UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 5

Título: “Desarrollo de un simulador para realizar las pruebas del manejador Modbus Omni”

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

Autor: **José Carlos Abraham Ravelo**
Tutor: **Ing. Adrián Carlos Moreno Borges**
Consultante: **Ing. Rubén Gómez Johnson**

Mayo, 2012

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Firma del Autor.

(José Carlos Abraham Ravelo)

_____

Firma del Tutor.

(Ing. Adrián Carlos Moreno Borges)

Agradecimientos:

No pudiera empezar este apartado sin mencionar a mi mamá y a mi papá. Sin ellos me hubiese sido imposible avanzar en la vida hasta llegar a este momento. Mi mamá siempre dispuesta a escuchar, siempre receptiva y presente. Y lo mejor de todo, sin importar que tan buena o mala puedan haber sido mis decisiones, brindándome su apoyo incondicional. Y mi padre como ejemplo de perseverancia, de imponerse ante las adversidades de la vida y sobre todo, siempre dispuesto a sacrificarlo todo por mí o por cualquiera de mis hermanos.

Además, siento que necesito agradecer a mi hermano "Hermano", a mi hermano Abelito, a mi hermana Anabel y a la recién llegada Leydi, con los cuales siempre he compartido un verdadero vínculo de hermandad. Luego mis abuelos, primos y tíos. Sobre todo de mi abuela Fidelina, mi abuelo Ramón, tío Pablo, tía Milagros y tía Celita, los cuales por razones geográficas han compartido mayor tiempo conmigo. Pero en general todos fueron parte de mi vida, y todos me enseñaron cosas nuevas y oportunas.

Luego quiero agradecer a mi mole, con la que aún siento la química del primer día. Química que le ha permitido tolerar mis defectos durante el final de nuestras carreras, quien me ha servido de aliento, cuando más apretaba la recta final y más que eso, quien ha aprendido a convivir con mi forma independiente y un tanto despreocupada de ser. Además, me abrió las puertas a una nueva familia a la cual he aprendido a querer y ya hasta extraño.

También me gustaría agradecer a mis amistades desde primer año Yanet, Sonia, mi pupilo Ernesto, Annabel, Leya, Lores. En especial a Yanet con la cual compartí desde cumpleaños, hasta los problemas personales de cada uno. A mis amistades más recientes Darlen, Gadied y Gretter, con los que, a pesar del poco tiempo, se han ganado un lugar en mi memoria y se han colado en mi lista de amigos. También quisiera mencionar a mis amistades del pre Yele, Jessica, Annalie.

Y por último, pero tan importante como el primero, me gustaría agradecer a la profesora Yirka y la profesora Zoraida, las cuales me han ayudado muchísimo con mi tesis y la vez brindándome su amistad. Además, necesito mencionar a mi tutor, a Tony, Pedro y todos los profesores del proyecto que de una forma u otra me ayudaron a desarrollar mi tesis. Y a mis compañeros de apartamento Luiso, Karel, Andrie, en fin, todo ese piquete, con los cuales compartí muy buenos momentos.

Resumen:

El proyecto GALBA, es un convenio de trabajo entre la Universidad de las Ciencias Informáticas y Petróleos de Venezuela Sociedad Anónima, empresa medular de la economía venezolana. Como parte del proyecto se le otorgó la tarea a la universidad de desarrollar el manejador Modbus Omni, producto necesario en el sistema de Control de Supervisión y Adquisición de Datos instalados en las instituciones de la compañía venezolana.

En el presente trabajo se describe el desarrollo de un simulador que funcionará como servidor en la comunicación a través del protocolo de comunicación Modbus Omni. Este producto es necesario debido a la carencia de una aplicación con estas características en la Línea de Adquisición, del Centro de Informática Industrial, perteneciente a la Universidad de las Ciencias Informáticas. Esta situación es desfavorable para dicha línea debido a que dificulta la etapa de pruebas del manejador Modbus Omni, proceso indispensable para lograr que dicho producto presente la menor cantidad de errores posible antes de ser entregado al cliente.

El proceso de desarrollo comienza con un estudio de los principales conceptos enmarcados dentro del proceso de adquisición y envío de datos a través de los puertos serie y Ethernet. Además se analizan las especificaciones del protocolo Modbus Omni. Se lleva a cabo un estudio de las tecnologías y herramientas de desarrollo afines con este tipo de investigación, para determinar cuáles utilizar en el diseño e implementación del simulador. Se exponen las características funcionales y no funcionales del simulador que guiarán el proceso de desarrollo, así como la arquitectura general que se seleccionó para desarrollar la aplicación y la descripción de cada uno de sus componentes. Por último, se concluye el proceso de desarrollo con la ejecución de las pruebas que garantizan la calidad y el funcionamiento correcto del simulador.

Palabras clave: Protocolo Modbus Omni, simulador, dispositivo maestro, dispositivo esclavo.

Índice.

Índice de tablas.

Índice de ilustraciones.

Introducción:

En la actualidad, la informática está presente en más de una esfera de la vida del ser humano y un área destacada, es la industrial. En la industria existen innumerables procesos que son difíciles o imposibles de realizar manualmente, por lo que se hace imprescindible automatizarlos. Parte de una de las soluciones a este problema es un sistema de Control de Supervisión y Adquisición de Datos (SCADA por sus siglas en inglés). A grandes rasgos se denomina SCADA a cualquier software que permita el acceso y control de datos remotos de un proceso, utilizando las herramientas de comunicación necesarias.

En el 2006, como parte del Convenio Integral de Cooperación Cuba – Venezuela, la Universidad de las Ciencias Informáticas (UCI) comenzó a trabajar en el proyecto SCADA PDVSA. El objetivo del proyecto es la implementación del sistema SCADA necesario en la industria petrolera venezolana; como única restricción se impuso que los sistemas tenían que implementarse con software libre y con la utilización de estándares abiertos. Con ello, el estado venezolano pretendía remplazar los antiguos proveedores de los sistemas SCADA instalados en Petróleos de Venezuela Sociedad Anónima (PDVSA), compañía petrolera de gran importancia en la economía de su país.

El proyecto aún está vigente y se conoce como Guardián del Alba o GALBA. La línea de Adquisición es uno de los departamentos de investigación y desarrollo en que está dividido este proyecto; una de su responsabilidad es implementar los *drivers* (manejador en español). Los manejadores, a través de un lenguaje de comunicación específico, más conocido como protocolo de comunicación, se encargan de garantizar que el SCADA intercambie datos con los dispositivos de campo encargados de monitorear los procesos que se desean controlar.

Dado el importante papel que juegan los manejadores dentro de un sistema SCADA, es imprescindible garantizar que estos posean la menor cantidad de errores posible; para ello en la línea de Adquisición del proyecto GALBA de la UCI, se utilizan varias estrategias. La primera variante de estas estrategias, es adquirir los dispositivos de campo, para así realizar las pruebas en ellos; pero esto no siempre es posible principalmente por el precio de la transacción. La segunda variante es adquirir *software* que simulen el comportamiento de dichos dispositivos (herramientas libres en Internet); esta variante tampoco brinda

resultados satisfactorios en todos los casos, debido a que en varias ocasiones dichos *software* no cumplen con todos los requisitos indispensables para realizar las pruebas o no se encuentran disponibles en Internet.

En el 2011, en la línea de Adquisición se comenzó la implementación del manejador Modbus Omni, dando lugar a la siguiente **situación problémica**: Al llegar a la etapa de pruebas, la línea no poseía los dispositivos de campo necesarios para probar el funcionamiento del manejador. Además, en los sitios consultados y en la documentación revisada, no existe un *software* que posea las funcionalidades necesarias para poder ser utilizado como sustituto de los dispositivos antes mencionados, en el proceso de detección de los posibles errores del manejador Modbus Omni. Estas deficiencias imposibilitan en gran medida la realización de la etapa de prueba del manejador, proceso indispensable para el cumplimiento del ciclo de desarrollo del producto.

De la situación problémica planteada se deriva la necesidad de solucionar el siguiente **problema investigativo:** ¿Cómo mejorar el proceso de detección de errores del manejador Modbus Omni, desarrollado en el proyecto GALBA?

Por lo cual el **objeto de estudio** de esta investigación es: el proceso de transferencia de datos a través de un protocolo de comunicación.

Para contribuir a la solución del problema se propone como **objetivo general:** desarrollar una aplicación, basada en tecnologías libres, que implemente el protocolo de comunicación Modbus Omni.

Todo lo cual precisa como **campo acción:** el envío y recepción de información utilizando el protocolo Modbus Omni.

Con vistas a dar cumplimiento al objetivo general se tienen las siguientes **tareas de investigación:**

- Elaboración del marco teórico de la investigación a partir del estado del arte existente actualmente sobre el tema.

- Estudio de las especificaciones del protocolo Modbus Omni.
- Selección de la metodología y tecnología de desarrollo acorde con las políticas de la línea de Adquisición.
- Implementación del simulador Modbus Omni.
- Realización de las pruebas del simulador Modbus Omni.

Donde la **idea a defender** es**:** con el desarrollo de un simulador que implemente el protocolo Modbus Omni y brindando la posibilidad de crear alteraciones en el proceso de comunicación, se podrá ayudar al equipo de prueba en el proceso de detección de los errores que pueda poseer el manejador Modbus Omni.

Y el **posible resultado** es**:** un simulador Modbus Omni que ayude a realizar las pruebas al manejador Modbus Omni. Además, al ser una herramienta desarrollada en la línea de Adquisición, si se le adicionan nuevas funcionalidades al manejador, sería posible agregarle nuevos comportamientos al simulador, para probar dichas funcionalidades. También brinda la posibilidad de realizar demostraciones de las capacidades del manejador Modbus Omni y de probar otras aplicaciones o dispositivos que se comuniquen a través del mismo protocolo.

1 Capítulo 1 Fundamentación Teórica.

Introducción:

En el capítulo se podrá encontrar la explicación de todos los elementos externos e internos que interactuarán o serán parte de una aplicación que utilice el protocolo Modbus Omni, para la comunicación con equipos externos a él. Algunos de estos elementos son: los recursos informáticos que brindan la posibilidad de recolectar, procesar y enviar datos correctamente y varios conceptos que acotan la manera en que se debe trabajar con el fin de culminar la investigación satisfactoriamente.

1.1 Modelo maestro/esclavo.

IBM define al modelo Maestro/Esclavo: "La tecnología que proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios de cómputo o cualquier otro recurso del grupo de trabajo y/o organización, en múltiples plataformas. El modelo soporta un medio ambiente distribuido en el cual los requerimientos de servicio hechos por estaciones de trabajo inteligentes o maestros, resultan en un trabajo realizado por otros computadores llamados esclavos". (1) El “maestro” es el que inicia un requerimiento de servicio. (1) El requerimiento inicial puede convertirse en múltiples requerimientos de trabajo a través de redes LAN o WAN. La ubicación de los datos o de las aplicaciones es totalmente transparente para el maestro. Y el “esclavo” es cualquier recurso de cómputo dedicado a responder a los requerimientos del “maestro”. (1) Los esclavos pueden estar conectados a los maestros a través de redes LAN o WAN, para proveer de múltiples servicios a los maestros (tales como impresión, acceso a bases de datos, fax, procesamiento de imágenes, etc.)

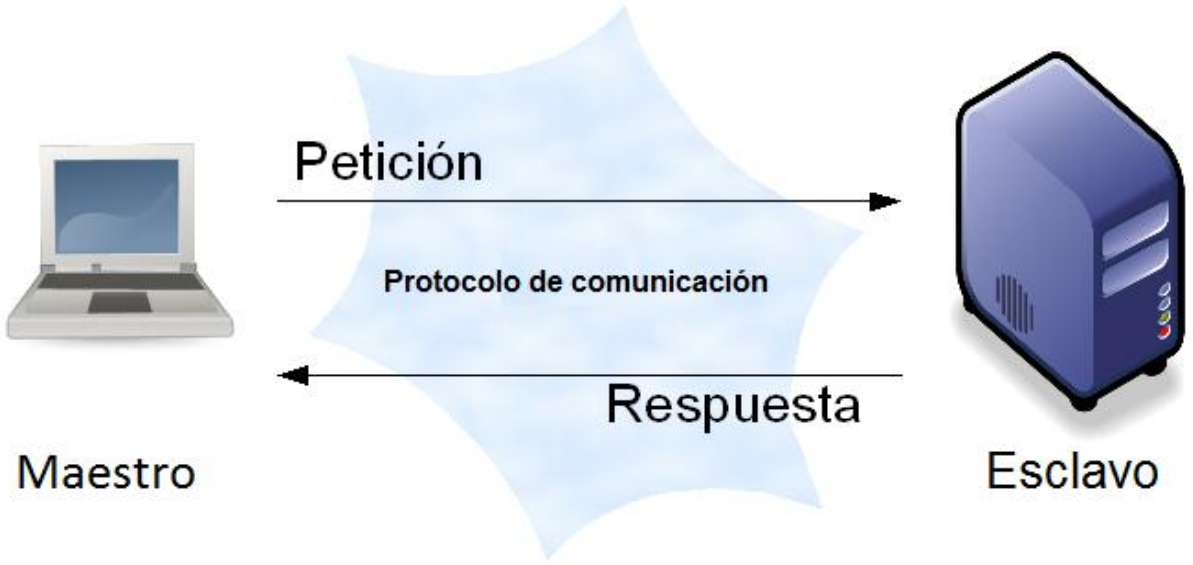

Ilustración 1. Modelo maestro/esclavo.

1.2 Comunicación TCP/IP.

El Protocolo de Control de Transmisión (TCP por sus siglas en inglés), es uno de los protocolos utilizados en internet. Fue creado entre los años 1973-1974 por Vint Cerf y Robert Kahn. Muchos programas dentro de una red de datos compuesta por ordenadores pueden usar TCP para crear conexiones entre ellos, a través de las cuales puede enviarse un flujo de datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y el mismo orden en que se transmitieron. También proporciona un mecanismo para distinguir distintas aplicaciones dentro de una misma máquina, a través del concepto de puerto. TCP proporciona soporte a muchas de las aplicaciones más populares de Internet, incluidas HTTP, SMTP y SSH. (2)

Todos los datos en este tipo de comunicación viajan en segmentos TCP, donde cada viaje se realiza a través de Internet en un datagrama IP. (2) El conjunto de protocolos se conoce frecuentemente como TCP/IP debido a que el TCP y el IP son los dos protocolos más importantes.

