hexadecimales del tipo 0-9, A-F. Los datos contenidos en la trama tienen que contener todo el mensaje y deben ser enviados continuamente. La integridad del mensaje es indicada en el contenido del CRC (Código de Redundancia Cíclica).

Inicio	**Dirección**	**Función**	**Datos**	**CRC**	**final**
Intervalo de silencio	**1 byte**	**1 byte**	**n bytes**	**2 bytes**	**Intervalo de silencio**

Figura 6: Representación de una trama Modbus RTU.

1.3.7 Modbus TCP.

Modbus TCP está diseñado para permitir la comunicación con equipamiento industrial como PLC, computadoras, paneles de operador, motores, sensores y otros tipos de dispositivos físicos de entrada y salida que se comunican a través de la red. (3)

Un sistema de comunicación sobre Modbus TCP puede incluir diferentes tipos de dispositivos:

- Cliente y servidor Modbus TCP conectados a una red TCP/IP. (3)
- La interconexión de dispositivos como puente, router o gateway para la interconexión de una red TCP/IP y una sub-red serial donde estarían los dispositivos finales que interpretan Modbus serial. (3)

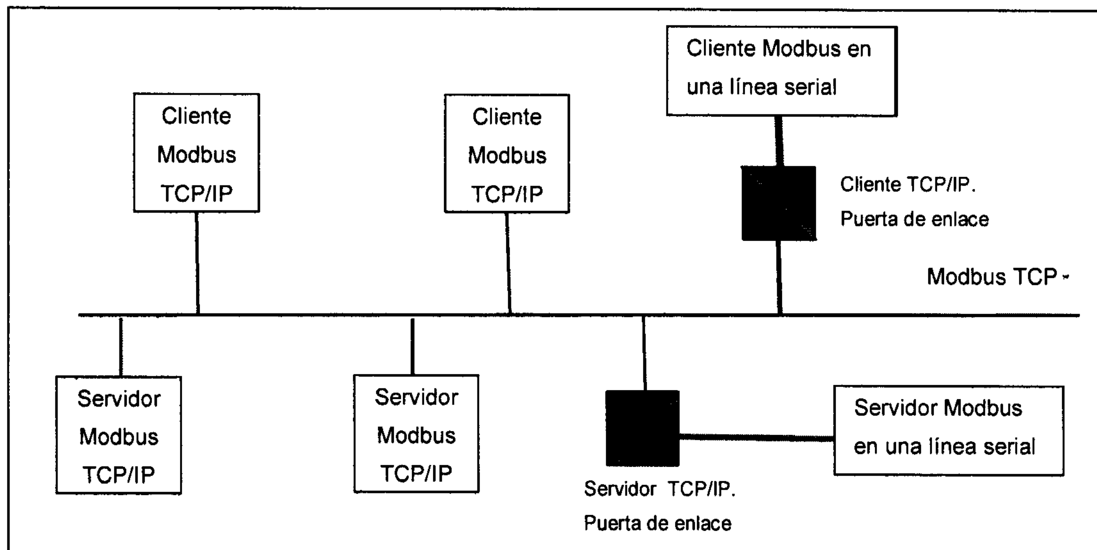

Figura 7: Representación de dispositivos comunicándose a través Modbus TCP.

Antes de seguir es necesario ver algunas definiciones de abreviaturas que se utilizarán a continuación.

- **ADU:** unidad de datos de aplicación.
- **IP:** protocolo de internet.
- **MB:** Modbus
- **MBAP:** protocolo de aplicación Modbus.
- **PDU:** Unidad de datos del protocolo.

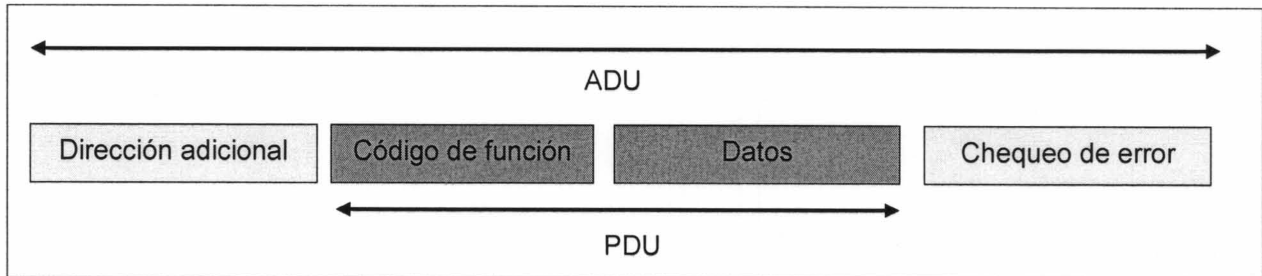

Figura 8: Trama Modbus/TCP general.

El cliente que inicializa una transacción Modbus construye un ADU Modbus el código de la función (Function code), le indica al servidor que tipo de acción realizar (4). A continuación veremos cómo se encapsula las solicitudes y respuestas sobre una red TCP/IP.

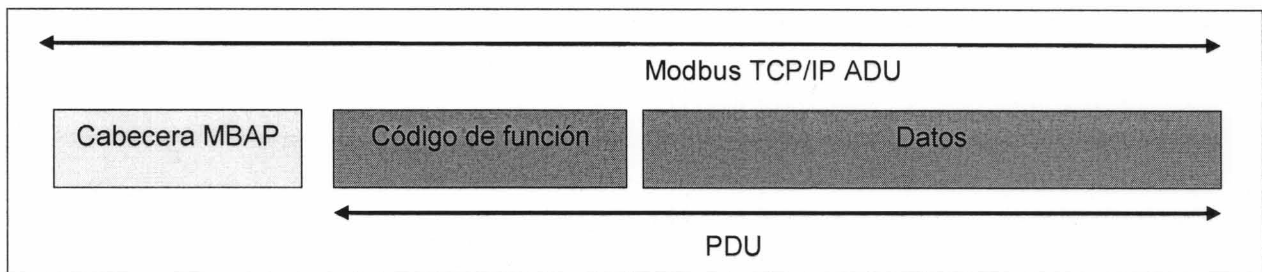

Figura 9: Encapsulación de las tramas sobre una red TCP/IP.

Campos.	Tamaño	Descripción.
Identificador de transacción.	2 Bytes.	Identificador de una trama, ya sea de solicitud o respuesta
Identificador del protocolo.	2 Bytes.	Es igual a 0 en el caso del protocolo Modbus.
Longitud.	2 Bytes.	Número de bytes que siguen a continuación.

Identificador de unidad.	1 Bytes.	Identificador del esclavo remoto a ser encuestado.

Tabla 1: Elementos que componen la cabecera MBAP.

1.3.8 Funciones asociadas.

Modbus ofrece 19 funciones diferentes. Estas están caracterizadas por una función asociada de tamaño 1 byte. No todos los dispositivos admiten todas las funciones asociadas. (5)

Código Hexadecimal.	**Descripción de la función asociada**
H01	**Leer n bits consecutivos de salidas.**
H02	**Leer n bits consecutivos de entrada.**
H03	**Leer n palabras consecutivas de salida.**
H04	**Leer n palabras consecutivas de entrada.**
H05	**Escribir 1 bit de salida.**
H06	**Escribir 1 palabra de salida.**
H07	Lectura de estado de error.
H08	Acceso a los contadores de diagnóstico.
H09	Modos de operación y carga/descarga remota.
H0A	Solicitud de reporte de operación.
H0B	Lectura del contador de eventos.
H0C	Lectura de los eventos de conexión.

HOD	Modos de operación y carga/descarga remota.
H0E	Solicitud de reporte de operación.
H0F	**Escritura de n bits de salida.**
H10	**Escritura de n palabras de salida.**
H11	Leer identificación.
H12	Modos de operación y carga/descarga remota.
H13	Restablecer esclavo después de error no resuelto.

Tabla 2: Funciones asociadas del protocolo.

1.3.9 Clasificación de los servicios en categorías.

Los servicios que brinda el estándar pueden ser clasificados en las siguientes categorías:

- Escritura o lectura de palabras o bits.
- Funciones para el diagnóstico de dispositivos.
- Funciones para el manejo de los modos de operación de dispositivos.

Las funciones principales aparecen en la tabla anterior en negrita: la palabra “0” es direccionada con la dirección “0”. La palabra “n” es direccionada con la dirección “n”. El bit “0” es direccionado con la dirección “0” y el bit n es direccionado con la dirección “n”.

1.4 Modelo cliente/servidor en Modbus TCP.

Anteriormente vimos que Modbus en su servicio de mensajería provee comunicación de tipo cliente/servidor entre dispositivos conectados a la red Ethernet mediante TCP/IP. Este modelo básicamente está basado en 4 tipos de mensajes. (6)

- solicitud Modbus: es el mensaje enviado por la red por parte del cliente para inicializar la transacción.
- confirmación Modbus: es el mensaje de solicitud recibido del lado del servidor.
- indicación Modbus: es el mensaje de respuesta a la solicitud del cliente, enviado por el servidor.
- respuesta Modbus: es el mensaje de respuesta recibido del lado del cliente.

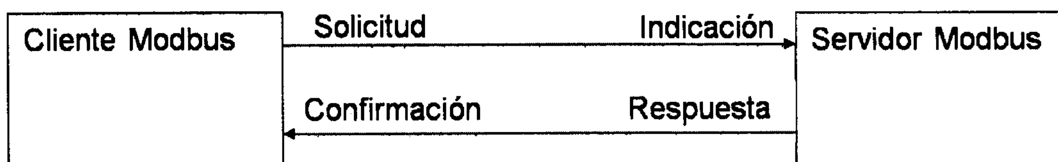

Figura 10: Modelo cliente/servidor en Modbus.

Este modelo es utilizado para el intercambio de información en tiempo real entre: (6)

- 2 aplicaciones de dispositivos.
- aplicación de dispositivo y dispositivo físico.
- aplicaciones HMI/SCADA y dispositivos físicos.
- una PC y programa de dispositivo.

1.5 Controladores Lógicos Programables (PLC).

Un autómata programable industrial (API) o Programable logic controller (PLC), es un equipo electrónico, programable en lenguaje no informático, diseñado para controlar en tiempo real y en ambiente de tipo industrial, procesos secuenciales. Es un dispositivo especial de ordenador, utilizados en la industria de

sistemas de control. Se utilizan en muchas industrias como las refinerías de petróleo, líneas de fabricación, sistemas de transporte y otros. (7)

Un PLC trabaja en base a la información recibida por los captadores y el programa lógico interno, actuando sobre los accionadores de la instalación. Poseen uno o varios protocolos de comunicación mediante el cual se puede acceder a la zona de memoria interna en la que almacenan los datos procesados y recibidos de los actuadores. Las personas suelen nombrar y clasificar estos dispositivos en dependencia del tipo de protocolo que utilice ej. PLC Modbus (8)

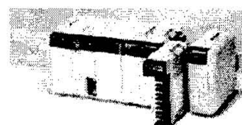

Figura 11 : Controlador lógico programable (PLC).

1.5.1 Campo de aplicación de los PLC.

El PLC por sus especiales características de diseño tiene un campo de aplicación muy extenso. La constante evolución del hardware y software amplía constantemente este campo para poder satisfacer las necesidades que se detectan en el espectro de sus posibilidades reales.

Su utilización se da fundamentalmente en aquellas instalaciones en donde es necesario un proceso de maniobra, control, señalización, por tanto, su aplicación abarca desde procesos de fabricación industriales de cualquier tipo a transformaciones industriales y control de instalaciones. (8)

Sus reducidas dimensiones, la extremada facilidad de su montaje, la posibilidad de almacenar los programas para su posterior y rápida utilización, la modificación o alteración de los mismos, hace que su eficacia se aprecie fundamentalmente en procesos en que se producen necesidades tales como: (8)

- Espacio reducido.
- Procesos de producción periódicamente cambiantes.
- Procesos secuenciales.
- Maquinaria de procesos variables.
- Instalaciones de procesos complejos y amplios.

- Chequeo de programación centralizada de las partes del proceso.

1.5.2 Funciones básicas de un PLC.

- **detección:** lectura de la señal de los captadores distribuidos por el sistema de fabricación.
- **mando:** elaborar y enviar las acciones al sistema mediante los accionadores y preaccionadores.
- **diálogo hombre máquina:** mantener un diálogo con los operarios de producción, obedeciendo sus consignas e informando del estado del proceso.
- **programación:** para introducir, elaborar y cambiar el programa de aplicación del autómata. El diálogo de programación debe permitir modificar el programa incluso con el autómata controlando la máquina.

Nuevas Funciones

- **redes de comunicación:** permiten establecer comunicación con otras partes de control. Las redes industriales permiten la comunicación y el intercambio de datos entre autómatas en tiempo real. En unos cuantos milisegundos pueden enviarse telegramas e intercambiar tablas de memoria compartida.
- **sistemas de supervisión:** permiten comunicarse con ordenadores provistos de programas de supervisión industrial. Esta comunicación se realiza por una red industrial o por medio de una simple conexión por el puerto serie del ordenador.
- **control de procesos continuos:** además de dedicarse al control de sistemas de eventos discretos los autómatas llevan incorporadas funciones que permiten el control de procesos continuos. Disponen de módulos de entrada y salida analógicas y la posibilidad de ejecutar reguladores PID que están programados en el autómata.
- **entradas-salidas distribuidas:** los módulos de entrada salida no tienen por qué estar en el armario del autómata. Pueden estar distribuidos por la instalación, se comunican con la unidad central del autómata mediante un cable de red.

- **buses de campo:** mediante un solo cable de comunicación se pueden conectar al bus captadores y accionadores, reemplazando al cableado tradicional. El autómata consulta cíclicamente el estado de los captadores y actualiza el estado de los accionadores.

1.6 Tendencias y desarrollo de la variante Modbus TCP.

El siguiente epígrafe aborda las principales tendencias y el desarrollo del protocolo de comunicación Modbus a nivel internacional y nacional.

Existen variadas organizaciones internacionales que se han destacado por el desarrollo y proliferación de actividades relacionadas con el protocolo, entre unas de ellas podemos encontrar a Modbus-IDA: grupo de usuarios y proveedores independientes de dispositivos de automatización que se destacan en:

- La participación en las actividades del estándar en internet.
- El aprendizaje y evolución del protocolo y todas sus variantes.
- El mantenimiento y la elaboración de un programa de pruebas de conformidad para asegurar una mayor interoperabilidad de dispositivos Modbus.
- Proporcionar información a los usuarios y proveedores por igual para ayudarles a tener éxito en el uso de Modbus.
- Participar en la educación y los esfuerzos promocionales incluyendo ferias comerciales, boletines informativos, páginas web, y otras actividades de divulgación.

Variado es el número de productos que a nivel internacional han sido desarrollados bajo las especificaciones del protocolo a continuación se describen algunos de los más populares.

Jamod.

Jamod es una representación de la implementación del protocolo de comunicación Modbus realizado completamente en el lenguaje de programación Java. Puede ser usado para construir aplicaciones máster y esclavo de diversas variantes: (9)

- Seriales: ASCII y RTU (en caso de aplicaciones máster solamente), BIN.
- IP: TCP y UDP.

El diseño de la librería es completamente orientado a objetos basados en abstracciones que podrían brindar un fácil entendimiento, reusabilidad y extensibilidad. Desde su versión 1.0 hasta la existente hoy en día la v1.2 se le han adicionados nuevas funcionalidades de particular interés. (9)

Utilizando Jamod se pueden crear de forma muy sencilla aplicaciones máster/esclavo al igual que especifica Modbus. Para la librería un máster es, en términos del paradigma de programación de redes, una aplicación cliente que establece una comunicación con un esclavo (servidor), enviando solicitudes a través de una conexión y esperando por una respuesta del esclavo a dicha solicitud. La forma que utiliza Jamod a la hora de realizar una transacción es totalmente síncrona.

A nivel nacional se tiene conocimiento de que algunos sistemas SCADA desarrollados, como por ejemplo EROS, presentan manejadores Modbus TCP/IP, pero de forma síncrona.

1.7 Tecnologías libres más utilizadas.

En este epígrafe se resaltan las tecnologías libres más utilizadas en la adquisición de datos, específicamente aquellas relacionadas con el acceso al medio físico mediante el cual se establecerá la comunicación con los dispositivos de campo, o sea para gestionar cómo y bajo qué condiciones serán enviado los datos por la red o por un canal serial por citar algunos. De las tecnologías descritas a continuación se determinará cual es la más adecuada para el desarrollo de la capa de acceso a datos.

