Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Implementación de Aplicación Cliente/Servidor sobre el sistema *Tankvision* en la refinería “Camilo Cienfuegos”.

Autor: Roberto Carlos Vergara Osorio.

Tutores: Ing. José Omar Padrón Ramos.

Ing. Yeiniel Suárez Sosa.

Consultante: Lic. Mario Moreira Martínez.

Santa Clara

2010

"Año 52 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Implementación de Aplicación Cliente/Servidor sobre el sistema *Tankvision* en la refinería “Camilo Cienfuegos”.

Autor: Roberto Carlos Vergara Osorio.

rvergara@uclv.edu.cu

Tutores: Ing. José Omar Padrón Ramos.

jpadron@uclv.edu.cu

Ing. Yeiniel Suarez Sosa.

yssosa@pdvcupetsa.cu

Consultante: Lic. Mario Moreira Martínez.

mmoreira@pdvcupetsa.cu

Santa Clara

2010

"Año 52 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

Autor

Roberto Carlos Vergara Osorio

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Tutor

José Omar Padrón Ramos.

Tutor

Yeiniel Suárez Sosa.

Jefe Dpto Automática

Boris Luis Martínez Jiménez.

.

Responsable de

Información Científico-Técnica.

El futuro tiene muchos nombres.

Para los débiles es lo inalcanzable,

Para los temerosos, lo desconocido.

Pero para los valientes

Es la oportunidad.

Víctor Hugo.

A Abuela, por lo que significa.

A Mamá, por estar siempre a mi lado en todos los momentos.

A papá, por su optimismo y enseñanzas.

A Lili, por su amor.

A Migue, por ser una razón muy fuerte en mi vida.

A Nathalí y Luany, mis hermanos, porque nunca me han faltado.

A Mandy, porque es especial.

A tía Alina y tío Carlis porque han sido como unos padres, con un amor

Sin límites.

A Alejandra, mi hermanita.

A abuelo Carlos, mi padre, porque sin su apoyo y sacrificio

Nada hubiera sido posible.

A Manina, Mariela y Laura, que a pesar de la distancia siempre están presentes

A kaki y abuela Magela, por estar siempre presentes en mi vida.

A abuela Olguita, por su amor y cariño constantes.

A tío Alejandro y Maye por su sinceridad y afecto siempre.

A Che, Abuelo Taurino y Abuelo Roberto que no están,

Pero que también son parte de la realización de este sueño.

A Danilo, que vive en mi corazón.

A toda mi familia por formar parte de este sueño durante estos cinco años y conducirme en los caminos de la vida.

A Yeiniel, que hizo suyo este proyecto con el empeño que lo caracteriza, y sin el cual no hubiera sido posible.

A José Omar, mi tutor y amigo que no dudó en apoyarme cuando lo necesité.

A Mayito, gracias por su apoyo.

A Joaquín, Gustavo, Lázaro y todo el grupo de la Universidad, que hemos sido un gran equipo.

A mi hermano Julio Rubén, Gracias

A todo el colectivo de profesores e la Facultad de Ingeniería Eléctrica de la UCLV por los conocimientos y las enseñanzas dadas durante estos años.

Al colectivo de técnicos e ingenieros de la refinería por su inestimable ayuda, en especial a Heinz.

A todas las personas que de una forma u otra colaboraron directa o indirectamente para que este proyecto llegara a su feliz culminación.

A todos Muchas Gracias.

TAREA TÉCNICA

Para la elaboración de este trabajo de diploma se realizaron las siguientes tareas:

1. Revisión bibliográfica del caso de estudio: Aplicación Cliente/Servidor empleando Modbus TCP.
2. Selección de las herramientas adecuadas para la realización del servicio.
3. Ilustración el proceso de diseño del software y su implementación.
4. Propuesta de aplicación cliente servidor empleando Modbus TCP para el sistema *Tankvision*.
5. Realización del informe.

Tutor
José Omar Padrón Ramos.

Tutor
Yeiniel Suarez Sosa.

Autor
Roberto Carlos Vergara Osorio.

RESUMEN

El presente trabajo trata sobre la implementación de una aplicación Cliente/Servidor sobre el sistema de monitoreo *Tankvision* instalado en el patio de tanques de la refinería de petróleo “Camilo Cienfuegos”. Este proyecto nace debido a la necesidad de explotar al máximo las capacidades de comunicación que brinda el sistema y tiene como objetivo principal crear un servicio que sea capaz de comunicarse, empleando el protocolo Modbus/TCP, con el dispositivo *Host Link* NXA 822 que actuará como esclavo. Dicho programa además de encargarse de la comunicación deberá escribir los datos que almacena el *Host Link* NXA 822 en una base de datos relacional para ser usados posteriormente con diversos fines.

TABLA DE CONTENIDOS

GLOSARIO DE TÉRMINOS

ADU *Application Data Unit*, Unidad de Datos de Aplicación.

ANSI *American National Standards Institute,* Instituto Nacional Estadounidense de Estándares.

API *Application Programming Interface*, Interfaz de Programación de Aplicaciones.

ASCII *American Standard Code for Information Interchange,* Código Estadounidense Estándar para el Intercambio de Información.

BD Base de Datos.

BSD *Berkeley Software Distribution,* Distribución de Software Berkeley.

CPU *Central Processing Unit,* Unidad Central de Procesamiento.

DAQ *Data Adquisition,* Adquisición de Datos.

DDE *Dinamic Data Exchange,* Intercambio Dinámico de Datos.

DI *Injection Dependency,* Inyección de Dependencia.

DLL *Dinamic Link Library,* Enlace Dinámico de Bibliotecas.

GUI *Graphical User Interface,* Interfaz Gráfica de Usuario.

IDE *Integrated Development Environments,* Ambiente de Desarrollo Integrado.

IP *Internet Protocol,* Protocolo de Internet.

JDBC *Java Database Conectivity* Conectividad Java-Base de Datos.

JDK *Java Developer's Kit,* Kit de Aplicaciones de Java.

JRE *Java Runtime Enviroinment,* Ambiente de Ejecución de Java.

LAN *Local Area Network,* Red de Área Local.

MBAP *Modbus Application Protocol,* Protocolo de Aplicación de Modbus.

MCP Movimiento de Crudos y Productos.

MIPS Millones de Instrucciones por Segundo.

NT *New Technology*, Nueva tecnología.

OLE *Object Linking and Embedding,* Objetos Enlazados y Unificados.

PC *Personal Computer,* Computadora Personal.

PDU *Protocol Data Unit,* Unidad de Datos de Protocolo.

PGDG *PostgreSQL Global Development Group,* Grupo Global de Desarrollo de PostgreSQL.

PL *Procedure Language*, Lenguaje de Procedimiento.

PLC *Programmable Logic Controller,* Controlador Lógico Programable.

RDBMS *Relational Data Base Management System,* Gestor de Bases de Datos Relacional.

RPC *Remote Procedure Call,* Llamada a Procedimiento Remoto.

RTU *Remote Terminal Unit*, Unidad Terminal Remota.

SCADA *Supervisory Control and Data Acquisition,* Supervisión Control y Adquisición de Datos.

SCD *SCM based Distribuited Clusting*, SCM Distribuido basado en Clusters

SCM *Software Configuration Manager*, Software Administrador de Configuración.

SQL *Structured Query Language,* Lenguaje de Consulta Estructurado.

TCP *Trasport Control Protocol,* Protocolo de Control de Transporte.

TD Tabla de datos.

UML *Unified Modeling Language,* Lenguaje Unificado de Modelado.

URL *Uniform Resource Locator,* Localizador Uniforme de Recursos.

XML *Extensible Markup Language,* Lenguaje de Marcas Extensible.

INTRODUCCIÓN

En el área de las comunicaciones en entornos industriales, la estandarización de protocolos es un tema en permanente discusión, donde intervienen problemas técnicos y comerciales. Cada protocolo está optimizado para diferentes niveles de automatización y en consecuencia responden al interés de diferentes proveedores, por ejemplo: *Fieldbus Foundation, Profibus* y *HART*, están diseñados para instrumentación de control de procesos, en cambio *DeviceNet* y SDC están optimizados para el mercado de los dispositivos discretos de detectores, actuadores e interruptores, donde el tiempo de respuesta y la frecuencia de repetición son factores críticos (Olaya 2002).

Cada protocolo tiene un rango de aplicación, fuera del cual disminuye el rendimiento y aumenta la relación costo prestación. Debido a la no aceptación de un protocolo estándar único en las comunicaciones industriales, los múltiples buses de campo han perdido terreno ante la incursión de tecnologías de comunicación emergentes, como Ethernet, en esta área (Olaya 2002).

La aceptación mundial de Ethernet en los entornos administrativos y de oficina ha generado el deseo de expandir su aplicación a la planta, por lo que se está moviendo rápidamente hacia el mercado de los sistemas de control de procesos y la automatización para la interconexión a nivel de campo de sensores y actuadores, reemplazando de esta forma a los buses de campo en las industrias. Es posible que con los avances de Ethernet y la tecnología emergente *Fast* Ethernet se pueda aplicar también al manejo de aplicaciones críticas de control (Olaya 2002).

Los buses de campo son una forma especial de LAN dedicada a aplicaciones de adquisición de datos y comando de elementos finales de control sobre la planta, que típicamente operan sobre cables de par trenzado de bajo costo. A diferencia de Ethernet, donde no se puede garantizar determinismo sobre la llegada de paquetes, los diseñadores optimizan los buses de campo para el intercambio de mensajes cortos de comando y de control con altísima seguridad y temporización estricta (Olaya 2002).

En las aplicaciones industriales, Ethernet es usado en conjunto con la pila de protocolos TCP/IP universalmente aceptada. TCP/IP es el conjunto de protocolos usado en Internet, suministrando un mecanismo de transporte de datos confiable entre máquinas y permitiendo interoperabilidad entre diversas plataformas. Usar TCP/IP sobre Ethernet a nivel de campo en la industria permite tener una verdadera integración con la Intranet corporativa, y de esta forma se ejerce un estricto control sobre la producción (Olaya 2002).

Objetivo General.

Implementar un servicio basado en la arquitectura Cliente/Servidor empleando el protocolo estándar de instrumentación sobre Ethernet: Modbus TCP, que sea capaz de establecer comunicación con el dispositivo *Host Link* NXA822, del sistema de supervisión *Tankvision,* y almacenar los datos del mismo en una base de datos relacional.

Objetivos Específicos.

- Implementar un software que se encargue de establecer la conexión y de realizar la adquisición de los datos desde el *Host Link* NXA822.
- Implementar una base de datos relacional de registros históricos que almacene esos datos, la cual será actualizada cada un tiempo específico.

Importancia.

Este proyecto deviene en una aplicación novedosa para el mundo de las comunicaciones industriales utilizando Ethernet, ya que se conocen pocos antecedentes del mismo.

La ejecución práctica del resultado del mismo revestirá gran importancia para la empresa debido a que se aprovechará la ventaja de almacenar la información que ofrece el sistema *Tankvision*, hasta ahora solo accesible por medio de un archivo XML a través del *Host Link* NXA822, en una base de datos que permitirá visualizarla de manera organizada y con todas las ventajas que ello implica, realizándose la comunicación de una manera sencilla, rápida y segura como promete el protocolo Modbus TCP, con lo que se mejorará el proceso de toma de decisiones y el sistema de monitoreo de tanques, todo lo cual contribuirá al perfeccionamiento del sistema de adquisición de datos en general, manteniendo una supervisión más estricta del proceso.

Organización del informe.

El informe está compuesto por tres capítulos, en el primer capítulo se hace una revisión bibliográfica del caso de estudio, donde se exponen conceptos importantes acerca de la arquitectura Cliente/Servidor en general, se describe el protocolo Modbus TCP y se hace referencia a cuestiones importantes relacionadas con las bases de datos y la programación multicapas así como una explicación del sistema *Tankvision.* En el segundo capítulo se describen las herramientas empleadas para la creación del software y la base de datos y se dan los detalles acerca de su diseño. En el tercer capítulo se enuncian lo experimentos diseñados para probar el servicio y se ofrece un manual de usuario para uso y explotación del mismo.

CAPÍTULO 1. APLICACIONES CLIENTE/SERVIDOR EN LA INDUSTRIA.

En este capítulo se abordan los aspectos teóricos implicados en el caso de estudio. El mismo está compuesto por seis epígrafes que recogen el estado del arte sobre las aplicaciones Cliente/Servidor en la industria. Se conceptualiza la filosofía y el funcionamiento de esta, se define y describe el protocolo Modbus TCP/IP así como la importancia de las bases de datos, mencionándose los gestores más utilizados, también se explica el funcionamiento del sistema *Tankvision* y se enumeran las herramientas a utilizar para llevar a cabo el proyecto planteado. Finalmente se dan las conclusiones parciales del capítulo.

1.1 Historia y definición.

En el mundo de TCP/IP las comunicaciones entre computadoras se rigen básicamente por lo que se llama modelo Cliente/Servidor, éste es un modelo que intenta proveer usabilidad, flexibilidad, interoperabilidad y escalabilidad en las comunicaciones (Márquez 2004). .

El término Cliente/Servidor fue usado por primera vez en 1980 para referirse a máquinas conectadas en red y empezó a ser aceptado como modelo de arquitectura oficial a finales del propio año. Desde el punto de vista funcional, se puede definir como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente, aún en entornos multiplataforma (Márquez 2004).

En el modelo Cliente/Servidor, el cliente envía un mensaje solicitando un determinado servicio a un servidor, y este envía uno o varios mensajes con la respuesta (provee el servicio). En un sistema distribuido cada máquina puede cumplir el rol de servidor para algunas tareas y el rol de cliente para otras (Márquez 2004).

La forma más estándar de aplicación y uso de sistemas Cliente/Servidor es mediante la explotación de las PC a través de interfaces gráficas de usuario; mientras que la administración de datos y su seguridad e integridad se deja a cargo de computadoras centrales de alto rendimiento (*mainframe*). Usualmente la mayoría del trabajo pesado se hace en el proceso llamado servidor y los procesos cliente sólo se ocupan de la interacción con el usuario, aunque esto puede variar. En otras palabras la arquitectura Cliente/Servidor es una extensión de programación modular en la que la base fundamental es separar una gran pieza de software en módulos, con el fin de hacer más fácil el desarrollo y mejorar su mantenimiento. Esta arquitectura permite distribuir físicamente los procesos y los datos en forma más eficiente, lo que en computación distribuida afecta directamente el tráfico de la red, reduciéndolo grandemente (Márquez 2004).

1.1.1 Ventajas.

Uno de los aspectos que más ha promovido el uso de sistemas Cliente/Servidor, es la existencia de plataformas de hardware cada vez más baratas, esta constituye a su vez una de las más palpables ventajas de este esquema: la posibilidad de utilizar máquinas considerablemente más baratas que las requeridas por una solución centralizada, basada en sistemas grandes. Además, se pueden utilizar componentes, tanto de hardware como de software, de varios fabricantes, lo cual contribuye considerablemente a la reducción de costos y favorece la flexibilidad en la implantación y actualización de soluciones (Márquez 2004).

El esquema Cliente/Servidor facilita la integración entre sistemas diferentes y comparte información, permitiendo por ejemplo, que las máquinas ya existentes puedan ser utilizadas pero empleando interfaces más amigables al usuario. De esta manera podemos integrar computadoras simples con sistemas medianos y grandes,

sin necesidad de que todos tengan que utilizar el mismo sistema operativo (Márquez 2004).

Al favorecer el uso de interfaces gráficas, los sistemas construidos bajo este esquema tienen una mayor y más intuitiva interacción con el usuario. En el uso de interfaces gráficas para el usuario, el esquema Cliente/Servidor presenta la ventaja, con respecto a uno centralizado, de que no es siempre necesario transmitir información gráfica por la red pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red (Márquez 2004).

Una ventaja adicional de la arquitectura Cliente/Servidor radica en que: es más rápido el mantenimiento y el desarrollo de aplicaciones, pues se pueden emplear las herramientas existentes (por ejemplo los servidores de SQL o las herramientas de más bajo nivel como los *sockets* y el RPC). La estructura modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones (Márquez 2004).

El esquema Cliente/Servidor contribuye además a proporcionar a los diferentes departamentos de una organización, soluciones locales, pero permitiendo la integración de la información relevante a nivel global (Márquez 2004).

1.2 El protocolo Modbus TCP.

Modbus TCP fue introducido por Schneider Automation como una variante de la familia Modbus, introducida por Modicon en 1979. Específicamente, el protocolo cubre el uso de mensajes Modbus en un entorno Intranet o Internet usando los protocolos TCP/IP (Olaya 2002).

En la actualidad hay cientos de dispositivos Modbus TCP disponibles en el mercado. El protocolo se ha convertido en un estándar industrial de facto debido a su simplicidad, bajo costo, necesidades mínimas en cuanto a componentes de hardware y sobre todo a que se trata de un protocolo abierto. Modbus TCP se emplea para intercambiar información entre dispositivos, así como monitorizarlos y gestionarlos, también se emplea para la gestión de entradas/salidas distribuidas, siendo el protocolo más popular entre los fabricantes de este tipo de componentes. La combinación de una red física versátil y escalable como Ethernet con el estándar

universal de interredes TCP/IP y una representación de datos independiente del fabricante, como Modbus, proporciona una red abierta y accesible para el intercambio de datos de un proceso (Buendía 2003).