1.3 Comunicación Serial.

El puerto serial permite la comunicación entre dos dispositivos a través de un canal exclusivo. Un bit se transmite durante un período de tiempo, por tanto, la velocidad en baudios (baudrate) de un puerto serie de la computadora es igual al número de bits por segundo que se transmiten o reciben. (3) Para enviar información codificada de esta manera, el transmisor y receptor deben estar a la misma frecuencia y sincronizados. Cada caracter se envía en una contenedor, que consiste en un bit “0” llamado un bit de inicio, seguido por el caracter mismo y luego, opcionalmente, un bit de paridad y después un bit “1” llamado bit de paro. (3) La lógica del bit bajo de inicio le dice al receptor que está empezando un contenedor, y la lógica del bit alto de parada denota el final del mismo.

1.4 Protocolo Modbus.

Modbus es un protocolo de comunicaciones situado en el nivel 7 (capa de aplicación) del Modelo OSI, basado en el modelo maestro/esclavo. (4) Es un protocolo público, exige poco desarrollo y permite el control de una red de dispositivos.

Existen versiones del protocolo Modbus para puerto serie y Ethernet (una de ellas es Modbus/TCP). En la versión serial se estudiaron dos variantes: Modbus RTU es una representación binaria compacta de los datos; Modbus ASCII es una representación legible del protocolo (debido a que no utiliza la notación binaria, sino una representación con caracteres ASCII). (4) El formato RTU finaliza la trama con una suma de control de redundancia cíclica (CRC por sus siglas en inglés), mientras que el formato ASCII utiliza una suma de control de redundancia longitudinal (LRC por sus siglas en inglés).

Cada dispositivo de la red Modbus posee una dirección única. (4) Cualquiera de ellos puede enviar instrucciones, aunque lo normal es habilitarlo sólo para un equipo *maestro*. Las instrucciones son mensajes o tramas enviado a través de la red, que funciona como un comando para indicar al receptor una operación a realizar. Cada trama Modbus contiene la dirección de la unidad receptora de la orden. Los dispositivos reciben los mensajes, pero sólo el destinatario la ejecuta (salvo un modo especial denominado "*Broadcast*"). Cada uno de los mensajes incluye información redundante que asegura su

integridad. Los comandos básicos Modbus permiten controlar un dispositivo para modificar los datos que estos contienen o bien extraer su contenido. (4) Estos datos se almacenan distintos tipos registros, los cuales varían su estructura según el tipo de datos que se desea almacenar.

1.5 Protocolo de comunicación Modbus Omni.

Modbus Omni es un protocolo de comunicación desarrollado por Enron Corporation. El mismo comparte grandes similitudes con el protocolo Modbus. Las principales diferencias entre los dos protocolos son la numeración de las direcciones de registro, el soporte de registros de 32 bits y cadenas de caracteres, así como la capacidad de transmitir registros de eventos y datos históricos. (5) La descripción de los tipos de registros soportados por este protocolo está reflejada en la siguiente tabla.

Tipo de registro	Tamaño de un registro en byte	Tipo de variable que representa
Mixed	Variable	Cadena de caracteres o símbolos
Status	2	Booleana (sus únicos valores válidos son verdadero o $FF00_{16}$ y falso o 0)
Short Intiger	2	Entera o numérica
Long Intiger	4	Entera o numérica
IEEE Floating Point	4	Entera o numérica
8-Char. ASCII String	8	Cadena de caracteres o símbolos
16-Char. ASCII String	16	Cadena de caracteres o símbolos
ASCII Text Buffers	Máximo de 64 cadenas de 128 bytes.	Cadena de caracteres o símbolos

Tabla 1. Tipos de registros Modbus Omni.

Este protocolo especifica un maestro y un máximo de hasta 247 *esclavos* en una línea de comunicación común. (5) A cada *esclavo* se le asigna una dirección de dispositivo única en el intervalo de 1 a 247. El maestro siempre inicia la transacción. Las transacciones son o bien un tipo de pregunta y respuesta (sólo se accede a un esclavo a la vez, ver Ilustración 2) o de tipo una emisión sin ninguna respuesta o *Broadcast* (todos los esclavo se accede al mismo tiempo y ninguno responde, ver Ilustración 3). (5) O sea una transacción comprende una consulta única y una respuesta única o una consulta de difusión. El

protocolo posee tres modos básicos de la transmisión de datos: Modbus Omni TCP/IP a través de la comunicación vía Ethernet y Unidad de terminal ASCII o Remota (RTU), que utiliza la comunicación serial. (5)

Ilustración 2. Transacciones de tipo pregunta y respuesta.

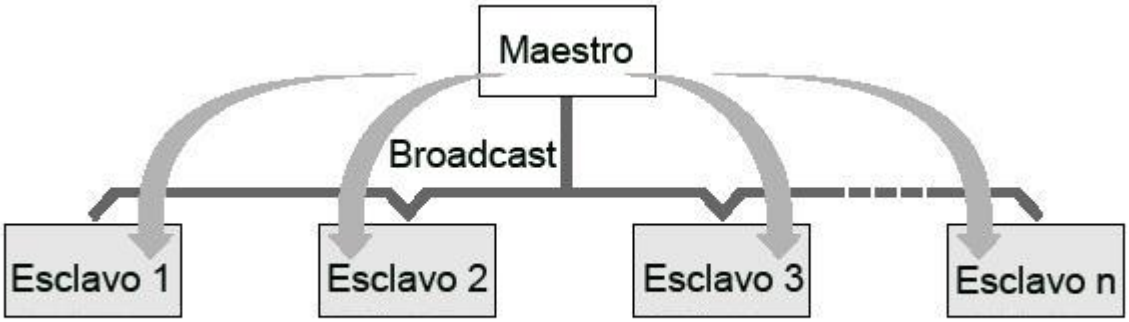

Ilustración 3. Transacciones de tipo emisión sin respuesta.

1.5.1 Trama Modbus Omni ASCII y formato del mensaje.

El modo de transmisión ASCII se logra mediante el uso de los dos puntos (:), caracter que indica el comienzo de una trama y el caracter CR (retorno del carro, por sus siglas en inglés) y el caracter LF (alimentación de línea, por sus siglas en inglés) para delinear el fin de la misma. (5) El caracter de LF también sirve como un caracter de sincronismo que indica que la estación transmisora está preparada para recibir una respuesta inmediata. El formato de los caracteres en este tipo de trama es ASCII, (5) o

sea el valor a transmitir es el código hexadecimal del caracter que se desea enviar. En la siguiente tabla se muestra la estructura de este tipo de trama. En ella se especifica el nombre del campo, valor predeterminado del campo (si posee) y el tamaño en bit del mismo.

Formato del mensaje ASCII						
Inicio	Dirección	Código de función	Datos	Error	Fin de trama	
´:´	-	-	-	-	CR	LF
7 bits	14 bits	14 bits	n x 14 bits	14 bits	7 bits	7 bits

Tabla 2. Formato del mensaje Modbus Omni ASCII.

1.5.2 Trama Modbus Omni RTU y formato de mensaje.

La sincronización de trama sólo puede ser mantenida en el modo de transmisión RTU mediante la simulación de un mensaje sincrónico. El 'OMNI' controla el tiempo transcurrido entre la recepción de caracteres. Si 3,5 intervalos de tiempo transcurren sin un nuevo caracter o de la finalización de la estructura, la trama se hace cero y los siguientes bytes serán procesados en busca de una dirección válida. (5) El formato de los caracteres en este tipo de trama es binario de 8 bits, o sea el valor a transmitir es el valor binario del caracter escrito. En la siguiente tabla se muestra la estructura de cada trama. En ella se especifica el nombre del campo y el tamaño en bit del mismo.

Formato del mensaje RTU			
Dirección	Código de función	Datos	Error
8 bits	8 bits	n x 8 bits	14 bits

Tabla 3. Formato del mensaje Modbus Omni RTU.

1.5.3 Trama Modbus Omni TCP/IP y formato de mensaje.

El formato de los caracteres en las tramas TCP es binario de 8 bits, o sea, el valor a transmitir es el valor binario del caracter escrito. Este tipo de trama posee tres campos extra que no están presentes en la comunicación serial. El primero es el campo “Transacción” que contiene el identificador de la transacción. Este valor será diferente para cada solicitud que envié el maestro y será el mismo que el esclavo colocará en el mensaje de respuesta. El segundo campo es “Protocolo” que contiene el identificador del protocolo y

siempre será 0. Y el tercer campo “Longitud” contendrá la longitud del resto del mensaje. Además, las tramas TCP no poseen el campo de “Error” ya que el protocolo TCP/IP se encarga de hacerlo. En la siguiente tabla se muestra la estructura de cada trama. En ella se especifica el nombre del campo y el tamaño en bit del mismo.

Formato del mensaje TCP					
Transacción	Protocolo=0	Longitud	Dirección	Código de función	Datos
16 bits	16 bits	8 bits	8 bits	8 bits	n x 8 bits

Tabla 4. Formato del Mensaje Modbus Omni TCP/IP.

1.5.4 Características de los campos de las tramas.

Las tres tramas poseen varios campos comunes, los cuales serán descritos en este epígrafe.

Campo “Dirección”.

Estos bits indican la dirección de usuario asignado al dispositivo esclavo que va a recibir el mensaje enviado por el maestro. Y cuando el esclavo envía una respuesta, este campo informa al maestro esclavo se está comunicando. (5) En el modo de difusión, la dirección es cero (0). Todos los esclavos interpretan esto como una instrucción para leer y tomar medidas, pero no emitir un mensaje de respuesta.

Campo “Código de Función”.

El campo de código de función (FC por sus siglas en inglés) le dice al esclavo direccionado cuál es la función a realizar. El bit de orden más alto del campo de código de función es fijado por el esclavo para indicar si es o no una respuesta normal lo que se le transmite al maestro. (5) Este bit permanece a 0 si el mensaje es una consulta o un mensaje de respuesta normal y se pone a 1 si es un mensaje de error. En la siguiente tabla se indican los FC válidos y el tipo de función que representan.

FC (decimal)	**Función a realizar.**
1	Leer múltiples registros de tipo *Status*.

2	Leer múltiples registros de tipo *Status* (FC = 02 es idéntico al FC = 01, pero se puede utilizar por los dispositivos de comunicación que no son compatibles con el FC = 01).
3	Leer un *Mixed* o múltiples registros de 16 o 32 bits.
4	Leer un *Mixed* o múltiples registros de 16 o 32 bits (FC = 04 es idéntico al FC = 03, pero se puede utilizar por los dispositivos de comunicación que no son compatibles con el FC = 03).
5	Escribir un registro de tipo *Status*.
6	Escribir un registro que no sea de tipo *Status*, ni *ASCII Text Buffer.*
15	Escribir múltiples registros de tipo *Status*.
16	Escribir un *Mixed* o múltiples registros de 16 o 32 bits.
65	Leer un paquete de un registro de tipo *ASCII Text Buffer.*
66	Escribir un paquete de un registro de tipo *ASCII Text Buffer.*

Tabla 5. Códigos de función validos en Modbus Omni.

Campo "Error".

Este campo permite que los dispositivos maestros y esclavos puedan ver si existen errores en la transmisión. (5) Un mensaje transmitido puede ser alterado ligeramente debido al ruido eléctrico u otras interferencias mientras se está en camino de una unidad a otra. La comprobación de errores asegura que el maestro y el esclavo no respondan a los mensajes que han sido modificados durante la transmisión. El campo de comprobación de errores utiliza una comprobación de redundancia longitudinal (LRC) en el modo Modbus Omni ASCII y una verificación de CRC-16 en el modo Modbus Omni RTU. Los bytes seleccionados incluyen la dirección del esclavo y todos los octetos hasta la comprobación de errores. La comprobación se realiza con los datos en el modo binario o modo RTU.

<u>El modo de LRC</u>

La comprobación de errores es un número binario de 8 bits representado y transmitido como los valores hexadecimales de dos caracteres ASCII. Para la comprobación de errores, en primer lugar, se separan el caracter ':', CR y LF y luego se convierten los caracteres ASCII de hexadecimal a binario. Por último, se añaden los bytes binarios (incluyendo la dirección del esclavo) y dos complementos de resultado. Cuando

un mensaje es recibido se calcula nuevamente el LRC y se compara con el LRC que fue recibido. Cualquier caracter no ASCII hexadecimal y el caracter ´:´, CR, LF, se ignoran en el cálculo de la LRC.(5)

El modo de CRC

El mensaje es considerado como un número binario continuo, cuyo bit más significativo (MSB) se transmite en primer lugar. El mensaje es pre multiplicado por 16 (desplazado a la izquierda de 16 bits), a continuación, dividido por x16 + x15 + x2 + 1, expresado como el número binario (11000000000000101). Los cociente entero se omiten y el resto de 16 bits, se añade al mensaje (MSB primero) como los dos bytes de verificación CRC. El mensaje resultante incluyendo la convención, cuando se divide por el mismo polinomio (x16 + x15 + x2 + 1) en el receptor le dará un resto cero si no se han producido errores.(5)

Campo "Datos".

El campo de datos contiene la información necesaria para que el esclavo pueda realizar la función especificada en el campo FC o contiene los datos recogidos por el esclavo en respuesta a una consulta. (5) Esta información puede ser cadenas de texto, valores o código de excepción.

1.5.5 Respuesta Excepción.

Los errores de programación u operación son aquellos que involucran datos ilegales en un mensaje, no hay respuesta o dificultan la comunicación con un esclavo. Estos errores resultan en una respuesta de excepción del esclavo, dependiendo del tipo de error. (5) Cuando un mensaje del maestro es recibido por el esclavo con algún tipo de error, éste envía una respuesta al maestro haciéndose eco de la dirección del esclavo, el código de la función (con bit alto en 1), código de excepción y los campos de error de verificación (en el caso de Modbus Omni RTU y Modbus Omni ASCII). Los códigos de excepción válidos para el protocolo Modbus Omni se muestran en la siguiente tabla.