1.7.1 Librería socket de C++.

Es una librería de transporte, que implementa clases en C++ bajo licencia GPL que funciona lo mismo en sistemas operativos Unix como en win32. Presenta características tales como soporte SSL, soporte para

direcciones IPv6, TCP y UDP socket, TCP encriptado, soporta además el protocolo HTTP y un avanzado nivel de tratamiento de errores. (10)

En el desarrollo de la librería fue decisión del creador, trabajar para que cada socket fuese tratado como objeto individual y así fue como nació la clase Socket. La clase Socket, en si misma, tiene todas las funciones necesarias para la resolución de direcciones, esto es muy importante ya que cuando se habla de direcciones de dispositivos se pueden referir a la dirección IP del mismo, o simplemente, a la dirección nombre del dispositivo. (10)

Aparte de algunas clases útiles en la librería, hay dos tipos especiales de clases que veremos a continuación. La clase Socket , con los diferentes IP y los protocolos de alto nivel de los que esta derivado; y la clase SocketHandler que actúa como controlador.

Un programa (cliente o servidor) consiste en una clase SocketHandler, que puede manejar una o más clases de tipo Socket. El SocketHandler mantiene la información de la lista de sockets, y llama a select() para obtener el siguiente evento (read / write / exception). También controla el progreso de las conexiones salientes.

Cuando el SocketHandler recibe un evento para un socket, ese evento es inmediatamente reportado a la clase Socket usando un método callback. Los métodos callback más básicos son OnRead(), OnWrite(), OnConnect(), OnAccept(). Aquí están el resto para completar la lista OnException(), OnDelete(), OnLine(), OnSSLInitDone(), OnRawData(), OnDetached().

La librearía consiste en diferentes clases de tipo Socket especializadas como por ejemplo TcpSocket, UdpSocket, SSLSocket. Para añadir funcionalidad, para hacer cosas tipo socket tanto de lectura como de escritura, una nueva clase debe ser creada que herede una de las clases tipo Socket.

La mayoría del tiempo, la clase SocketHandler puede manejar por si misma la mayoría de los requerimientos de programación cliente/servidor. Pero cuando los sockets tienen que interactuar, resulta más práctico crear una nueva clase derivada de SocketHandler. (10) Posee clases para el trabajo con hilos de ejecución, así como mecanismos para controlar áreas o regiones críticas entre ellos, tal es el caso de las clases **Semaphore, Mutex y Lock.**

1.7.2 Qt.

Qt es un framework escrito en C++ creado por la compañía Trolltech bajo licencias, pública y propietaria, que permite crear aplicaciones con interfaces gráficas, además de poseer un módulo para la comunicación vía TCP/IP (QtNetwork), posee un poderoso mecanismo para el trabajo con objetos, permitiendo adicionar y eliminar propiedades de los mismos en tiempo de ejecución. Qt es totalmente orientado a objetos, fácil de usar, extensible y multi-plataforma (soportada en Windows, Unix y derivados). Qt es un producto creado por la compañía Trolltech.

El módulo QtNetWork provee un conjunto de clases que facilitan la programación sobre la red, dentro de este conjunto de clases encontramos algunas que se encuentran a un alto nivel pues implementan aplicaciones a nivel de protocolos específicos de red como son el caso de QHttp y QFtp, mientras encontramos otras de más bajo nivel como QTcpSocket y QUdpSocket. (11)

Actualmente Qt va por la versión 4.4 permitiendo desarrollar más avanzadas interfaces de usuario, enriquecido con multimedia, web y de contenidos y de servicios además de utilizar las aplicaciones de escritorio a través de los sistemas operativos y dispositivos embebidos. (12)

1.7.3 Asio C++ Library.

Asio es una librería multiplataforma para la implementación de aplicaciones de red, que brinda un desarrollo consistente de un mecanismo asíncrono de flujos de entrada y salida, usando un moderno enfoque de C++. Entre la amplia gama de funcionalidades sobre la red que brinda, da soporte para la resolución de direcciones, aceptación de nuevas conexiones, socket orientados a datagrama y funcionalidades de temporizador. Su licencia está estrechamente relacionada con la licencia de Boost ya que depende de la misma para su ejecución, la librería en sí, trata de abordar los siguientes objetivos: (13)

- **portabilidad.** La librería puede ser utilizada en los sistemas operativos más usados a nivel mundial, además se le han realizado pruebas sobre los compiladores más populares sobre dichos sistemas operativos, como son el caso de:
    - 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 ésta. (13)
- **eficiencia.**
- **modelo de Berkeley sockets**. Es ampliamente entendido y aplicado en la librería.
- **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.

La librería ofrece mecanismos síncronos y asíncronos. A partir de la versión 1.35.0 de Boost se incluye la versión 1.0.0 de Asio entre su conjunto de librerías.

1.8 Herramientas de desarrollo empleadas.

Para el desarrollo de la capa de acceso se necesitan herramientas como los entornos de desarrollo integrados (IDE). Estos IDE 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.

Dentro de los lenguajes de programación, se escogió para el desarrollo de la capa de protocolo, el C++, lenguaje de programación con el cual se han implementado los entregables de la línea de manejadores, e incluso el resto de los módulos del proyecto.

El lenguaje de programación C++ surge en 1980 de la mano de Bjarne Stroustrup, fue diseñado con el objetivo de adicionarle al lenguaje C nuevas características, como es el caso de las clases y funciones virtuales, tipos de datos genéricos, la posibilidad de poder declarar variables en cualquier punto del programa y sobre todo un auténtico motor de objetos con herencia múltiple que permite combinar la programación imperativa de C, con la programación orientada a objetos. Gran importancia en la evolución de C++ fue la incorporación de la librería STL años más tardes, incorporando clases contenedoras y algoritmos genéricos que hacen al C++ una potencia única entre los lenguajes de alto nivel. Este lenguaje de programación es multiplataforma, o sea se puede usar en diferentes sistemas operativos aunque requiera de diferentes compiladores en cada unos de ellos, ejemplo:

- Windows: MingW (GCC para Windows)
- Linux (u otros UNIX): gcc

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

1.9 Metodología de desarrollo de software.

En cuanto a la metodología de desarrollo de software empleada, se utilizó Rational Unified Process (RUP).

Las principales características de RUP son:

- centrado en la arquitectura
- dirigido por casos de uso
- iterativo e incremental

RUP define cuatro fases para el desarrollo del software: inicio, elaboración, construcción y transición. Además define un grupo de flujos de trabajo básicos: Modelamiento del Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba y Despliegue; incluye flujos de trabajo de apoyo como Gestión del Cambio y la Configuración, Gestión del Proyecto y Ambiente.

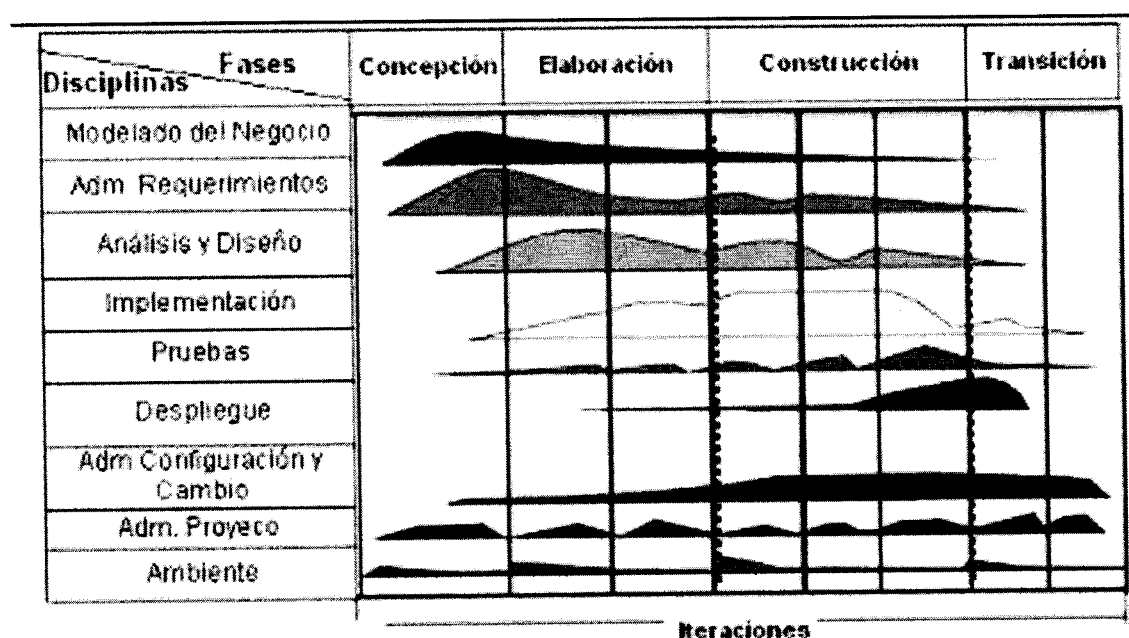

Figura 12: Fases y flujos de trabajos de RUP.

RUP define como lenguaje de modelado al UML (Unified Modeling Language) utilizado para especificar, visualizar, construir y documentar artefactos de un sistema de software. Es bueno aclarar que el UML es

solo un lenguaje de modelado, no una metodología, ni proceso como muchos confunden. Entre los diagramas que propone UML para modelar un sistema nos encontramos:

- **Diagramas de estructura estática:** Describen las propiedades estructurales del sistema.
    - Diagrama de clases: Conjunto de clases, interfaces y colaboraciones; así como sus colaboraciones.
    - Diagrama de objetos: Conjunto de objetos y sus relaciones.
    - Diagrama de casos de uso: Conjunto de casos de uso y actores y sus relaciones.
- **Diagramas de comportamiento:**
    - Diagramas de interacción (secuencia y colaboración): Objetos y sus relaciones, incluyendo los mensajes que pueden ser enviados entre ellos.
    - Diagrama de estados: Muestra una máquina de estado que consta de estados, transiciones, eventos y actividades.
    - Diagrama de actividad: Es un tipo especial de diagrama de estados que muestra el flujo de actividades dentro de un sistema.
- **Diagramas de implementación:**
    - Diagrama de componentes: Organización y las dependencias entre un conjunto de componentes.
    - Diagrama de despliegue: Configuración de nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos.

2 Capitulo 2: Características del sistema.

Introducción.

Este capítulo refleja la solución que se propone para resolver el problema científico identificado. Se realiza un análisis de cómo se ejecutan actualmente los procesos, las causas que originan la situación problemática. Se describen los procesos que serán objeto de automatización. Se incluyen algunos aspectos del diseño que sirven de punto de partida para la implementación de la capa de acceso a datos.

2.1 Flujo actual de los procesos.

El manejador Modbus existente en el módulo de manejadores del proyecto SCADA es totalmente síncrono. A través de dicho manejador se realizan todas las operaciones de lectura y escritura de variables que existen en los dispositivos Modbus, que se encuentran en el campo de acción del sistema. El hilo de ejecución que invoca unas de las operaciones mencionadas anteriormente se bloquea hasta tanto no culmine la operación, bien sea porque haya ocurrido un error, porque todo se haya realizado correctamente o por haber expirado por time-out.

Cada solicitud realizada al manejador implica:

1. Generar tramas Modbus de acuerdo a la solicitud realizada.
2. Enviar la trama al dispositivo Modbus seleccionado.
3. Esperar la respuesta del dispositivo. (Tr)
4. Decodificar la trama recibida de acuerdo al protocolo.
5. Retornar la respuesta recibida.

En la acción número tres, es donde se bloquea el hilo de ejecución, esperando por la respuesta del dispositivo un tiempo determinado, el cual denominaremos Tr (tiempo de respuesta).

Existen dos variantes fundamentales para la implementación usando el modelo sincrónico visto anteriormente, una de ellas es cuando se crea un hilo de ejecución para atender las solicitudes que se realizan a varios dispositivos (ver Figura 14) y la otra cuando se crea un hilo de ejecución por cada dispositivo a ser encuestado (ver Figura 15).

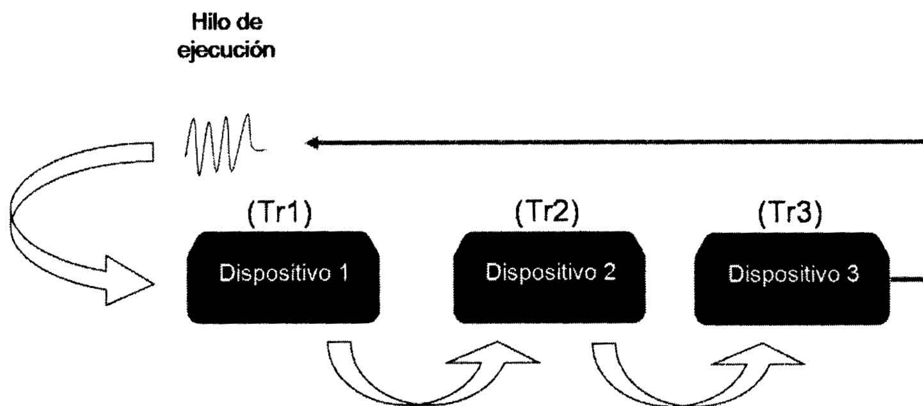

Figura 13: Variante de un hilo de ejecución que atienda todos los dispositivos.

En el primer caso el hilo atiende las solicitudes de todos los dispositivos de manera secuencial, o sea primeramente atiende el dispositivo número uno, luego el dos y así sucesivamente. La variante planteada es sumamente ineficiente, ya que, la incorporación de nuevos dispositivos a ser atendidos retrasa considerablemente la ejecución del ciclo de atenciones a los dispositivos, debido a la latencia de encuestar a cada uno.

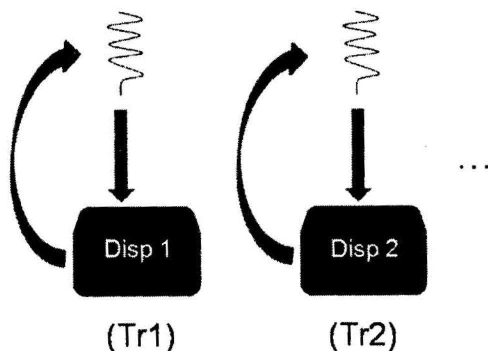

Figura 14: Variante de un hilo por cada dispositivo a ser atendido.

En este segundo caso se resuelve el inconveniente de la primera variante, pues un dispositivo es atendido por un hilo independiente. Esta variante es fácil de programar porque se sigue la secuencia operacional y eficiente para un número pequeño de dispositivos. El problema recae cuando el número de dispositivos a atender sea numeroso, se necesitarían numerosos hilos de ejecución. En las operaciones de cómputo, el cambio de contexto entre hilos de ejecución es una operación costosa, los microprocesadores para simular la multitarea necesitan ejecutar en pequeños intervalos de tiempo, pequeñas porciones de cada una de las tareas. En el proceso de cambio de tareas conocido como cambio de contexto tienen que realizar operaciones muy costosas como son el caso de:

1. Salvar el estado del programa que se estaba ejecutando.
2. Seleccionar otro programa para ejecutar.
3. Restaurar el estado del programa seleccionado.
4. Ejecutar el programa seleccionado.

2.2 Objeto de automatización.

Según la especificación del estándar Modbus, existen alrededor de 19 funciones que pueden ser enviadas como solicitud a un dispositivo Modbus, de ellas solo se automatizarán las que se consideran más importantes en el proceso de intercambio de datos, éstas serian:

- H01: Leer n bits consecutivos de salidas.
- H02: Leer n bits consecutivos de entrada.
- H03: Leer n palabras consecutivas de salida.
- H04: Leer n palabras consecutivas de entrada.
- H05: Escribir 1 bit de salida.
- H06: Escribir 1 palabra de salida.
- H0F: Escritura de n bits de salida.
- H10: Escritura de n palabras de salida.