Modbus TCP simplemente encapsula una trama Modbus en un segmento TCP (figura 1.1), este último proporciona un servicio orientado a conexión fiable, lo que significa que toda consulta espera una respuesta (Buendía 2003).

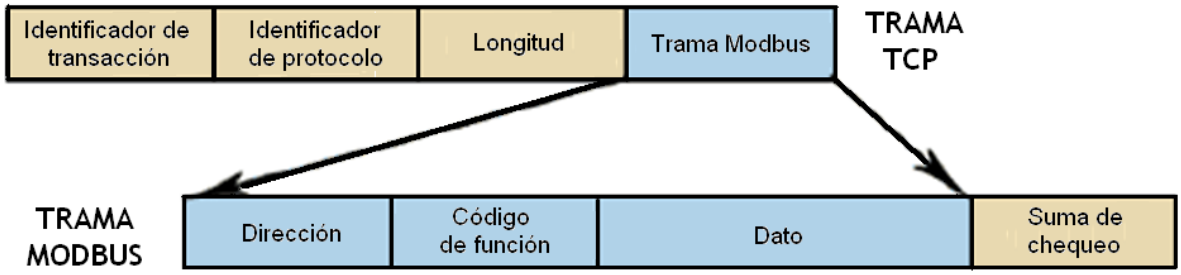


Figura 1.1 Encapsulamiento de la trama Modbus en una trama TCP.

1.2.1 Modbus TCP en el modelo Cliente/Sevidor.

La comunicación Cliente/Servidor empleando Modbus TCP está basada en cuatro tipos de mensajes mostrados en la figura 1.2.

- Petición (*Request*) (Mensaje enviado por el cliente para iniciar la comunicación).
- Indicación (*Indication*) (Mensaje de petición recibido por el servidor).
- Respuesta (*Response)* (Respuesta enviada por el servidor).
- Confirmación (*Confirmation*) (Mensaje de respuesta recibido por el cliente).

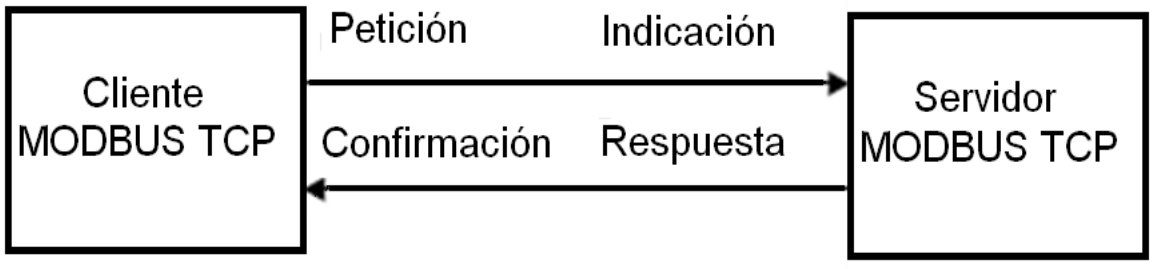


Figura 1.2 Esquema de comunicación Cliente/Servidor empleando Modbus TCP.

La comunicación sobre Modbus TCP puede incluir diferentes tipos de dispositivos (figura 1.3), tanto clientes y servidores conectados directamente a la red TCP/IP como la utilización de *bridge*, *router* o *gateway* para la interconexión entre una red TCP/IP y una sub red serie que permita a dispositivos de este tipo formar parte de la arquitectura Cliente/Servidor (Modbus-IDA 2006).

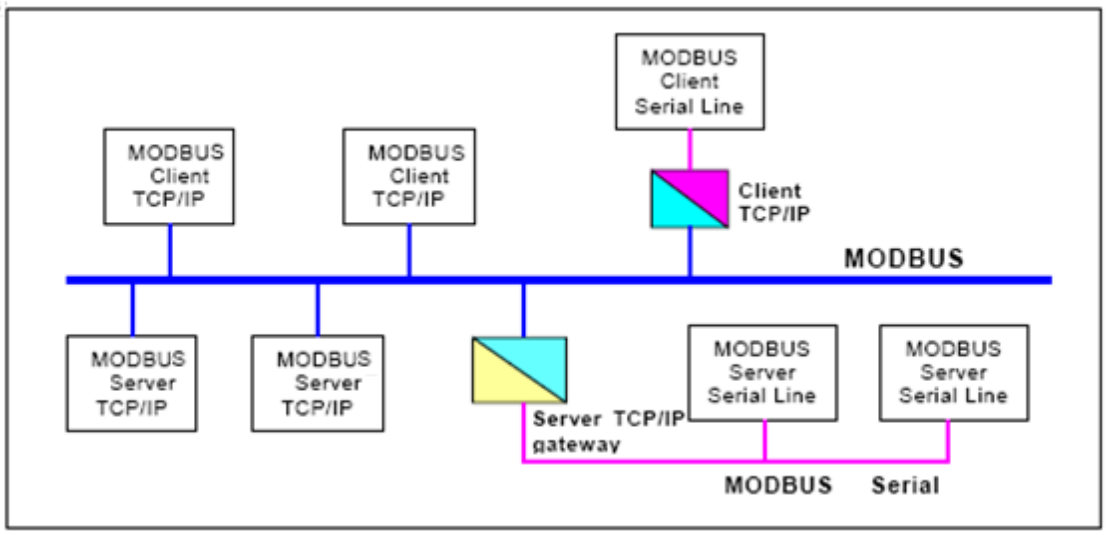


Figura 1.3 Esquema de conexión de diferentes dispositivos sobre una red TCP/IP.

Básicamente el protocolo Modbus consta de un PDU independientemente de la capa de comunicación en que se implemente. El *mapping* del protocolo en un tipo de red específico introduce lo que se conoce como ADU (figura 1.4). El cliente que inicia el intercambio construye su propia ADU. El *Function Code* (Código de Función) indica al servidor la acción a tomar (Modbus-IDA 2006).

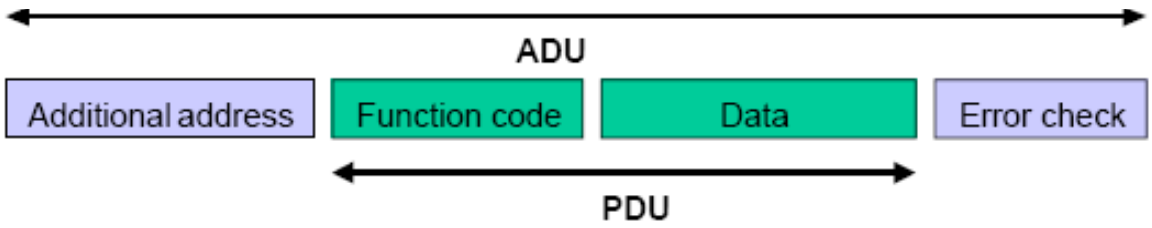


Figura 1.4 Trama genérica Modbus.

Cuando Modbus se encapsula en TCP/IP se utiliza un encabezamiento (*header*) para identificar la ADU, identificado como MBAP (figura 1.5). Este encabezamiento consta de cuatro campos que son detallados en la Tabla 1.1 (Modbus-IDA 2006).

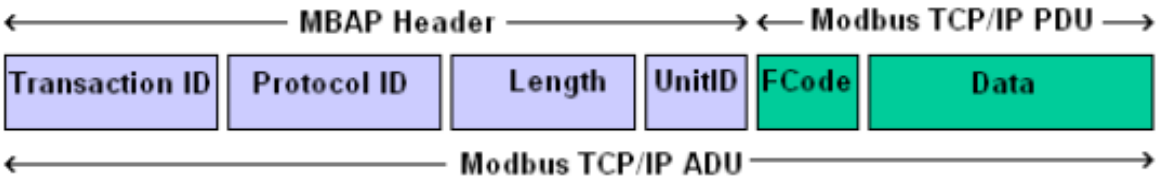


Figura 1.5 Trama de Modbus TCP.

Tabla 1.1 Descripción de los campos del encabezamiento MBAP.

Campo	Extensión	Descripción	Cliente	Servidor
Identificador de transacción	2 bytes	Identifica la transacción de una petición o respuesta	Inicializado por el cliente	Copiado por el servidor desde que se recibe la petición
Identificador de protocolo	2 bytes	0 para el protocolo Modbus	Inicializado por el cliente	Copiado por el servidor desde que se recibe la petición
Longitud	2 bytes	Número de bytes siguientes	Inicializado por el cliente (petición)	Inicializado por el servidor (respuesta)
Identificador de unidad	1 byte	Identifica esclavo remoto conectado en puerto serie o en otro bus	Inicializado por el cliente	Copiado por el servidor desde que se recibe la petición

1.2.2 Modbus TCP como alternativa eficiente.

Las ventajas cuando se emplea Modbus TCP dependen básicamente de la red y el hardware. Si es usado sobre Internet, las prestaciones serán las correspondientes a tiempos de respuesta en Internet, que no siempre serán las deseables para un sistema de control, sin embargo pueden ser suficientes para la comunicación destinada a depuración y mantenimiento, evitando así desplazamientos al lugar de la instalación. Si disponemos de una Intranet de altas prestaciones con conmutadores Ethernet de alta velocidad, la situación es totalmente diferente. (Buendía 2003)

En teoría, Modbus TCP, transporta datos alrededor de un 60% de eficiencia cuando se trasfieren registros en bloque, y puesto que 10 Base T proporciona unos 1.25 Mbps de datos, la velocidad de transferencia de información útil será:

1.25Mbps / 2 * 60% = 375 000 registros por segundo.

En 100 Base T la velocidad es 10 veces mayor.

Todo esto suponiendo que se están empleando dispositivos que pueden dar servicio en la red Ehernet aprovechando todo el ancho de banda disponible (Buendía 2003).

En los ensayos prácticos realizados por Schneider Automation utilizando un PLC Momentum con entradas/salidas Ethernet, se demostró que se podían escanear hasta 4000 bloques entradas/salidas por segundo, cada uno con hasta 16 entradas/salidas analógicas de 12 bits o 32 entradas/salidas digitales (se pueden actualizar 4 registros base por milisegundo). Aunque estos resultados están por debajo del límite teórico calculado anteriormente, debe saberse que el dispositivo se probó con una CPU de baja velocidad (80186 a 50MHz con 3 MIPS). Además, el abaratamiento de las computadoras y el desarrollo de redes Ethernet cada vez más

rápidas, permite elevar las velocidades de funcionamiento, a diferencia de otros buses que están inherentemente limitados a una sola velocidad (Buendía 2003).

Otras ventajas de utilizar Modbus TCP son:

- Es escalable en complejidad. En dispositivos que requieran un propósito simple necesita solo implementar uno o dos tipos de mensaje.
- Es simple para administrar y expandir. No se requiere usar herramientas de configuración complejas cuando se añade una nueva estación a una red Modbus TCP.
- No es necesario equipo o software propietario de algún vendedor. Cualquier sistema, computadora o microprocesador con una pila de protocolos TCP/IP puede usarlo.
- Puede ser usado para comunicarse con dispositivos Modbus, usando productos de conversión que no requieren configuración.
- Es de muy alto desempeño, limitado típicamente por la capacidad del sistema operativo instalado para comunicarse.
- Altas tasas de transmisión son fáciles de lograr sobre una estación única, cualquier red puede ser construida para lograr tiempos de respuesta garantizados en el rango de los milisegundos (Olaya 2002).

1.3 Conectividad entre Aplicaciones.

En las aplicaciones Cliente/Servidor juega un papel fundamental las comunicaciones existentes entre las aplicaciones, esta es una premisa fundamental en los sistemas modernos, tanto en la propia computadora como con el exterior de la misma, ya sea por software o por hardware. El sistema operativo Windows ofrece varias posibilidades para la comunicación entre aplicaciones, las cuales son explotadas por los sistemas de supervisión actuales.

A continuación se dan a conocer ejemplos de algunas de las conectividades más usadas (Ballesteros 2009).

- Ejecución de comandos del sistema.
- Manejo de ficheros.
- Enlace dinámico de bibliotecas (DLL).
- Intercambio dinámico de datos (DDE).
- Enlace con objetos (OLE)
- Conexión a redes, uso del protocolo TCP/IP.
- Uso de *drivers* específicos para adquisición de datos (DAQ).
- Trabajo con bases de datos y lenguaje de consulta (SQL).

Seguidamente se explican algunas de ellas:

Ejecución de comandos.

En muchas aplicaciones se pueden ejecutar comandos del sistema operativo, como: obtener de manera dinámica el directorio activo, crear directorios, ejecutar otra aplicación, ejecutar una línea de comandos similar a como se realiza desde un sistema operativo, etc. (Ballesteros 2009).

Manejo de ficheros.

Existe un gran número de funciones para el trabajo con ficheros, entre las que se encuentran: abrir, leer, escribir, cerrar, copiar, mover información sobre un fichero, etc. La información puede almacenarse en binario, ASCII u otro código, no obstante, en muchos casos se brindan utilitarios para la lectura, escritura de cadenas de caracteres, estructuras de datos etc., para facilitar el trabajo común con ficheros. Siempre hay que tener presente cerrar los ficheros cada vez que se abran, ya que en tiempo real si un fichero queda abierto, ocupa un espacio en memoria que puede bloquear la aplicación (Ballesteros 2009).

Enlace dinámico de bibliotecas (DLL).

El enlace dinámico de bibliotecas es un mecanismo que enlaza aplicaciones a bibliotecas en tiempo de ejecución, permaneciendo estas en su propio fichero, sin ser copiadas al fichero ejecutable de la aplicación. Las DLL se enlazan con la aplicación cuando esta se está ejecutando y pueden tener enlaces con otras. Cada

DLL puede contener varias funciones y usan un formato que es común a una gran variedad de sistemas, por lo que las aplicaciones sobre Windows tendrán acceso a una gran cantidad de funciones realizadas con diferentes aplicaciones (Ballesteros 2009).

Intercambio dinámico de datos (DDE).

DDE es un protocolo para intercambiar datos de manera dinámica entre aplicaciones de Windows que lo permitan. Esto hace que se puedan enviar datos de forma dinámica, en tiempo de ejecución, entre dos aplicaciones. Para este tipo de conectividad una aplicación debe actuar como servidor y la otra como cliente. El cliente establece la conexión, si no hay errores, realiza el intercambio de los datos necesarios y al final del mismo se cierra. Existen DDE para redes, lo que posibilita la transmisión de datos entre máquinas distantes (Ballesteros 2009).

Enlace con objetos.

En aplicaciones sobre Windows pueden ser utilizados métodos y funciones de una aplicación por otra logrando con ello una gran conectividad. Con OLE se brinda una gran flexibilidad para ello y muchas aplicaciones hacen uso de este para intercambiar tipos de datos abstractos. OLE es un ambiente unificado de servicios basados en objetos, con la capacidad de que estos servicios son hechos a la medida. Su arquitectura puede extenderse arbitrariamente con el único propósito de permitir una rica integración entre componentes (Ballesteros 2009).

1.4 Persistencia de datos en la industria.

Una base de datos se define como una serie de datos organizados y relacionados entre sí, y un conjunto de programas que permitan a los usuarios acceder y modificar esos datos. Las bases de datos proporcionan la infraestructura requerida para los sistemas de apoyo a la toma de decisiones y para los sistemas SCADA, ya que éstos explotan la información contenida en las bases de datos de una organización para realizar dicho proceso, por este motivo es importante conocer la forma en que están estructuradas y su manejo. Uno de los propósitos principales de un sistema de base de datos es proporcionar a los usuarios una visión abstracta

de los datos, es decir, el sistema esconde ciertos detalles de como se almacenan y mantienen los datos (Proaño 2006).

Las bases de datos proporcionan las siguientes ventajas:

- Independencia de los datos respecto a los tratamientos y viceversa, lo que evita el importante esfuerzo que origina la reprogramación de las aplicaciones cuando se producen cambios en los datos.
- Coherencia de los resultados, con lo que se elimina el inconveniente de las divergencias en los datos, debidas a actualizaciones no simultáneas en todos los archivos.
- Mejor disponibilidad de los datos para el conjunto de los usuarios junto a una mayor transparencia respecto a la información existente.
- Mayor valor informativo, debido a que los distintos elementos están interrelacionados.
- Mejor y más normalizada documentación de la información, la cual está integrada con los datos.
- Mayor eficiencia en la recuperación, validación y entrada de los datos al sistema (Proaño 2006).

1.4.1 Gestores de bases de datos más utilizados.

Microsoft SQL Server.

Diseñado desde su inicio para trabajar en entornos Internet e Intranet, Microsoft SQL Server es un gestor de base de datos relacional de código abierto que se ha convertido en el más popular del mundo (Proaño 2006).

Microsoft SQL Server revoluciona el concepto de base de datos para la industria ya que reúne en un sólo producto la potencia necesaria para cualquier aplicación empresarial crítica junto a herramientas de gestión que reducen al mínimo el costo de propiedad.