Códigos de Excepción	Descripción.
1	Código de función inválido.
2	Dirección de registro inválida.

3	Ilegal valor de dato.
4	Los datos no se pueden escribir.
5	Necesita una contraseña.

Tabla 6. Códigos de excepción de Modbus Omni.

1.6 Características de la simulación, y su relación con los simuladores.

La simulación es una imitación de las operaciones de un sistema o proceso real a lo largo del tiempo (Sistemas complejos). (6) También podemos definir como simulación la representación de un proceso o fenómeno mediante otro proceso más simple, el cual permite analizar las características del proceso real. (7) De forma concreta, la simulación es utilizar modelos simplificados de un proceso o sistema real, con el objetivo de reproducir de manera suficiente aquellos comportamientos del proceso o sistema que interesa estudiar, pero descartando aquellos otros que no se consideran representativos y que pudieran complicar el desarrollo. Pero no es sólo esto, ya que se pueden encontrar ejemplos de simulación en el actuar cotidiano de cada persona. Por ejemplo, se puede ver la simulación cuando una maestra somete a sus pupilos a un examen extra con vistas a constatar los posibles resultados en un examen del ministerio, o en aplicaciones recreativas, o en la construcción de infraestructuras por medio de maquetas, hasta el entrenamiento virtual de los pilotos de combate.

En síntesis el término “simulación” es muy amplio y existen varios enfoques para analizarlo. Esta investigación está centrada en la simulación como herramienta informática. En la actualidad el incremento de la potencia del *hardware*, unido a la bajada de precio del mismo y el desarrollo a gran escala de *software*, hacen que la accesibilidad de los *software* de simulación haya llegado a un nivel doméstico (un ejemplo de ello son los juegos donde se simula un ambiente virtual con fines recreativos). Por otra parte, la simulación de procesos es una herramienta de gran utilidad en la ingeniería industrial, gracias a que permite representar un proceso mediante otro más simple y entendible.

Partiendo de este análisis se define como simulador un artefacto que permite la simulación de un sistema, reproduciendo su comportamiento en el tiempo. El mismo requiere de modelos de validez, que verifique su correcto funcionamiento. A pesar de esto no obtiene resultados exactos, pero permite modelar sistemas complejos. Un simulador no tiene por qué representar virtualmente todo el sistema real, puede reproducir

sólo algunos de su subprocesos. Además, un mismo sistema real puede tener varios modelos. También puede responder ciertas interrogantes para apoyar el diseño de sistemas reales. De forma general la simulación no resuelve los problemas por sí misma, sino que ayuda a identificar los problemas relevantes y evaluar cuantitativamente las soluciones alternativas.

1.6.1 ¿Por qué son necesarios los modelos de simulación o prototipos?

La experimentación de un sistema real o procesos no siempre es factible. En muchas ocasiones la ejecución del sistema encuentra uno o varios obstáculos que pueden llegar hasta paralizar su implementación. Uno de los obstáculos más comunes son los de índole económica. En la esfera industrial cotidianamente se implementan aplicaciones para automatizar la producción; estas aplicaciones son concebidas para garantizar que las operaciones en zonas críticas del proceso se ejecuten correctamente. Por ello, es indispensable garantizar que la misma posea la menor cantidad de errores posibles, ya que de ocurrir alguno podría ocasionar pérdidas para la institución. La solución óptima sería instalar la aplicación en un local de prueba, pero el monto de este tipo de pruebas no siempre puede ser costeado por la institución. Por tanto, en vez de instalar la aplicación en un local real, se adquieren o desarrollan ambientes virtuales que simulan el local real y sobre estos prueban las aplicaciones. Además, la representación real de algunos sistemas puede ser sencillamente imposible; uno de los causantes de este tipo de situación son las limitaciones tecnológicas, ya que existen procesos que a pesar del desarrollo tecnológico actual aún no se pueden hacer. También existen sistemas, que a pesar de ser necesarios, no se puede predecir con exactitud si el resultado final será el esperado o traerá consecuencias negativas, una de las soluciones viables es crear un simulador que imite su comportamiento para evitar un posible colapso. También puede ser útil cuando sólo se necesita hacer estudios y/o experimentos de un sistema, por ello un caso común donde son necesarios los simuladores es cuando se desarrolla un nuevo producto.

1.6.2 Áreas de aplicación de los simuladores.

Las esferas donde su puede aplicar la simulación son muy amplias y diversas, en el área industrial se destacan: Los sistemas de Computadoras, para evaluar el *hardware* o los requisitos de *software*. Las telecomunicaciones, para diseñar sistemas de comunicación o protocolos para mensajería, etc. La

industria del transporte y la energía, para diseñar autopistas, metros, puertos, cableados, etc. Aplicaciones Militares y Navales, con el objetivo de evaluar nuevas armas o tácticas. En la fabricación de productos, con el fin de diseñar y analizar políticas de planificación, inventarios, prototipos, etc.

1.6.3 ¿Cuándo simular?

Como regla general, la simulación es apropiada cuando desarrollar un modelo real es muy difícil o costoso o quizás aún imposible. Además, cuando el sistema tiene una o más variables aleatorias relacionadas. También se puede simular cuando la dinámica del sistema es extremadamente compleja y no se cuenta con los recursos necesarios para hacerlo o cuando el objetivo es observar el comportamiento del sistema sobre un período de tiempo, y no es factible instalar el sistema real. Para hacer demostraciones o poner a prueba algún proceso.

1.6.4 Ventajas de la simulación:

A continuación se muestran algunas de las ventajas que trae aparejada la implementación de un sistema de simulación. (7)

- Se puede experimentar con nuevos diseños sin tener que construirlos.
- Mejorar el funcionamiento de sistemas reales complejos.
- Disminuir inversiones y gastos de operación.
- Reducir el tiempo de desarrollo de un sistema.
- Minimizar el riesgo de que el sistema se comportará de forma inesperada.
- Conocer oportunamente hechos relevantes y efectuar cambios en el momento oportuno.
- Un modelo de simulación puede ser más amplio y robusto con respecto a los cambios en las características de los parámetros de entrada que un modelo analítico que sólo es válido bajo un conjunto de suposiciones.
- Flexibilidad para modelar las cosas tal como son (no importa si son difíciles de explicar y complicadas).
- Diagnosticar problemas.

- Permite modelar la incertidumbre (la única cosa segura es que nada es seguro, o sea, peligro de ignorar la variabilidad y la incertidumbre, validez del modelo).

1.6.5 Desventajas de la simulación:

A pesar de la versatilidad y usabilidad de este tipo de sistemas, poseen algunas inconvenientes. Entre ellos se encuentran: (7)

- Puede ser costosa y consumir mucho tiempo inicialmente.
- Construir modelos precisa un entrenamiento especial.
- Los resultados pueden ser difíciles de interpretar.
- Por lo general son ignorados los factores humanos y tecnológicos.
- Peligro de poner demasiada confianza en los resultados de la simulación.

1.7 Otros simuladores.

Se analizaron varias herramientas con el objetivo de determinar si las mismas reúnen las condiciones necesarias para ayudar en el proceso de prueba del manejador Modbus Omni. A continuación se describen algunos de estos productos y el por qué no se pudieron utilizar.

Modbus Slave: Esta aplicación fue desarrollada en la línea de Adquisición. Fue implementada siguiendo la lógica del resto de los manejadores implementados en la línea, pero que no funciona como maestro, sino como un esclavo. Esta aplicación ha sido probada en reiteradas ocasiones y ha demostrado su valía. La deficiencia encontrada en este producto que no lo hace óptimo para la tarea encomendada es que es un esclavo del protocolo Modbus, por tanto no posee el mismo mapa de memoria, ni tiene soporte para registros 32 bits o cadenas de caracteres. Además, esta aplicación sólo puede comunicarse a través de la variante Modbus TCP/IP. Por tanto, se descartó la idea de utilizar esta herramienta.

Modsim: Esta herramienta está concebida para comportarse como un esclavo dentro de una red de comunicación Modbus. A diferencia del Modbus Slave si soporta la comunicación por un puerto serie, pero

no tiene soporte para registros 32 bits o cadenas de caracteres. Además, no permite la simulación de eventos, ni redes seriales. Por tanto, se desechó la opción de utilizarlo.

MatrikonOPC Server for Omni Flow Computers: El servidor OPC para Computadores de flujo OMNI proporciona conectividad interoperable y segura de flujo OMNI 3000 y 6000 y sus equipos de medición de datos críticos. El servidor OPC para Computadores de flujo OMNI ofrece a los usuarios la capacidad de escoger de una lista generada automáticamente de las etiquetas y la eliminación de la tarea de trazar manualmente las direcciones Modbus. También proporciona un acceso seguro a los reportes de texto OMNI, el apoyo a los canales de comunicación redundantes o dispositivos, y la optimización de paquetes de datos. A pesar de ello no soporta la generación de algunos de los eventos necesarios y no es una herramienta libre, por tanto, no se puede acceder a su código para modificarlo y agregarle estas funcionalidades. Además es una aplicación para el sistema operativo Windows.

1.8 Estilo arquitectónico basado en capas.

La arquitectura basada en capas se enfoca en la distribución de roles y responsabilidades de forma jerárquica, proporcionando una forma muy efectiva de separación de responsabilidades. El rol indica el modo y tipo de interacción con otras capas, y la responsabilidad indica la funcionalidad que está siendo desarrollada. (8) El estilo de arquitectura basado en capas se identifica por las siguientes características: Describe la descomposición de servicios de forma que la mayoría de la interacción ocurre solamente entre capas vecinas. Las capas de una aplicación pueden residir en la misma máquina física (misma capa) o puede estar distribuido sobre diferentes computadores (n-capas). Los componentes de cada capa se comunican con otros componentes en otras capas a través de interfaces muy bien definidas. Este modelo ha sido descrito como una “pirámide invertida de re-uso” donde cada capa agrega responsabilidad y abstracción a la capa directamente sobre ella.

1.9 Herramientas de desarrollo empleadas.

Para el desarrollo de esta aplicación es necesaria la utilización de varias herramientas. Primeramente se necesita un entorno de desarrollo integrado o por sus siglas en inglés IDE que no es más que un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios. Un IDE es un entorno

de programación que ha sido empaquetado como un programa de aplicación, por lo general consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario, conocida también como GUI (del inglés Graphical User Interface). Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Las principales características con que deben contar los IDE son: flexibilidad, estabilidad y documentación existente.

Se utilizó C++ como lenguaje de programación para el desarrollo. C++ surge en 1980 gracias a Bjarne Stroustrup. Es un lenguaje imperativo orientado a objetos derivado del C. En realidad es un súper conjunto de C, que nació para añadirle cualidades y características de las que carecía, como son: las clases y el polimorfismo, los tipos de datos genéricos, la posibilidad de declarar variables en cualquier lugar del programa y además un motor de objetos con herencia múltiple, que permite combinar la programación imperativa de C, con la programación orientada a objetos. El resultado es que como su ancestro, sigue muy ligado al *hardware* subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. (9)

Se utilizó el IDE QT Creator: es un IDE para el desarrollo de programas mediante las librerías de Qt (Quasar Technologies) la cual utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de *bindings*. Además, incluye un depurador visual, un sistema integrado GUI y un diseñador de formularios. Las características del editor incluyen resaltado de sintaxis y autocompletado, pero no tabulaciones. Qt Creator utiliza el compilador de C++ a partir de la colección de compiladores de GNU en Linux y FreeBSD. (10)

El objetivo de esta investigación es desarrollar una aplicación que intercambie datos a través de una red; para ello se utilizó la biblioteca TransporProvider desarrollada en la línea de manejadores del proyecto GALBA. La misma está basada en la Asio C++ Library. Su principal función es permitir el intercambio de información a través del puerto serie o del protocolo TCP/IP. Esta biblioteca posibilita el intercambio asíncrono, dándole una mayor eficiencia en tiempo y recursos a los manejadores que la emplean. Su interfaz es intuitiva y fácil de utilizar. El TransportProvider puede ser utilizado por cualquier sistema que necesite comunicarse con dispositivos de campo. Es multiplataforma, ya que puede ser utilizada en

cualquiera de los sistemas operativos más conocidos y compilada con cualquiera de los compiladores de C++.

La herramienta CASE utilizada fue el Visual Paradigm for UML para el modelado de la solución. La decisión está basada en las facilidades que brinda esta aplicación de representar todos los tipos de diagramas de clases, generar código desde diagramas, y generar documentación. Además, es un *software* multiplataforma. También se estudió la herramienta Rational Rose, pero se descartó debido a que no es multiplataforma y no es una herramienta libre. Y como lenguaje de modelado se utilizó UML (siglas inglesas de Lenguaje de Modelado Unificado), el cual es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Ofrece un estándar para describir un plano del sistema, incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Es un lenguaje de modelado para especificar o para describir métodos o procesos. (11)

1.10 Metodología de desarrollo de *software*.

Se entiende por metodología de desarrollo una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del *software*. La finalidad de una metodología de desarrollo es garantizar la eficacia (cumplir los requisitos iniciales) y la eficiencia (minimizar las pérdidas de tiempo) en el proceso de generación de *software*. (12)

Las metodologías, en los últimos tiempos, se han dividido en dos grupos: los llamados métodos pesados y los métodos ligeros. La diferencia fundamental entre ambos es que mientras los métodos pesados intentan conseguir el objetivo común por medio de orden y documentación, los métodos ligeros (también llamados metodologías ágiles) tratan de mejorar la calidad del *software* por medio de una comunicación directa e inmediata entre las personas que intervienen en el proceso.