2.3 Propuesta del sistema.

Para ilustrar la solución que se propone usaremos un ejemplo simple de la vida real. Supongamos que un profesor desea evaluar a un grupo de estudiantes. Si se utiliza un modelo sincrónico para el método EVALUAR la secuencia de pasos a seguir sería la siguiente: el profesor le entrega la prueba al primer estudiante, espera a que el estudiante termine la prueba y la entregue (en este proceso el profesor queda bloqueado, sin poder realizar otra acción hasta tanto el estudiante no le entregue el examen) y finalmente cuando recibe la prueba terminada la califica y solo entonces procede a realizar el mismo procedimiento con el segundo estudiante y así sucesivamente. Obviamente se requeriría mucho tiempo para evaluar a todos los estudiantes. El modelo asíncrono que es el más natural se implementaría de la siguiente manera: el profesor le entregaría la prueba al primer estudiante y sin esperar respuesta alguna pasaría a entregarle la prueba al segundo estudiante y así hasta que todos los estudiantes hayan recibido su prueba. El profesor ahora espera porque algún estudiante le avise que terminó el examen (aviso asíncrono). Cuando se recibe este aviso se le retira la prueba al estudiante y se procede a calificarla. Si varios estudiantes entregan al mismo tiempo o en momentos de tiempo cercanos sus trabajos ya terminados se colocan en una lista de trabajos pendientes por calificar y se van calificando en dependencia del orden de esa lista.

Es fácil ver que este segundo enfoque es más productivo. También puede apreciarse un patrón general que se manifiesta cuando se pasa de un modelo síncrono a uno asíncrono. Cuando queremos pasar de un modelo síncrono a otro asíncrono debemos detectar el punto de bloqueo del método síncrono y dividirlo en tres partes, lo que ocurre antes del bloqueo (invocación), la operación de entrada/salida que genera el bloqueo y la parte del método que se ejecuta después de completada la operación de entrada salida (callback), además es bueno aclarar que se debe convertir la operación de entrada/salida en una operación asíncrona de entrada/salida.

Basándose en lo mencionado anteriormente se llevó el modelo sincrónico del manejador Modbus, al modelo asincrónico, arrojando los siguientes resultados.

Se crearon varias etapas en la ejecución del mecanismo, una de ellas es la denominada ***Invocación***. En la cual se realizarían las dos primeras acciones del modelo sincrónico visto anteriormente; generar la trama Modbus en dependencia de la solicitud realizada y el envío de la misma al dispositivo seleccionado.

Una segunda etapa donde se ejecutarían operaciones de entrada y salida asíncronas soportadas por la mayoría de los sistemas operativos. Como tercera y última etapa denominada callback, donde se ejecutarían las dos últimas acciones del modelo sincrónico; decodificar la trama recibida de acuerdo al protocolo y retornar la respuesta recibida. La siguiente figura (Figura 15) muestra como queda estructurado las acciones a realizar en el modelo asincrónico propuesto.

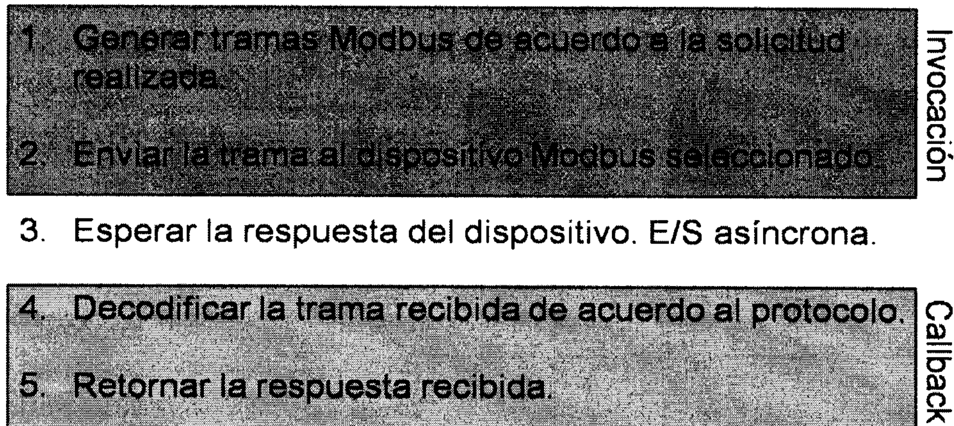

Figura 15: Estructuración de las acciones a realizar en el modelo asincrónico propuesto.

De forma general el sistema propuesto tiene las siguientes características.

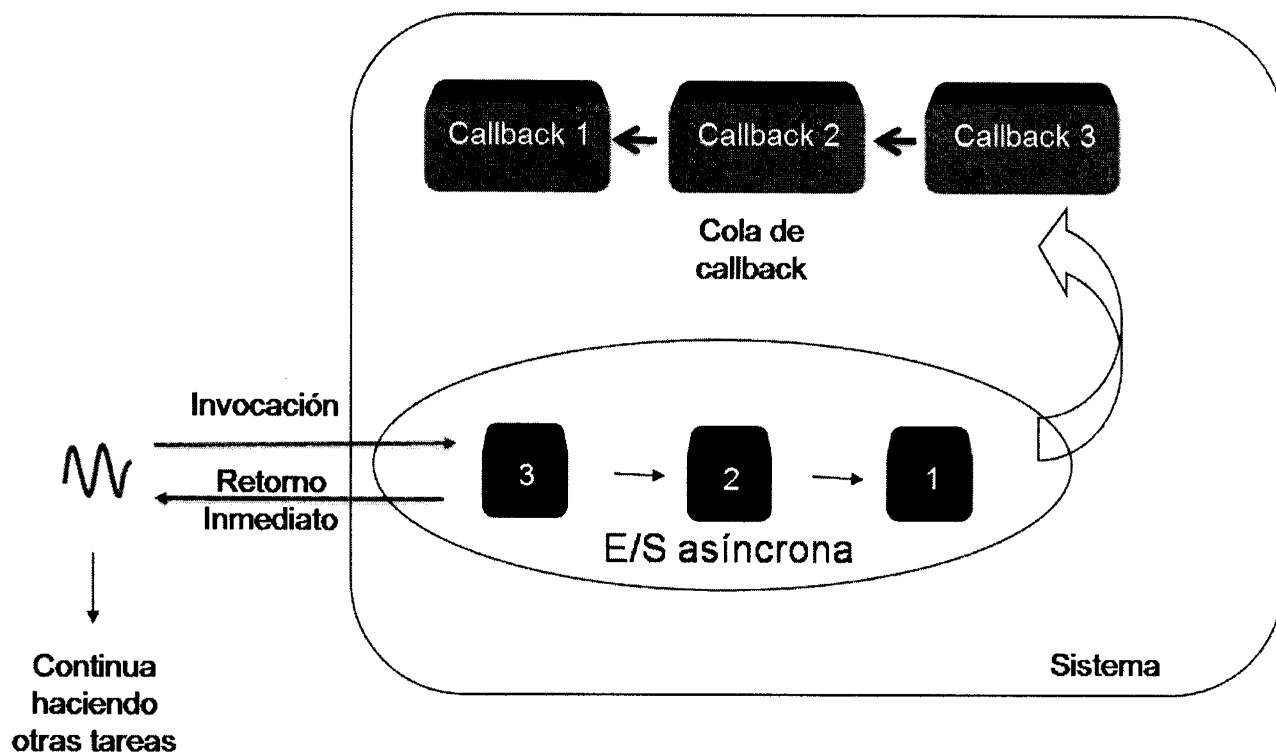

Figura 16: Representación del modelo asíncrono propuesto.

Un hilo de ejecución realiza una invocación al sistema, ya vimos anteriormente, en la Figura 15, que en la invocación es donde se realizan las acciones de generar la trama Modbus y enviarla al dispositivo, luego de realizada la invocación, se retorna inmediatamente, de esta forma el hilo que realizó la solicitud puede continuar realizando otras tareas, incluso, puede seguir realizando invocaciones al sistema. Luego dentro del sistema se ejecutarían de fondo las solicitudes (ver en la Figura 16 los objetos enumerados en rojo), a través de las operaciones de entrada/salida asíncronas y donde se espera por la respuesta del dispositivo. Al concluir la realización de una de éstas solicitudes se crea un callback (ver en la Figura 16 los objetos enumerados en verde), listos para ser ejecutados y de esta forma avisar que se culminó o completó la solicitud realizada.

El desarrollo de la propuesta de solución analizada anteriormente, plantea varios aspectos a resolver:

1. La implementación de los métodos para codificar y decodificar las operaciones Modbus.
2. La selección de una librería de socket asíncrona para implementar el modelo.
3. El contexto de ejecución de los callback.
4. La implementación de los callback en el modelo.

En cuanto a la implementación de los métodos para codificar y decodificar las operaciones Modbus. Es necesaria la implementación de una clase que se encargue de ensamblar y desensamblar mensajes Modbus y que la misma de soporte para las funciones formarían parte del objeto de automatización [2.2].

En cuanto a la selección de una librería de socket asíncrona para implementar el modelo. Se utilizará Asio C++ Library [1.7.3], ya que como se analizó en el primer capitulo, la librería presente una gran portabilidad, escalabilidad, eficiencia, modelo de Berkeley Sockets, es fácil de usar, posee bases para una mayor abstracción, además de ser una librería probada y utilizada en la implementación de los manejadores del proyecto SCADA.

Refiriéndose al contexto de ejecución de los callback, ya que los mismos no podrían ser ejecutados por el hilo que realiza la invocación, ni por el hilo que ejecutaría de fondo las operaciones de entrada/salida asíncronas. Se propone el uso del patrón thread pool (estanque o piscina de hilos en español).

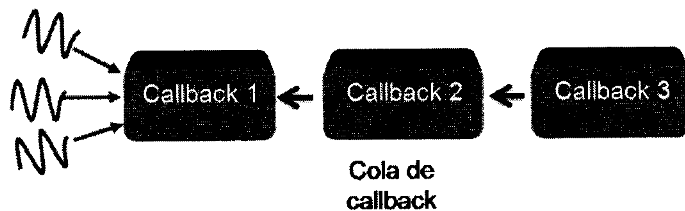

Figura 17: Representación del patrón thread pool, en la ejecución de la cola de callback presente en la propuesta de solución.

El patrón thread pool se caracteriza por tener:

- Un conjunto de hilos previamente creados.
- El número de hilos puede ser configurado para obtener el mejor rendimiento.

- Los hilos que terminan la ejecución de una tarea, se incorporan y se mantienen listos para la ejecución de nuevas tareas.

Con respecto al cuarto y último aspecto, la implementación de los callback en el modelo. Debido a que C++ no presenta delegados, se propuso la utilización de las súper clases para la implementación del modelo.

2.4 Reglas del Negocio.

El módulo de clases representará a los dispositivos Modbus TCP. Cuando se crean los dispositivos se debe especificar el modo de operación permitido a usar (modo ASYNC para las operaciones de lectura y escritura asíncronas, SYNC para las síncronas). En caso de no especificarse el modo de operación se tomará el asíncrono por defecto.

Los dispositivos poseen varias propiedades, sino se establecen las mismas se tomarán valores por defecto. A continuación se muestra una tabla con los valores que se tomaran por defecto para cada propiedad.

Descripción de la propiedad	Valor por defecto
Identificador del esclavo del dispositivo a encuestar.	1
Tiempo de espera para la conexión.	1000 milisegundos.
Número de reintentos en caso de error (para modelo síncrono).	3 reintentos.
Intervalo entre reintentos.	1000 milisegundos.
Tiempo de espera en las lecturas.	1000 milisegundos.
Dirección base de los 4 tipos de variables (Coils, Input, Inputs Registers, Holding Registers).	0

Cantidad máxima de variables de cada tipo en el dispositivo.	9999
Puerto de comunicación.	502

Tabla 3: Tabla de valores por defecto según la propiedad.

Antes de llamar a las funciones de escritura y lectura se debe especificar la dirección IP (IPv4) del dispositivo físico a encuestar.

2.5 Modelo del dominio.

A continuación se representa y se describen los conceptos fundamentales del dominio del sistema.

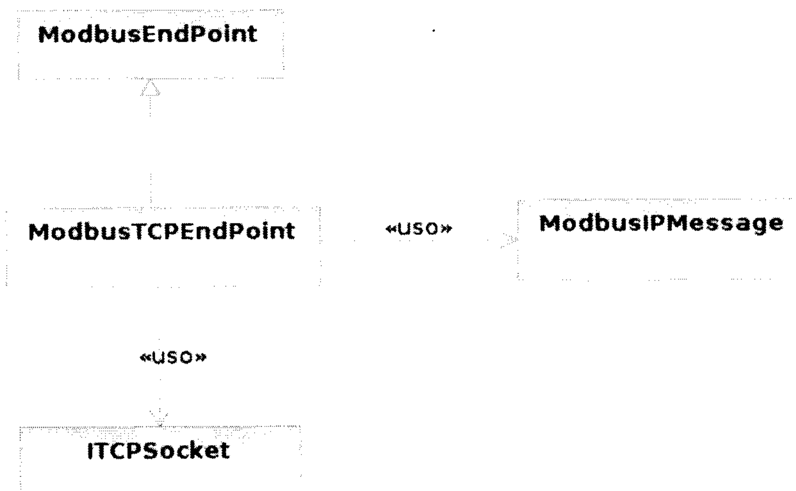

Figura 18: Modelo del dominio.

2.5.1 Glosario de términos del dominio.

ModbusEndPoint: define características generales que podrían ser utilizadas para cualquier representación de dispositivo Modbus. Dentro de estas características se incluyen las propiedades del transporte que se utiliza. Sirve de interfaz genérica para las operaciones de lectura y escritura de variables síncronas y asíncronas de cualquier tipo de dispositivo. Es básicamente quien chequea que se cumplan

todas las restricciones que especifica el estándar Modbus en cuanto a las cantidades de variables permitidas en cada transacción.

ModbusTCPEndPoint: es básicamente la representación de un dispositivo Modbus TCP. Posee además de las características heredadas de ModbusEndPoint, características específicas de los dispositivos TCP y también del transporte que éste utiliza, tal es el caso de la dirección del host con el cual se desea comunicar vía TCP/IP y el puerto. Utilizan como medio de confección e interpretación de tramas al **ModbusTCPMessage** y para el envío y recepción de las tramas a través de un medio físico, el concepto de **ITCPSocket**.

ModbusTCPMessage: el concepto encapsula todo lo relacionado con la confección e interpretación de un mensaje o trama Modbus TCP, las cuales son usadas para ser enviadas a través de la red o recibidas del mismo medio. Cuando se le envía una solicitud a la capa de protocolo, se transforma dicha solicitud a como especifica el estándar que se deben mandar las tramas para que el dispositivo físico entienda la petición que se le realizó, luego cuando se recibe un mensaje de parte del dispositivo físico en cuestión, es el encargado de descifrar lo que el dispositivo quiso transmitir.

ITCPSocket: es la representación de un socket, a través de él son enviadas y recibidas las tramas o mensajes sobre la red hacia el dispositivo físico en cuestión. Da soporte para que los envíos y las recepciones se puedan realizar de dos formas, síncronas y asíncronas.

2.6 Especificación de requerimientos del software.

En el presente epígrafe se exponen los requerimientos funcionales y no funcionales del sistema.

2.6.1 Requerimientos funcionales.

1- **Lectura de variables del dispositivo**. Se deben brindar funcionalidades para la lectura de variables para los 4 tipos que establece el protocolo Modbus (Coils, Inputs, Holding Registers y Registers). Es necesario que estas funcionalidades se puedan ejecutar de varios modos, síncrono y asíncrono, en dependencia de las exigencias y necesidades del que utilice la capa de acceso a datos.

2- **Escritura de variables del dispositivo**. La capa de acceso a datos debe soportar funcionalidades de escritura para las variables de tipo Coils y Holding Registers, según especifica Modbus. Éstas funcionalidades deben poder ejecutarse de forma síncrona y asíncrona en dependencia de las exigencias y necesidades del que utilice la capa de acceso a datos.
3- **Configuración de dispositivos y redes.** La capa de acceso a datos debe ofrecer funciones para obtener y modificar la configuración de sus parámetros y de los parámetros de los dispositivos y redes.
4- **Variables con información de diagnóstico.** Se deben brindar funcionalidades que permitan obtener información de diagnóstico con relación a las operaciones realizadas a cierto dispositivo, lo cual puede ayudar a determinar el estado del mismo y el comportamiento que ha presentado en el transcurso del tiempo.
5- **Mensajes de Error.** La capa de acceso a datos debe proporcionar una función que traduzca los códigos de error específicos que devuelven sus funciones a mensajes textuales.

2.6.2 Requerimientos no funcionales.

- **Usabilidad.**
- **Confiabilidad.**
- **Portabilidad.** La librería debe poder ser ejecutada por los sistemas operativos más utilizados, así como en la mayoría de los compiladores de C++.
- **La capa debe de tener un 100% de disponibilidad.**
- **Especificación de los recursos utilizados.** Se debe especificar, la cantidad de recursos del Sistema Operativo (% del CPU, memoria, hilos, objetos del núcleo) que consume en dependencia del número de redes y dispositivos que atiende.