Entre sus principales características se encuentran:

- Escalabilidad: Se adapta a las necesidades de la empresa, soportando desde unos pocos usuarios a varios miles.
- Potencia: Microsoft SQL Server es la mejor base de datos para Windows NT Server. Posee los mejores registros de los *benchmarks*.
- Gestión: Posee una interfaz gráfica que reduce la complejidad innecesaria de las tareas de administración y gestión de la base de datos.
- Orientada al desarrollo: Compatible con muchas herramientas como Visual Basic, Visual C++, Visual J++, y otras (Ibercom 2007).

Este gestor constituye un ejemplo clásico entre los motores de base de datos multihilos, los cuales se encargan de resolver por sí mismos el problema de compartir los recursos de procesamiento, en lugar de dejarlo en manos del sistema operativo (Proaño 2006).

Oracle.

Oracle es un gestor de base de datos relacional (en inglés RDBMS), fabricado por Oracle Corporation, para la gestión de bases de datos. Es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hacen que sólo se vea en empresas muy grandes y multinacionales, por norma general. Para el desarrollo de aplicaciones en Oracle se utiliza el lenguaje de quinta generación PL/SQL, bastante potente para tratar y gestionar bases de datos, aunque también se suele utilizar SQL (Proaño 2006).

Oracle es sin duda uno de los mejores gestores que existen. Es un sistema robusto que garantiza la seguridad e integridad de los datos y una correcta ejecución de las transacciones sin causar inconsistencias. Ayuda a administrar y almacenar grandes volúmenes de datos, proporcionando estabilidad y escalabilidad, y es multiplataforma. Aunque su dominio en el mercado de servidores empresariales ha sido casi total hasta hace poco, recientemente sufre la competencia de gestores de bases de datos comerciales y de la oferta de otros con licencia Software Libre como PostgreSQL o MySQL. Las últimas versiones de *Oracle* han sido certificadas para poder trabajar bajo Linux (Proaño 2006).

PostgreSQL.

PostgreSQL es también un sistema de gestión de base de datos relacional, orientado a objetos y de Software Libre, publicado bajo la licencia BSD. Como muchos otros proyectos de código abierto PostgreSQL no es manejado por una sola compañía, sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales denominada PGDG que trabajan en su desarrollo (Pérez 2009).

Sus principales características son:

- **Alta concurrencia:** mediante un sistema de acceso concurrente multiversión permite que, mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos.
- **Integridad de los datos:** claves primarias, llaves foráneas con capacidad de actualizar en cascada o restringir la acción y restricción *not null*.
- **Resistencia a fallas:** escritura adelantada de registros para evitar pérdidas de datos en caso de fallos por energía, sistema operativo o hardware.
- **Compatibilidad:** Windows, Linux, Unix, BSD's, Mac OS X, Solaris, AIX, Irix, HP/UX
- **Características operativas:** corrección, fiabilidad, eficiencia, integridad y facilidad de uso.
- **Capacidad para soportar cambios:** facilidad de mantenimiento, flexibilidad y facilidad de prueba.

(Pérez 2009).

1.5 Aplicaciones multicapas.

La programación en múltiples capas es la técnica más efectiva en aplicaciones empresariales, debido a la fácil administración que implica el dividir los componentes de la aplicación en capas y la rapidez que esto imprime en programas basados en el modelo Cliente/Servidor. Esta filosofía consiste en dividir los

componentes primarios de la aplicación, programarlos por separado y luego unirlos en tiempo de ejecución (Zapata 2004).

En la actualidad existen dos tipos de arquitecturas sobre las cuales se apoyan los programadores para estos tipos de aplicaciones: la arquitectura de dos capas y la arquitectura de tres capas. A continuación se hará una breve descripción de cada una.

1.5.1 Arquitectura de dos capas.

En una arquitectura de dos capas la aplicación se divide en dos componentes. El primero de ellos reside en la máquina cliente y casi siempre lo compone la interfaz gráfica de usuario; este a su vez se comunica con el otro componente, que llama a la funcionalidad del sistema de base de datos en la máquina servidor mediante instrucciones del lenguaje de consultas (figura 1.6). Los estándares de interfaces de programas de aplicación como ODBC y JDBC se usan para la interacción entre el cliente y el servidor (Zapata 2004).

Figura 1.6 Arquitectura de dos capas.

1.5.2 Arquitectura de tres capas.

En una arquitectura de tres capas, la máquina cliente actúa simplemente como frontal y no contiene ninguna llamada directa a la base de datos, sino que se comunica con un servidor de aplicaciones, usualmente mediante una interfaz de formularios. El servidor de aplicaciones, a su vez, se comunica con el sistema de base de datos para acceder a los mismos (figura 1.7) (Zapata 2004).

En este caso se pueden actualizar las funciones de la aplicación sin tener que cambiar el servidor de aplicaciones y también la aplicación, que establece las acciones a realizar bajo determinadas condiciones, se incorpora en el servidor de aplicaciones, en lugar de ser distribuida a múltiples clientes (Zapata 2004).

Resulta mucho más práctico programar las capas por separado, sobre todo si se tiene un equipo de trabajo de diferentes programadores o el proyecto es para una empresa con varias sucursales con políticas independientes (Zapata 2004).

Figura 1.6 Arquitectura de tres capas.

1.6 El sistema *Tankvision*.

Tankvision es un sistema de control de inventario y monitoreo de tanques, diseñado por la firma alemana Endress Hauser, fundamentalmente para la industria petroquímica. Su funcionamiento se basa en servicios de ingeniería extensiva y dispositivos de campo inteligentes con la utilización de buses de campo estandarizados y software de solución, todo sobre un sistema Web (Endress+Hauser 2010).

Tankvision ha sido creado fundamentalmente para la optimización del almacenamiento y entrega de datos de procesos ofreciendo un inventario de datos de tanques al usuario final a través de un servicio Web (Endress+Hauser 2010).

1.6.1 Características.

Algunas de sus características son:

- Provee variables medidas.
- Realiza y proporciona cálculos de inventario.
- Muestra gráficos de tendencia en tiempo real e histórico.
- Muestra e imprime reportes de inventario.
- Muestra eventos y alarmas.

Como sistema, *Tankvision* ofrece las siguientes prestaciones:

- Interfaz Web configurable.
- No requiere software adicional para ser instalado.
- Es escalable.
- Interfaz de usuario en varios idiomas.
- Comunicación por medio de protocolo estándar de red.

(Endress+Hauser 2010).

1.6.2 Diseño.

El sistema *Tankvision* consta de cuatro módulos:

- **Examen de tanques** (*Tank Scanning*). Se encarga de la adquisición de los datos desde los dispositivos de campo y proporciona la entrada para el cálculo de inventario.
- **Cálculo de Inventario** (*Inventory Calculation*). Realiza el cálculo de inventario basado en la medición de los datos.
- **Almacenamiento de Datos** (*Data Concentration*). Almacena los datos para generar gráficos de tendencias históricas y reportes.
- **Enlace** (*Host Link*). Proporciona una interfaz de comunicación para un DCS.

Físicamente *Tankvision* se divide en tres unidades que se encargan de ejecutar los cuatro módulos mencionados anteriormente, cada unidad es responsable de una operación específica y todas se encuentran conectadas mediante un enlace Ethernet (figura1.7).

La arquitectura física es la siguiente.

NXA 820 (*Tank Scanner*).

Se encarga de la adquisición de los datos medidos por la instrumentación de campo y de proporcionarlos a otra unidad del sistema o visualizarlos, así como del cálculo de inventario y la generación de gráficos de tendencia, tanto históricos como de tiempo real.

NXA 821 (*Data Concentrator*).

Almacena un resumen de los datos medidos y calculados de varios NXA820 para cierto intervalo de tiempo, también genera reportes y tendencias históricas.

NXA 822 (*Host Link*).

Proporciona un enlace Modbus (RTU o TCP/IP) para un DCS.

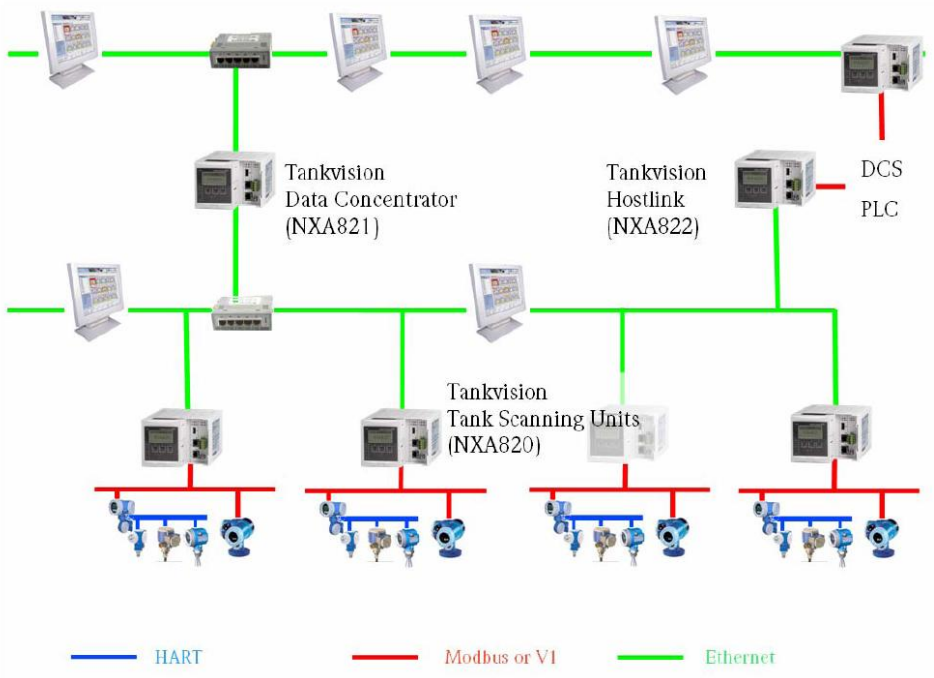


Figura1.7 Arquitectura física de *Tankvision*.

En la refinería de Cienfuegos existe la posibilidad de la observación de los datos medidos y calculados por este sistema, por medio de un archivo XML a través del *Host Link* NXA822 o de los concentradores NXA820, vía Web por las características antes mencionadas y solo a través de la red de control de la planta, pero no se cuenta con un sistema de adquisición de datos, ni con una base de datos dedicada específicamente a almacenarlos.

1.7 Herramientas a utilizar.

El software será escrito en lenguaje Java con el empleo de la plataforma Eclipse, la base de datos será creada con PostgreSQL 8.4.

1.8 Conclusiones parciales del capítulo.

El análisis bibliográfico realizado permite llegar a las siguientes conclusiones:

- El movimiento de Ethernet a la planta ha revolucionado el mundo de las comunicaciones industriales y promete expandir su uso a las más disímiles aplicaciones del control de procesos.
- Con el modelo Cliente/Servidor se alcanza interoperabilidad logrando una verdadera integración con la red administrativa y ejerciendo un mejor monitoreo sobre la producción.
- El esquema Cliente/Servidor empleando el protocolo Modbus TCP/IP proporciona un mecanismo de transporte de datos confiable.
- Las tecnologías y plataformas analizadas permiten un trabajo eficiente para lograr los objetivos planteados.

CAPÍTULO 2. DISEÑO DE LA APLICACIÓN.

En este capítulo se tratan los aspectos más importantes acerca del diseño del servicio en general, en cuanto a la confección del software y de la base de datos describiendo las herramientas fundamentales empleadas en su creación.

2.1 Diseño del Software.

2.1.1 El lenguaje de programación.

Para la selección del lenguaje de programación se han tenido en cuenta las diferentes tecnologías que se usan en la actualidad para la programación de aplicaciones Cliente/Servidor. Existen múltiples lenguajes de programación que pueden ser usados para la programación de este tipo de aplicaciones. La plataforma Microsoft.NET permite crearlas con múltiples variantes que brindan potentes funcionalidades. No obstante se ha elegido a Java, principalmente por ser un lenguaje de alto nivel, ejecutable en máquina virtual e independiente de la plataforma y del sistema operativo prestando muchas funcionalidades para TCP/IP.

Java fue creado y lanzado al mercado por Sun Microsystems Inc. a principios de 1995, fecha a partir de la cual se difunde a una velocidad vertiginosa, añadiéndosele funcionalidad para TCP/IP. Es un lenguaje de desarrollo de propósito general orientado a objetos y aunque comenzó a ser conocido como un lenguaje de programación de *applets* (componente de una aplicación que se ejecuta en el contexto de otro programa) puede ser utilizado para construir cualquier tipo de proyecto. Su sintaxis es muy parecida a la de C y C++ pero no constituye una evolución de éste ni es un C++ mejorado (Fernández 2004).

Otra característica importante de *Java* es que los programas ejecutables creados por el compilador, son independientes de la arquitectura, o sea, se ejecutan indistintamente en una gran variedad de equipos con diferentes microprocesadores y sistemas operativos. Es público, con posibilidades de adquirir JDK gratis. Además, las aplicaciones son fiables, ya que posee un gestor de seguridad con el que puede restringir el acceso a los recursos del sistema, es capaz de gestionar permisos y criptografía. También garantiza la seguridad frente a virus a través de redes locales e Internet. (Fernández 2004)

2.1.2 La plataforma Eclipse.

Eclipse es básicamente una plataforma creada para entornos de desarrollo integrados (en inglés IDE), que permite además de integrar diversos lenguajes, introducir otras aplicaciones que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, etc., todo esto gracias a una arquitectura basada en *plugins* (enchufes) además aprovecha al máximo las potencialidades de Java para la escritura de herramientas (Object Tehcnology International 2003).

También permite las ventajas siguientes:

- Apoya la construcción de una variedad de herramientas para el desarrollo de aplicaciones.
- Apoya un conjunto ilimitado de proveedores de herramientas, incluyendo los proveedores de software independientes.
- Facilita la integración sin fisuras de herramientas entre distintos tipos de contenidos y proveedores.
- Apoya las aplicaciones en entornos de desarrollo con y sin GUI.
- Es ejecutable en una amplia gama de sistemas operativos (Windows y Linux).

La principal función de la plataforma Eclipse es dotar a los usuarios de herramientas con los mecanismos a utilizar, y las reglas a seguir, que conducen a

la perfección las herramientas integradas. Estos mecanismos están expuestos a través de interfaces API bien definidas, clases y métodos. La plataforma también proporciona bloques de construcción útiles y los marcos que faciliten el desarrollo de nuevas herramientas (Object Tehcnology International 2003).

2.1.3 Ingeniería del software.

En el diseño del sistema se han tenido en cuenta diferentes diagramas de casos de uso así como diagramas de secuencia. En la figura 2.1 se muestra el diagrama de casos de uso generales del sistema, en el mismo la “Aplicación Cliente” es un actor que obtiene los datos a partir de una conexión implementando un *master* Modbus TCP con el *Host Link* NXA822, dichos datos serán almacenados luego, en una base de datos relacional por el propio software para su uso con diversos fines.

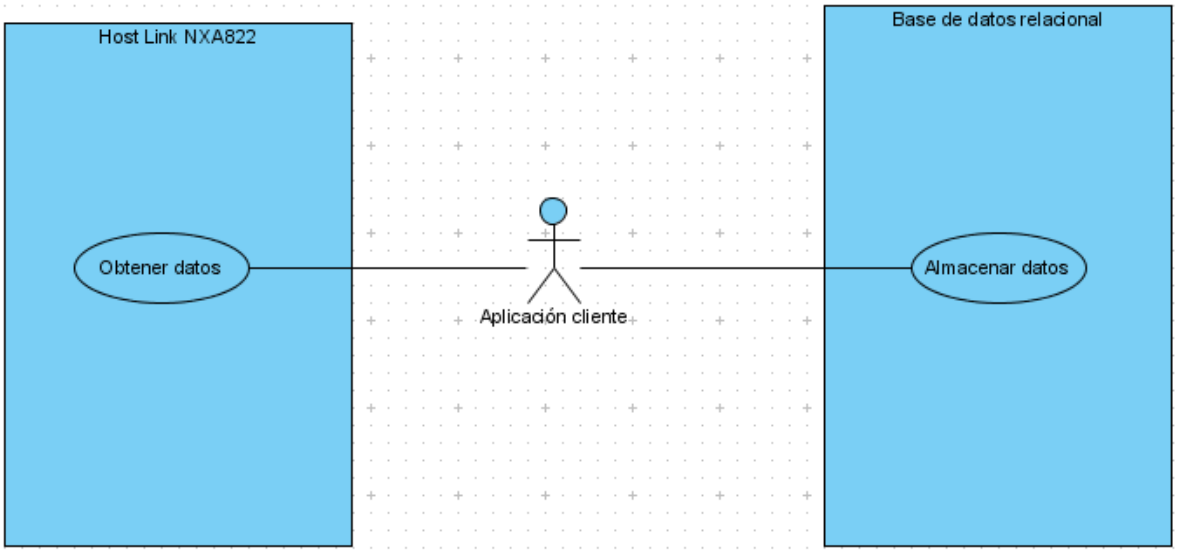


Figura 2.1 Diagrama de casos de uso generales del sistema.