1.10.1 Programación Extrema (XP).

XP: es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el

cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (13)

XP, como toda metodología ágil, intenta reducir la complejidad del *software* por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción. XP intenta minimizar el riesgo de fallo del proceso por medio de la disposición permanente de un representante competente del cliente a disposición del equipo de desarrollo. El representante debería estar en condiciones de contestar rápida y correctamente a cualquier pregunta del equipo de desarrollo de forma que no se retrase la toma de decisiones, de ahí lo de competente. XP define “Historias de Usuarios” como base del *software* a desarrollar. Estas historias las escribe el cliente y describen escenarios sobre el funcionamiento del *software*, que no sólo se limitan a la GUI si no también pueden describir el modelo, dominio, entre otros aspectos. A partir de las Historias de Usuarios (HU) y de la arquitectura perseguida se crea un plan de entregables entre el equipo de desarrollo y el cliente. (13)

Para cada entregable se discutirán los objetivos de la misma con el representante del cliente y se definirán las iteraciones necesarias para cumplir con los objetivos del entregable. El resultado de cada iteración es un programa que se transmite al cliente para que lo juzgue. Sobre la base de su opinión se definen las siguientes iteraciones del proyecto y si el cliente no está contento se adaptará el plan de entregables e iteraciones hasta que el cliente dé su aprobación y el *software* esté a su gusto. (13)

En XP se programará sólo la funcionalidad que es requerida para el entregable actual. Es decir, una gran flexibilidad y capacidad de configuración sólo será implementada cuando sea necesaria para cumplir los requerimientos del entregable. Se sigue un diseño evolutivo con la siguiente premisa: conseguir la funcionalidad deseada de la forma más sencilla posible. De ahí una variación educada del famoso KISS (mantén las cosas tan sencillas como sea posible). Este diseño evolutivo hace que no se le dé apenas importancia al análisis como fase independiente, puesto que se trabaja exclusivamente en función de las necesidades del momento. (13)

Roles XP.

Los roles de acuerdo con la propuesta original de Beck son:

- Programador. El programador escribe las pruebas unitarias y produce el código del sistema.
- Cliente. Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.
- Encargado de pruebas (Tester). Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- Encargado de seguimiento (Tracker). Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- Entrenador (Coach). Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- Consultor. Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
- Gestor (Big boss). Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

Proceso XP.

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

- El cliente define el valor de negocio a implementar.
- El programador estima el esfuerzo necesario para su implementación.
- El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
- El programador construye ese valor de negocio.
- Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el *software* o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.
El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción o Implementación, Mantenimiento y Muerte del Proyecto.

1.10.2 Proceso Unificado de Racional (RUP).

RUP es un método pesado que intenta reducir la complejidad del *software* por medio de estructura y la preparación de las tareas pendientes en función de los objetivos de la fase y actividad actual. RUP es uno de los procesos más generales de los existentes actualmente, ya que en realidad está pensado para adaptarse a cualquier proyecto, y no tan sólo de *software*. (14)

RUP se basa en casos de uso para describir lo que se espera del *software* y está muy orientado a la arquitectura del sistema, basándose en UML como herramienta principal. Es un proceso muy general y muy grande, por lo que antes de usarlo habrá que adaptarlo a las características de la empresa. Por suerte ya hay muchos procesos descritos en internet que son versiones reducidas del RUP. (14)

Fases de RUP.

RUP define un total de 4 fases. En cada fase se ejecutarán una o varias iteraciones (de tamaño variable según el proyecto) y dentro de cada una de ellas seguirá un modelo de cascada para los flujos de trabajo que requieren las nuevas actividades anteriormente citadas. Las fases son:

- Inicio (puesta en marcha)
- Elaboración (definición, análisis, diseño)
- Construcción (implementación)
- Transición (fin del proyecto y puesta en producción)

Flujos de trabajo de RUP.

RUP define nueve flujos de trabajo a realizar en cada fase del proyecto. El proceso define una serie de roles que se distribuyen entre los miembros del proyecto y que definen las tareas de cada uno y el resultado (artefactos) que se espera de ellos. Los flujos son:

- Modelado del negocio.
- Requisitos.
- Análisis y diseño.
- Implementación.
- Prueba.
- Despliegue.
- Gestión de cambios y configuración.
- Gestión de proyectos.
- Gestión del entorno.

1.11 Selección de la metodología.

Durante la etapa de concepción del proyecto entre el cliente y el proveedor se acordó, que la tarea principal del mismo era desarrollar una aplicación funcional, a pesar del poco tiempo de desarrollo que se contaba. La documentación generada, únicamente abordaría la descripción de los componentes principales del sistema. Por ello se acordó por ambas que una vez terminado el desarrollo de la aplicación, el proveedor entregaría al cliente el producto terminado junto con un pequeño número de artefactos, indispensables para el entendimiento del *software*.

De las metodologías estudiadas la más adecuada a este tipo de trabajo es XP, la cual está centrada en el programador y en la programación intensiva. Además, la misma sólo genera los artefactos indispensables durante el proceso de desarrollo. Esto se ajusta a lo acordado con el cliente. En la siguiente tabla se muestra una comparación entre XP y RUP sobre la cual se basó esta selección.

Metodología Rational Unified Process (RUP).	**Metodología Extreme Programming (XP).**
RUP Forma disciplinada de asignar tareas y responsabilidades en una empresa de desarrollo (quién hace qué, cuándo y cómo). Método pesado Un cambio en las etapas de vida del sistema incrementaría notablemente el costo.	XP Nace en busca de simplificar el desarrollo del *software* y que se lograra reducir el costo del proyecto. Método ligero: No produce demasiados gastos sobre las actividades de desarrollo, y no impide el avance de los proyectos. Reduce el costo del cambio en las etapas de vida del sistema.
Requiere un grupo grande de programadores para trabajar con esta metodología. RUP es un marco del proyecto que describe una clase de los procesos que son iterativos e incrementales. RUP define un grupo de las actividades y artefactos que usted necesita construir. RUP es el proceso de desarrollo más general de los existentes actualmente. Los procesos de RUP estiman tareas y horario del plan midiendo la velocidad de iteraciones concerniente a sus estimaciones originales. Las iteraciones tempranas de proyectos conducidos	Se requiere de un grupo pequeño de programadores para trabajar con esta metodología entre 2 – 15 personas y estas irán aumentando conforme sea necesario. Sus programadores pueden ser ordinarios. Combina las que han demostrado ser las mejores prácticas de desarrollo de *software*, y las lleva al extremo. Se rediseñará todo el tiempo, dejando el código siempre en el estado más simple posible. Se harán pruebas todo el tiempo, no sólo de cada nueva clase (pruebas unitarias), sino que también los clientes comprobarán que el proyecto va satisfaciendo los requisitos (pruebas funcionales).

RUP se enfocan fuertemente sobre arquitectura del *software*; la puesta en práctica rápida de características se retrasa hasta que se ha identificado y se ha probado una arquitectura firme. RUP proporciona muchas ventajas sobre XP le da énfasis en los requisitos y el diseño. La ventaja principal de RUP es que se basa todo en las mejores prácticas que se han intentado y se han probado en el campo. (En comparación con XP que se basa en las prácticas inestables que utilizaron juntas se evita que se derribe).	Las pruebas de integración se efectuarán siempre, antes de añadir cualquier nueva clase al proyecto, o después de modificar cualquiera existente (integración continua). Las iteraciones serán radicalmente más cortas de lo que es usual en otros métodos, esto permite beneficiarse de la retroalimentación tan a menudo como sea posible.

Tabla 7. Comparación entre RUP y XP

2 Capítulo 2 Diseño de la solución propuesta.

Introducción.

El objetivo de este capítulo es iniciar el proceso de desarrollo del sistema propuesto como solución. Esta etapa estará guiada por la metodología XP. Además, se presentan los principales artefactos generados en ella.

2.1 Características de la comunicación con los PLC.

El flujo normal de eventos del proceso de comunicación entre los dispositivos de campo (PLC) y un sistema que acceda sus servicios se rige por el modelo maestro/esclavo. El objetivo de este tipo de sistema es que un operador central pueda controlar diversas actividades, cada una de ellas será controlada por un dispositivo de campo, el cual, para los efectos prácticos del sistema será tratado como un esclavo y el sistema instalado en el puesto de trabajo del operador cumplirá la función del maestro. Al iniciar el sistema en el maestro se configurarán todas las direcciones de los esclavos a quien éste controlará. Una vez configurado el maestro, el operador podrá enviar mensajes solicitando la lectura o escritura de los registros de un esclavo. Ambas partes podrán estar comunicándose vía Ethernet o serial. Los mensajes enviados, tanto por el dispositivo maestro como por los dispositivos esclavos, serán concebidos según las especificaciones del protocolo de comunicación que ambos utilizan. O sea, los mensajes enviados por el manejador Modbus Omni y las respuestas de los dispositivos de campos, serán las tramas Modbus Omni RTU o Modbus Omni ASCII o Modbus Omni TCP/IP descritas en el capítulo 1.

2.2 Propuesta de solución.

Desarrollar un simulador capacitado para recolectar, interpretar y responder las solicitudes que un maestro le envíe a través del protocolo Modbus Omni; para lo cual el sistema se comportará como un esclavo y tendrá como objetivo probar un nuevo *software*.

Para ello el simulador brindará la opción de simular redes seriales, o sea, en el sistema se podrá crear más de un esclavo, cada uno con su propio canal de comunicación. Esta opción permitirá al maestro

comunicarse de forma independiente con cada uno de los esclavos creados. Además el sistema mostrará la información referente a cada canal de comunicación y cada esclavo, estos datos serán actualizados cada un segundo. También se desea emular cambios complejos en la comunicación, por tanto el simulador brindará la opción de generar cuatro tipos de eventos capaces de alterar dicha comunicación.

El primer tipo de evento soportado por la aplicación es “Caída sincrónica de dispositivos” y tiene como objetivo simular la caída de los servicios de un grupo de esclavos (previamente seleccionados por el usuario). El segundo se denomina “Caída asincrónica de dispositivos”, el cual es similar al anterior, pero sólo uno de los esclavos dentro de un grupo seleccionado por el usuario, simulará la caída de sus servicios. “Ruido en el canal” es el tercer tipo de evento y simulará ruido en el canal de comunicación de uno o varios esclavos seleccionados por el usuario, o sea, los elementos asociados a este evento enviarán tramas incorrectas al maestro, situación que puede ocasionarse cuando en el medio de comunicación existen interferencias (por ejemplo ruido eléctrico) que alteran el contenido de los mensajes que viajan a través de él. Y el último tipo de evento se nombró “Variación de un registro” y brindará la opción de cambiar el valor de los registros de un esclavo o varios esclavos a partir de fórmulas matemáticas, o sea, el usuario proporcionará una función matemática (que sólo posean un valor variable) por ejemplo “x+7” por cada registro que desee cambiar. Luego el sistema evaluará un valor de “x” (x inicialmente vale 0 y se incrementara en uno cada vez que se ejecute el evento) en la función, para obtener un resultado y asignárselo al registro en cuestión.

2.3 Fase de exploración.

El ciclo de vida del proyecto según la metodología seleccionada comienza en esta fase. En ella el usuario define las historias de usuario (HU), asignándole a cada HU una prioridad y los programadores se encargan de realizar las estimaciones correspondientes. Además el equipo de desarrollo se familiariza con cada una de las herramientas y tecnologías que utilizará en el desarrollo del sistema y también explora diferentes posibilidades de conformar la arquitectura. (13)

2.3.1 Características funcionales del sistema. Historias de Usuarios (HU).

Las funcionalidades del sistema son modelados en XP como HU. Estas deben ser redactadas por el cliente, aunque los desarrolladores pueden brindar también su ayuda en su confección. El contenido que ellas abarcan debe ser concreto y sencillo. A continuación, aparecen los títulos de las HU generadas por el cliente.

- HU 1: Mostrar el contenido de los registros de un dispositivo. (ver Anexo 1 tabla 26).
- HU 2: Modificar el contenido de un registro. (ver Anexo 1 tabla 27).
- HU 3: Modificar el contenido de un registro colocándole un valor obtenido a través de una fórmula matemática. (ver Anexo 1 tabla 28).
- HU 4: Cambiar la configuración del mapa de memoria de los dispositivos de un canal de comunicación. (ver Anexo 1 tabla 29).
- HU 5: Cargar la configuración del mapa de memoria de los dispositivos de un canal de comunicación. (ver Anexo 1 tabla 30).
- HU 6: Salvar la configuración del mapa de memoria de los dispositivos de un canal de comunicación. (ver Anexo 1 tabla 31).
- HU 7: Simular fallas de caída de dispositivos sincrónicas. (ver Anexo 1 tabla 32).
- HU 8: Simular fallas de caída de dispositivos aleatorias. (ver Anexo 1 tabla 33).
- HU 9: Simular ruido en el canal. (ver Anexo 1 tabla 34).
- HU 10: Simular redes seriales. (ver Anexo 1 tabla 35).

2.3.2 Características no funcionales del sistema.

Las características no funcionales del sistema son aquellas que especifican criterios que pueden usarse para juzgar la operación de un sistema, en lugar de sus comportamientos específicos. Por tanto, se refieren a todas las particularidades del sistema que ni describen información a guardar, ni funciones a realizar.

Más específicamente el simulador deberá contar con una ayuda que garantice que un usuario inexperto pueda operar con más facilidad el *software*. Para su correcto funcionamiento se necesitará haber instalado con antelación la biblioteca TrasportProvider y el framework Qt. Podrá ser utilizado en máquinas que

posean un microprocesador Pentium 4 o superior. Además deberán poseer al menos 512 mb de RAM, se sugiere 1 gb de RAM o más para lograr una mayor eficiencia. El *software* debe ser confiable, estable y poseer una interfaz de usuario amigable.

2.4 Fase de planificación.

Durante esta fase el cliente y los desarrolladores acuerdan cuando será la posible fecha de entrega del producto. Estas operaciones se realizan partiendo de la prioridad dada por el usuario a cada HU y los programadores realizan la estimación del esfuerzo necesario para realizar cada una de ellas. (13) La unidad de medida seleccionada para representar el esfuerzo del grupo de desarrolladores al implementar una HU es el Punto Estimado (PE), que representa una semana de 6 días de desarrollo. A partir de este momento los miembros de los equipos de desarrollo trabajan el tiempo planeado sin ningún tipo de interrupción.

2.4.1 Estimación de esfuerzo.

Historias de usuarios.	Estimación de esfuerzo en PE.
HU 1	1.5
HU 2	1.5
HU 3	2
HU 4	2.5
HU 5	1
HU 6	1
HU 7	1
HU 8	1
HU 9	1
HU 10	3

Tabla 8. Estimación de esfuerzo.