2.7 Modelo de casos de usos del sistema.

2.7.1 Actor del sistema.

Actores.	Justificación.
Programador	Es el que hará uso y se beneficiará con las funcionalidades que brinda el módulo de clases.

Tabla 4: Actor del sistema.

2.7.2 Casos de uso del sistema.

Principales casos de uso.

1. CU Configurar dispositivo.
2. CU Leer variables de forma síncrona.
3. CU Leer variables de forma asíncrona.
4. CU Escribir datos de forma síncrona.
5. CU Escribir datos de forma asíncrona.

Casos de uso secundarios.

6. CU Obtener información de diagnóstico.
7. CU Interpretar código de error.

2.7.3 Diagrama de caso de usos del sistema.

En la siguiente figura (*Figura 15*) se puede apreciar el diagrama de caso de uso del sistema.

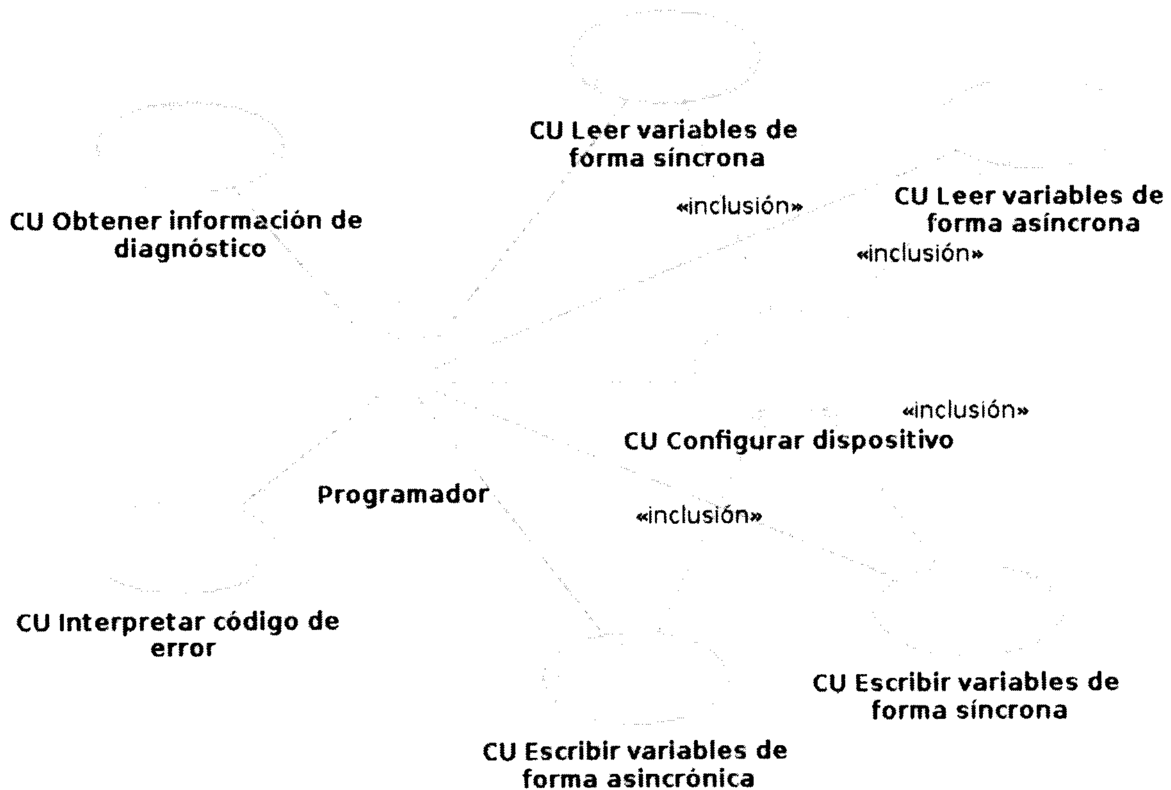

Figura 19: Diagrama de CUS.

2.7.4 Especificación de los CU en formato expandido.

A continuación se describen los casos de usos más importantes.

CU Configurar dispositivo.

Caso de uso

Nombre	CU Configurar dispositivo.
Propósito	Obtener una representación de un dispositivo listo para el intercambio de información.
Actores	Programador.
Resumen	El caso de uso se inicializa cuando el actor decide configurar un dispositivo para leer o escribir datos. En caso de que la representación del dispositivo necesitada por el actor ya este creada solo tendría que configurar su propiedades en dependencia de sus necesidades.
Referencias	RF3.
Precondiciones	
Acción del actor	**Respuesta del sistema**
1 – Si se ha creado un dispositivo. Decide establecer las propiedades del dispositivo.	2 - Establece las propiedades del dispositivo en cuestión.
Flujo alternativo	
Acción del actor	**Respuesta del sistema**
1 – En caso de que no. Decide crear una representación de un dispositivo.	2 – Crea una representación del dispositivo.
3 - Decide establecer las propiedades del dispositivo.	4 - Establece las propiedades del dispositivo especificadas por el actor.
Prioridad	Critico

Tabla 5: Descripción del CU configurar dispositivo.

CU Leer variables de forma síncrona.

Caso de uso		
Nombre	CU Leer variables de forma sincrónica.	
Propósito	Obtener datos del dispositivo bloqueando el hilo de ejecución hasta tanto responda el dispositivo o expire el time-out.	
Actores	Programador.	
Resumen	El caso de uso se inicializa cuando el actor decide leer sincrónicamente valores del dispositivo. Al llamar a la función de lectura de forma síncrona el hilo de ejecución se bloquea hasta tanto no se termine de completar la operación. Se podrán leer variables de tipo Coils, Inputs, Registers y Holding Registers.	
Referencias	RF1.	
Precondiciones	CU Configurar dispositivo.	
Acción del actor		Respuesta del sistema
1 - Manda a leer datos del dispositivo.		2 - Chequea que los parámetros para la solicitud son válidos.
		3 – Si son válidos, crea la solicitud de lectura o trama correspondiendo a los parámetros especificados por el actor.
		4 – Ejecuta la transacción síncrona, (entiéndase el proceso de envío de solicitud y espera de una

	confirmación por parte del dispositivo).
	5 – Si no ocurre errores en la transacción, se interpreta el mensaje enviado por el dispositivo.
	6 – En caso de no existir errores de trama, se copian los datos solicitados para que sean devueltos al actor.
Flujo alternativo	
Acción del actor	**Respuesta del sistema**
	3 – No son válidos, devuelve un código de error indicando que hubo problemas con los parámetros para la realización de la operación.
	5 – En caso de error, se intenta realizar la transacción una vez más, y así sucesivamente hasta agotar el número máximo de reintentos en caso de error.
	6 - En caso de error, se intenta realizar la transacción una vez más, y así sucesivamente hasta agotar el número máximo de reintentos en caso de error.
Prioridad	Critico

Tabla 6: Descripción del CU Leer variables de forma sincrónica.

CU Leer variables de forma asincrónica.

Caso de uso		
Nombre	CU Leer variables de forma asincrónica.	
Propósito	Obtener datos del dispositivo sin necesidad de bloquear el hilo de ejecución.	
Actores	Programador.	
Resumen	El caso de uso se inicializa cuando el actor decide leer asincrónicamente valores del dispositivo. Al llamar a la función de lectura asíncrona el hilo de ejecución retorna inmediatamente. Se podrán leer variables de tipo Coils, Inputs, Registers y Holding Registers	
Referencias	RF1.	
Precondiciones	CU Configurar dispositivo.	
Acción del actor		Respuesta del sistema
1 - Solicita leer datos del dispositivo asíncronamente.		2 - Chequea que los parámetros para la solicitud son válidos.
		3 – Si son válidos, crea la solicitud de lectura o trama correspondiendo a los parámetros especificados por el actor.
		4 – Ejecuta la transacción asíncrona, (entiéndase el proceso de envío de solicitud).
		5 – Devuelve un código indicando que la solicitud se realizo correctamente.

Flujo alternativo		
Acción del actor		Respuesta del sistema
		3 – No son válidos, devuelve un código de error indicando que hubo problemas con los parámetros establecidos para la realización de la operación.
Prioridad	Critico	

Tabla 7: Descripción del CU Leer variables de forma asincrónica.

CU Escribir variables de forma sincrónica.

Caso de uso		
Nombre	CU Escribir variables de forma sincrónica.	
Propósito	Escribir datos al dispositivo sincrónicamente.	
Actores	Programador.	
Resumen	El caso de uso se inicializa cuando el actor decide escribir sincrónicamente valores al dispositivo. Al llamar a la función de escritura sincrónica el hilo de ejecución se bloquea hasta que no se complete la operación. Se podrán escribir variables de tipo Coils, y Holding Registers.	
Referencias	RF2.	
Precondiciones	CU Configurar dispositivo.	
Acción del actor		Respuesta del sistema
1 - Solicita escribir datos al dispositivo de		2 - Chequea que los parámetros para la solicitud

forma sincrónica.	son válidos.
	3 – Si son válidos, crea la solicitud de escritura o trama, en dependencia de los parámetros especificados por el actor.
	4 – Ejecuta la transacción síncrona, (entiéndase el proceso de envío de solicitud y espera de la confirmación del dispositivo).
	5 – Si no ocurre errores en la transacción, se interpreta el mensaje enviado por el dispositivo.
	6 – Se verifican que confirmación que envió el dispositivo coincide con la solicitud que se realizó.
Flujo alternativo	
Acción del actor	**Respuesta del sistema**
	3 – No son válidos, devuelve un código de error indicando que hubo problemas con los parámetros establecidos para la realización de la operación.
	5 – En caso de error, se intenta realizar la transacción una vez más, y así sucesivamente hasta agotar el número máximo de reintentos en caso de error.

	6 - En caso de error, se intenta realizar la transacción una vez más, y así sucesivamente hasta agotar el número máximo de reintentos en caso de error.
Prioridad	Critico

Tabla 8: Descripción del CU Escribir variables de forma sincrónica.

CU Escribir variables de forma asincrónica.

Caso de uso	
Nombre	CU Escribir variables de forma asincrónica.
Propósito	Escribir datos al dispositivo asincrónicamente.
Actores	Programador.
Resumen	El caso de uso se inicializa cuando el actor decide escribir asincrónicamente valores al dispositivo. Al llamar a la función de escritura asíncrona el hilo de ejecución retorna inmediatamente. Se podrán escribir variables de tipo Coils, y Holding Registers.
Referencias	RF2.
Precondiciones	CU Configurar dispositivo.
Acción del actor	Respuesta del sistema
1 - Solicita escribir datos al dispositivo de forma asincrónica.	2 - Chequea que los parámetros para la solicitud son válidos.

	3 – Si son válidos, crea la solicitud de escritura o trama, en dependencia de los parámetros especificados por el actor.
	4 – Ejecuta la transacción asíncrona, (entiéndase el proceso de envío de solicitud).
Flujo alternativo	
Acción del actor	**Respuesta del sistema**
	3 – No son válidos, devuelve un código de error indicando que hubo problemas con los parámetros estbalecidos para la realización de la operación.
Prioridad	Critico

Tabla 9: Descripción del CU Escribir variables de forma asincrónica.

3 Capitulo 3: Diseño e implementación del sistema.

Introducción.

La primera parte del capítulo encierra el diagrama de clase del sistema, propuesto como resultado del refinamiento de las etapas anteriores. Posteriormente se muestran los diagramas de secuencia, elaborados a partir de los casos de uso que intervienen en el desarrollo del módulo. Se presentan además los componentes físicos, que se traducen en los ficheros .h y .cpp correspondientes a la implementación en C++. Además se elabora el diagrama de despliegue del sistema.

3.1 Diagrama de clases del diseño.

Figura 20: Diagrama de clases del diseño.

3.2 Descripción de las clases del diseño.

Descripción de la clase Protocol.

Nombre: Protocol	
Tipo de clase: Entidad	
Atributo	**Tipo**
Buffer	LPByteArray
bufferSize	DWord
msgSize	DWord
headerSize	DWord
timeOut	DWord
connectionTimeOut	DWord
State	State
Para cada responsabilidad:	
Nombre:	Protocol()
Descripción:	Constructor por defecto.
Nombre:	getBuffer()
Descripción:	Función que devuelve un puntero al buffer actual que se esta utilizando.
Nombre:	getBufferSize()
Descripción:	Devuelve el tamaño del buffer en bytes.
Nombre:	getMsgSize()
Descripción:	Devuelve el tamaño del mensaje en bytes.

Nombre:	getHeaderSize()
Descripción:	Devuelve el tamaño del encabezado del mensaje en bytes.
Nombre:	getTimeOut()
Descripción:	Devuelve el tiempo de espera para el recibo de los mensajes, en milisegundos.
Nombre:	getConnectionTimeOut()
Descripción:	Devuelve el tiempo de espera para el establecimiento de la conexión, en milisegundos.
Nombre:	getState()
Descripción:	Devuelve el estado de la máquina de estado que usa el transporte.
Nombre:	setBuffer(LPByteArray buffer)
Descripción:	Establece el buffer que se utilizará para los mensajes.
Nombre:	setBufferSize(DWord bufferSize)
Descripción:	Establece el tamaño del buffer, en bytes.
Nombre:	setMsgSize(DWord msgSize)
Descripción:	Establece el tamaño del mensaje, en bytes.
Nombre:	setHeaderSize(DWord headerSize)
Descripción:	Establece el tamaño del encabezado de los mensajes, en bytes.
Nombre:	setTimeOut(DWord timeOut)
Descripción:	Establece el tiempo máximo de espera para la recepción de los mensajes, en milisegundos.

Nombre:	setState(State newState)
Descripción:	Establece el estado de la máquina de estado del transporte.
Nombre:	setConnectionTimeOut(DWord timeOut)
Descripción:	Establece el tiempo máximo de espera para el establecimiento de la conexión.
Nombre:	OnError (DWord errorCode)
Descripción:	Evento que se dispara cuando ocurre un error en la transacción síncrona.
Nombre:	OnConnected ()
Descripción:	Evento que se dispara cuando se establece la conexión de forma síncrona.
Nombre:	ConnectedHandler(DWord error, IHandlerParameter* handler)
Descripción:	Evento que se dispara cuando se estableció la conexión con el dispositivo en la transacción asíncrona.
Nombre:	OnReadyHeader ()
Descripción:	Evento que se dispara cuando se recibe el encabezado del mensaje del dispositivo, en la transacción síncrona.
Nombre:	OnReadyMsg ()
Descripción:	Evento que se dispara cuando se recibe el resto del mensaje enviado por el dispositivo, en la transacción síncrona.
Nombre:	ReadHandler(LPByteArray data, DWord size, DWord error, IHandlerParameter* handler)
Descripción:	Evento que se dispara cuando se recibe un mensaje del dispositivo en la transacción asíncrona.

Nombre:	OnDisconnect ()
Descripción:	Evento que se dispara cuando se desconecta del dispositivo.
Nombre:	OnMsgWritten ()
Descripción:	Evento que se dispara cuando se envía el mensaje al dispositivo
Nombre:	WriteHandler(DWord error, IHandlerParameter* handler)
Descripción:	Evento que se dispara cuando se ha enviado un mensaje al dispositivo en la transacción asíncrona.
Nombre:	OnTimedOut(IHandlerParameter* handler)
Descripción:	Evento que se dispara en caso que exista una expiración del time-out en la transacción asíncrona.

Tabla 10: Descripción de la clase Protocol.

Descripción de la clase ModbusEndPoint.