En la figura 2.2 se muestra el actor “Administrador” que administra el servicio. Este actor se encarga de la instalación y el mantenimiento correcto del mismo.

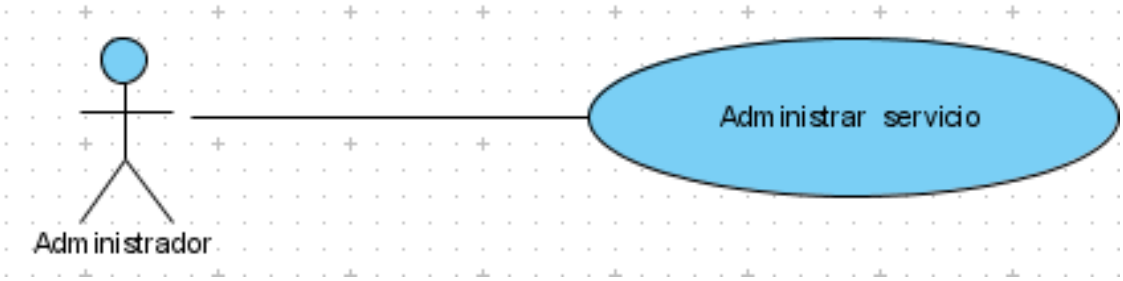


Figura 2.2 Diagrama de caso de uso “Administrar servicio”.

El caso de uso ““Administrar servicio” es una generalización de “Actualizar datos de configuración” quien a su vez lo es de los casos de uso: “Cargar datos de comunicación”, “Cargar datos de transacción” y “Cargar datos JDBC” (figura 2.3).

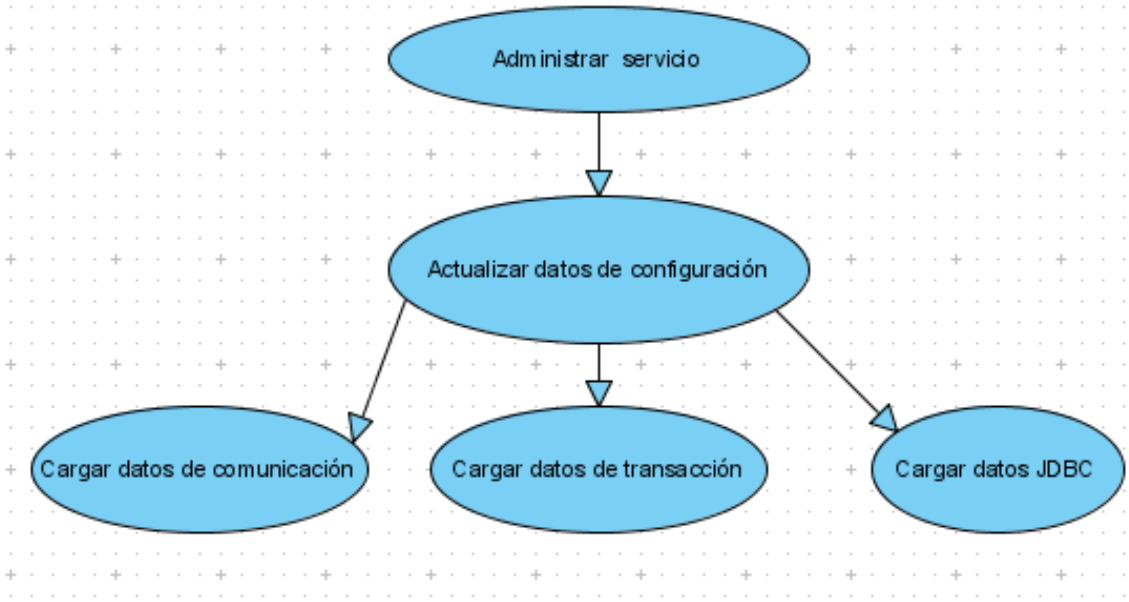


Figura 2.3 Generalización del caso de uso “Administrar servicio”.

El caso de uso “Cargar datos de comunicación” es una generalización de los casos de uso: “Cargar IP esclavo”, “Cargar ID esclavo” y “Cargar puerto” (figura 2.4).

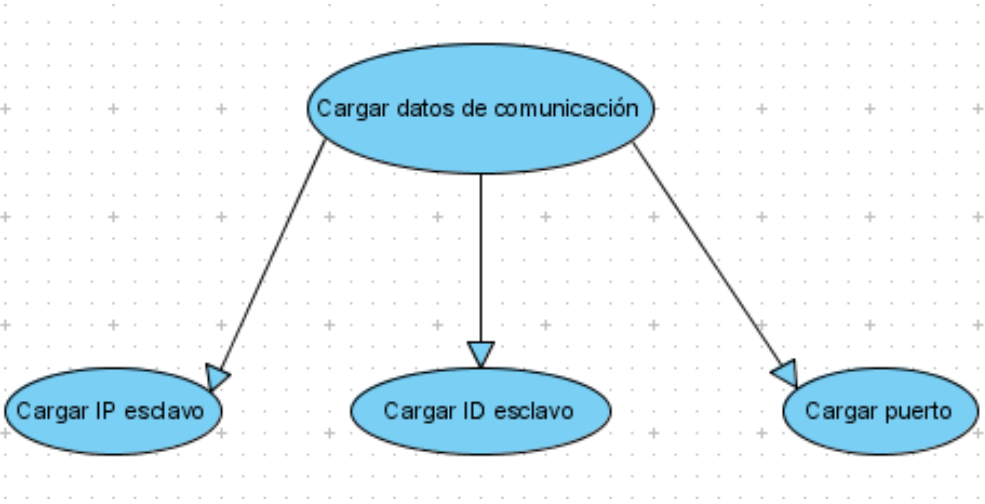


Figura 2.4 Generalización del caso de uso “Cargar datos de configuración”.

Donde se cargan la dirección IP y el ID del *Host Link* NXA822 así como el puerto para iniciar la comunicación.

El caso de uso “Cargar datos de transacción” es una generalización de los casos de uso “Tomar referencia”, “Cargar registros a contar”, “Cargar período de muestreo” y “Cargar solicitudes de comunicación” (figura 2.5).

Aquí se obtiene el registro de referencia en memoria a partir del cual se va a inicializar la adquisición de los datos, la cantidad de registros a contar a partir de la referencia señalada, el período de muestreo y el número de solicitudes de comunicación al esclavo.

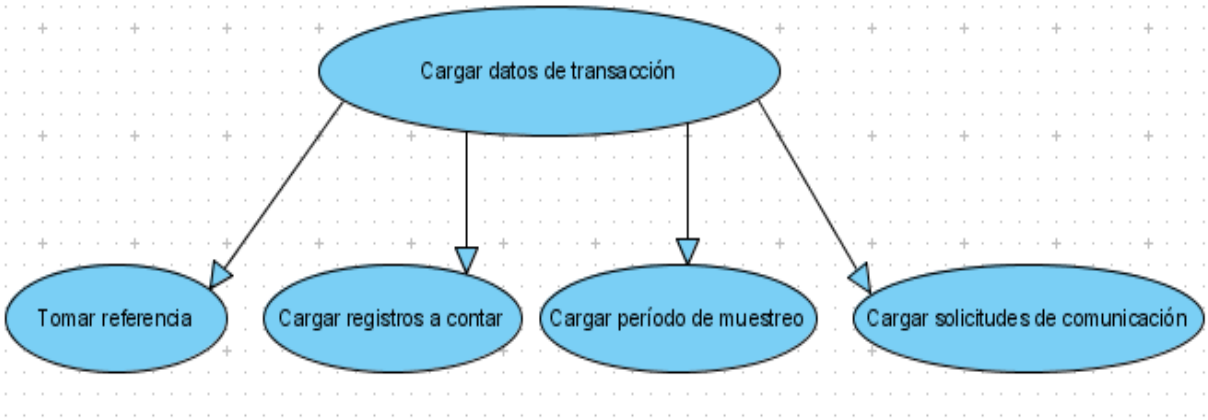


Figura 2.5 Generalización del caso de uso “Cargar datos de transacción”.

El caso de uso “Cargar datos JDBC” es una generalización de los casos de uso “Tomar formato fecha”, “Tomar URL de BD”, “Tomar nombre TD”, “Cargar máxima entrada por tabla” y “Cargar *driver* JDBC”(figura 2.6).

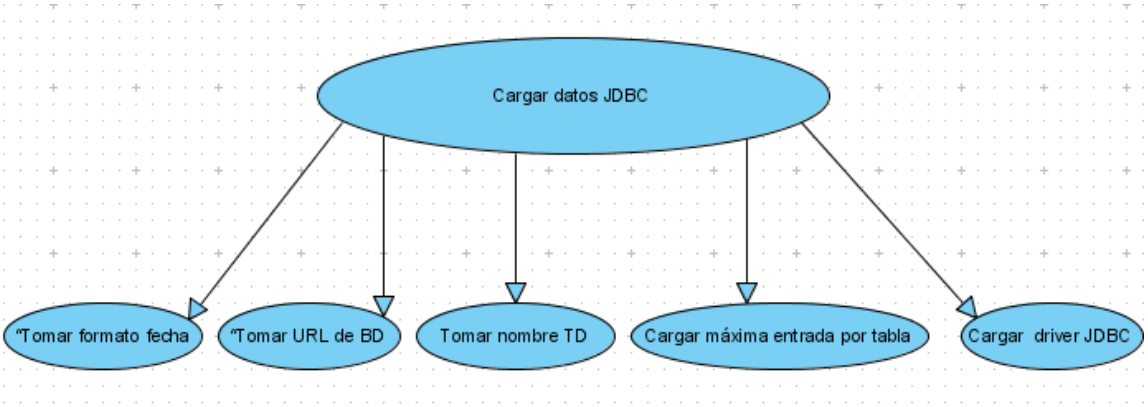


Figura 2.6 Generalización del caso de uso “Cargar datos JDBC”.

En estos casos de uso se realizan las siguientes acciones: se obtiene el formato en que se observará la fecha y la hora de la adquisición de los datos, se obtiene la URL de la base de datos así como el nombre de la tabla de datos a crear, la máxima entrada por tablas (tamaño de la tabla de datos) y el nombre del *driver* para la comunicación con la base de datos, respectivamente.

En la figura 2.5 se muestra un diagrama de secuencia general del servicio, primero se realiza el pedido de un segmento de registros por parte del software al *Host Link* NXA822 y luego él mismo realiza la inserción de los datos.

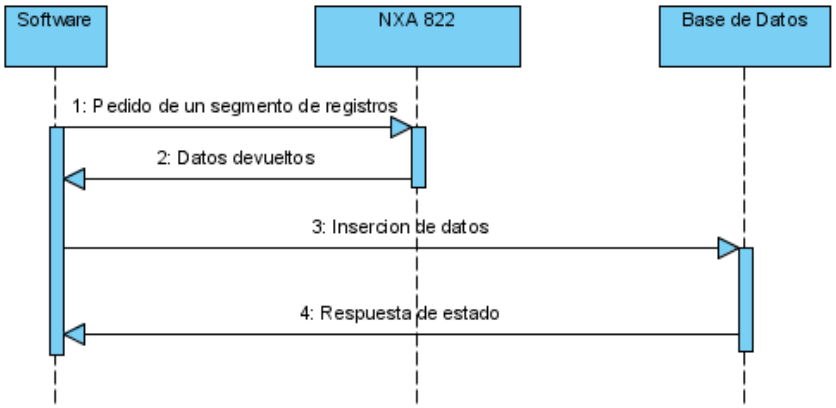


Figura 2.5 Diagrama de secuencia general del sistema.

En la figura 2.6 se muestra en un diagrama de secuencia la interacción de los objetos para realizar la comunicación y la adquisición de los datos.

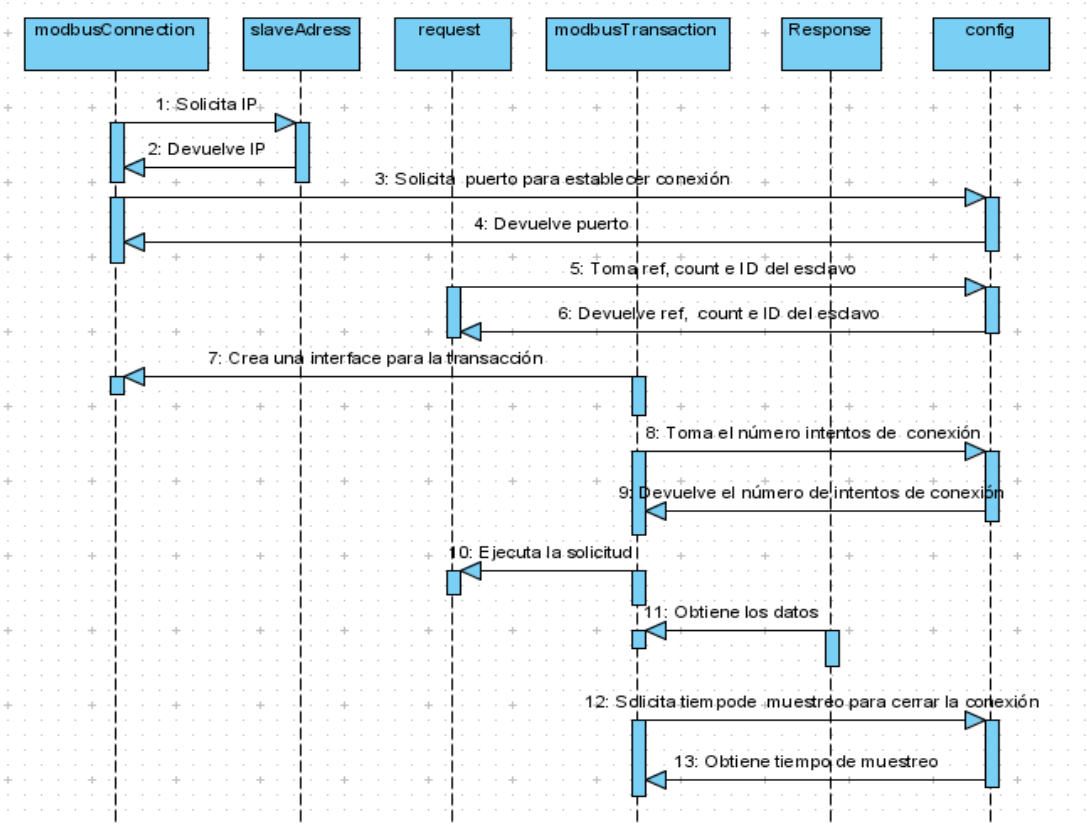


Figura 2.6 Diagrama de secuencia de la adquisición de los datos.

El objeto ***config*** porta los parámetros de configuración para la conexión y la transacción, la comunicación se establece con ***modbusConection*****,** cuando solicita IP y puerto de comunicación, ***request*** toma la referencia y la cantidad de registros que va a leer a partir de ella, ***modbusTransaction*** crea una interface de comunicación con ***modbusConection*** y lee de ***config*** el número de veces que va a solicitar la conexión, ejecuta la solicitud a ***request*** y es accedido por ***response*** para obtener los datos. Finalmente solicita el tiempo de muestreo para el cierre de la conexión, la cual se realiza por medio de una excepción.

La conexión con la base de datos se explica en el diagrama de secuencia de la figura 2.7. Primeramente se carga el *driver* para la comunicación JDBC, para

luego crear un objeto de conexión con la función ***getConnection,*** posteriormente se produce un objeto ***Statement*** con ***CreateStatement*** para realizar la ejecución de sentencias SQL sobre la base de datos.