2.4.2 Plan de iteraciones.

Después de haber sido descritas e identificadas las HU se propone el plan de iteraciones. El objetivo de este epígrafe es describir las iteraciones que serán necesarias para desarrollar el sistema, así como que HU están relacionadas con ellas. Una vez confeccionado este plan los desarrolladores tienen una línea de trabajo y deberán ganarse la confianza del cliente. En base a lo antes mencionado se decide realizar el sistema en tres iteraciones, las cuales se detallan a continuación:

Iteración 1: El objetivo de esta iteración es crear los dispositivos. En ellos se podrá almacenar información y acceder a ella. Este proceso debe ejecutarse lo más óptimo posible para lograr que la comunicación con el maestro que acceda a sus servicios, sea eficiente.

Iteración 2: El objetivo de esta iteración es lograr que los dispositivos conformen una red serial, donde cada uno actúe como un ente individual. Además lograr que los dispositivos puedan recibir, interpretar y responder solicitudes enviados por la red desde un maestro.

Iteración 3: El objetivo de esta iteración es lograr que las redes y los dispositivos creados puedan simular una serie de eventos que puedan afectar la comunicación con los dispositivos maestros.

2.4.3 Plan de entregables.

A partir del plan de iteraciones y las HU, se confeccionó el plan de entregas. El mismo tiene como objetivo definir las diferentes entregas que se harán y el orden de las mismas. Cada fila de la siguiente tabla representa una HU y cada columna una iteración. Cada entregable está situado en la fila de las HU que se implementan en él y en la columna de la iteración en que se elaborará.

Historias de Usuario	Iteración 1	Iteración 2	Iteración 3
HU 1	Entregable 1		
HU 2	Entregable 1		
HU 3			Entregable 3
HU 4		Entregable 2	
HU 5		Entregable 2	
HU 6		Entregable 2	
HU 7			Entregable 3
HU 8			Entregable 3
HU 9			Entregable 3
HU 10		Entregable 2	

Tabla 9. Plan de entregables

3 Capítulo 3 Diseño del sistema.

En este capítulo se describen el diseño del sistema. Según la metodología XP este proceso se debe realizar utilizando las tarjetas CRC y descarta la utilización de los diagramas de clases del diseño; pero para lograr una mejor comprensión de la estructura del sistema y en coordinación con el cliente, se decidió utilizar esta segunda variante.

3.1 Arquitectura del sistema.

El estilo arquitectónico seleccionado para el desarrollo de esta aplicación fue el “Estilo basado en capas” dado que sus características se ajustan al tipo de trabajo que se desea hacer. En el diseño del simulador se identificaron 3 capas: Capa de aplicación, Capa de protocolo y Capa de transporte. En el siguiente diagrama se muestra su jerarquía.

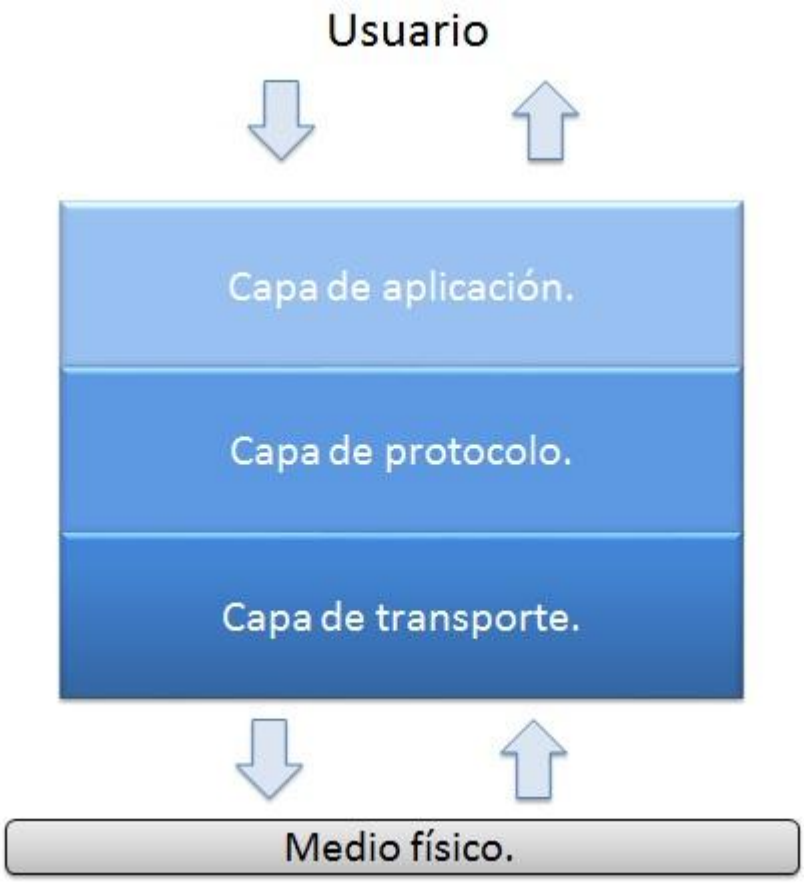

Ilustración 4. Arquitectura del Simulador Modbu Omni.

3.1.1 Capa aplicación.

Esta capa se concibió para servir de enlace entre la capa de protocolo y el usuario. Su objetivo específico es enviar a la capa de protocolo las órdenes del usuario y mostrar al usuario la información generada en la capa de protocolo. De esta forma, la manera en que el usuario interactúa con el simulador puede ser cambiada con facilidad sin tener que cambiar el funcionamiento interno del sistema. Para ello se utilizaron las clases y funcionalidades que brinda el framework de Qt para el desarrollo de GUI.

3.1.2 Capa de protocolo.

La capa de protocolo tiene como objetivo garantizar la lógica de comunicación entre el maestro que solicita los servicios y los dispositivos lógicos del sistema. Para ello se definieron dos componentes fundamentales, donde cada uno se encarga de cumplir un determinado número de tareas diferentes, pero interrelacionadas. Estos componentes son: “Lógica del protocolo” encargado de gestionar toda la información referente a cada transacción y “Gestor de eventos” encargado de generar eventos que puedan alterar la comunicación con el dispositivo maestro. Separar estas dos funcionalidades tiene la ventaja de que se pueda modificar o ampliar cualquiera de las dos partes sin afectar a la otra, un ejemplo específico sería el caso de necesitar añadir o modificar un evento determinado. En el siguiente diagrama se muestra como estos dos componentes se relacionan dentro de la capa de protocolo.

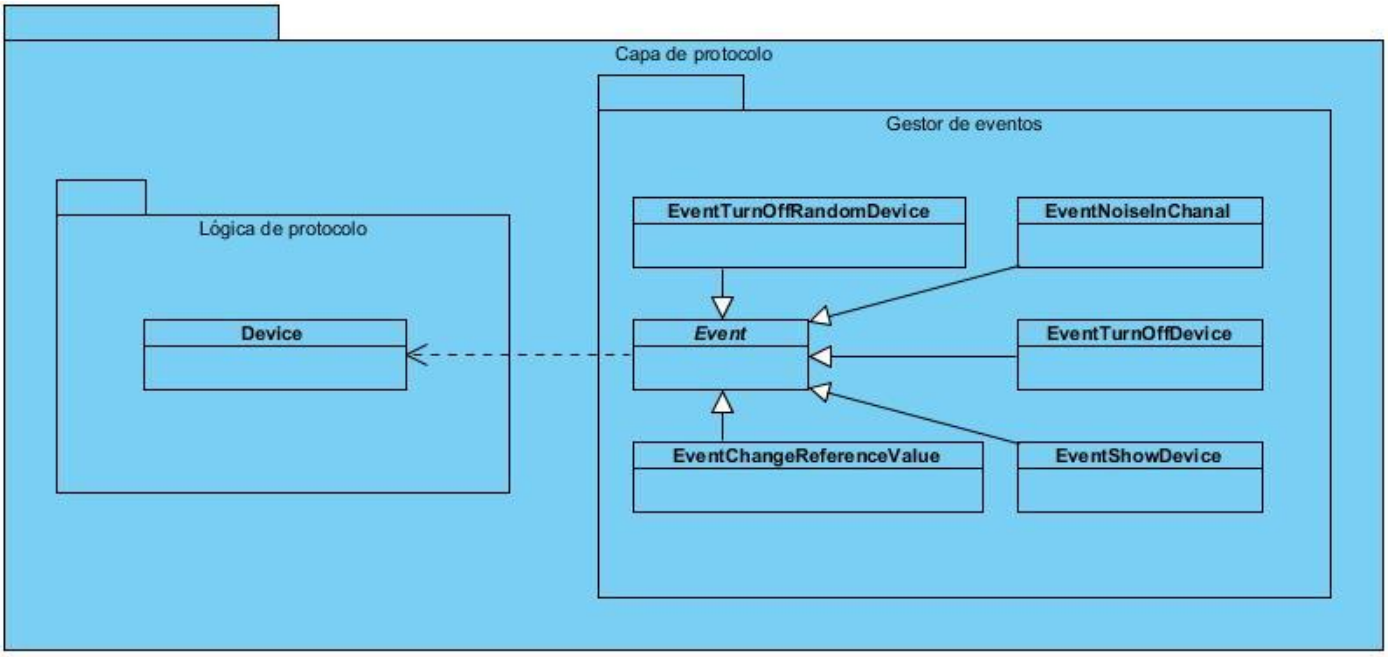

Ilustración 5. Capa de protocolo.

3.1.2.1 Lógica del protocolo

En la implementación de este componente se definieron dos clases fundamentales. La clase Connect encargada del manejo de la lógica en la comunicación por un canal exclusivo. Y la clase SessionTCP

encargada de manejar la lógica de la comunicación TCP/IP. Además fueron necesarias otro grupo de clases sobre las que se apoyarían las ya mencionadas para cumplir con su función. En el siguiente diagrama se muestran las clases de esta capa y las relaciones entre ellas.

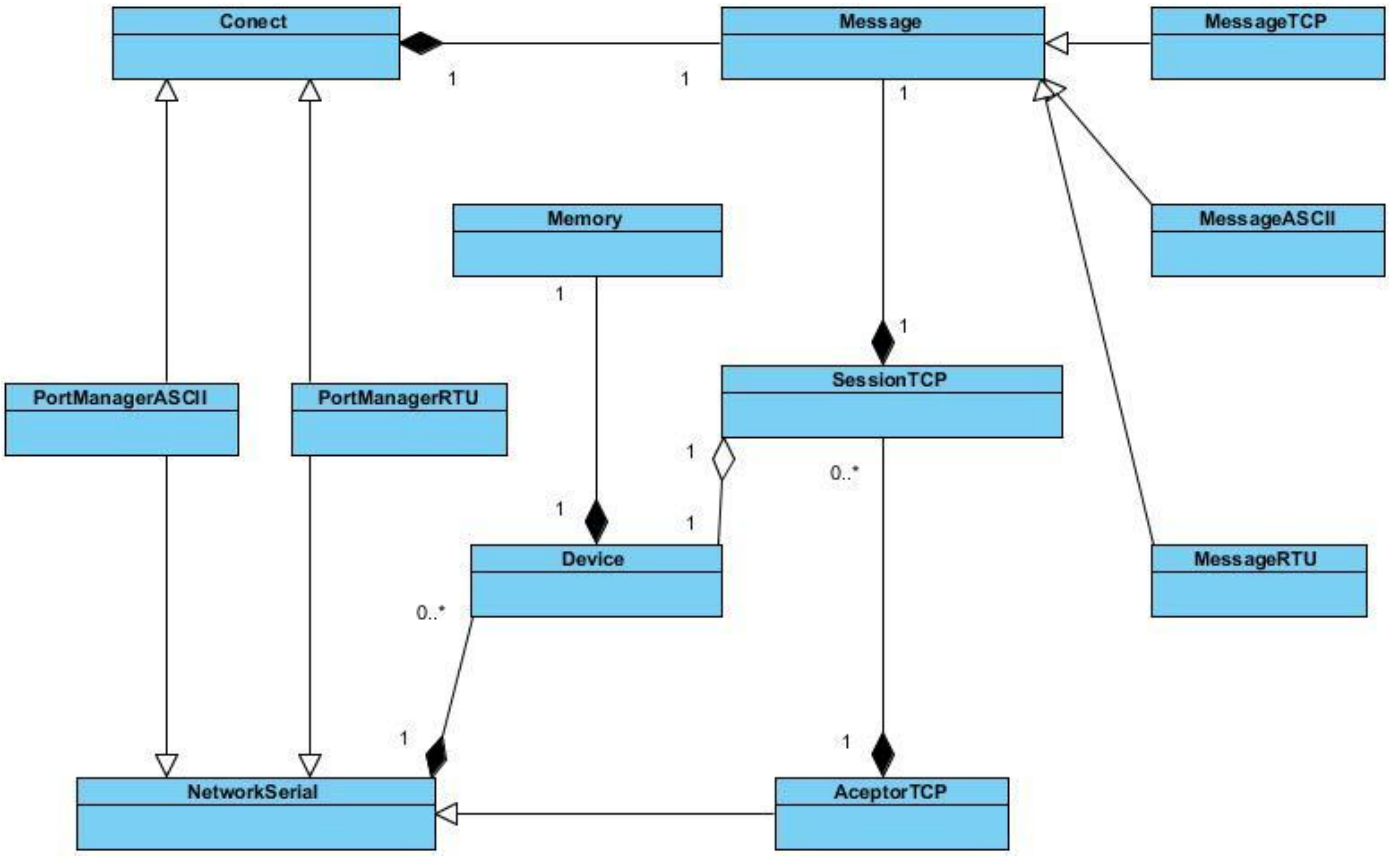

Ilustración 6. Componente lógica de protocolo.

A continuación se describirán con más detalles las funcionalidades de este grupo de clases divididas por funciones.

Almacenamiento de información.

Para simular más de un esclavo era necesario manipular información diferenciada para cada uno de ellos. Como solución a este problema se implementó la clase Device, que no es más que la abstracción lógica

del esclavo para el sistema. Esta posee una serie de atributos que representan el estado del mismo y toda la información de los registros asociados a él.