Nombre: ModbusEndPoint	
Tipo de clase: Entidad	
Atributo	**Tipo**
readBuffer	LPByteArray
writeBuffer	LPByteArray
myHeaderSize	DWord
Message	ModbusMessage*

Error	DWord
responseSize	DWord
parametersMutex	QMutex
useSingleWrite	bool
isWaitProcessingRequestError	bool
endPointID	DWord
Type	ModelType
Sniffer	ISniffer*
snifferEnabled	bool
transportAtrib	TCPTransport*
slaveID	Byte
numberOfRetries	DWord
retriesInterval	DWord
baseAddress[]	DWord
maxRegisters[]	DWord
maxRegistersInTransaction[][]	DWord
dignosticInfo	EndPointDiagnosticInfo
Para cada responsabilidad:	
Nombre:	ModbusEndPoint (ISniffer* snifferParam)

Descripción:	Constructor por defecto. En el se inicializa el sniffer con los valores del parámetro pasado.
Nombre:	~ModbusEndPoint()
Descripción:	Destructor por defecto.
Nombre:	useThread(void* thread)
Descripción:	Cambia el contesto al dispositivo, de esta forma se podría usar el mismo endpoint en otro hilo de ejecución.
Nombre:	syncReadRegisters(ModbusRegisterTypes registerType, DWord reference, AbstractVector& data)
Descripción:	Lee variables del dispositivo de forma síncrona, retorna un código de error en caso de que falle la operación, 0 en caso contrario.
Nombre:	asyncReadRegisters(ModbusHandler* handler)
Descripción:	Lee variables del dispositivo de forma asíncrona.
Nombre:	syncWriteRegisters (ModbusRegisterTypes registerType, DWord reference, AbstractVector& data)
Descripción:	Escribe valores al dispositivo de forma síncrona.
Nombre:	asyncWriteRegisters(ModbusHandler* handler)
Descripción:	Escribe valores al dispositivo de forma asíncrona.
Nombre:	getMaxRegisters(ModbusRegisterTypes registerType)
Descripción:	Devuelve la cantidad máxima de variables del tipo especificado.
Nombre:	setMaxRegisters(ModbusRegisterTypes registerType, DWord value)

Descripción:	Establece la cantidad máxima de variables del tipo especificado.
Nombre:	getMaxRegistersInTransaction(ModbusRegisterTypes registerType, ModbusOperations operation)
Descripción:	Devuelve la cantidad de variables permitidas en cada operación.
Nombre:	setMaxRegistersInTransaction(ModbusRegisterTypes registerType, ModbusOperations operation, DWord value)
Descripción:	Establece la cantidad máxima de variables permitidas en cada transacción.
Nombre:	setBaseAddress(ModbusRegisterTypes registerType, DWord newBaseAddress)
Descripción:	Establece la dirección base del bloque de variables especificado.
Nombre:	getBaseAddress(ModbusRegisterTypes registerType)
Descripción:	Devuelve la dirección base del bloque de variables especificado.
Nombre:	setNumberOfRetries(DWord newRetriesNumber)
Descripción:	Establece la cantidad de reintentos en caso de error, en las transacciones.
Nombre:	getNumberOfRetries()
Descripción:	Devuelve la cantidad de reintentos en caso de error.
Nombre:	setRetriesInterval(DWord newRetriesInterval)
Descripción:	Establece el intervalo de tiempo de espera entre reintentos en caso de error.
Nombre:	getRetriesInterval()
Descripción:	Devuelve el intervalo de tiempo de espera entre reintentos en caso de error.
Nombre:	setSlaveID(Byte newSlaveID)

Descripción:	Establece el identificador de esclavo a encuestar.
Nombre:	getSlaveID()
Descripción:	Devuelve el identificador del esclavo encuestado.
Nombre:	setEndPointID(DWord newEndPointID)
Descripción:	Establece el identificador de la representación del dispositivo creado.
Nombre:	getEndPointID()
Descripción:	Devuelve el identificador de la representación del dispositivo creado
Nombre:	setUseSingleWrite(bool value)
Descripción:	Establece el valor de la propiedad que indica si se usará cuando sea necesaria la operación de escritura simple.
Nombre:	getUseSingleWrite()
Descripción:	Devuelve el valor de la propiedad que indica si se usará la operación de escritura simple cuando sea necesario.
Nombre:	getErrorString(DWord errorCode)
Descripción:	Devuelve un string con la información del código de error pasado por parámetro interpretado.
Nombre:	getDiagnosticInfo()
Descripción:	Devuelve una estructura con la información de diagnóstico generada.
Nombre:	resetDiagnosticInfo()
Descripción:	Restablece los valores de la estructura que posee la información de diagnóstico.

Nombre:	Checker(ModbusRegisterTypes registerType, ModbusOperations operation, DWord reference, DWord count)
Descripción:	Funcionalidad auxiliar que chequea si los parámetros para la realización de las operaciones de lectura y escritura, tanto síncronas como asíncronas son correctos.
Nombre:	enableSniffer(bool enable)
Descripción:	Habilita o deshabilita las funcionalidades del sniffer en dependencia del valor del parámetro. Si es true queda habilitado, deshabilitado en caso contrario.
Nombre:	getTransport()
Descripción:	Devuelve un apuntador a la instancia del transporte utilizado.
Nombre:	setTransport(TCPTransport* transport)
Descripción:	Establece la instancia del transporte a utilizar.
Nombre:	getModelType()
Descripción:	Devuelve el modelo de operación con el cual se inicializó el dispositivo.

Tabla 11: Descripción de la clase ModbusEndPoint.

Descripción de la clase ModbusIPEndPoint.

Nombre: ModbusIPEndPoint	
Tipo de clase: Entidad	
Atributo	**Tipo**
hostName[]	char

Port	DWord
Para cada responsabilidad:	
Nombre:	ModbusIPEndPoint (ISniffer* snifferParam)
Descripción:	Constructor por defecto. En el se inicializa el sniffer con los valores del parámetro pasado.
Nombre:	~ModbusIPEndPoint()
Descripción:	Destructor por defecto.
Nombre:	setHost(const LPStr& hostName)
Descripción:	Establece el host del dispositivo a ser encuestado.
Nombre:	getHost()
Descripción:	Devuelve el host del dispositivo encuestado.
Nombre:	setPort(Word newPort)
Descripción:	Establece el puerto mediante el cual se establecerá la comunicación.
Nombre:	getPort()
Descripción:	Devuelve el puerto de comunicación.

Tabla 12: Descripción de la clase ModbusIPEndPoint.

Descripción de la clase ModbusTCPEndPoint.

Nombre: ModbusTCPEndPoint
Tipo de clase: Entidad

Atributo		Tipo
Para cada responsabilidad:		
Nombre:	ModbusTCPEndPoint (ISniffer* snifferParam, ModelType type)	
Descripción:	Constructor por defecto. En el se inicializa el sniffer con los valores del parámetro pasado y el modo de operación (síncrono o asíncrono) para las operaciones de lectura o escritura.	
Nombre:	~ModbusTCPEndPoint()	
Descripción:	Destructor por defecto.	

Tabla 13: Descripción de la clase ModbusTCPEndPoint.

Descripción de la clase ModbusMessage.

Nombre: ModbusMessage	
Tipo de clase:	
Atributo	**Tipo**
slaveID	Byte
currentPos	DWord
readBuffer	LPByteArray
writeBuffer	LPByteArray
High	DWord
Low	DWord

Para cada responsabilidad:	
Nombre:	ModbusMessage (LPByteArray writeBuffer, LPByteArray readBuffer, Byte slaveID)
Descripción:	Constructor por defecto. En él se inicializan los buffers que serán utilizados en el ensamblado y desensamblado de los mensajes, además del identificador del esclavo en cuestión.
Nombre:	~ModbusMessage ()
Descripción:	Destructor por defecto.
Nombre:	assembleRead(ModbusRegisterTypes registerType, Word reference, Word quantity)
Descripción:	Ensambla todas las tramas para la solicitud de lectura. Retorna la cantidad de bytes que conforma la trama.
Nombre:	assembleSingleWrite(ModbusRegisterTypes registerType, Word reference, Word value)
Descripción:	Ensambla todas las tramas de escritura simple. Retorna la cantidad de bytes que conforma la trama.
Nombre:	assembleWrite(ModbusRegisterTypes registerType, Word reference, AbstractVector& data)
Descripción:	Ensambla todas las tramas de escritura múltiple. Retorna cantidad de byte que conforma la trama.
Nombre:	disassembleRead(ModbusRegisterTypes registerType, AbstractVector& data)
Descripción:	Desensambla todas las tramas de lectura. Retorna un posible error de lectura o de dispositivo.

Nombre:	disassembleWrite(ModbusRegisterTypes registerType, AssociatedOperations writeOperation, Word& reference, Word& value)
Descripción:	Desensambla todas las tramas de escritura. Retorna un posible error de escritura o de dispositivo.
Nombre:	isErrorResponse()
Descripción:	Chequea si la trama de recibida es errónea o no.
Nombre:	isWaitProcessingRequestError()
Descripción:	Chequea si el dispositivo se encuentra procesando la solicitud enviada.
Nombre:	setSlaveID(const Byte newSlaveID)
Descripción:	Establece el identificador del dispositivo con el que nos conectaremos.
Nombre:	getSlaveID()
Descripción:	Devuelve el identificador del esclavo en cuestión.
Nombre:	writeByte (const Byte value)
Descripción:	Utilidad que escribe un byte en el buffer de escritura.
Nombre:	writeWord (const Word value)
Descripción:	Utilidad que escribe una palabra (2 bytes) al buffer de escritura.
Nombre:	writeVectorData(const ModbusRegisterTypes registerType, AbstractVector& vector)
Descripción:	Escribe los datos de un vector al buffer de escritura.
Nombre:	readByte()

Descripción:	Lee un byte del buffer de lectura.
Nombre:	readWord()
Descripción:	Devuelve una palabra (2 bytes) del buffer de lectura.
Nombre:	readVectorData(const ModbusRegisterTypes registerType, AbstractVector& vector)
Descripción:	Devuelve los datos del buffer de lectura en un vector.
Nombre:	writeHeader()
Descripción:	Funcionalidad abstracta. Escribe la cabecera de una trama en el buffer de escritura.
Nombre:	writeErrorCheck()
Descripción:	Funcionalidad abstracta. Escribe los datos de chequeo de errores en el buffer de escritura.
Nombre:	readHeader()
Descripción:	Funcionalidad abstracta que lee los datos de la cabecera de los mensajes.
Nombre:	readErrorCheck(const DWord bodySize)
Descripción:	Devuelve los datos para el chequeo de errores del buffer de lectura.

Tabla 14: Descripción de la clase ModbusMessage.

Descripción de la clase ModbusIPMessage.

Nombre: ModbusIPMessage

Tipo de clase:		
Atributo		**Tipo**
transactionID		Word
byteLength		DWord
Para cada responsabilidad:		
Nombre:	ModbusIPMessage (LPByteArray writeBuffer, LPByteArray readBuffer, Byte slaveID)	
Descripción:	Constructor por defecto. En el se inicializan los buffers que serán utilizados en el ensamblado y desensamblado de los mensajes, además del identificador del esclavo en cuestión.	
Nombre:	~ModbusIPMessage ()	
Descripción:	Destructor por defecto.	
Nombre:	getFrameSize()	
Descripción:	Devuelve el tamaño de la trama leída.	
Nombre:	getRequestTransactionId()	
Descripción:	Devuelve el identificador de la solicitud en transacción.	
Nombre:	getResponseTransactionId()	
Descripción:	Devuelve el identificador de la respuesta en la transacción.	

Tabla 15: Descripción de la clase ModbusIPMessage.

Descripción de la clase TCPTransport.

<table>
<tr><td colspan="2">Nombre: TCPTransport</td></tr>
<tr><td colspan="2">Tipo de clase:</td></tr>
<tr><td>Atributo</td><td>Tipo</td></tr>
<tr><td>Sniffer</td><td>ISniffer*</td></tr>
<tr><td>Protocol</td><td>Protocol*</td></tr>
<tr><td>Provider</td><td>ITransportProvider*</td></tr>
<tr><td>Transport</td><td>ITCPSocket*</td></tr>
<tr><td>msgSize</td><td>DWord</td></tr>
<tr><td colspan="2">Para cada responsabilidad:</td></tr>
<tr><td>Nombre:</td><td>TCPTransport(ISniffer* snifferParam, ModelType type)</td></tr>
<tr><td>Descripción:</td><td>Constructor por defecto. En el se inicializan los parámetros del sniffer y el modelo de operación a utilizar.</td></tr>
<tr><td>Nombre:</td><td>~TCPTransport ()</td></tr>
<tr><td>Descripción:</td><td>Destructor por defecto.</td></tr>
<tr><td>Nombre:</td><td>setProtocol (Protocol* protocol)</td></tr>
<tr><td>Descripción:</td><td>Establece el protocolo que usara el transporte.</td></tr>
<tr><td>Nombre:</td><td>getProtocol ()</td></tr>
<tr><td>Descripción:</td><td>Devuelve el protocolo usado en el transporte.</td></tr>
<tr><td>Nombre:</td><td>runSyncTransaction (State initialState)</td></tr>
</table>

Descripción:	Ejecuta una transacción síncrona en dependencia del estado pasado por parámetro.
Nombre:	runAsyncTransaction(IHandlerParameter* parameter)
Descripción:	Inicializa una transacción asíncrona, en dependencia del estado pasado por parámetro.
Nombre:	getSniffer()
Descripción:	Devuelve el sniffer utilizado por el transporte.
Nombre:	isConnected()
Descripción:	Devuelve verdadero, si se esta conectado con el dispositivo.
Nombre:	disconnect()
Descripción:	Se desconecta del dispositivo.
Nombre:	setHost(char* hostName)
Descripción:	Estable el host del dispositivo con el cual se establecerá la comunicación.
Nombre:	getHost()
Descripción:	Devuelve el host del dispositivo en cuestión.
Nombre:	setPort(unsigned short port)
Descripción:	Establece el puerto de comunicación.
Nombre:	getPort()
Descripción:	Devuelve el valor del puerto de comunicación.
Nombre:	onTimedOut(IHandlerParameter*)

Descripción:	Handler ejecutado cuando expira una operación (conexión, lectura) por time-out. Este handler es ejecutado desde la librería de transporte implementada.
Nombre:	readHandler(unsigned char* data, unsigned long bytesRead, unsigned long error, IHandlerParameter*)
Descripción:	Handler ejecutado después de haber leído datos del socket de forma asíncrona. Este handler es ejecutado desde la librería de transporte implementada.
Nombre:	writeHandler(unsigned long bytesWrite, unsigned long error, IHandlerParameter*)
Descripción:	Handler ejecutado después de haber escribido datos al socket de forma asíncrona. Este handler es ejecutado desde la librería de transporte implementada.
Nombre:	connectHandler(unsigned long error, IHandlerParameter*)
Descripción:	Handler ejecutado después de haberse establecido la conexión. Este handler es ejecutado desde la librería de transporte implementada.

Tabla 16: Descripción de la clase TCPTransport.

Descripción de la clase ModbusHandler.

Nombre: ModbusHandler	
Tipo de clase:	
Atributo	**Tipo**
data	AbstractVector*
registerType	ModbusRegisterTypes

reference	DWord
Para cada responsabilidad:	
Nombre:	ModbusHandler ()
Descripción:	Constructor por defecto.
Nombre:	~ ModbusHandler ()
Descripción:	Destructor por defecto.
Nombre:	setData(AbstractVector& data)
Descripción:	Establece el vector que se usará en las operaciones de lectura o escritura asíncronas. En caso de una lectura es donde se le depositan los datos al usuario y en la escritura es donde el usuario almacena los valores a escribir. En ambos casos el vector debe ser construido por el usuario
Nombre:	AbstractVector& getData()
Descripción:	Devuelve el vector asociado.
Nombre:	setRegisterType(ModbusRegisterTypes newRegisterType)
Descripción:	Establece el tipo de variables que se leerán o escribirán de forma asíncrona (Coils, Inputs, Inputs Registers, Holding Registers).
Nombre:	ModbusRegisterTypes getRegisterType()
Descripción:	Devuelve el tipo de variable asociado a la operación.
Nombre:	setReference(DWord newReference)
Descripción:	Establece la referencia a partir de la cual se leerán o escribirán valores en el dispositivo físico.

Nombre:	DWord getReference()
Descripción:	Devuelve la referencia asociada a la operación.

Tabla 17: Descripción de la clase ModbusHandler.

Descripción de la interfaz ITransportProvider.

Nombre: ITransportProvider	
Tipo de clase:	
Atributo	**Tipo**
_instance	ITransportProvider*
Para cada responsabilidad:	
Nombre:	instance()
Descripción:	A través de la funcionalidad se accede a la única instancia que se crea de la propia clase.
Nombre:	~ITransportProvider()
Descripción:	Destructor por defecto.
Nombre:	setThreadCount(unsigned int amount)
Descripción:	Establece la cantidad de hilos que ejecutaran las funciones del transporte.
Nombre:	getThreadCount()
Descripción:	Devuelve la cantidad de hilos que ejecutan las funciones del transporte.
Nombre:	getTCPTransport(Type type)

Descripción:	Devuelve una apuntador a una instancia creada de ITCPSocket, el parámetro indica el modelo de operación (síncrono o asíncrono).
Nombre:	getAsyncTimer()
Descripción:	Devuelve un apuntador de una instancia creada de IAsyncTimer.
Nombre:	getStrand()
Descripción:	Devuelve un apuntador de una instancia creada de IStrand.
Nombre:	getAcceptor()
Descripción:	Devuelve un apuntador de una instancia creada de IAcceptor.
Nombre:	deleteObject(IObject* object)
Descripción:	Destruye una instancia dentro de la librería.
Nombre:	stop()
Descripción:	Detiene la ejecución de cualquier operación dentro de la librería.
Nombre:	getErrorInformation(unsigned long error)
Descripción:	Devuelve una cadena de caracteres con la descripción de un error dado dentro de la librería.