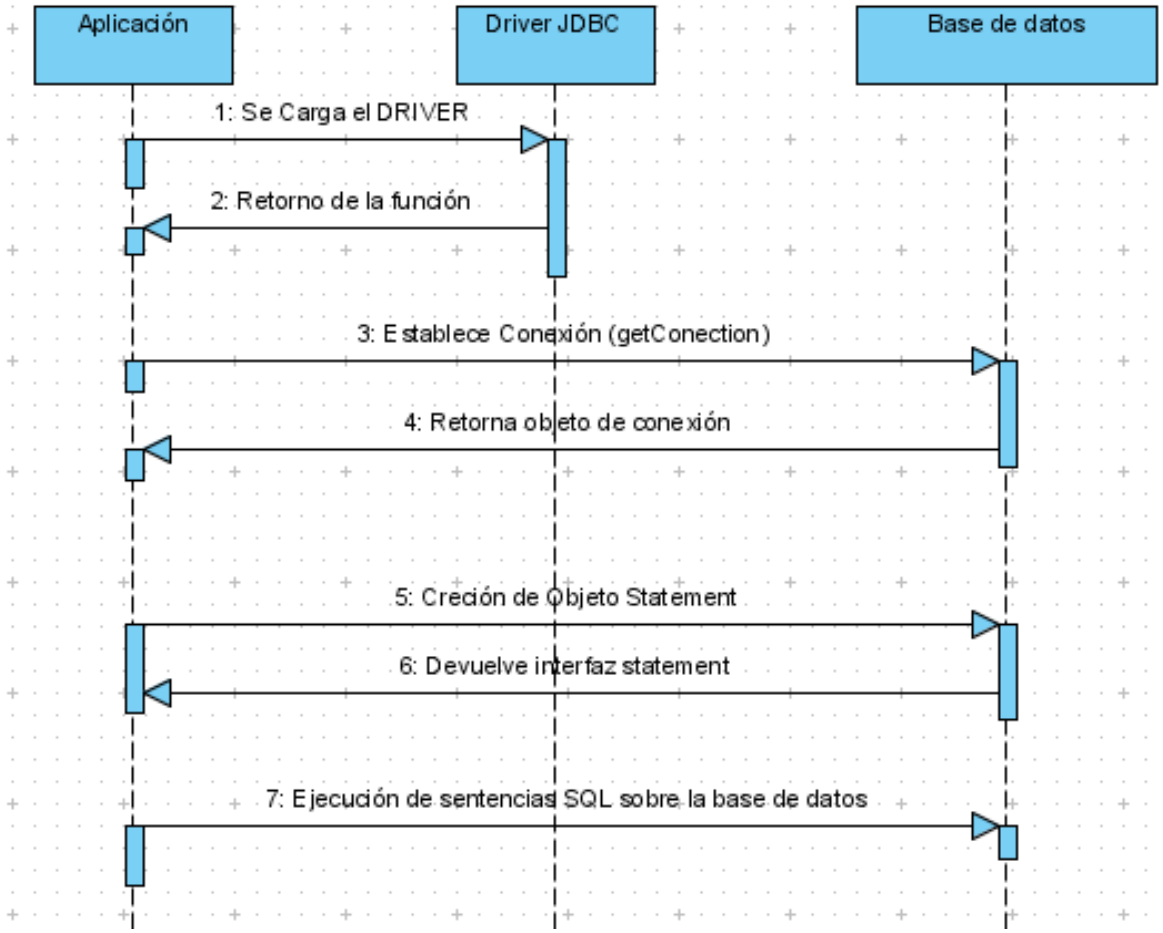


Figura 2.7 Establecimiento de la conexión con la base de datos.

2.1.4 Herramientas adicionales utilizadas en la implementación y estructura lógica del software.

En la escritura del código se han empleado diferentes librerías (Figura 2.7). Primeramente, ***commons logging 1.1.1*** la cual actúa como un puente a diferentes librerías y un complemento a la generación de código que retribuye calidad a las aplicaciones que lo usan, además permite registrar estados de ejecución, de tal

forma que sus usos más frecuentes son: registros de uso, prueba y detección de errores. Este componente normalmente se utiliza para quitar dependencias en tiempo de compilación o ejecución a cualquier paquete de *logs (*registros*)*, y ofrecer un interfaz común para generarlos independientemente de la librería elegida (Apache_Commons 2008).

De la librería ***commons logging*** se usa el paquete ***org.apache.commons.logging*** que envuelve múltiples API de *loggings*, con las clases ***Log****,* ***LogFactory*** ***LogSource*** y ***LogConfigurationException*** *entre otras, así como* el paquete ***org.apache.commons.logging.impl*** para la implementación de las mismas (Apache_Commons 2008).

La biblioteca ***jamod 1.2*** permite la implementación del protocolo Modbus TCP con los paquetes:

- ***net.wimpi.modbus.msg*** que proporciona las interfaces y las clases que encapsulan los mensajes Modbus en un objeto de manera orientada.
- ***net.wimpi.modbus.io*** para crear interfaces de entrada/salida para el transporte.
- ***net.wimpi.modbus.net*** para la implementación del protocolo en la red.

También se emplean los módulos ***beans*** y ***context*** entre otros de ***springframework*** para el uso de la inyección de dependencia (DI) así como ***xmldriver*** para el *mapping* con archivos XML y ***sqljdbc*** para la conexión e interacción con la base de datos.

El software cuenta con dos ficheros fuente. El primero es el llamado ***Aplication*** donde se cargan los datos de configuración, se ejecutan los mensajes correspondientes para abrir la conexión, se realiza la transacción de los datos implementando la función 04 de Modbus, (*Read Input Register*) para luego cerrar la conexión.

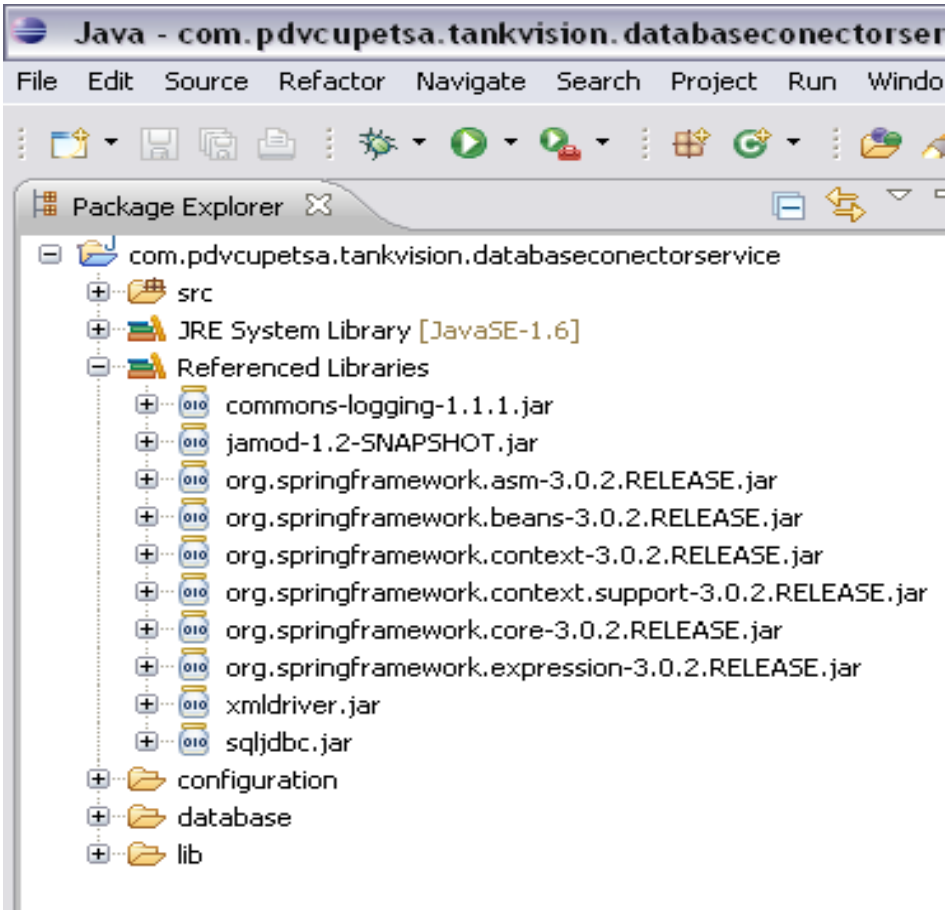


Figura 2.7 Árbol de librerías empleadas.

También se carga el *driver* JDBC se establece la comunicación con la base de datos, y se manipula la misma por medio de sentencias SQL (creación de tablas de datos y su almacenamiento en la tabla de tablas).

En el fichero ***AplicationConfiguration*** se almacenan los parámetros de configuración: período de muestreo de los datos, dirección IP e ID del esclavo y el puerto de comunicación, el registro de referencia para iniciar la transacción y el número de registros que se van a contar a partir de este, número de intentos de la comunicación, así como la conexión URL con la base de datos, el nombre del *driver* de conexión JDBC y el formato de fecha y hora de la adquisición de los datos, la máxima entrada por tabla de datos y el nombre de la misma, para uso de ***Aplication.***

2.2 La Base de Datos.

2.2.1 PostrgeSQL 8.4.

La base de datos se creó con PostgreSQL en su versión 8.4, debido a que este cumple con los requerimientos necesarios para el correcto desempeño de esta aplicación.

PostgreSQL 8.4 es de alto desempeño, soporta la arquitectura Cliente/Servidor y es desarrollado con licencia de Software Libre. El núcleo PostgreSQL 8.4 está disponible en código fuente y varios formatos binarios (PostgreSQL 2009).

Entre las principales características que presenta esta versión se encuentran:

- Implementación de funciones avanzadas en estándar ANSI SQL, añadiendo extensiones no estándares para permitir a los usuarios realizar tareas más complejas a través de sus características objeto-relacionales.
- Es más fácil de administrar.
- Ofrece mayor seguridad, con la utilización de herramientas fáciles de usar que hacen de la conexión y el control de acceso un servicio más flexible.
- Añade nuevas características de supervisión y herramientas que aumentan la idoneidad para entornos de gran empresa.
- Implementación de nuevas funciones: matemáticas, estadísticas, XML, fecha y hora y operaciones de manipulación de texto para los usuarios. (PostgreSQL 2009).

2.2.2 Estructura y Diseño.

La base de datos relacional consta de un diseño sencillo que, descansando sobre una estructura modular, facilita su uso, para la realización de la misma se tuvieron en cuenta los siguientes aspectos:

1. Se necesita un registro de datos histórico que almacene los valores de un número determinado de variables (actualmente dos: **P_LEVEL** (Nivel de producto) y **P_TEMP** (Temperatura del producto)) de los casi 80 tanques

de crudo, productos intermedios y productos terminados del patio de tanques de la refinería.

2. El número determinado de variables del cual se habla en el punto uno, depende de la configuración dada al *Host Link* NXA822, otras variables a almacenar pueden ser: **V_PRESS** (Presión de vapor), **W_LEVEL** (Nivel de agua en el fondo), **V_TEMP** (Temperatura del vapor), etc. Por tanto se necesita funcionalidad y flexibilidad en el servicio para cuando sea cambiada la configuración del *Host Link* NXA822 y se necesiten leer más variables.
3. El *Host Link* NXA822 almacena los datos, para una configuración dada, uno a continuación del otro en sus registros de lectura a partir del 30 000, mostrándolos al usuario por medio un archivo XML en grupos de *Data Concentrator* NXA820 (los cuales a su vez atienden un número de tanques cada uno (figura 2.8)). Básicamente el software lo que hace es escanear esos registros y tomar los datos que hay en ellos en ese momento, para luego insertarlos en la base de datos, de manera que cuando no solo se guarde temperatura y nivel para cada tanque, y se cambie la configuración del *Host Link* NXA822 cambiará la posición del dato en cada registro.
4. También es necesaria la flexibilidad cuando sean añadidos nuevos tanques al sistema.

Por todo lo antes expuesto se decidió dar la siguiente estructura a la base de datos.

Serán creadas tantas tablas de datos como sean necesarias, cada una de ellas se irá creando y llenando con los datos hasta un número límite prefijado en el fichero de configuración. Cada tabla de datos consta de tres columnas: en la primera se hallan los identificadores de los tanques, en la segunda los nombres de las variables a medir para cada tanque y en la tercera los valores de las mismas.

```
<?xml version="1.0" ?>
- <FG4HL_MODBUS_PARAM_MAP CRC="0">
    <MAP_ORIENTATION>Data</MAP_ORIENTATION>
  - <MAP_ELEMENTS>
    - <ELEMENT>
        <!-- Reg 0  Level  -->
        <Name>P_LEVEL</Name>
        <Scalar>1.0</Scalar>
        <Offset>0.0</Offset>
        <Packing_Format>IEEE754</Packing_Format>
      </ELEMENT>
    - <ELEMENT>
        <!-- Reg 2  Temperature  -->
        <Name>P_TEMP</Name>
        <Scalar>1.0</Scalar>
        <Offset>0.0</Offset>
        <Packing_Format>IEEE754</Packing_Format>
      </ELEMENT>
    </MAP_ELEMENTS>
  - <BLOCKS>
      <BLOCK_START>30001</BLOCK_START>
    </BLOCKS>
  - <OVERRIDEBLOCKS>
      <BLOCK_START>40001</BLOCK_START>
    </OVERRIDEBLOCKS>
  - <TANKS>
    - <TANK>
        <IP_ADDR>NXA820_TKSC01</IP_ADDR>
        <ID>1</ID>
      </TANK>
    - <TANK>
        <IP_ADDR>NXA820_TKSC01</IP_ADDR>
        <ID>2</ID>
      </TANK>
    - <TANK>
        <IP_ADDR>NXA820_TKSC01</IP_ADDR>
        <ID>3</ID>
      </TANK>
    - <TANK>
        <IP_ADDR>NXA820_TKSC01</IP_ADDR>
        <ID>4</ID>
      </TANK>
    - <TANK>
```

Figura 2.8 Segmento de Archivo XML donde se muestran los datos almacenados por el *Host Link* NXA822.

Cada tabla una vez finalizada, será almacenada en una tabla de tablas que contendrá en la primera columna los nombres de las tablas de datos creadas hasta ese momento y en la segunda la fecha en que se empezó a escribir cada una, así el usuario podrá acceder a los datos de acuerdo a la fecha en que fueron obtenidos y almacenados. Los nombres de las tablas de datos comenzarán con **"*points_XX*"** done XX será el número del orden en que fueron creadas.

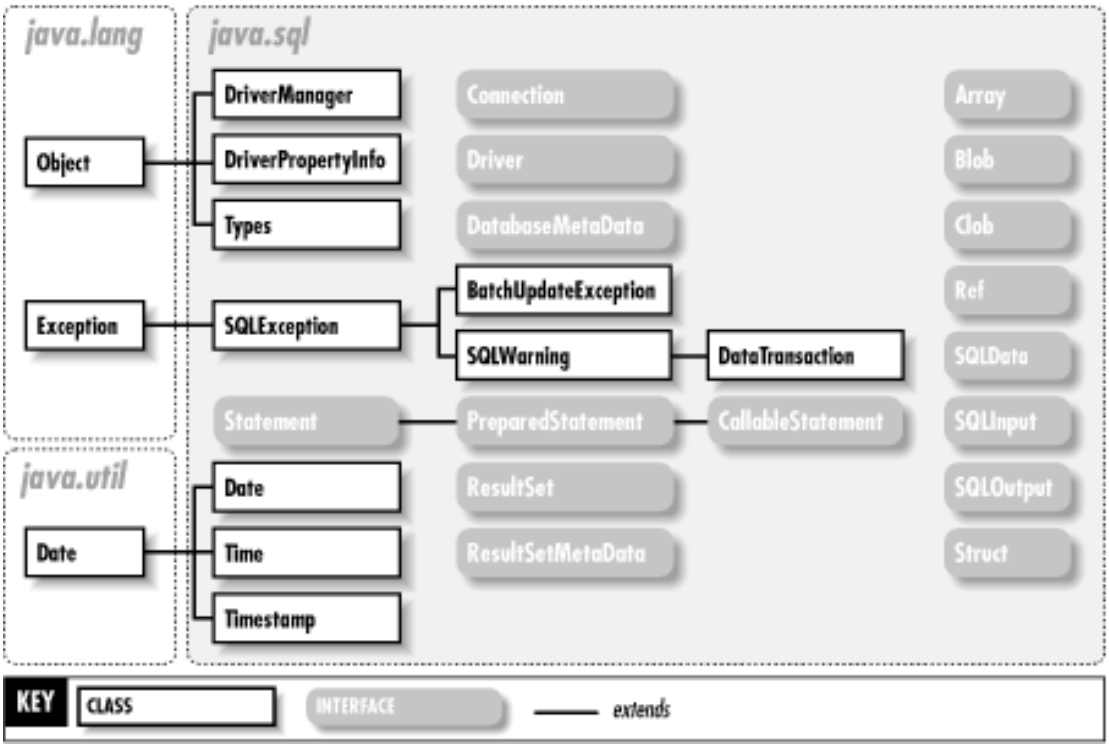


Figura 2.9 Jerarquía de clases de java.sql.

Para acceder e interactuar con la base de datos se utilizó el paquete ***java.sql***, el cual contiene toda la API de JDBC que envía declaraciones a bases de datos relacionales y recupera los resultados de la ejecución de las sentencias. En la figura 2.9 se muestra la jerarquía de clases de este paquete, la interfaz de ***Driver*** representa una aplicación específica JDBC para un sistema de base de datos determinado, ***Conection*** representa una conexión a una base de datos específica. Las declaraciones, ***PreparedStatement*** y ***CallableStatement*** son interfaces de apoyo a la ejecución de diversos tipos de sentencias SQL. ***ResultSet*** es un conjunto de resultados devueltos por la base de datos en respuesta a una consulta SQL. La interfaz ***ResultSetMetaData*** proporciona metadatos acerca de un conjunto de resultados, mientras que ***DatabaseMetaData*** los proporciona sobre la base de datos en su conjunto (O'Reilly 2001).

2.3 Software Libre.

De forma general la implementación de la aplicación se basa en su mayor parte en la selección de Software Libre, tanto en el soporte técnico como en los lenguajes de desarrollo. Las razones de esta selección se argumentan a continuación.

El Software Libre permite a los usuarios ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el producto, lo cual incluye:

- Libertad de usar el programa con cualquier propósito.
- Libertad de estudiar cómo funciona el programa y adaptarlo a sus necesidades.
- Libertad de distribuir copias.
- Libertad de mejorar el programa y hacer públicas la mejoras de modo que toda la comunidad se beneficie.