Para almacenar y gestionar los datos de un dispositivo lógico se agregó la clase Memory. Para la implementación de esta clase se necesitó superar varios obstáculos. Primeramente el volumen de datos a almacenar para un sólo dispositivo es considerable. Además, al brindar la opción de cambiar el mapa de memoria este volumen podría incrementarse. Por ello no se podía crear a partir de los tipos de datos convencionales del lenguaje C++ (ya que en computadoras de pocos recursos la aplicación sería lenta y cargaría el procesador de la máquina).

La primera solución fue guardar la información en fichero siguiendo un formato lógico y entendible. Esta decisión se tomó con el objetivo de que el fichero fuese entendible aún sin tener la aplicación ejecutándose, pero al correr la aplicación y guardar en el fichero un caso extremo, de un dispositivo, el fichero llegó a tener un tamaño de 512 mb, lo cual afectaba el rendimiento de la aplicación.

La segunda variante consistía en crear un fichero de forma dinámica por cada instancia de la clase Memory, dentro del cual se guardaría la información referente a la memoria creada. Para ello se utilizó las funcionalidades de la Biblioteca Estándar de C++, más específicamente de las funcionalidades *write* y *read* de las clases fstream (15) y ostream (15) contenida en dicha biblioteca. Luego la clase poseería una estructura donde se guardarían la posición en el fichero que ocupan los datos de todos los registros válidos (ver tabla 1). Crear esta estructura consume menos recursos que crear espacio de memoria para los datos de cada registro. De esta forma, cuando se quisiera leer o escribir un registro, se buscaría en la estructura de almacenamiento la posición en el fichero de los datos del registro en cuestión, y luego se escribiría o leería dicha posición en el fichero. Utilizando este método y el mismo caso extremo inicial, el fichero que llegaba a pesar 512 mb, se redujo a 3,9 mb. Por último, cada objeto de tipo Device posee una Memory único, en el siguiente diagrama se muestra esta relación.

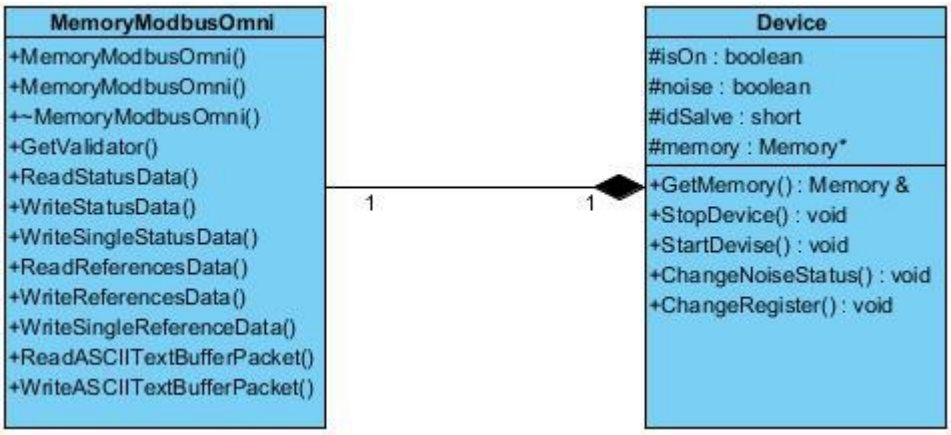

Ilustración 7. Clases relacionadas con el almacenamiento de datos.

Análisis de tramas.

Para lograr comprensión de una trama se implementaron las clases Message, MessageASCII, MessageRTU y MessageTCP. Su principal función es desarmar las tramas entrantes, detectando los posibles errores que pueda contener y armar el mensaje de respuesta que se enviará al maestro. Estas tareas se dividieron entre las cuatro clases antes mencionadas. La clase Message ensambla y desarticula los elementos comunes entre los tres tipos de tramas definidos en el protocolo y las tres restantes ejecutan las tareas específicas de una de estos tipos. En el siguiente diagrama se muestran estas clases y su relación.

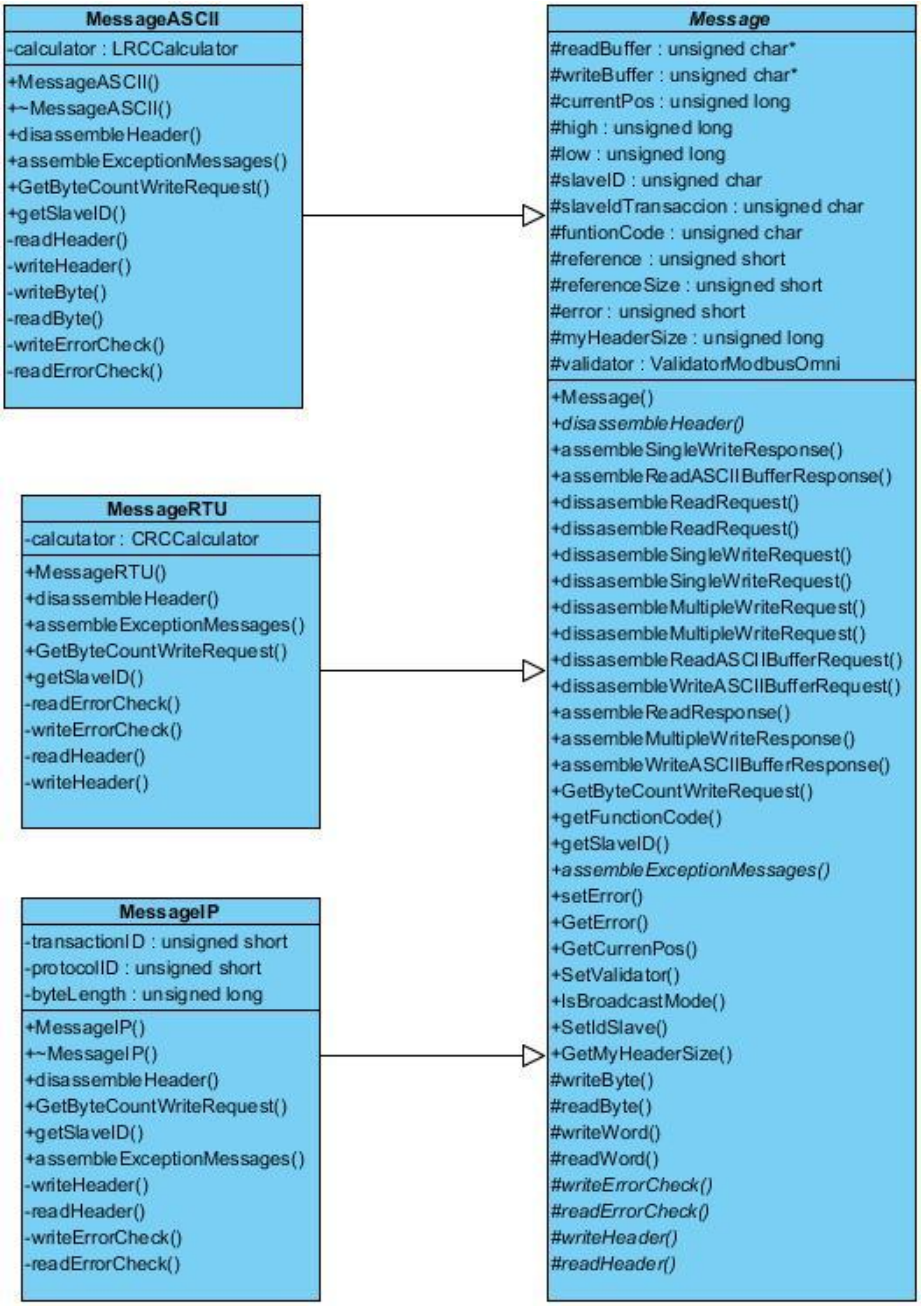

Ilustración 8. Clases relacionadas con el análisis de tramas.

Simulación de una red serial por un puerto serie.

Para el cumplimiento del HU 10 y lograr simular una red de dispositivos que se conectan utilizando el mismo puerto serial, se necesitó crear un controlador de puerto ya que si se le asigna el puerto a uno de los dispositivos el resto quedará sin conexión. Con este objetivo fueron creadas las clases PortManagerASCII y PortManagerRTU, hijas de la clase Connect abordada en los epígrafes anteriores. Estas contienen todos los dispositivos lógicos que se comunican a través del puerto controlado por él y funciona como intermediario entre el puerto serie y los dispositivos asociados a él. Cuando una trama es recibida, el controlador de puerto interpreta la solicitud para obtener el identificador del esclavo al cual es dirigido el mensaje, la operación que se desea realizar sobre él y los datos necesarios para ejecutar la operación. Este proceso se ejecuta valiéndose de una instancia de la clase MessageASCII o MessageRTU, dependiendo del tipo de manejador de puerto que sea. Luego utilizando la información obtenida de la trama, el manejador de puerto ejecuta dichas operaciones sobre el dispositivo lógico correcto y elabora el mensaje de respuesta adecuado. Por último, se encarga de enviar la respuesta al emisor. Para lograr una mayor claridad en el código y poder reutilizar funciones comunes en ambos controladores de puerto se agregó la clase NetworkSerial como padre de ellos. Dicha clase tiene la función de almacenar una serie de dispositivos lógicos y gestionar la adición, búsqueda y eliminación de estos. En el siguiente diagrama se muestra esta relación.

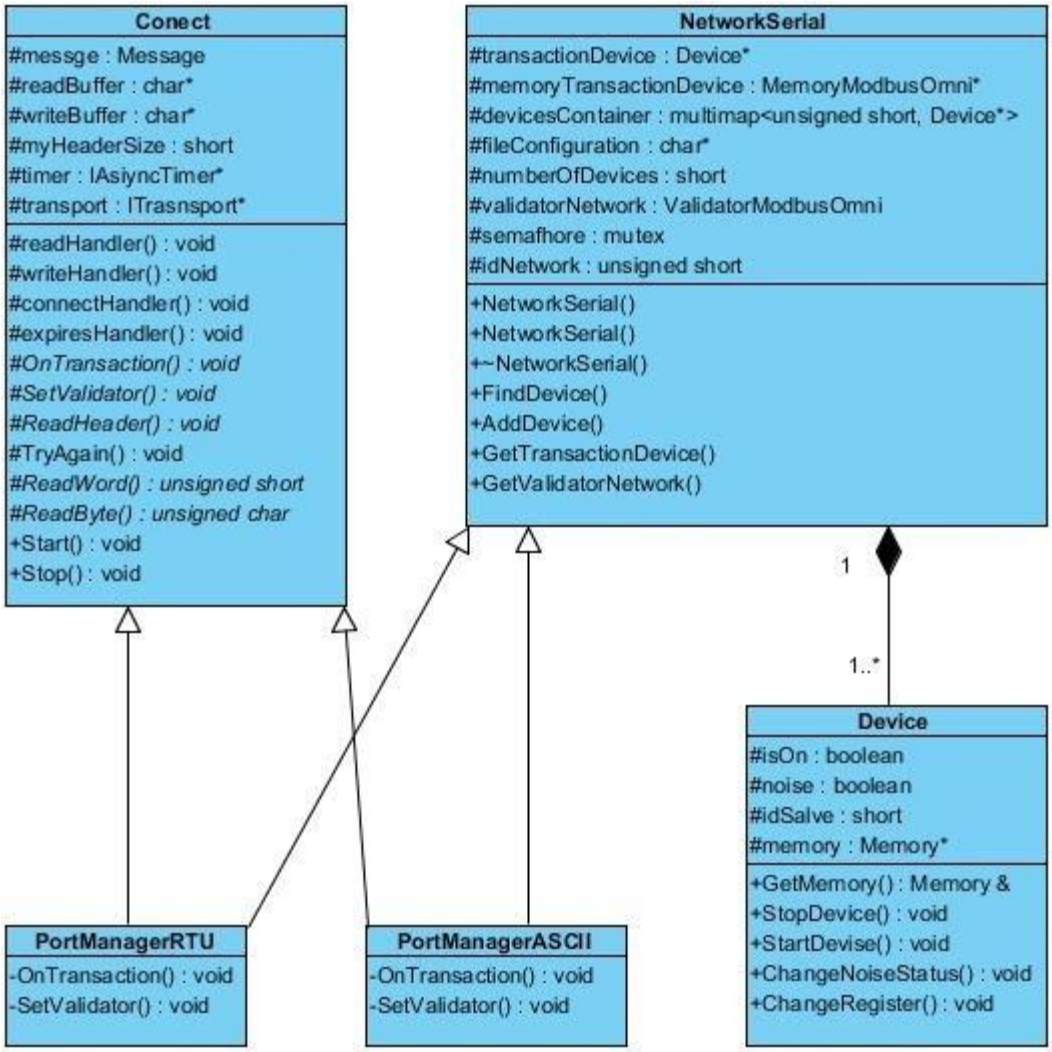

Ilustración 9. Simulación de una red serial por un puerto serie.

Gestión de conexiones por un puerto TCP/IP.

Dadas las características de este tipo de comunicación y con el objetivo de cumplir con el HU 10, se hizo necesario implementar la clase AceptorTCP. Esta tendrá la responsabilidad de gestionar todas las conexiones que arriben a un puerto determinado. Al llegar una solicitud de conexión, éste crearía una instancia de la clase SessionTCP asignándole dicha conexión, y esta se encargaría de gestionar las tramas que lleguen por este medio. Además la clase AceptorTCP, hereda de la clase NetworkSerial, de

esta forma adquiere las funcionalidades para almacenar dispositivos lógicos y gestionar la adición, búsqueda y eliminación de estos; cabe destacar que cada instancia del AceptorTCP funciona como un canal de comunicación TCP/IP y sólo se le podrá agregar un dispositivo lógico a dicho canal.