Tabla 18: Descripción de la clase ITransportProvider.

Descripción de la clase ITCPSocket

Nombre: ITCPSocket		
Tipo de clase:		

Atributo		Tipo
Para cada responsabilidad:		
Nombre:	setHost(char* hostName)	
Descripción:	Establece el host del dispositivo con el cual se iniciará la comunicación.	
Nombre:	getHost()	
Descripción:	Devuelve el host utilizado.	
Nombre:	setPort(unsigned short port)	
Descripción:	Establece el puerto de comunicación.	
Nombre:	getPort()	
Descripción:	Devuelve el puerto utilizado.	
Nombre:	isConnected()	
Descripción:	Retorna verdadero en caso de que exista la conexión con el dispositivo.	
Nombre:	disconnect()	
Descripción:	Desconecta el socket del dispositivo.	
Nombre:	sync_connect(unsigned long timeout)	
Descripción:	Establece una conexión síncrona con el dispositivo.	
Nombre:	asyn_connect(unsigned long timeout, ITCPSocketHandler* handler, IHandlerParameter* parameter)	
Descripción:	Establece una conexión con el dispositivo de forma asíncrona, la función retorna inmediatamente, cuando se completa la operación es ejecutado el evento	

	connectHandler del ITCPSocketHandler pasado por parámetro.
Nombre:	sync_read(unsigned char* buffer, unsigned long& size, unsigned long timeout)
Descripción:	Lee datos del socket de forma síncrona.
Nombre:	async_read(unsigned char* buffer, unsigned long size, unsigned long timeOut, ITCPSocketHandler* handler, IHandlerParameter* parameter)
Descripción:	Lee datos del socket de forma asíncrona, la función retorna inmediatamente, cuando se completa la operación es ejecutado el evento readHandler del ITCPSocketHandler pasado por parámetro.
Nombre:	sync_write(unsigned char* buffer, unsigned long& size)
Descripción:	Escribe datos al socket de forma síncrona.
Nombre:	async_write(unsigned char* buffer, unsigned long size, ITCPSocketHandler* handler, IHandlerParameter* parameter)
Descripción:	Escribe datos al socket de forma asíncrona, la función retorna inmediatamente, cuando se completa la operación es ejecutado el evento writeHandler del ITCPSocketHandler pasado por parámetro.
Nombre:	available()
Descripción:	Retorna la cantidad de bytes disponibles para lectura.
Nombre:	cancel()
Descripción:	Cancela las operaciones del socket.

Tabla 19: Descripción de la clase ITCPSocket.

Descripción de la clase ITransportHandler.

Nombre: ITransportHandler	
Tipo de clase:	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	onTimedOut(IHandlerParameter* parameter)
Descripción:	Evento que es invocado cuando expira por time-out las operaciones de lectura o conexión asíncronas del ITCPSocket. El IHandlerParameter* pasado por parámetro es el mismo que se le pasó a las funciones de lectura o escritura asíncronas cuando fueron llamadas.
Nombre:	readHandler(unsigned char* data, unsigned long bytesRead, unsigned long error, IHandlerParameter* parameter)
Descripción:	Evento que se llama cuando se concluye la operación de lectura asíncrona, ya sea por algún error o porque terminó satisfactoriamente. Son devueltos como parámetros el buffer donde se encuentran los datos leídos, la cantidad de bytes leídos, un posible error, y el mismo parámetro adicional que se le pasó a la función de lectura.
Nombre:	writeHandler(unsigned long bytesWrite, unsigned long error, IHandlerParameter*)
Descripción:	Evento que se llama cuando se concluye la operación de escritura asíncrona, ya sea por algún error o porque terminó satisfactoriamente. Son devueltos como parámetros la cantidad de bytes escritos, un posible error, y el mismo parámetro adicional que se le paso a la función de escritura.
Nombre:	connectHandler(unsigned long error, IHandlerParameter*)

Descripción:	Evento que se llama cuando se concluye la operación de conexión asíncrona, ya sea por algún error o porque terminó satisfactoriamente. Son devueltos como parámetros, un posible error, y el mismo parámetro adicional que se le pasó a la función de conexión.

Tabla 20: Descripción de la clase ITransportHandler.

Descripción de la clase IAcceptor.

Nombre: IAcceptor	
Tipo de clase:	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	async_accept(IAcceptorHandler* handler)
Descripción:	Acepta conexiones entrantes, la función retorna inmediatamente, cuando se acepta una conexión es invocado el evento acceptHandler del IAcceptorHandler pasado por parámetro.
Nombre:	initAcceptor(unsigned long newPort)
Descripción:	Inicializa el acceptor con un puerto determinado, mediante el cual se pondrá a escuchar y aceptar nuevas conexiones.

Tabla 21: Descripción de la clase IAcceptor.

Descripción de la clase IAsyncTimer.

Nombre: IAsyncTimer

Tipo de clase:		
Atributo		**Tipo**
Para cada responsabilidad:		
Nombre:	expires_from_now(ITimerHandler* handler, unsigned int milliseconds)	
Descripción:	La función cuando expira el tiempo especificado por el parámetro "milliseconds" ejecuta el evento expiresHandler de la clase ITimerHandler. La función retorna inmediatamente.	
Nombre:	cancel()	
Descripción:	Cancela las operaciones de la clase.	

Tabla 22: Descripción de la clase IAsyncTimer.

Descripción de la clase IStrand.

Nombre: IStrand		
Tipo de clase:		
Atributo		**Tipo**
Para cada responsabilidad:		
Nombre:	post(IStrandHandler* handler)	
Descripción:	Solicita la ejecución del handler dado y retorna inmediatamente.	
Nombre:	dispatch(IStrandHandler* handler)	
Descripción:	Solicita la invocación del handler dado.	

Tabla 23: Descripción de la clase IStrand.

3.3 Diagramas de secuencia.

A continuación se muestran los diagramas de secuencia construidos, a través de los casos de uso anteriormente especificados y desarrollados. Se mostrarán los diagramas de secuencia de los principales casos de uso.

Diagrama de secuencia del CU Configurar dispositivo.

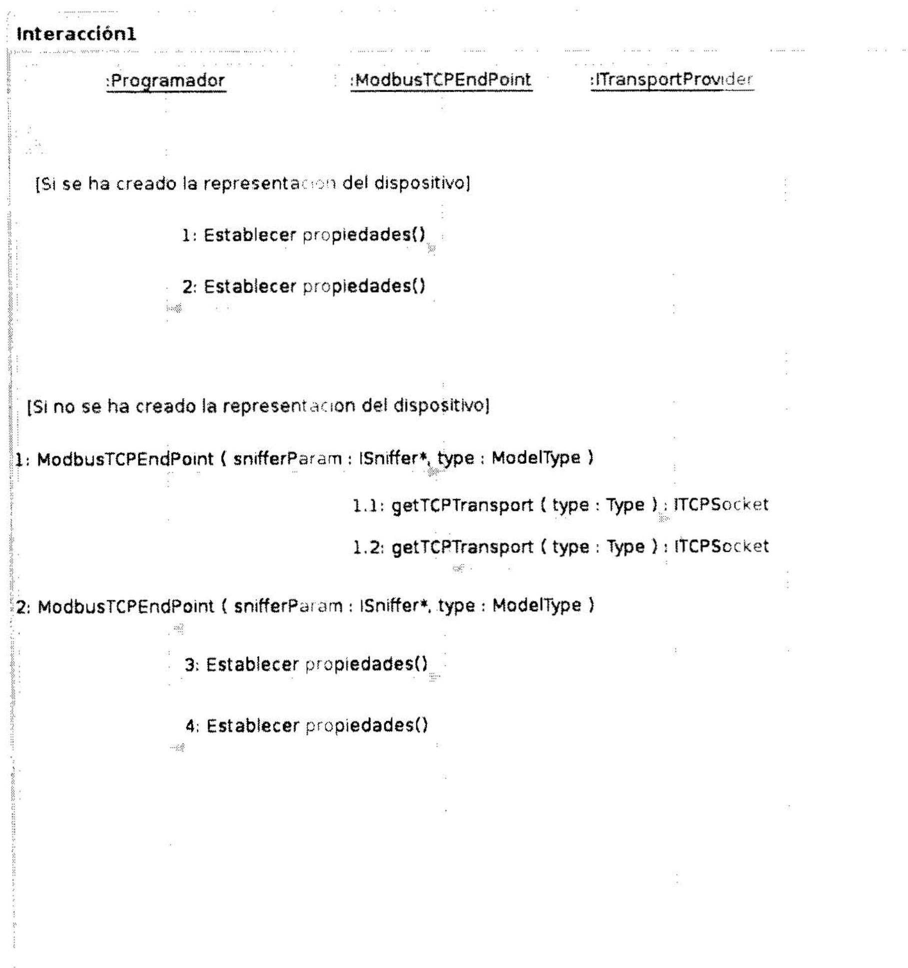

Figura 21: Diagrama de secuencia del CU Configurar dispositivo.

Diagrama de secuencia del CU Leer variables de forma síncrona.

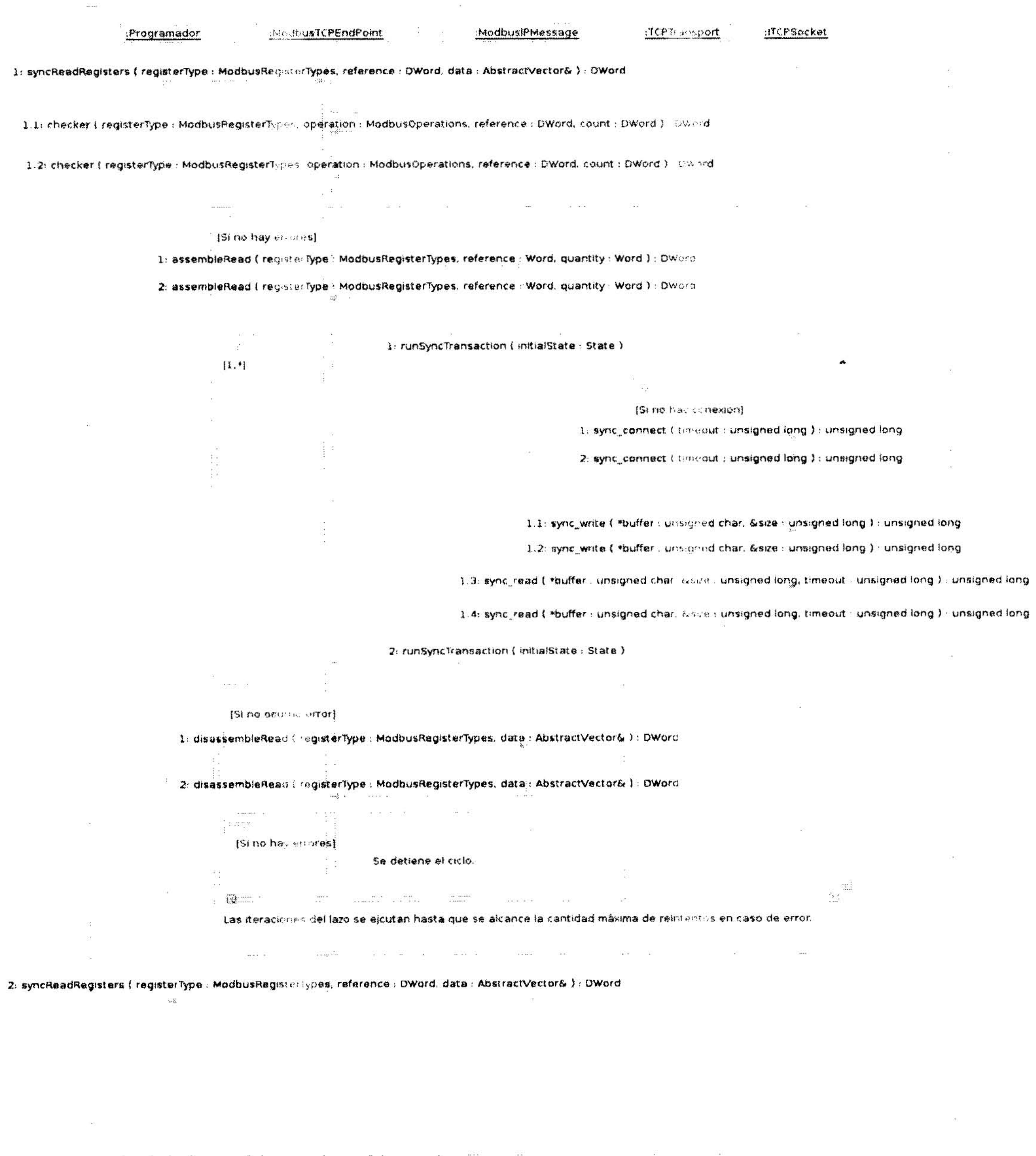

Figura 22: Diagrama de secuencia del CU Leer variables de forma síncrona.

Diagrama de secuencia del CU Leer variables de forma asíncrona.

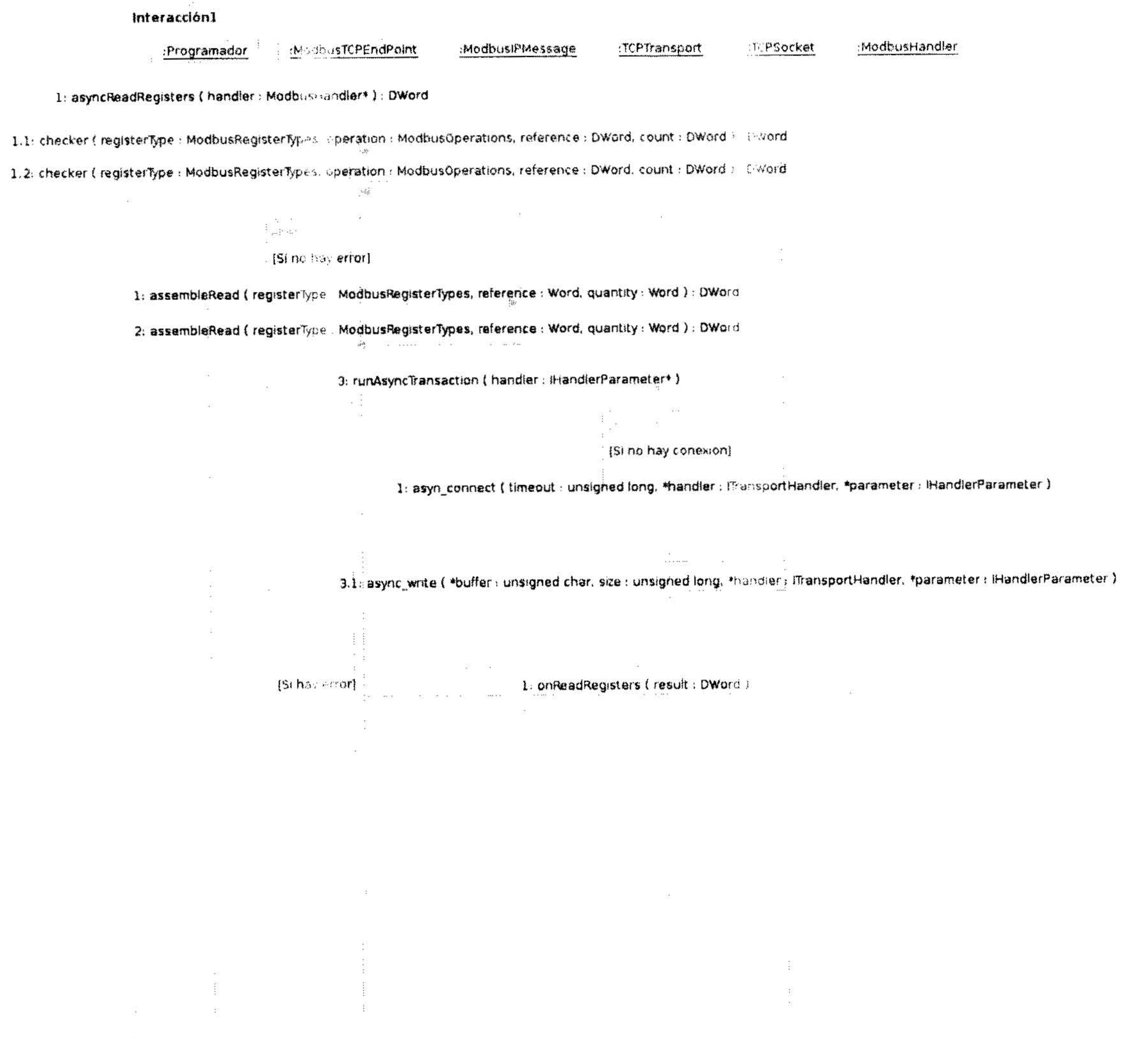

Figura 23: Diagrama de secuencia del CU Leer variables de forma asíncrona.