(Rodríguez 2008).

Entre las ventajas en el uso del Software Libre podemos enunciar:

- Beneficios sociales y tecnológicos a quien los usa y al país en general
- Ahorro en la adquisición de licencias.
- Combate efectivo a la copia ilícita de software.
- Eliminación de barreras presupuestarias.
- Tiempos de desarrollo menores, por la amplia disponibilidad de herramientas y bibliotecas.

(Rodríguez 2008).

2.4 Conclusiones parciales del capítulo.

El uso de los métodos y herramientas antes mencionadas y el análisis de los resultados obtenidos hasta el momento nos permiten arribar a las siguientes conclusiones.

- Al implementar el protocolo Modbus TCP con la librería ***jamod*** se evidencia la facilidad y flexibilidad del mismo y con ello la razón de su alta aceptación en entornos industriales.

- PostgreSQL 8.4 brinda las potencialidades suficientes para la realización y explotación de la base de datos con la calidad requerida por este tipo de aplicación.

- La escritura del código en Software Libre posibilita su viabilidad, flexibilidad adaptabilidad así como su conocimiento público para posteriores aportes y renovaciones.

CAPÍTULO 3. EXPERIMENTOS Y MANUAL DE USUARIO DEL SERVICIO *DATABASECONECTORSERVICE*.

En este capítulo se muestran los pasos a seguir en dos experimentos diseñados para validar los resultados obtenidos, así como un manual de usuario para la correcta explotación de este servicio.

3.1 Diseño de experimentos.

Han sido diseñados dos experimentos con el fin de probar la funcionalidad del servicio.

Experimento No 1: Conexión remota con otra PC.

Para la realización de este experimento es necesario ejecutar los siguientes pasos:

1. Ejecutar la instalación del software ModSim32, simulador de un esclavo empleado por Endress Hauser para este tipo de pruebas.

2. En el menú ***File*** seleccionar ***New*** para acceder a la ventana de trabajo, luego configurar allí los parámetros ***Adress, Length*** y ***Device id*** tal como se muestra en la figura 3.1, también configurar el ***Function Code***.

3. Configurar la conexión Modbus TCP entrando el puerto de comunicación (figura 3.2).

4. Ejecutar ***Databaseconectorservice*** (ver manual de usuario, sección: “Configuración y ejecución del servicio”) desde otra PC en la red, el IP del esclavo será el de la propia PC donde corre ModSim32.

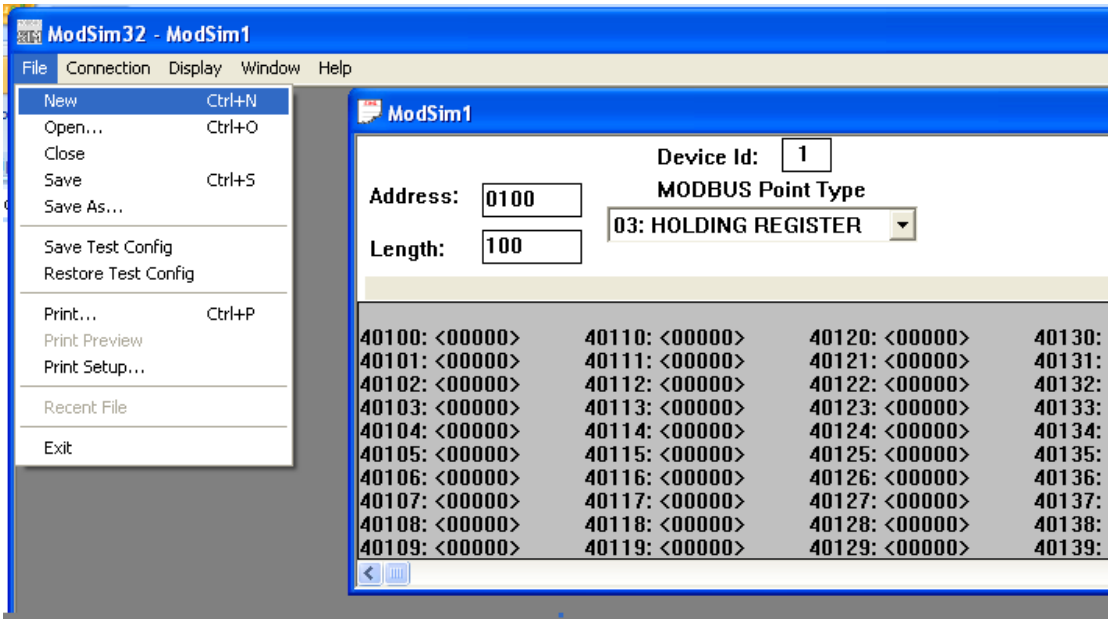


Figura 3.1 Ventana principal de ModSim32.

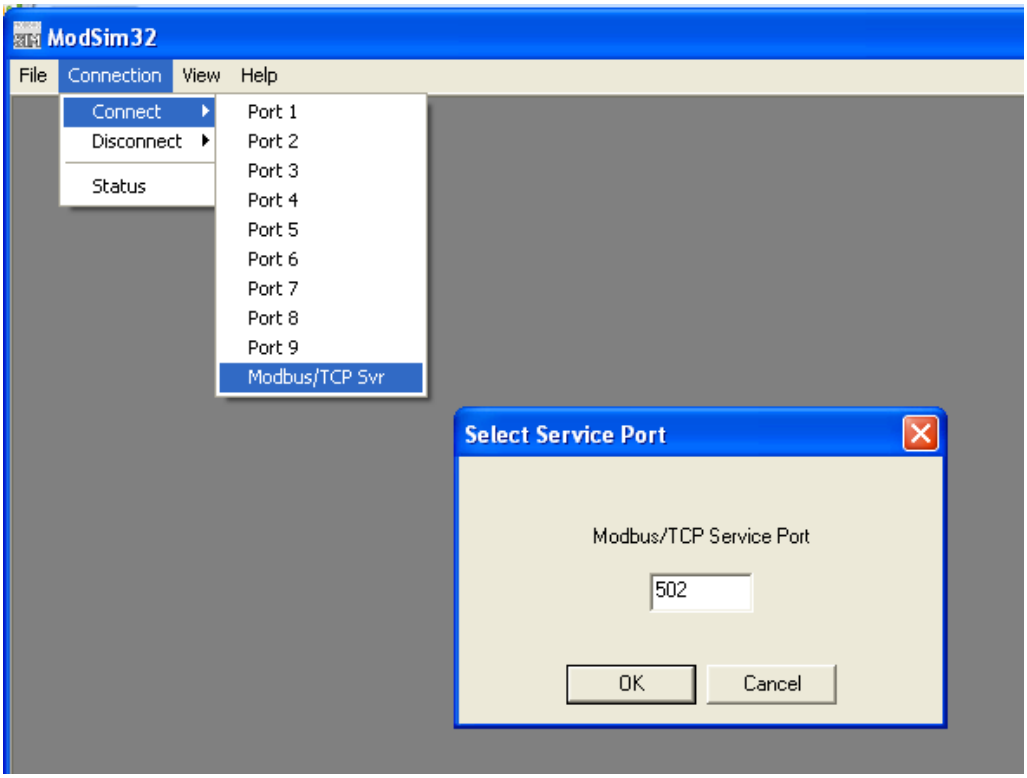


Figura 3.2 Configuración de ModSim32 para Modbus TCP.

Con la ejecución de este experimento se obtuvieron lo resultados esperados demostrándose la funcionalidad del servicio.

Experimento No 2: Conexión Con el *Host Link* NXA822.

Para la realización de este experimento es necesario ejecutar los siguientes pasos:

1. Configurar el *Host Link* NXA822 como esclavo (ver manual de usuario, sección: “Configuración del *Host Link* NXA822”).

2. Ejecutar ***Databaseconectorservice*** (ver manual de usuario, sección: “Configuración y ejecución del servicio”) desde otra PC en la red de control o conectado directamente al *Host Link* NXA822 vía LAN.

Este experimento no pudo ser ejecutado por razones de seguridad del *Tankvision*.

3.2 Manual de usuario.

Configuración y ejecución del servicio.

Este es un manual de usuario dirigido al administrador del servicio.

Databaseconectorservice es un servicio que, basado en la arquitectura Cliente/Servidor sobre Modbus TCP, ha sido creado para la adquisición de datos, desde el *Host Link* NXA822 del sistema *Tankvision* y su posterior almacenamiento en una base de datos relacional creada con PostgreSQL 8.4. No se garantiza un correcto desempeño y funcionamiento del mismo en aplicaciones fuera del marco especificado anteriormente.

Las aplicaciones programadas en lenguaje Java, necesitan una máquina virtual para ejecutarse, por eso antes de correr esta aplicación en un servidor es necesario instalar el JRE disponible en http://java.softonic.com/.

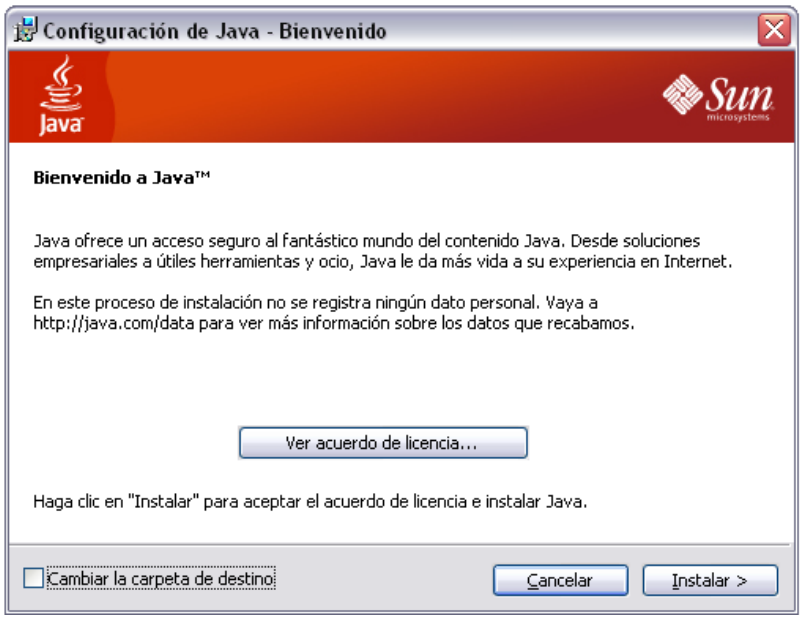


Figura 3.3 Ventana del programa de instalación de JRE.

JRE es la máquina virtual básica de Java, sin ella ningún programa Java lograría ejecutarse ya que su instalador se encarga de integrar Java con el sistema. Sólo es necesario pulsar "Instalar" y luego "Siguiente" hasta que la instalación se haya completado (figura 3.3). Para configurarla, se encontrará, en el Panel de Control, un icono dedicado exclusivamente a Java. JRA soporta los formatos *.java* y *.jar*. Para utilizarla se necesitan sistemas operativos Windows 2000 en adelante (Intershare 2010).

El paquete del servicio ***Databaseconectorservice*** consta de tres carpetas y el fichero ejecutable (figura 3.4).

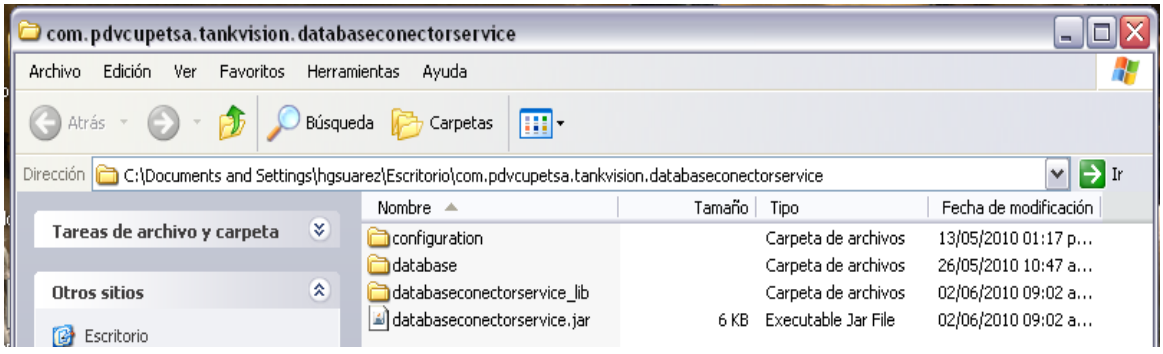


Figura 3.4 Paquete de ***Databaseconectorservice***.

En la carpeta ***Configuration*** se encuentra un archivo XML para la misma, los parámetros a configurar se muestran a continuación (figura 3.5).

- Período de muestreo de los datos.
- Dirección IP del esclavo.
- Puerto de comunicación.
- ID del esclavo.
- Número de solicitudes de comunicación.
- Registro de referencia.
- Numero de registros a contar.
- URL de la base de datos.
- Nombre del *driver* JDBC.
- Formato de fecha y hora.
- Máxima entrada por tabla de datos.
- Nombre de la tabla de datos.

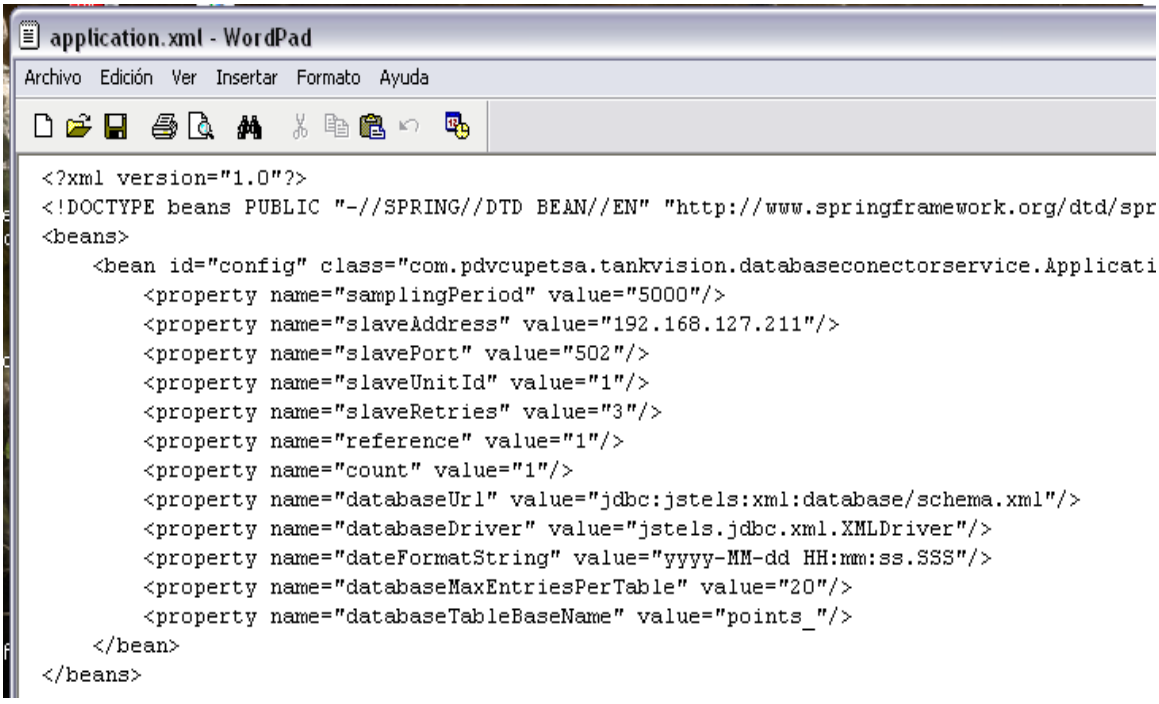


Figura 3.5 Fichero XML de configuración de ***Databaseconectorservice***.

En la carpeta ***Database*** se encuentra el fichero ***point_0.xml***, el cual se llena con los datos adquiridos al mismo tiempo que estos son insertados a la base de datos, la carpeta ***Databaseconectorservice_lib*** contiene las librerías utilizadas.

Teniendo en cuenta que esta aplicación correrá como un servicio no se le ha sido desarrollada una interfaz gráfica, la instalación de la misma se hará por la consola de Windows, situándose el usuario primeramente en el directorio donde se encuentra el ejecutable de la aplicación y luego ejecutando los comandos ***java -jar*** seguido del nombre de la aplicación como sigue: ***databaseconectorservice.jar*** (figura 3.6).