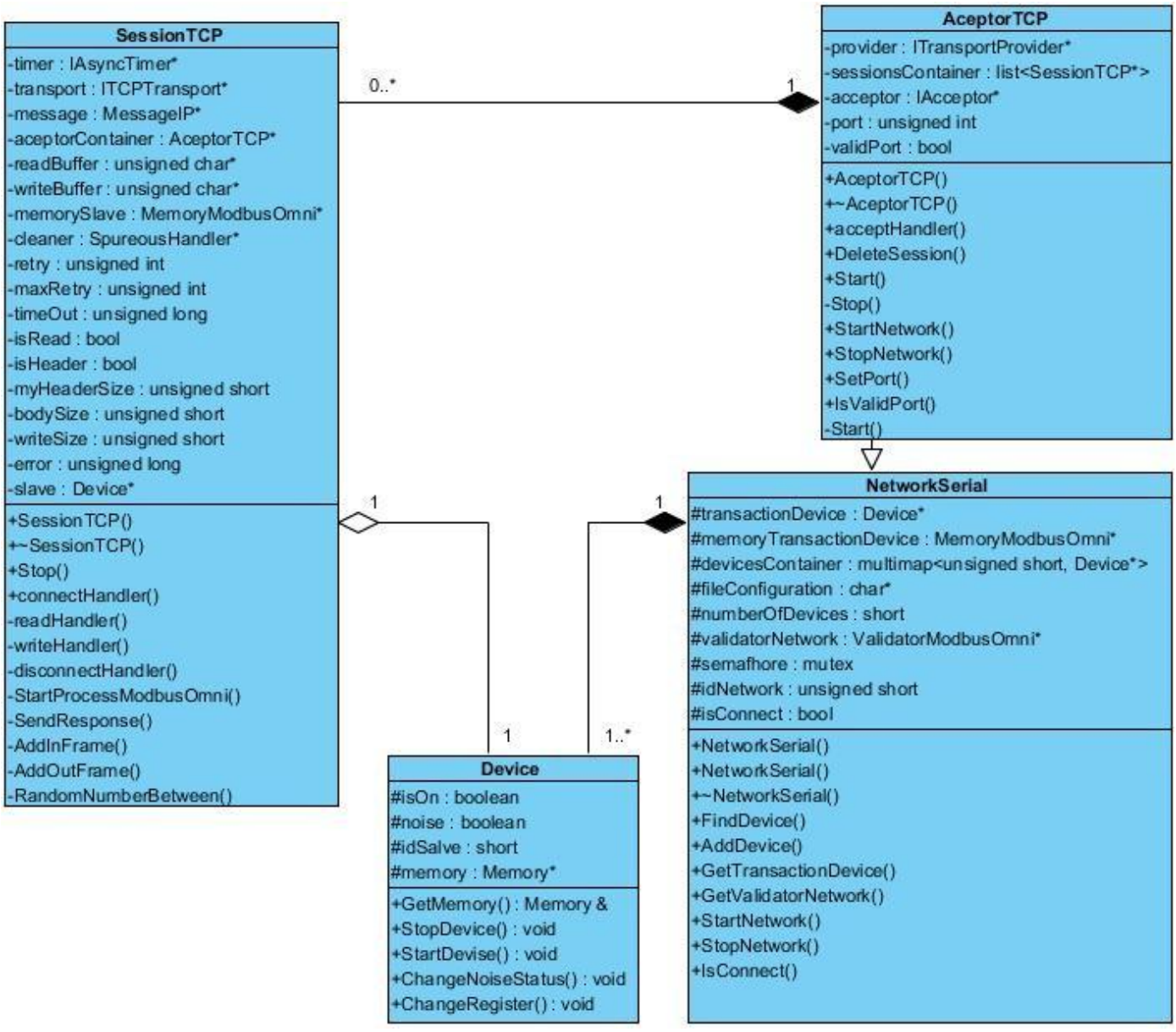

Ilustración 10. Simulación de una red serial por un puerto TCP/IP.

3.1.2.2 Gestor de eventos.

Para la ejecución de cada uno de los eventos que serán soportados por el simulador se implementó una clase.

Con el objetivo de cumplir con el HU 1 se definió la clase EventShowDevice, o sea esta clase se encarga de recolectar los datos que serán mostrados al usuario. Para ello la misma posee dos funciones básicas: leer los datos del dispositivo asociada a ella y notificar en forma de “callback” a la capa de aplicación que los datos han sido actualizados. Estas operaciones alternan su ejecución cada 500 milisegundos. Como resultado la información mostrada al usuario es actualizada cada un segundo.

Con el objetivo de cumplir con el HU 3 se definió la clase EventChangeReferenceValue. Esta clase se crea dado un intervalo de tiempo estático, una fórmula matemática, una referencia de tipo Short Intiger, Long Intiger o Floting Pointer (ver tabla 1) de la memoria de un dispositivo lógico y una variable ‘x’, de valor inicial 0. Cada vez que se cumpla su intervalo de tiempo se calculará el resultado de la fórmula matemática evaluada en ‘X’ y el resultado se escribirá en el registro seleccionado; por último se incrementará el valor de ‘X’ en 1. Si el resultado obtenido es mayor que el máximo admisible por el tipo del registro seleccionado, se le asignará 0 al registro y a la variable ‘X’. Sólo se define este evento para las variables de tipo numéricas (ver tabla 1), ya que no se puede evaluar una variable de tipo cadena de caracteres (ver tabla 1) o una variable de tipo booleana (ver tabla 1) en una fórmula matemática.

Con el objetivo de cumplir con el HU 7 se definió la clase EventTurnOffDevice. Esta clase se crea dado un grupo de dispositivos lógicos, un intervalo de tiempo y una bandera de estado que indicará si se desea apagar o encender los dispositivos asociados. Cada vez que se cumpla dicho intervalo de tiempo, si el valor de la bandera es ‘apagar’ todos los dispositivos lógicos ligados al evento serán apagados y de lo contrario serán encendidos. Cada vez que se ejecute el evento la bandera cambiará su valor de ‘apagar’ a ‘encender’ o viceversa.

Con el objetivo de cumplir con el HU 8 se definió la clase EventTurnOffRandomDevice. Esta clase se crea dado un grupo de dispositivos lógicos y un intervalo de tiempo. Cada vez que se cumpla dicho intervalo de tiempo, se seleccionará un dispositivo lógico del grupo de forma aleatoria. Luego si este dispositivo está encendido se apagará y si no se encenderá.

Con el objetivo de cumplir con el HU 9 se definió la clase EventNoiseChanal. Esta clase se crea dado un grupo de dispositivos lógicos, un intervalo de tiempo y una bandera de estado que indicará si se desea simular o no ruido en el canal. Cada vez que su cumpla dicho intervalo de tiempo, si el valor de la bandera es ‘simular’ todos los dispositivos lógicos ligados al evento comenzarán a agregar caracteres inválidos en las tramas que envíe y si el valor es ‘no simular’ los dispositivos comenzarán a enviar tramas correctas. Cada vez que se ejecute el evento la bandera cambiará su valor de ‘simular’ a ‘no simular’ o viceversa.

Estos 5 tipos de eventos tienen en común la gestión del tiempo; o sea, cada vez que se cumpla un intervalo determinado se ejecutan un conjunto de operaciones. Esta gestión se implementó en la clase Event, de la cual heredan todos los eventos, además esta clase obliga a sus hijos a definir un método Run en el cual se definirán las operaciones a realizar una vez que se quiera ejecutar el evento. De esta forma, si se desea agregar un nuevo evento sólo se debe agregar otro hijo a Event y si se desea cambiar un evento ya definido sólo se tiene que reimplementar su método Run. En el siguiente diagrama se muestra esta relación.

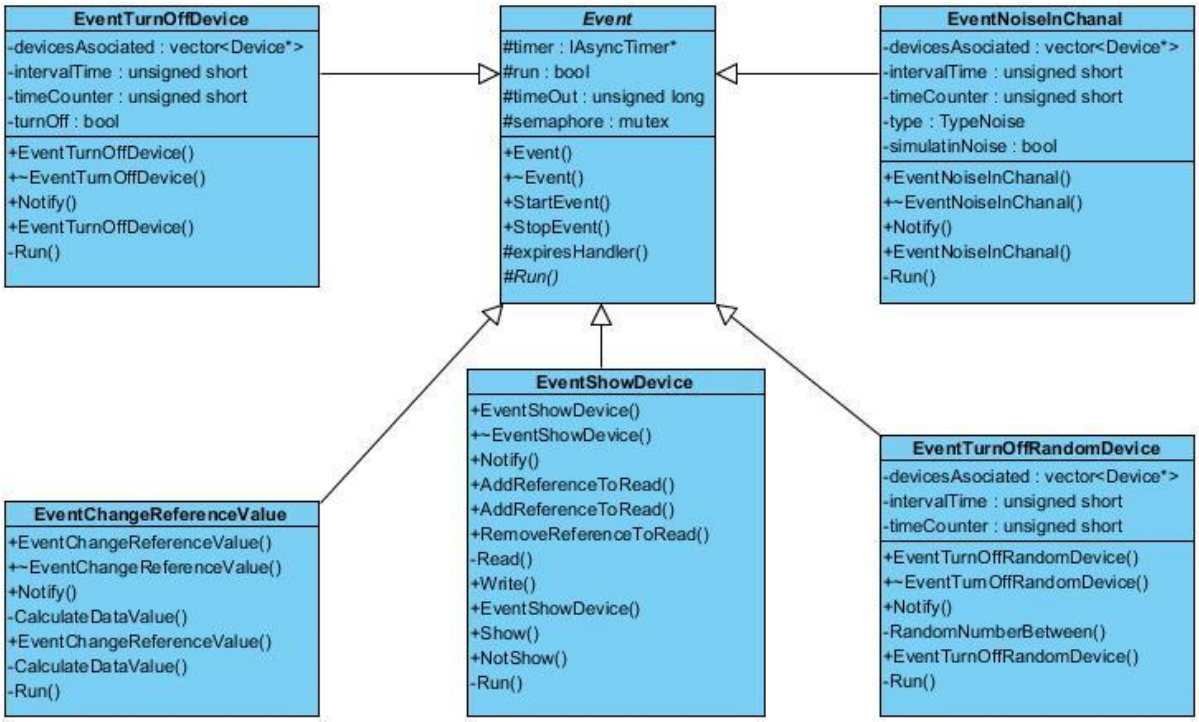

Ilustración 11. Componente gestor de eventos.

3.1.3 Capa transporte.

La capa de transporte se encarga del intercambio de información entre el medio físico y la capa de protocolo. Para ello se utilizó las funcionalidades de la biblioteca TransportProvider. Esta herramienta brinda una serie de interfaces las cuales el programador puede utilizar para acceder a sus funcionalidades. La información es recibida en forma de “callback” a través de métodos específicos proporcionados por la biblioteca; estos métodos pueden o no ser reimplementados por el programador, dependiendo si se desea o no realizar una operación una vez que la biblioteca termine de ejecutar una función determinada y se denominan métodos handler. Para la implementación del simulador se utilizó la interfaz ITransport que brinda la opción de escribir y leer datos de un puerto y los objetos que recibirían esta información se hicieron hijos de ITransporHandler, interfaz que posee los métodos handler relacionados con estas operaciones. De igual forma se utilizó la interfaz IAsiyncTimer para el manejo del tiempo e ITimerHandler que brinda los handler pertinentes. Y el IAceptor e IAceptorHandler para la gestión

de las conexiones que lleguen a un puerto Ethernet. En el siguiente diagrama muestra cómo se relaciona la capa de protocolo con esta capa.

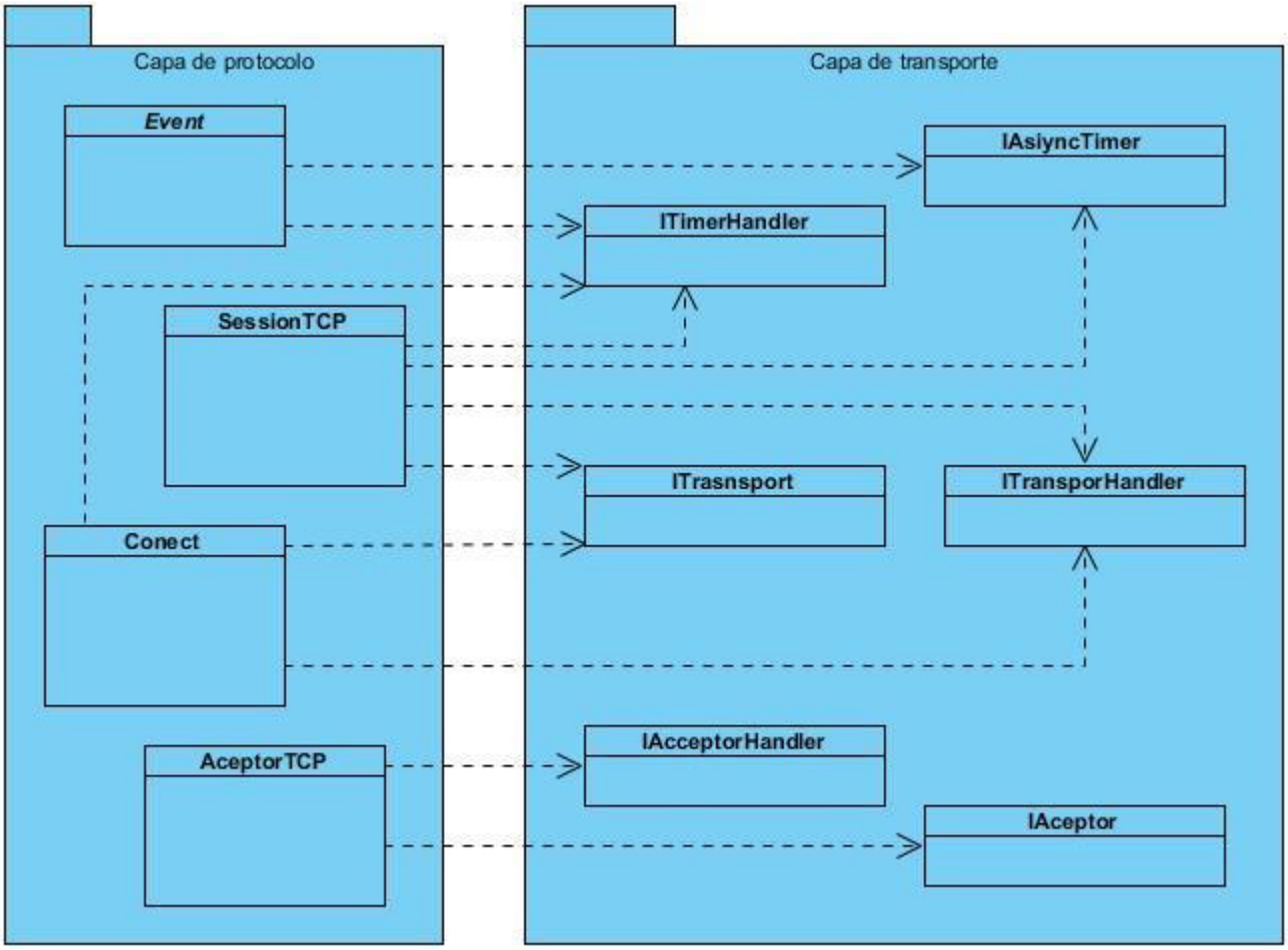

Ilustración 12. Relación entre la capa de transporte y la de protocolo.