Diagrama de secuencia del CU Escribir variables de forma síncrona.

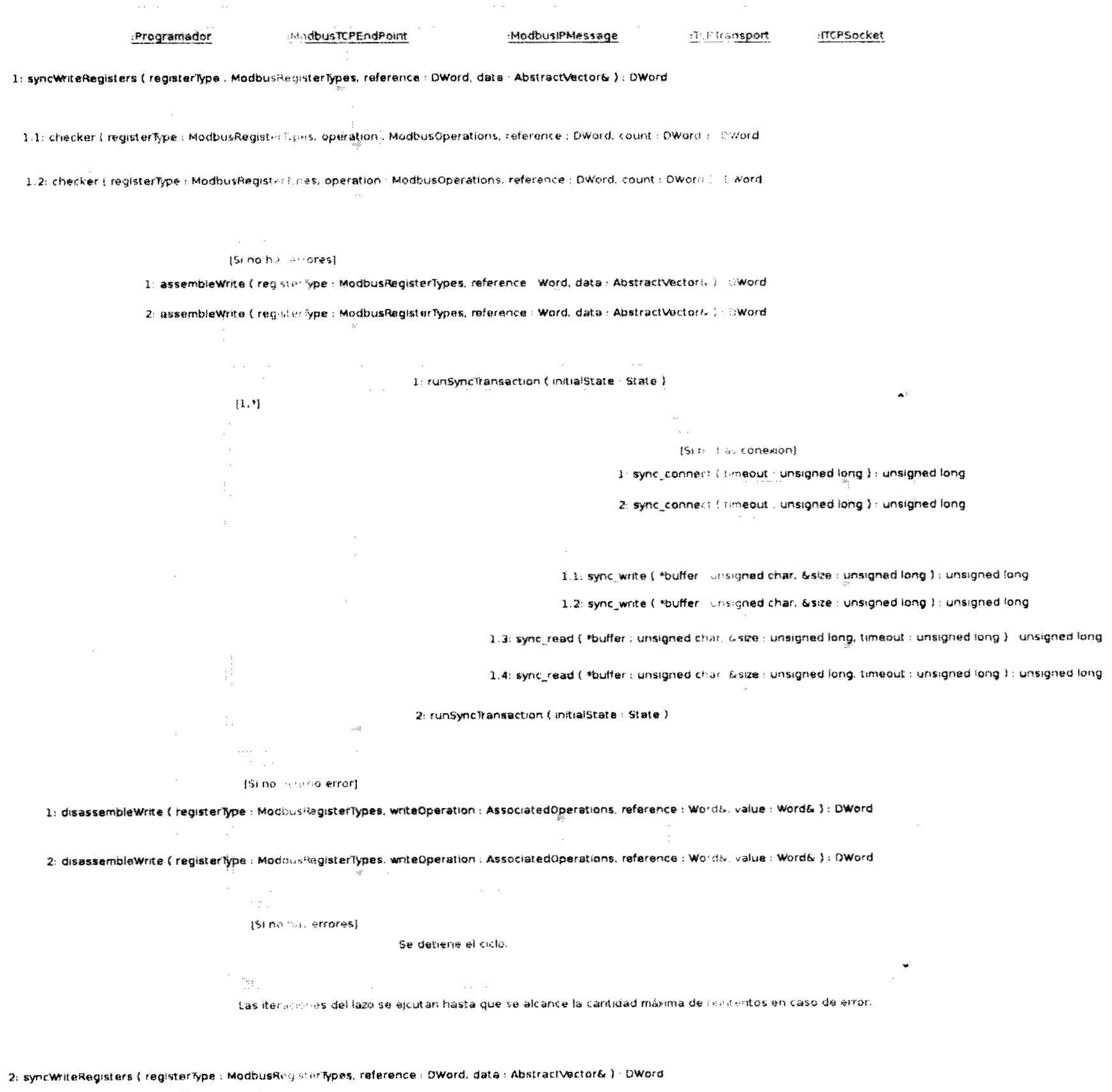

Figura 24: Diagrama de secuencia del CU Escribir variables de forma síncrona.

Diagrama de secuencia del CU Escribir variables de forma asíncrona.

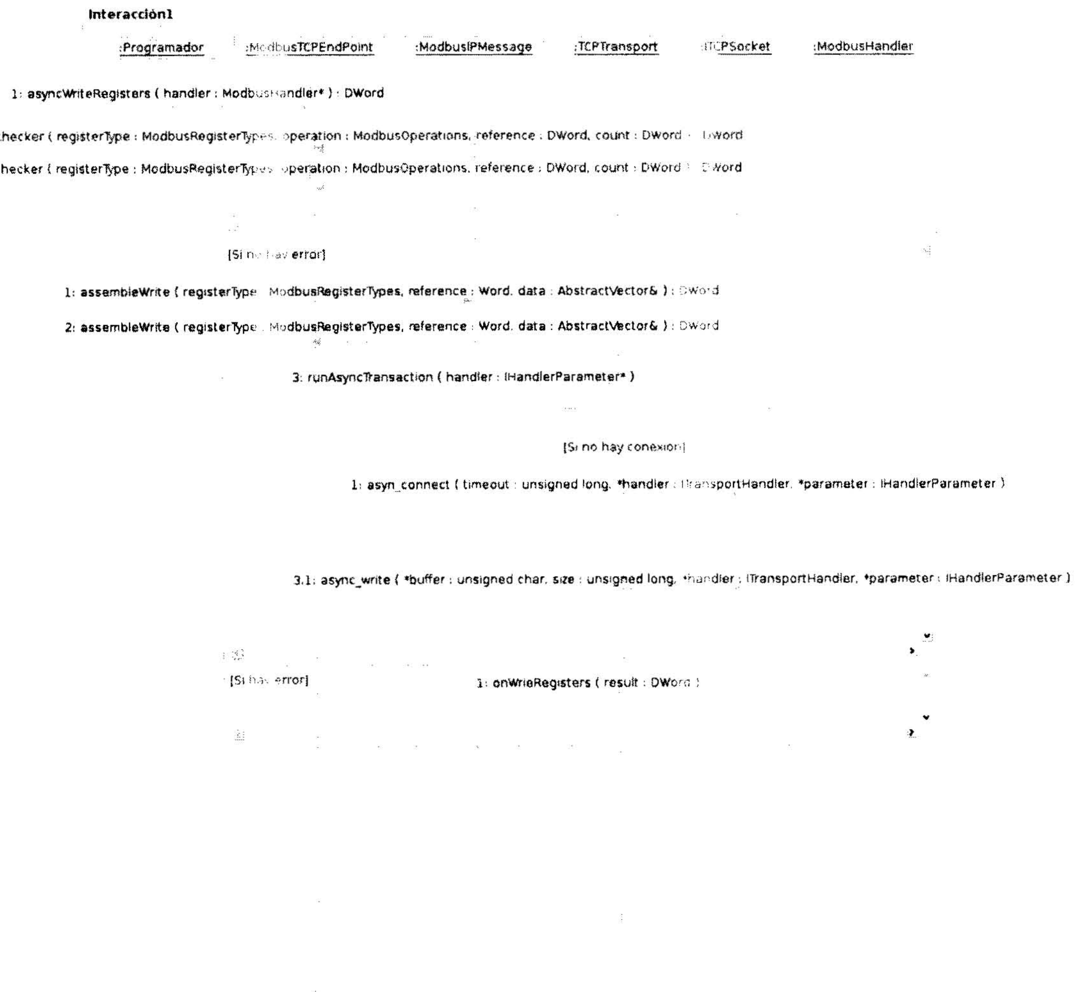

Figura 25: Diagrama de secuencia del CU Escribir variables de forma asíncrona.

3.4 Estándar de codificación.

El código de la capa de acceso a datos sigue algunos estándares propuestos por los grupos de desarrolladores de C++. Esta programado en inglés, debido que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático.

El conocimiento y descripción de los estándares seguidos para el desarrollo contribuirán a un mejor entendimiento del código fuente.

Nombre de los ficheros.

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

NameOfUnits.h

NmaeOfUnit.cpp

Interfaces.

El nombre de las interfaces se indicará de la siguiente manera: IMyInterface, indicando con la "I" que se refiere a una interfaz.

Clases.

Los nombres de las clases deben tener la siguiente forma: MyClass

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

Métodos o función miembro.

Los métodos de clases deben seguir la siguiente estructura: myFunction

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

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.

Tipos de datos definidos.

El nombre de los tipos de datos que se definan deberá comenzar con letra mayúscula similar al formato usado para el nombre de las clases, que en si es un tipo de datos definido por el programador.

Ejemplo.

DWord

AbstractVector

3.5 Diagrama de despliegue.

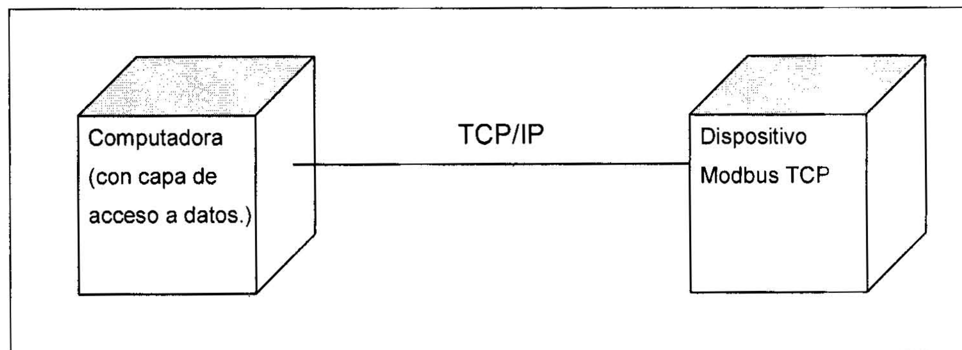

Figura 26: Diagrama de despliegue.

3.6 Diagrama de componentes.

En el siguiente diagrama de componentes se muestran solo los ficheros .h más importantes que se relacionan. Es bueno tener en cuenta que cada fichero .h, donde se define una clase, le corresponde un componente .cpp que implementa sus funcionalidades.

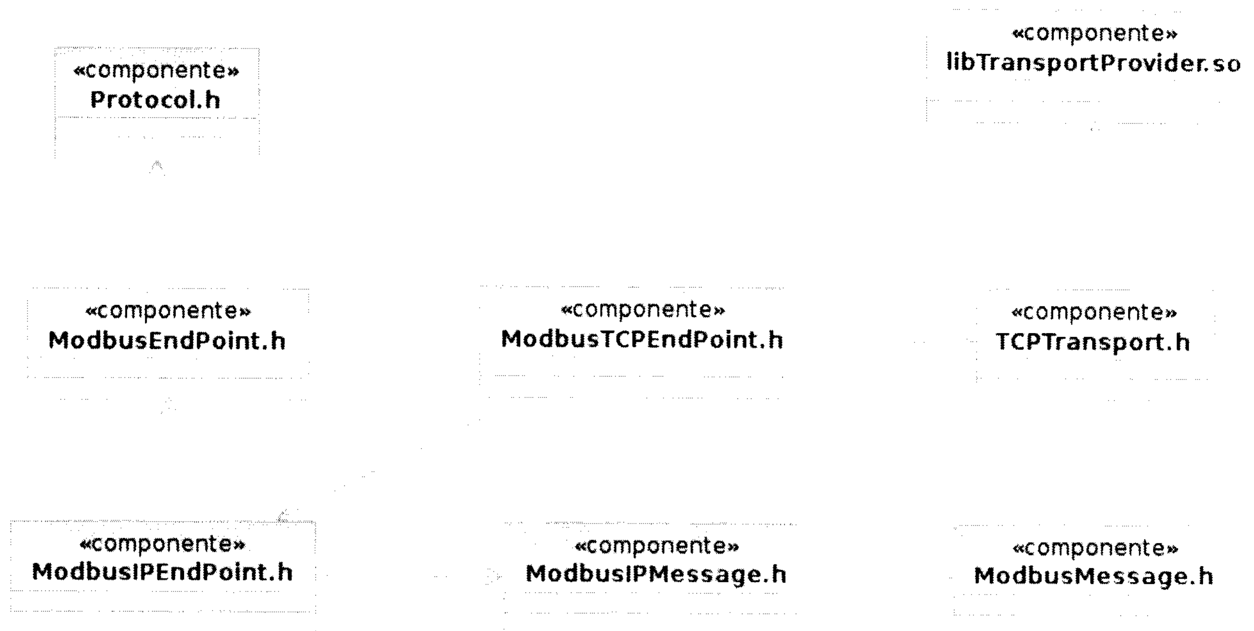

Figura 27: Diagrama de componentes.

Conclusiones

Se obtuvo un módulo de clases completamente funcional, que cumple con los requisitos planteados, la utilización de la librería de transporte Asio C++ Libraries permitió acceder a datos dispuestos en dispositivos Modbus TCP de forma fácil y eficiente. Además del módulo asíncrono implementado, se agregó el modelo sincrónico existente en la capa de protocolo Modbus del proyecto SCADA, para así el que use la capa de acceso a datos pueda elegir cuál de los dos mecanismos es el más idóneo a utilizar.

La capa de acceso a datos se adapta a diferentes y variados entornos de acción y ejecución. Además del manejador Modbus del proyecto SCADA, la misma podría ser usada por cualquier otro sistema que necesite comunicarse con dispositivos Modbus TCP.

Recomendaciones.

Aspectos fundamentales que se recomiendan del trabajo:

- Integrar a la capa de protocolo las variantes del protocolo Modbus restantes, ASCII y RTU.
- Agregar funcionalidades de control de modem y chequeo del estado del dispositivo.
- Posibilitar el envío de tramas a través del protocolo UDP.
- Integrar la capa de acceso a datos al manejador Modbus, del módulo de manejadores del proyecto SCADA.

Referencia bibliografía

1. Modelo OSI. *monografias.com.* [En línea] [Citado el: 24 de Enero de 2008.] http://www.monografias.com/trabajos13/modosi/modosi.shtml.

2. **Comer, Douglas E.** *Redes Globales de Información con Internet y TCP/IP.* Ciudad de La Habana : Editorial Pueblo y Educacion, 2005.

3. *Introduction to MODBUS.* [Technical Tutorial] 2002.

4. MODBUS APPLICATION PROTOCOL SPECIFICATION. *Modbus-IDA.* [En línea] 28 de Diciembre de 2006. [Citado el: 23 de Febrero de 2008.] http://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf.

5. **MODICON.** *Modicon Modbus Protocol Reference Guide.* Massachusetts : s.n., 1996.

6. Modbus Messaging Implementation Guide v1.doc. *MODBUS.ORG.* [En línea] 08 de Mayo de 2002. [Citado el: 03 de Diciembre de 2007.] http://www.modbus.org.

7. Introduction to Programmable Controllers. *PLC dev Tools for PLC Programming.* [En línea] 2005. [Citado el: 29 de mayo de 2008.] http://www.plcdev.com/definition_of_a_plc.

8. EL PLC. *AUTÓMATAS PROGRAMABLES.* [En línea] Diciembre de 2001. [Citado el: 27 de febrero de 2008.] http://www.sc.ehu.es/sbweb/webcentro/automatica/WebCQMH1/PAGINA%20PRINCIPAL/PLC/plc.htm#Introducci%F3n.

9. Welcome to jamod. *Jamod.* [En línea] 23 de marzo de 2007. [Citado el: 20 de abril de 2008.] http://jamod.sourceforge.net/.

10. **Hedström, Anders.** Introduction. *C++ Socket library tutorial.* [En línea] [Citado el: 30 de marzo de 2008.] http://www.alhem.net/Sockets/tutorial/using.html.