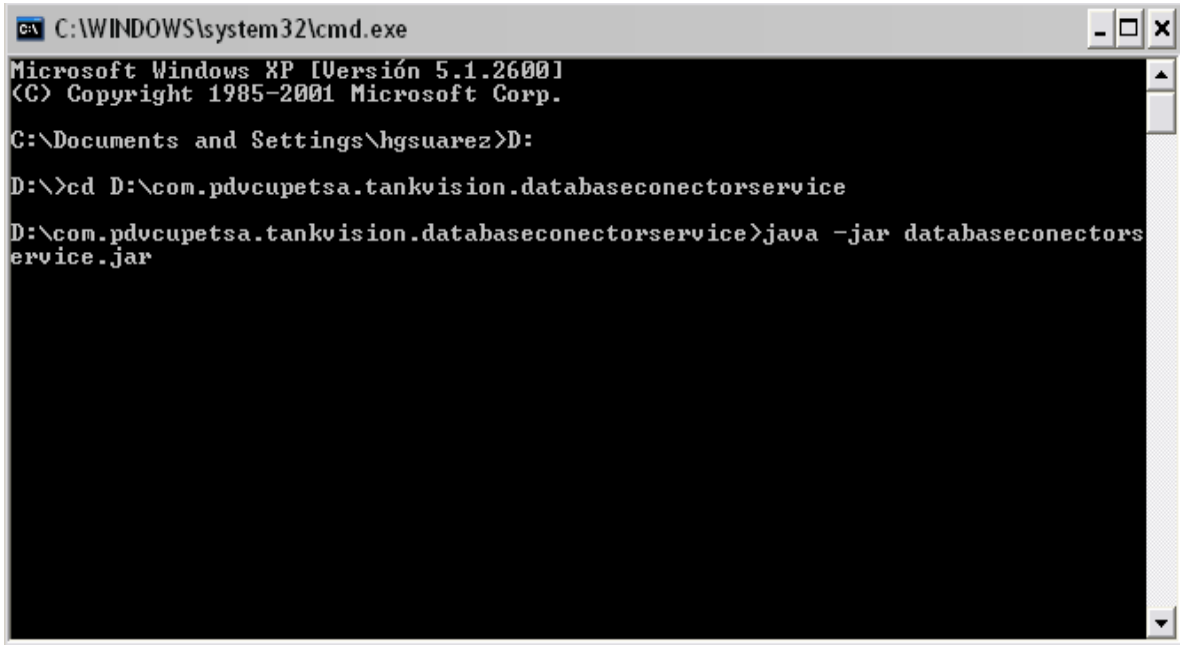


Figura 3.6 Ejecución de ***Databaseconectorservice*.**

Configuración del *Host Link* NXA822.

Para configurar el *Host Link* NXA822 deben seguirse los siguientes pasos.

1. Dar clic en el menú ***System*** en el panel izquierdo de la ventana principal de *Tankvision* (figura 3.7) y luego seleccionar ***Global Settings***.

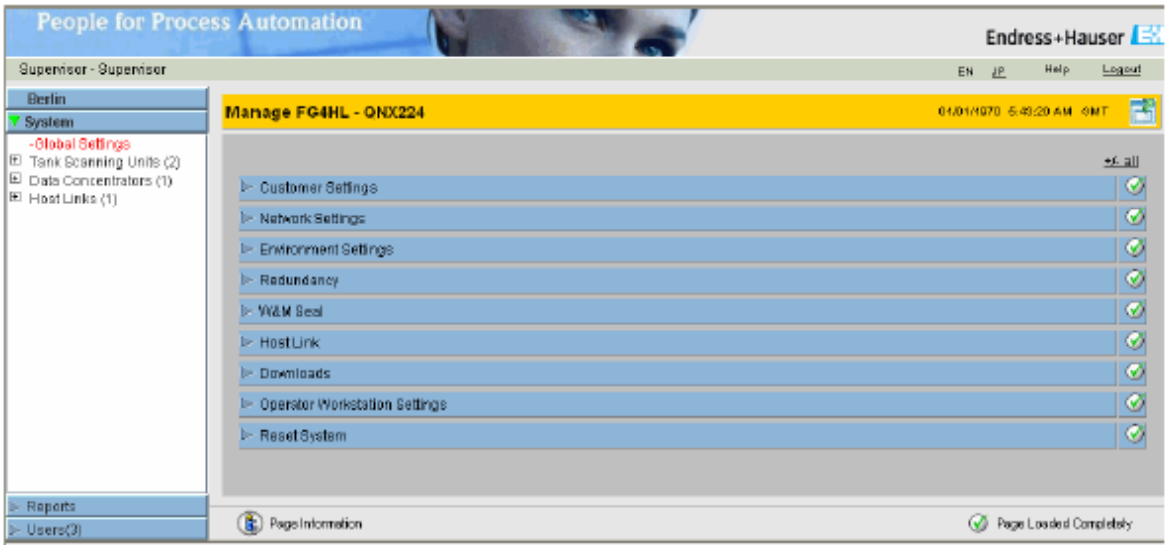


Figura 3.7 Ventana principal de Tankvision.

2. En ***Host Link*** elegir ***Host Link Configuration*** y luego seleccionar ***Modbus TCP*** (figuras 3.8 y 3.9).

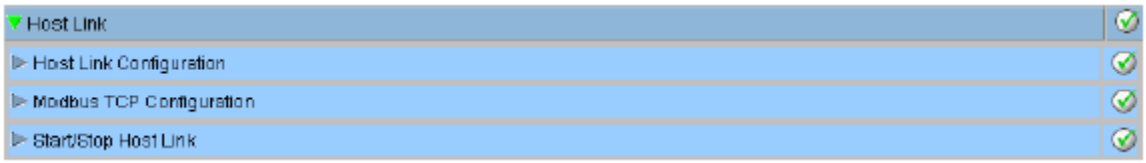


Figura 3.8 Submenú ***Host Link***.

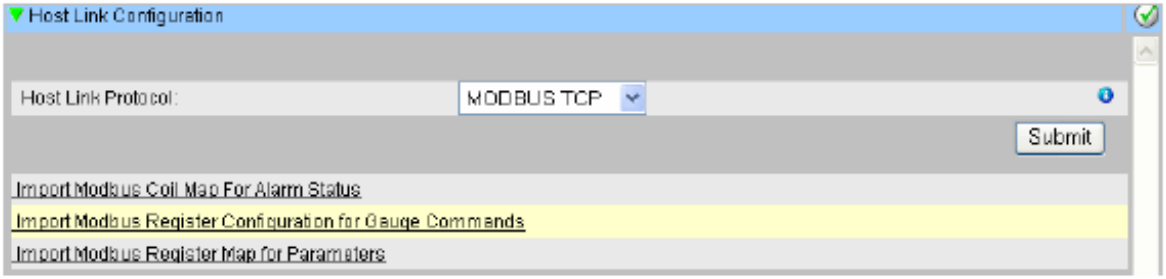


Figura 3.9 Submenú ***Host Link Configuration*.**

3. Luego en el submenú ***Modbus TCP Configuration*** configurar IP, ID y puerto de comunicación (figura 3.10).

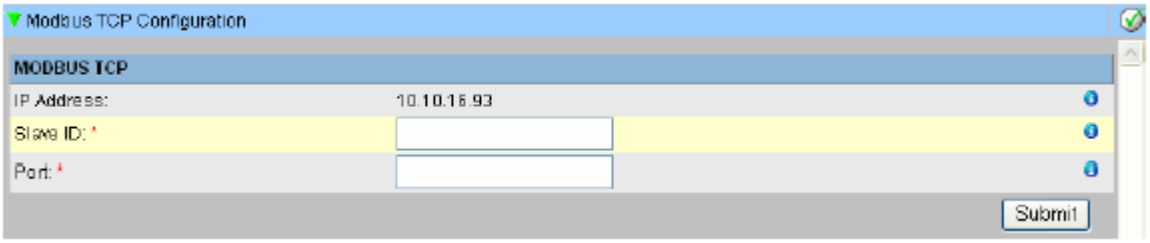


Figura 3.10 Submenú ***Modbus TCP Configuration*.**

Estos pasos garantizan la correcta configuración del *Host Link* NXA 822 como un esclavo Modbus TCP.

3.3 Valoración económica.

El servicio representa un ahorro considerable para la entidad debido a que el uso de software propietario para este tipo de aplicaciones representa un gasto de divisa representativo, lo que unido a la novedad que implica el proyecto, proporciona un ahorro por concepto de divisa a la empresa de entre unos 8000 y 10 000 USD.

3.4 Conclusiones parciales del capítulo.

Los logros alcanzados permiten arribar a las siguientes conclusiones.

- Los resultados obtenidos en experimentos realizados satisfacen los objetivos planteados.
- A través del análisis económico puede observarse que se garantiza un ahorro considerable a la entidad.
- El Manual de Usuario posee todos los requerimientos para el desarrollo y explotación del servicio por un administrador.

CONCLUSIONES.

- Con esta aplicación se logra la comunicación con el dispositivo *Host Link* NXA822 y el almacenamiento de los datos, en una base de datos, dándose cumplimiento a los objetivos planteados.
- Con la implementación de este sevicio se hace un aporte valiosos al sistema supervisorio *Tankvision* de la refienería de pétroleo “Camilo Cienfuegos” al dotar al mismo de una base de datos que muestra el estado de las principales variables de los tanques, lo que conlleva a un aumento de la calidad y una mejora considerable en el monitoreo del proceso.
- Con este proyecto se contribuye al ahorro por concepto de divisas con la empresa mixta PDVCupet S.A., al dotar a la misma de un servicio que sustituye la necesidad de adquisición de un software propietario.

RECOMENDACIONES

- Se propone el desarrollo de aplicaciones de este tipo en otros lenguajes de programación.
- Se recomienda la implementación de una interfaz gráfica que permita la comunicación con la base de datos desde la red administrativa de la empresa.
- Se recomienda la utilización este servicio como componente del sistema SCADA para el área de MCP de la refinería, actualmente en proyecto, o para uso de otras aplicaciones de esta índole en esa industria.

REFERENCIAS BIBLIOGRÁFICAS.

Apache_Commons. (2008). "Commons logging." from http://commons.apache.org/logging/.

Ballesteros, R. L. (2009). Conectividad entre aplicaciones. Computación Aplicada a la Automatización. Universidad Central "Marta Abreu" de Las Villas, Cuba.**:** 1-8.

Buendía, M. J. (2003). "Tema 7: Protocolo Modbus " Comunicaciones Industriales, from http://www.dte.upct.es/personal/manuel.jimenez/docencia/GD6_Comunic_Ind/pdfs/Tema%207.pdf.

Endress+Hauser (2010). Description of Instrument Functions Tankvision NXA820, NXA821, NXA822.

Fernández, O. B. (2004). "Introducción al lenguaje de programación Java: Una Guía básica." from http://www3.uji.es/~belfern/pdidoc//IX26/Documentos/introJava.pdf.

Ibercom, S. L. (2007). "Caracteristicas de SQL Server." from https://www.ibercom.com/soporte/index.php?_m=knowledgebase&_a=viewarticle&kbarticleid=996.

Intershare, S. L. (2010). "Softonic." from http://java.softonic.com/.

Márquez, B. (2004). "Implementación de un reconocedor de voz gratuito al sistema de ayuda a invidentes Dos-Vox en español." Departamento de Ingeniería en Sistemas Computacionales, Escuela de Ingeniería, from http://catarina.udlap.mx/u_dl_a/tales/navegacion/titulo.html.

Modbus-IDA. (2006). "Modbus Messaging on TCP/IP Implementation Guide." from http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf.

O'Reilly. (2001). "Java Enterprise in a Nutshell." from http://docstore.mik.ua/orelly/java-ent/jenut/ch18_01.htm.

Object Tehcnology International, I. (2003). "Eclipse Pataform Thecnical Overview ", from http://eclipse.org/whitepapers/eclipse-overview.pdf.

Olaya, A. F. R. (2002). "Implementación de una red Modbus TCP." Escuela de Ingeniería Eléctrica y Electrónica, Facultad de Ingeniería, from www.univalle.edu.co/~telecomunicaciones/.../tg_AndresRuiz.pdf.

Pérez, R. G. (2009). "Introducción al Sistema de Gestión de Base de Datos PostgreSQL." from http://www.monografias.com/trabajos-pdf2/sistema-gestion-base-datos-postgresql/sistema-gestion-base-datos-postgresql.pdf.

PostgreSQL. (2009). from http://www.postgresql.org/about/press/features84.

Proaño, D. J. B. (2006). "Analisis comparativo de bases de datos de código abierto vs codigo cerrado (Determinación de índices de comparación)." from http://www.willydev.net/InsiteCreation/v1.0/willycrawler/2008.05.12.articulo.comparacion%20bases%20de%20datos%20open%20y%20propietarias.pdf.

Rodríguez, Y. (2008). Diseño e implementación de pataforma informática para la gestión y organización de información. Santa Clara, Universidad Central "Marta Abreu" de Las Villas.**:** 40-41.

Zapata, D. E. V. (2004). "Introducción a la programación multicapas." from http://www.elguille.info/colabora/puntonet/jevergara_Multitier.htm.

ANEXOS.

Configuración del mapa Modbus para el *Host Link* NXA822.

Solutions Tankvision

Endress+Hauser
People for Process Automation

Advisory Note

MODBUS host link map configuration for NXA822

Marketing Services - Business Development

P.O. Box 1261
79690 Maulburg
Germany

Phone: ++49 7622 28 2091

Fax: ++49 7622 28 15091

E-mail: toni.kaesbeck@pcm.endress.com

1. Introduction

A typical tank farm uses control units or a DCS, which automates and controls some or the entire tank farm operation. Such a system (also called host system) typically needs tank inventory data.

Tankvision has a dedicated component for host communication: The NXA822 host link with a MODBUS slave software. Tankvision makes the data acquired from gauges available via:

- One serial port with MODBUS RTU mode on either physically EIA-232(RS) and EIA-485(RS) port
- MODBUS TCP/IP Ethernet

Tank data that is made available via the host interface includes measured and calculated inventory data and alarm status. The host interface also allows the host system to send gauge commands and monitor gauge command status.

The NXA822 has a web page to download XML definition files.

2. DEFINITION OF THE MODBUS MAP FOR TANK PARAMETERS

2.1. TYPE OF MODBUS MAPS

The register assignment is done via different XML description files:

- MODBUS register map for parameters (measured and calculated inventory data is available via MODBUS registers (3X or 4X reference). This can be either individual registers or automatically generated tank maps.
- MODBUS coil map for alarm and gauge status (Alarm status shall be made available via MODBUS coils (0X or 1X Reference)), see separate advisory note.
- MODBUS gauge command register configuration to issue gauge commands from a host (via a MODBUS WRITE command)

Once the MODBUS map for registers is downloaded, the configuration page gives a summary of the defined registers. An upload is currently not available via the configuration pages.

2.2. OVERVIEW OF PRINCIPLE OF REGISTER DEFINITION

Registers can be defined to describe the registers read from the NXA822 with the following parameters:

```
<MAP_ENTRY>
            <IP_ADDR>NXA820_1</IP_ADDR>
            <!—Note that although it says IP_ADDR the name of the box (System
            Tag) has to be entered and not the IP address.You can find the
            SystemTag in the "Manager System" Configuration page ?>
            <Tank_Id>1</Tank_Id>
                  <!— The Tank Id is the sequential number of the tank on the
                  original NXA820 ?>
            <Param_Name>P_Level</Param_Name>
                  <!— Tank Paramter name from the list below ?>
            <Register>40001</Register>
                  <!— MODBUS 3xxxx or 4xxxx register address, not Offset! ?>
            <Scalar>1.0</Scalar>
            <Offset>0.0</Offset>
                  <!— Optional Scaling?>
            <Packing_Format>IEEE754</Packing_Format>
                  <!— See the for a list of data types?>
</MAP_ENTRY>
```

2.3. DIFFERENCE BETWEEN <BLOCK> AND <OVERWRITE_BLOCK>

The host link allows reading tank data over MODBUS. At the same time, you can perform 'manual override' of tank measured data. Now, from MODBUS perspective, all read-only parameters are placed in 3X reference and all r/w parameters are placed in 4X reference. Therefore, all parameters present in <BLOCK> are

treated read-only. If you want to manually override measured data e.g. level, it needs to be placed into <OVERWRITE_BLOCK> block. The register address is validated accordingly.

The OVERWRITE blocks maps parameter which can be manually overridden by host via host link. Normal BLOCK is read/only and goes into MODBUS 3X reference and parameters under OVERWRITE block are R/W and go into MODBUS 4X reference.