11. **Trolltech.** *Qt Assistant Manual.* 2008.

12. Learn More About Qt . *Trolltech.* [En línea] 2008. [Citado el: 01 de abril de 2008.] http://trolltech.com/.

13. **Kohlhoff, Christopher M.** Rationale. *Asio C++ Library.* [En línea] 2003. [Citado el: 20 de marzo de 2008.] http://asio.sourceforge.net/asio-1.0.0/doc/asio/design/rationale.html.

Anexos

Anexo 1. Especificación del CU Obtener información de diagnóstico.

Caso de uso	
Nombre	CU Obtener información de diagnóstico.
Propósito	Obtener información acerca de las operaciones realizadas y el comportamiento de las mismas.
Actores	Programador.
Resumen	El caso de uso se inicializa cuando el actor decide obtener la información de diagnóstico de la representación de un dispositivo previamente creado y mediante el cual se han realizado operaciones de lectura y escrituras.
Referencias	RF6.
Precondiciones	CU Configurar dispositivo.
Acción del actor	Respuesta del sistema
1 – Solicita obtener información de diagnóstico de las operaciones realizadas.	2 – Devuelve la información de diagnóstico (cantidad de operaciones realizadas, operaciones satisfactorias y algunos tipos de errores).
Prioridad	Baja

Tabla 24: Especificación del CU Obtener información de diagnóstico.

Anexo 2. Especificación del CU Interpretar código de error.

<table>
<tr><td colspan="3">Caso de uso</td></tr>
<tr><td>Nombre</td><td colspan="2">CU Interpretar código de error.</td></tr>
<tr><td>Propósito</td><td colspan="2">Obtener descripción de un código de error devuelto.</td></tr>
<tr><td>Actores</td><td colspan="2">Programador.</td></tr>
<tr><td>Resumen</td><td colspan="2">El caso de uso se inicializa cuando el actor decide obtener la información descriptiva de un código de error devuelto, por parte de la representación del dispositivo previamente creado.</td></tr>
<tr><td>Referencias</td><td colspan="2">RF7.</td></tr>
<tr><td>Precondiciones</td><td colspan="2">CU Configurar dispositivo.</td></tr>
<tr><td colspan="2">Acción del actor</td><td>Respuesta del sistema</td></tr>
<tr><td colspan="2">1 – Solicita obtener descripción de un código de error.</td><td>2 – Chequea que el código de error es válido.
3 – De ser válido, se devuelve información descriptiva del código de error indicado.</td></tr>
<tr><td colspan="3">Flujo alternativo</td></tr>
<tr><td colspan="2">Acción del actor</td><td>Respuesta del sistema</td></tr>
<tr><td colspan="2"></td><td>3 – Se indica que el código de error no se puede interpretar.</td></tr>
<tr><td>Prioridad</td><td colspan="2">Baja</td></tr>
</table>

Tabla 25: Especificación del CU Interpretar código de error.

Anexo 3. Descripción de la clase AbstractVector.

Nombre: AbstractVector	
Tipo de clase:	
Atributo	**Tipo**
size	DWord
bitSize	DWord
capacity	DWord
byteSize	DWord
buffer	LPByteArray
Para cada responsabilidad:	

Nombre:	getSize()
Descripción:	Devuelve la cantidad de elementos del vector.
Nombre:	getByteSize()
Descripción:	Devuelve la cantidad de bytes que ocupa el vector.
Nombre:	getCapacity()
Descripción:	Devuelve la cantidad máxima de elementos del vector.
Nombre:	resize(DWord newSize)
Descripción:	Cambia el tamaño del vector.
Nombre:	getInteger(DWord index, DeviceVarType varType)

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

Tabla 26: Descripción de la clase AbstractVector.

Anexo 4. Descripción de la clase WordVector.

Nombre: WordVector	
Tipo de clase:	
Atributo	**Tipo**
high	int
low	int

first	int
second	int
third	int
forth	int
machineLittleEndian	bool
Para cada responsabilidad:	
Nombre:	WordVector(DWord size)
Descripción:	Construye un vector de registros de 16 bits, con el tamaño indicado.
Nombre:	getInteger(DWord index, DeviceVarType varType)
Descripción:	Devuelve un entero ubicado a partir del índice indicado.
Nombre:	getFloat(DWord index, DeviceVarType varType)
Descripción:	Devuelve un flotante ubicado a partir del índice indicado.
Nombre:	setInteger(DWord index, DeviceVarType varType, long long value)
Descripción:	Almacena un entero en la posición indicada.
Nombre:	setFloat(DWord index, DeviceVarType varType, double value)
Descripción:	Establece un flotante en la posición indicada.
Nombre:	getData ()
Descripción:	Devuelve un arreglo de bytes con los datos del vector.
Nombre:	setEndianness(Endianness value)

Descripción:	Cambia el orden de los bytes en los mensajes que se enviarán al dispositivo.

Tabla 27: descripción de la clase WordVector.

Anexo 5. Descripción de la clase BitVector.

Nombre: BitVector	
Tipo de clase:	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	BitVector (DWord size)
Descripción:	Construye un vector de bits, con el tamaño indicado.
Nombre:	getInteger(DWord index, DeviceVarType varType)
Descripción:	Devuelve un entero ubicado a partir del índice indicado.
Nombre:	getFloat(DWord index, DeviceVarType varType)
Descripción:	Devuelve un flotante ubicado a partir del índice indicado.
Nombre:	setInteger(DWord index, DeviceVarType varType, long long value)
Descripción:	Almacena un entero en la posición indicada.
Nombre:	setFloat(DWord index, DeviceVarType varType, double value)
Descripción:	Establece un flotante en la posición indicada.
Nombre:	getData ()

Descripción:	Devuelve un arreglo de bytes con los datos del vector.

Tabla 28: descripción de la clase BitVector.

Glosario de términos.

A:

Aplicaciones HMI/SCADA: proporciona a los operadores las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos.

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.

B:

Boost: Ofrece un conjunto de librerías portables para código en C++.

Broadcast: (Difusión). Sistema de entrega que proporciona la copia de un paquete dado a todos los anfitriones conectados para la difusión del paquete.

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

C:

Cliente/servidor: Modelo de interacción en un sistema distribuido en el que un programa, en una localidad, envía una solicitud a otro programa en otra localidad y espera una respuesta. El programa solicitante se conoce como cliente y el que atiende la solicitud servidor.

CRC: El CRC es un código de detección de error, cuyo cálculo es una larga división de computación en el que se descarta el cociente y el resto se convierte en el resultado, con la importante diferencia de que la

aritmética que usamos conforma que el cálculo utilizado es el arrastre de un campo finito, en este caso los bits.

F:

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

H:

Handler: Manipulador. Rutina de software que realiza una determinada tarea. Por ejemplo, cuando se detecta un error, se llama a un manipulador de error para recuperarse de esa condición. En este contexto el handler es utilizado en forma de función y de estructura o clase.

HTTP: El protocolo de transferencia de Hipertexto es el pegamento que une el World Wide Web. El servicio HTTP en un host permite que usuarios a distancia puedan acceder a los ficheros que almacena si éstos conocen su dirección exacta. El protocolo HTTP define un sistema de direcciones basado en Localizadores Uniformes de Recursos (URL).

I:

Internet: Conexión de redes de conmutación de paquetes interconectadas por ruteadores, junto con los protocolos TCP/IP permiten que la red funcione como una sola red virtual extensa.

IP: (Internet Protocol). Estándar que define los datagramas IP como la unidad de información que pasa a través de una red de redes y proporciona las bases para el servicio de entrega de paquetes sin conexión y con el mejor esfuerzo.

IPv6: IPv6 es el siguiente paso a IPv4 y, entre otras muchas características, soluciona el problema de direccionamiento.

ISO: (International Organization for Standardization). Organización internacional que bosqueja, discute, propone y especifica estándares para los protocolos de red. ISO es mejor conocido por su modelo de referencia de 7 capas que describe la organización conceptual de los protocolos.

L:

LabView: es una herramienta gráfica para pruebas control y diseño mediante la programación Es usado principalmente por ingenieros y científicos para tareas como; adquisición de datos, control de instrumentos, automatización industrial o PAC (Controlador de Automatización Programable).

Licencia GPL: (GNU General Public License). Las licencias de la mayoría del software están diseñadas para eliminar su libertad de compartir y modificar dicho software. Por contra, la GNU General Public License (GPL) está diseñada para garantizar su libertad de compartir y modificar el software. Software libre para garantizar la libertad de sus usuarios. Esta licencia GNU General Public License (GPL) se aplica en la mayoría de los programas realizado por la Free Software Foundation (FSF, Fundación del Software Libre) y en cualquier otro programa en los que los autores quieran aplicarla.

LRC: es un proceso utilizado para verificar la exactitud de los datos almacenados o transmitidos. LRC utiliza un bit de paridad para comprobar y asegurarse de que todos los datos están presentes.

P:

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.

R:

Router: (Ruteador). Computadora dedicada o dispositivo, de propósito especial que se conecta a dos o más redes y envía paquetes de una red a la otra. En particular un ruteador IP, envía datagramas IP entre las redes a las que está conectado. Utiliza las direcciones de destino de un datagrama para decidir el próximo salto al que enviará el datagrama.

S:

SMTP: EL protocolo SMTP es el estándar para la distribución de correo electrónico en Internet. Es un protocolo orientado a texto que utiliza los servicios de TCP/IP para recibir correo desde un cliente y para transferir mensajes desde un servidor a otro de forma fiable.

Socket: Designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiarse cualquier flujo de datos, generalmente de manera fiable y ordenada. Un socket queda definido por una dirección IP, un protocolo y un número de puerto.

SSH: (Secure Shell). Es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos, y también puede ejecutar programas gráficos en ella.

T:

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.

Trolltech: (antiguamente conocido como **Quasar Technologies**) es una compañía de software fundada en el año 1994 en Oslo, Noruega. Su principal actividad es proveer herramientas y bibliotecas de desarrollo de software, así como servicio experto de consulta. Su producto más popular es Qt (*Quasar Toolkit*), una librería multiplataforma para la creación de entornos gráficos.

U:

UDP: (User Datagram Protocol). Protocolo estándar TCP/IP que permite a un programa de aplicación en una máquina enviar un datagrama hacia el programa de aplicación en otra maquina. El UDP utiliza el Protocolo de Internet (IP) para entregar datagramas.

Índice de Figuras y Tablas.

Índice de Figuras.

FIGURA 1: CAPAS DEL MODELO OSI. ... 5

FIGURA 2: ENCUESTA RESPUESTA. ... 8

FIGURA 3: DIFUSIÓN. ... 8

FIGURA 4: FORMATO DE LA TRAMA ENCUESTA/RESPUESTA. ... 10

FIGURA 5: REPRESENTACIÓN DE UNA TRAMA MODBUS ASCII. ... 10

FIGURA 6: REPRESENTACIÓN DE UNA TRAMA MODBUS RTU. ... 11

FIGURA 7: REPRESENTACIÓN DE DISPOSITIVOS COMUNICÁNDOSE A TRAVÉS MODBUS TCP. ... 12

FIGURA 8: TRAMA MODBUS/TCP GENERAL. ... 13

FIGURA 9: ENCAPSULACIÓN DE LAS TRAMAS SOBRE UNA RED TCP/IP. ... 13

FIGURA 10: MODELO CLIENTE/SERVIDOR EN MODBUS. ... 16

FIGURA 11 : CONTROLADOR LÓGICO PROGRAMABLE (PLC). ... 17

FIGURA 12: FASES Y FLUJOS DE TRABAJOS DE RUP. ... 25

FIGURA 13: VARIANTE DE UN HILO DE EJECUCIÓN QUE ATIENDA TODOS LOS DISPOSITIVOS. ... 28

FIGURA 14: VARIANTE DE UN HILO POR CADA DISPOSITIVO A SER ATENDIDO. ... 28

FIGURA 15: ESTRUCTURACIÓN DE LAS ACCIONES A REALIZAR EN EL MODELO ASINCRÓNICO PROPUESTO. ... 31

FIGURA 16: REPRESENTACIÓN DEL MODELO ASÍNCRONO PROPUESTO. ... 32

FIGURA 17: REPRESENTACIÓN DEL PATRÓN THREAD POOL, EN LA EJECUCIÓN DE LA COLA DE CALLBACK PRESENTE EN LA PROPUESTA DE SOLUCIÓN. ... 33

FIGURA 18: MODELO DEL DOMINIO. ... 35

FIGURA 19: DIAGRAMA DE CUS. ... 39

FIGURA 20: DIAGRAMA DE CLASES DEL DISEÑO. ... 49

FIGURA 21: DIAGRAMA DE SECUENCIA DEL CU CONFIGURAR DISPOSITIVO. ... 77

FIGURA 22: DIAGRAMA DE SECUENCIA DEL CU LEER VARIABLES DE FORMA SÍNCRONA. ... 78

FIGURA 23: DIAGRAMA DE SECUENCIA DEL CU LEER VARIABLES DE FORMA ASÍNCRONA. ... 78

FIGURA 24: DIAGRAMA DE SECUENCIA DEL CU ESCRIBIR VARIABLES DE FORMA SÍNCRONA. ... 80

FIGURA 25: DIAGRAMA DE SECUENCIA DEL CU ESCRIBIR VARIABLES DE FORMA ASÍNCRONA. ... 80

FIGURA 26: DIAGRAMA DE DESPLIEGUE. ... 82

FIGURA 27: DIAGRAMA DE COMPONENTES. ... 83

Índice de tablas.

TABLA 1: ELEMENTOS QUE COMPONEN LA CABECERA MBAP 14

TABLA 2: FUNCIONES ASOCIADAS DEL PROTOCOLO. 15

TABLA 3: TABLA DE VALORES POR DEFECTO SEGÚN LA PROPIEDAD. 35

TABLA 4: ACTOR DEL SISTEMA. 38

TABLA 5: DESCRIPCIÓN DEL CU CONFIGURAR DISPOSITIVO. 41

TABLA 6: DESCRIPCIÓN DEL CU LEER VARIABLES DE FORMA SINCRÓNICA. 42

TABLA 7: DESCRIPCIÓN DEL CU LEER VARIABLES DE FORMA ASINCRÓNICA. 44

TABLA 8: DESCRIPCIÓN DEL CU ESCRIBIR VARIABLES DE FORMA SINCRÓNICA. 46

TABLA 9: DESCRIPCIÓN DEL CU ESCRIBIR VARIABLES DE FORMA ASINCRÓNICA. 47

TABLA 10: DESCRIPCIÓN DE LA CLASE PROTOCOL. 53

TABLA 11: DESCRIPCIÓN DE LA CLASE MODBUSENDPOINT. 58

TABLA 12: DESCRIPCIÓN DE LA CLASE MODBUSIPENDPOINT. 59

TABLA 13: DESCRIPCIÓN DE LA CLASE MODBUSTCPENDPOINT. 60

TABLA 14: DESCRIPCIÓN DE LA CLASE MODBUSMESSAGE. 63

TABLA 15: DESCRIPCIÓN DE LA CLASE MODBUSIPMESSAGE. 64

TABLA 16: DESCRIPCIÓN DE LA CLASE TCPTRANSPORT. 67

TABLA 17: DESCRIPCIÓN DE LA CLASE MODBUSHANDLER. 69

TABLA 18: DESCRIPCIÓN DE LA CLASE ITRANSPORTPROVIDER. 70

TABLA 19: DESCRIPCIÓN DE LA CLASE ITCPSOCKET. 72

TABLA 20: DESCRIPCIÓN DE LA CLASE ITRANSPORTHANDLER. 74

TABLA 21: DESCRIPCIÓN DE LA CLASE IACCEPTOR. 74

TABLA 22: DESCRIPCIÓN DE LA CLASE IASYNCTIMER. 75

TABLA 23: DESCRIPCIÓN DE LA CLASE ISTRAND. 75

TABLA 24: ESPECIFICACIÓN DEL CU OBTENER INFORMACIÓN DE DIAGNÓSTICO. 88

TABLA 25: ESPECIFICACIÓN DEL CU INTERPRETAR CÓDIGO DE ERROR. 89

TABLA 26: DESCRIPCIÓN DE LA CLASE ABSTRACTVECTOR. 91

TABLA 27: DESCRIPCIÓN DE LA CLASE WORDVECTOR. 93

TABLA 28: DESCRIPCIÓN DE LA CLASE BITVECTOR. 94