2.4. STATUS OF MODBUS REGISTERS READ VIA HOST LINK

When the MODBUS map is prepared via XML - either tank oriented, block oriented or element oriented, parameter status is present in the MODBUS in all three register layout for 3xxxx. In any MODBUS register layout, the parameter status is always present in the immediately following register of parameter value. Refer example below:

If product level packing is IEEE754, it takes 2 registers, then layout would contains following data

30001 level value – LSB	40001 level value - LSB
30002 level value – MSB	40002 level value - MSB
30003 level status	

If product level packing is IEEE754SWAPPED, it takes 2 registers, then layout contains following data

30001 level value – MSB	40001 level value - MSB
30002 level value – LSB	40002 level value - LSB
30003 level status	

If product level packing is INT16, it takes 1 register, then layout would contain following data

30001 level value	40001 level value
30002 level status	

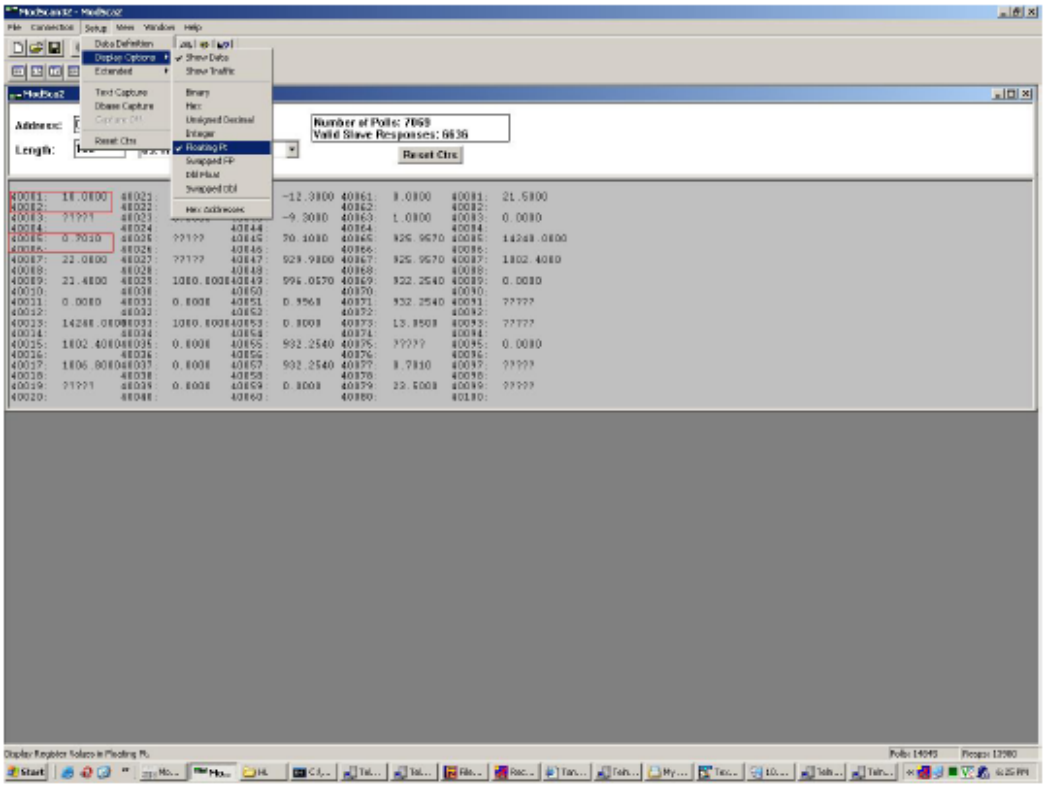
Since status cannot be over-ridden, it is not present in registers with 4xxxxx address.[1]

Please note, that only reading with command 04 and evaluating the 3rd STATUS WORD for status by the DCS allows detection of failures in tank gauges!

[1] There is a defect entered in Clearquest to make the status transmission option for both commands 03 and 04.

Solutions Tankvision

Endress+Hauser
People for Process Automation



Tankvision Solutions

Endress+Hauser
People for Process Automation

3. READING PARAMETERS

3.1. DATA TYPES SUPPORTED

MODBUS host link register map stores parameter value based on parameter packing format. Table 1 specifies parameter packing format supported by MODBUS host link[2].

The data types which are supported in NXA822 host link are[3]:

1. "INT16"
2. "UINT16"
3. "INT32"
4. "UINT32"
5. "UNSIGNDBL"
6. "DECFLOAT16"
7. "MDPII"
8. "IEEE754"
9. "IEEE754SWAPPED"

Type	XML tag for 'Packing_Format'	MODBUS Register Count	Minimum	Maximum	Description
16-bit signed integer	INT16	1	-32768	32767	Number loaded into single register
16-bit unsigned integer	UINT16	1	0	65535	Number loaded into single register
32-bit signed integer	INT32	2	-2147483648	2147483647	High order word followed by low order word
32-bit unsigned integer	UINT32	2	0	4294967295	HIGH order word followed by Low order word
Unsigned double integer	UNSIGNDBL	2	0	65535.9999	1st (lower address) register stores the integer part and 2nd (consecutive higher order) register stores the fraction
IEEE754 floating point	IEEE754	2	-3.402823466 E+38	3.402823466 E+38	HIGH order word of IEEE representation followed by LOW order word

[2] The data type definition is not identical in NXA820 (MODBUS) and NXA822

[3] As in email Rewa Tikekar, November 27th, 2007

Solutions Tankvision

Endress+Hauser
People for Process Automation

IEEE754 Floating point swapped	IEEE754SWAPPED	2	-3.402823466 E+38	3.402823466 E+38	LOW order word of IEEE representation followed by HIGH order word
16-bit Decimal Float	DECFLOAT16	1	-1023E7	1023E7	Floating point encoded as 16-bit decimal number
MDP-II	MDPII	2	0	99999999	1st (lower address) register stores the lower four digits and 2nd (consecutive higher order) register stores upper four digits

Table 1: Parameter Value Packing Format

3.2. PARAMETERS STATUS ENCODING

The MODBUS host link MODBUS register map contains the parameter status in the MODBUS register immediately following the register containing the parameter value. The Tankvision parameter status value is updated in the register holding the status based on the actual parameter status.

Parameter status string in gauge definition file on NXA822	Parameter status value in host link MODBUS register	Description
VALID	655	Parameter status valid or OK
FAIL	656	Parameter being scanned from gauge failed (after repetitive timeout on NXA820)
MANUAL	657	Parameter value manually overridden
INITIALIZED	675	Parameter initialized; but yet to be scanned from gauge (on NXA820)
NODATA	676	Parameter scan 'killed' i.e. mode changed to manual; but no value is overridden yet
INVALIDDATA	677	Data is invalid
TIMEOUT	678	Parameter scan timed out (on NXA820)
OLDDATA	679	Parameter data is too old
LASTVALIDVALUE	680	Parameter value is 'last valid value'

Parameter status is present in a single register in the MODBUS register map. The host link can read it by creating a tag with data type WORD or UINT16.

In general, if a parameter is not present in the tank object and is requested in the NXA822 XML for MODBUS, it is initialized to 32767. That is why S_LEVEL is transmitted by NXA822 as 32767, if it is not present in the MODBUS register for the corresponding tank on NXA820. The NXA822 can read S_LEVEL, by adding tag for same in Tank parameter XML on HL, provided the associated TSIC reads S_LEVEL.

Solutions Tankvision

Endress+Hauser
People for Process Automation

3.3. TANK PARAMETERS AVAILABLE

The following shows a list of all the tank parameters, which can be assigned to a MODBUS register in the NXA822 MODBUS host link:

622	P_LEVEL	Product Level
623	S_LEVEL	Secondary Level (zweiter Füllstand evtl. zur Überfüllsicherung)
624	W_LEVEL	Water Bottom Level
625	P_TEMP	Product Temperatur
626	V_TEMP	Vapour Temperatur
660	A_TEMP	Ambient Temperatur
627	V_PRESS	Vapour Pressure
628	P_OBS_D	Observed density
661	P_REF_D	Reference Density
700	P_PRESS_A	P1 Bottom pressure transmitter, Druckaufnehmer "bottom" (absolut)
701	P_PRESS_G	P1 Bottom pressure transmitter, Druckaufnehmer "bottom" (relativ)
702	P_PRESS	Middle Pressure (ist im Moment in der Diskussion) == P2
703	V_PRESS_A	P3 Top pressure transmitter, Druckaufnehmer in der Gasphase (absolut)
704	V_PRESS_G	P3 Top pressure transmitter, Druckaufnehmer in der Gasphase (relativ)
717	TOT_OBS_VOL	Total observed Volume
718	REM_CAP_TANK	Remaining capacity
719	AVAIL_VOL	available Volumen
720	SED_W_VOL	Sediment water Volumen
721	P_LVLCHNG_RATE	Level change rate (in mm)
722	TOTOBS_FLW_RATE	total observed flow rate
723	NETSTD_FLW_RATE	net standard flow rate
724	TOTMASS_FLW_RATE	total mass flow rate
725	FREE_W_VOL	free water volume
726	GROSS_OBS_VOL	gross observed volume
752	TOT_STD_VOL	total standard volumen
754	VCF	volume correction factor
756	MASS_VAPR	mass vapour
760	NET_WGHT_AIR	net weight air
761	NET_STD_WGHT	net standard weight
762	F_ROOF_ADJUS	floating roof adjustment
763	F_ROOF_POS	floating roof position
774	TNK_SHELL_CORR	tank shell correction
727	GROSS_STD_VOL	gross standard volume
728	NET_STD_VOL	net standard volume
729	P_MASS	Product mass
730	TOT_MASS	total mass (product mass + liquid equivalent of gas phase)
1592	VAPOUR_ROOM_VOL	
1634	TEMP_1	Spot temperature element 1[4]
1634	TEMP_2	Spot temperature element 2

[4] Spot temperature element 1 is the lowest

1634	TEMP_3	Spot temperature element 3
1634	TEMP_4	Spot temperature element 4
1634	TEMP_5	Spot temperature element 5
1634	TEMP_6	Spot temperature element 6
1634	TEMP_7	Spot temperature element 7
1634	TEMP_8	Spot temperature element 8
1634	TEMP_9	Spot temperature element 9
1634	TEMP_10	Spot temperature element 10
1634	TEMP_11	Spot temperature element 11
1634	TEMP_12	Spot temperature element 12
1634	TEMP_13	Spot temperature element 13
1634	TEMP_14	Spot temperature element 14
1634	TEMP_15	Spot temperature element 15
1634	TEMP_16	Spot temperature element 16

Object names, that can be assigned, will be listed in the Tankvision user manual, too.

3.4. Decimal Float representation

The FLOAT format is not user readable

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Description	Sign	Exponent				Mantissa										

Example: Number 12300 shall be packed in16-bit decimal float format as mentioned in this example.

Normalized number: 123×10^2

Part	Value	Binary representation
Sign bit	0	0
Exponent	2	0010
Mantissa	123	00001111011

Number in binary format: 0001000001111011

Number is Hexadecimal format: 107B

Thus, number packed into a 16-bit MODBUS register is $107B_{16}$.

3.5. Exception handling

In general, if a parameter is not present in the tank object and is requested in the NXA822 XML for MODBUS, it is initialized to 32767. S_LEVEL is transmitted by NXA822 as 32767, if it is not present in the MODBUS register for the corresponding tank on NXA820.

HL can read S_LEVEL, by adding tag for same in Tank parameter XML on HL, provided the associated TSIC reads S_LEVEL.

4. Overwriting of tank parameters

```
<MAP_ENTRY>
      <IP_ADDR>228-NXA820</IP_ADDR>
      <Tank_Id>1</Tank_Id>
      <Param_Name>P_LEVEL</Param_Name>          --parameter
      <ParamRegister>30001</ParamRegister>     --Read Status in decimal for-
mat from master(takes three registers)
      <OverRideRegister>40009</OverRideRegister>     --write value in floating
point from master(takes two registers)
      <Scalar>1.0</Scalar>
      <Offset>0</Offset>
      <Packing_Format>IEEE754</Packing_Format>
</MAP_ENTRY>
```

This tag writes parameter value and reads status to a specified location.

5. PARAMETER MAPS

The XML maps above allow individual description of MODBUS register. Tankvision offers two additional ways to minimize the effort in describing the MODBUS map:

- MODBUS map for parameters' (Simple Approach)

in case of simple approach, MODBUS register configuration for any data element is specified separately and individually

- MODBUS map for parameters' (With Data Orientation)

XML with data orientation, the XML file for MODBUS register map shall specify data elements and tanks for which host interface is to be provided separately

- MODBUS map for parameters' (With Element Orientation)

XML with data orientation, the XML file for MODBUS register map shall specify data elements and tanks for which host interface is to be provided separately

5.1. XML FILE FOR TANK PARAMETERS WITHOUT ORIENTED MAPS

The following table shows the example of an XML description to read level and temperature:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com)
<FG4HL_MODBUS_PARAM_MAP CRC="0">
      <MAP ENTRY>
            <IP ADDR>NXA820 1</IP ADDR>
                  <!—Note that although it says IP ADDR the name of the box
                     has to be entered and not the IP address (V0.1.58)?>
            <Tank_Id>1</Tank_Id>
            <Param_Name>P_Level</Param_Name>
            <Register>40001</Register>
            <Scalar>1.0</Scalar>
            <Offset>0.0</Offset>
            <Packing_Format>IEEE754</Packing_Format>
      </MAP_ENTRY>
      <MAP_ENTRY>
```

Tankvision Solutions

Endress+Hauser
People for Process Automation

```
            <IP_ADDR>NXA820_2</IP_ADDR>
            <Tank_Id>1</Tank_Id>
            <Param_Name>P_TEMP</Param_Name>
            <Register>40003</Register>
            <Scalar>1.0</Scalar>
            <Offset>0.0</Offset>
            <Packing_Format>IEEE754</Packing_Format>
      </MAP_ENTRY>
      <MAP_ENTRY>
            :
      </MAP_ENTRY>
</FG4HL_MODBUS_PARAM_MAP>
```

5.1.1. MOBDUS MAPS WITH DATA ORIENTATION

Block offset is available with 'DATA' orientation i.e all parameters(present in xml) from tank 1 are displayed then all parameters(present in xml) from tank2 are displayed and so on....
As Observed on 40001…(Holding registers)
Example: `No. of registers used by Tank 1 :`
`= no. of parameters in tank 1 * Register count for specified packing format for the parameters.`

5.1.2. MOBDUS MAPS WITH ELEMENT ORIENTATION

Block offset is available with 'ELEMENTS' orientation i.e first parameters from all tanks(in xml) are displayed then second parameter from all tanks(in xml) are displayed and so on....

`Example: No. of registers used by Parameter 1 : = no. of tanks * Register count for specified packing format for that parameter.`

```
<?xml version="1.0" ?>
 <FG4HL_MODBUS_PARAM_MAP CRC="0">
 <MAP_ORIENTATION>Elements</MAP_ORIENTATION>
  <MAP_ELEMENTS>
  <ELEMENT>
  <Name>P_LEVEL</Name>
  <Scalar>1.0</Scalar>
  <Offset>0.0</Offset>
  <Packing_Format>IEEE754</Packing_Format>
  </ELEMENT>
  ……. . . . .

   <BLOCKS>
   <BLOCK_START>30001</BLOCK_START>
   </BLOCKS>
   <OVERRIDEBLOCKS>
   <BLOCK_START>40001</BLOCK_START>
   </OVERRIDEBLOCKS>
   <TANKS>
   <TANK>
   <IP_ADDR>228-NXA820</IP_ADDR>
   <ID>1</ID>
   </TANK>
```

Example:

Solutions Tankvision

Endress+Hauser
People for Process Automation

Block	Register Address	Data element	Remark
40001	40001	P_Level_Inn for TNK-001	Block starts at 40000
	40003	P_Level_Inn for TNK-002	
	40005	P_Level_Inn for TNK-003	
	40007	P_Level_Inn for TNK-004	
	40009	P_Level_Inn for TNK-005	
40011	40011	P_Level_Ull for TNK-001	Next block starts at 40010
	40013	P_Level_Ull for TNK-002	
	40015	P_Level_Ull for TNK-003	
	40017	P_Level_Ull for TNK-004	
	40019	P_Level_Ull for TNK-005	
40021	40021	P_Temp for TNK-001	No next block address available; data starts immediately after the end of earlier block
	40023	P_Temp for TNK-002	
	40025	P_Temp for TNK-003	
	40027	P_Temp for TNK-004	
	40029	P_Temp for TNK-005	
:	:	:	:

As of V00.02.03-127 there a no options for spare registers or tank offsets available. Both is entered as an enhancement request.

5.1.3. BLOCK OFFSETS AND SPARE AREAS

Spare areas in the MODBUS register mapping can be kept via BLOCKS definition

```
You can have in the first block a spare area from 40017 to 40050
     e.g   T-0001 level, temperature, S_LEVEL etc. from 40001 to 40016,
                  empty spare 40017 ... 40050
            then T-002 starting from 40051..40066
                  empty spare 40067 ... 40100
            T-003 from 40101 to 40116
                  empty spare 40117 ... 40150
```

Using a XML map the Holding registers after the last EELEMNT defined will remain spare area.

```
        <ELEMENT>
          <Name>S_LEVEL</Name>
          <Scalar>1.0</Scalar>
          <Offset>0.0</Offset>
          <Packing_Format>IEEE754</Packing_Format>
        </ELEMENT>

<BLOCKS>

        <BLOCK_START>30001</BLOCK_START>

        <BLOCK_START>30051</BLOCK_START>

        <BLOCK_START>30101</BLOCK_START>

        <BLOCK_START>30151</BLOCK_START>

        <BLOCK_START>30201</BLOCK_START>

</BLOCKS>
```

Solutions Tankvision

Endress+Hauser
People for Process Automation

```
<OVERRIDEBLOCKS>
        <BLOCK_START>40001</BLOCK_START>
        <BLOCK_START>40051</BLOCK_START>
        <BLOCK_START>40101</BLOCK_START>
        <BLOCK_START>40151</BLOCK_START>
        <BLOCK_START>40201</BLOCK_START>
</OVERRIDEBLOCKS>
      <TANKS>
      -. ..
```