4 Capítulo 4 Implementación y pruebas del sistema.

En este capítulo comienza la implementación del sistema. Para describir este proceso se utilizan las Tareas de Ingeniería. Una vez que este proceso culmine se realizan las pruebas de aceptación sobre el producto terminado.

4.1 Fase de Implementación.

Durante esta fase se procede a implementar las HU relacionadas con las tres iteraciones descritas. Este proceso está guiado por el plan de iteraciones y descrito con la utilización de las Tareas de Ingeniería. En la metodología XP se define como Tarea de Ingeniería al conjunto de acciones asociadas a cada HU. Estas permiten organizar el proceso de implementación además de posibilitar que sea conocido el grado de complejidad de cada historia de usuario teniendo en cuenta la cantidad de tareas asociadas a ella. En cada una de las tareas se da a conocer el programador asignado, así como el tiempo necesario para su realización, lo que facilita la estimación del tiempo que abarcará cada historia de usuario en implementarse, de acuerdo con su complejidad. (13)

4.1.1 Iteración 1

HU	**Tareas de ingeniería asociadas.**			
Número	PE	Nombres	Inicio	Fin
1	1	1. Obtener los registros de un dispositivo. (ver Anexo 2 tabla 36).	1-3-2012	7-3-2012
	0.5	2. Mostrar los registros. (ver Anexo 2 tabla 37).	8-3-2012	10-3-2012
2	1	1. Modificar el valor de un registro en memoria. (ver Anexo 2 tabla 38).	12-3-2012	17-3-2012
	0.5	2. Cambiar el valor de un registro. (ver Anexo 2 tabla 39).	19-3-2012	21-3-2012

Tabla 10. Tareas de ingeniería de la iteración 1.

4.1.2 Iteración 2.

HU	Tareas de ingeniería asociadas.			
Número	PE	Nombres	Inicio	Fin
HU 4	0.5	1. Cambiar el mapa de memoria. (ver Anexo 2 tabla 42).	22-3-2012	24-3-2012
	1	2. Cambiar el mapa de memoria de un canal de comunicación. (ver Anexo 2 tabla 43).	26-3-2012	31-3-2012
	1	3. Crear un nuevo mapa de memoria para un canal de comunicación. (ver Anexo 2 tabla 44).	2-4-2012	7-4-2012
HU 5	1	1. Crear un dispositivo a partir de una configuración de memoria guardada. (ver Anexo 2 tabla 45).	9-4-2012	14-4-2012
HU 6	1	1. Guardar la configuración de un canal de comunicación para un fichero. (ver Anexo 2 tabla 46).	16-4-2012	21-4-2012
HU 10	2	1. Crear un canal de comunicación a partir de un tipo de comunicación (TCP, ASCII o RTU). (ver Anexo 2 tabla 53).	23-4-2012	5-5-2012
	1	2. Crear un nuevo canal de comunicación. (ver Anexo 2 tabla 54).	7-5-2012	12-5-2012

Tabla 11. Tareas de ingeniería de la iteración 2.

4.1.3 Iteración 3.

HU	Tareas de ingeniería asociadas.			
Número	PE	Nombres	Inicio	Fin
HU 3	0.5	1. Adicionar un evento. (ver Anexo 2 tabla 40).	14-5-2012	16-5-2012

	1.5	2. Crear un evento que modifique el valor de un registro. (ver Anexo 2 tabla 41).	17-5-2012	26-5-2012
HU 7	0.5	1. Cambiar el estado de encendido-apagado de un dispositivo. (ver Anexo 2 tabla 47).	28-5-2012	30-5-2012
	0.5	2. Adicionar un evento al sistema. (ver Anexo 2 tabla 48).	31-5-2012	2-6-2012
HU 8	0.5	1. Cambiar el estado de encendido-apagado de un dispositivo. (ver Anexo 2 tabla 49).	4-6-2012	6-6-2012
	0.5	2. Adicionar un evento al sistema. (ver Anexo 2 tabla 50).	7-6-2012	9-6-2012
HU 9	0.5	1. Crear ruido en un canal. (ver Anexo 2 tabla 51).	11-6-2012	13-6-2012
	0.5	2. Crear ruido en el canal de un dispositivo. (ver Anexo 2 tabla 52).	14-6-2012	16-6-2012

Tabla 12. Tareas de ingeniería de la iteración 3.

4.2 Diagrama de despliegue.

Los diagramas de despliegue describen la arquitectura física del sistema durante la ejecución y la topología del sistema: la estructura de los elementos de *hardware* y *software* que ejecuta cada uno de ellos. El diagrama de despliegue está constituido por nodos físicos en los cuales se ejecutan los componentes. (16)

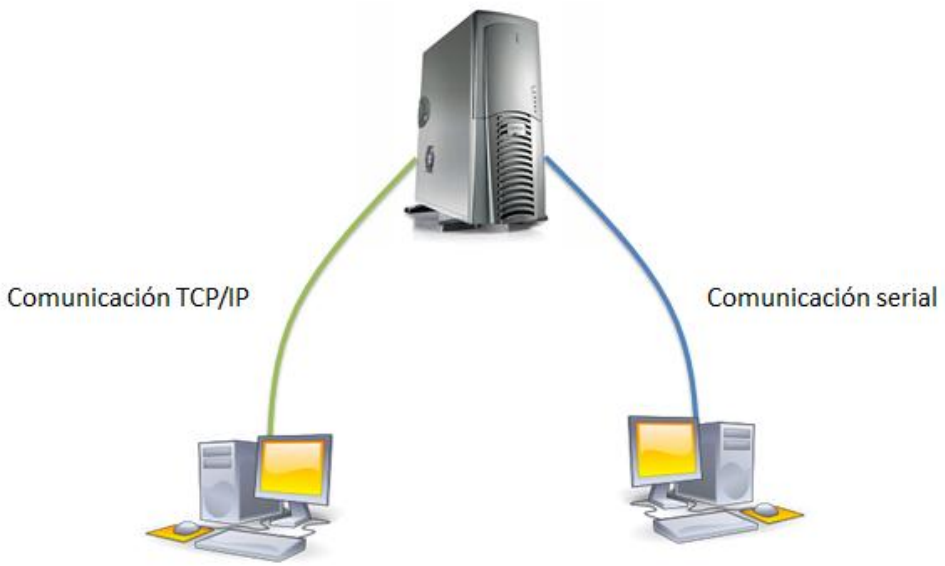

Ilustración 13. Diagrama de despliegue.

4.3 Pruebas.

Las pruebas son procesos que se llevan a cabo con el objetivo de verificar la calidad de un producto, detectando los posibles errores que puede presentar. Existen disímiles tipos de pruebas, y cada una de ellas fue diseñada con el objetivo de evaluar diferentes aspectos de un producto. En XP se definen las pruebas unitarias y las pruebas de aceptación.

Las pruebas unitarias son aquellas que los desarrolladores ejecutan con el objetico de validar que el código sea funcional. Por otra parte, el objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario determinar su aceptación, desde el punto de vista de su funcionalidad y rendimiento. Las pruebas de aceptación son definidas por el usuario del sistema y preparadas por el equipo de desarrollo, aunque la ejecución y aprobación final corresponde al usuario. Una HU sólo se considerará terminada cuando haya pasado correctamente todas las pruebas de aceptación.

4.3.1 Pruebas de aceptación.

Caso de prueba de aceptación.	
Código: HU1_P1	**HU:** 1
Nombre: Mostrar el contenido de varios registros similares y su progreso**.**	
Descripción: Esta prueba tiene como objetivo evaluar el correcto funcionamiento de la opción de monitorear de registros de un dispositivo durante el tiempo.	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo TCP, y le asignará el puerto 10000. • Paso 2: El usuario agregará un dispositivo al canal de comunicación creado. • Paso 3: El usuario activa en el dispositivo la opción de mostrar el contenido de los registros 3001, 3002,3007 de tipo *ShortIntiger*. • Paso 4: El usuario agrega un evento de cambio de valor de un registro utilizando la fórmula matemática “x+1”, con intervalo de tiempo 5 segundos, a cada uno de los registros seleccionados en el Paso 3. • Paso 5: El usuario pone a correr el simulador. • Paso 6: Verificar que el valor inicial de cada registro sea 0 y luego cada 5 el valor de cada uno se incremente en uno. • Paso 7: El usuario espera 15 minutos y detendrá el simulador.	
Resultado esperado: Se observa que el valor de los registros seleccionados y sus cambios en el tiempo, estén acorde a la configuración del sistema, o sea luego de 15 minutos el valor de los registros 3001, 3002 y 3007 tiene que ser 180.	
Evaluación de la prueba: Satisfactoria**.**	

Tabla 13. Prueba de aceptación 1.

Caso de prueba de aceptación.	
Código: HU1_P2	**HU**: 1
Nombre: Mostrar el contenido de registros diferentes.	
Descripción: Esta prueba tiene como objetivo evaluar el correcto funcionamiento de la opción de poder ver el valor de cualquier tipo de registro.	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo RTU, y le asignará el puerto "/dev/ttyUSB0". • Paso 2: El usuario agregará un dispositivo al canal de comunicación creado. • Paso 3: El usuario activa en el dispositivo la opción de mostrar el contenido de los registros 1001 de tipo *Status*, 3001 de tipo *ShortIntiger*, 4101 de tipo *ASCIIString8Char*, 5101 de tipo *LongIntiger*, 6001 de tipo *FlotingPointer*, 9001 de tipo *ASCIStringBuffer* (packet 1) y 14001 de tipo *ASCIIString16Char*, • Paso 4: El usuario escribirá un valor inicial que cada uno de los registros seleccionados en el Paso 3. • Paso 5: El usuario pone a correr el simulador. • Paso 6: Verificar que el valor inicial de cada registro sea igual que el valor inicial entrado. • Paso 7: El usuario espera 15 minutos y detendrá el simulador.	
Resultado esperado: Se observa el valor de los registros seleccionados sean iguales a los valores iniciales entregado por el usuario.	
Evaluación de la prueba: Satisfactoria**.**	

Tabla 14. Prueba de aceptación 2.

Caso de prueba de aceptación.	
Código: HU2_P1	**HU**: 2
Nombre: Modificar el contenido de cualquier tipo de registro.	

Descripción: Esta prueba tiene como objetivo evaluar el correcto funcionamiento de la opción modificar el contenido de cualquier tipo de registro.
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo RTU, y le asignará el puerto "/dev/ttyUSB0". • Paso 2: El usuario agregará un dispositivo al canal de comunicación creado. • Paso 3: El usuario pondrá a correr el simulador. • Paso 4: El usuario ejecutará en una máquina remota una aplicación que se comunique a través del protocolo Modbus Omni vía serial RTU • Paso 5: El usuario mandará a leer de los registros 1001 de tipo *Status*, 3001 y 3002 de tipo *ShortIntiger*, 4201 de tipo *ASCIIString8Char*, 5301 y 5302 de tipo *LongIntiger*, 6001 de tipo *FlotingPointer*, 9002 de tipo *ASCIStringBuffer* (*packet* 3) y 14007 de tipo *ASCIIString16Char*. • Paso 6: Verificar que los valores iniciales leídos sean 0, en todos los casos. • Paso 7: El usuario mandará a escribir en los registros 1001 de tipo *Status*, 3001 y 3002 de tipo *ShortIntiger*, 4201 de tipo *ASCIIString8Char*, 5301 y 5302 de tipo *LongIntiger*, 6001 de tipo *FlotingPointer*, 9002 de tipo *ASCIStringBuffer* (*packet* 3) y 14007 de tipo *ASCIIString16Char*, valores aleatorios. • Paso 8: Volver a ejecutar el Paso 5. • Paso 9: Verificar que los valores leídos sean los escritos en el Paso 7.
Resultado esperado: Se observa el valor de los registros seleccionados sean iguales a los valores iniciales entrados por el usuario.
Evaluación de la prueba: Satisfactoria**.**

Tabla 15. Prueba de aceptación 3.

Caso de prueba de aceptación.

Código: HU2_P2	**HU**: 2
Nombre: Modificar continuamente el contenido de un registro	
Descripción: Esta prueba tiene como objetivo evaluar el correcto funcionamiento de la opción modificar el contenido de un registro reiteradas veces.	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo ASCII, y le asignara el puerto "/dev/ttyUSB0". • Paso 2: El usuario agregará un dispositivo al canal de comunicación creado. • Paso 3: El usuario pondrá a correr el simulador. • Paso 4: El usuario ejecutará en una maquina remota una aplicación que se comunique a través del protocolo Modbus Omni ASCII. • Paso 5: El usuario mandará a leer el registro 1307 de tipo *Status*. • Paso 6: Verificar que el valor iniciales leído sea 0. • Paso 7: El usuario mandará a escribir en el registro 1307 de tipo *Status* un valor aleatorio. • Paso 8: Volver a ejecutar el paso 5. • Paso 9: Verificar que los valores leídos sean los escritos en el Paso 7.	
Resultado esperado: Se observa el valor de los registros seleccionados sean iguales a los valores entrado por el usuario.	
Evaluación de la prueba: Satisfactoria**.**	

Tabla 16. Prueba de aceptación 4.

Caso de prueba de aceptación.	
Código: HU3_P1	**HU**: 3
Nombre: Modificar el contenido de un registro dado un valor admisible.	
Descripción: El objetivo de esta prueba es verificar que el valor de los registros de un dispositivo pueda ser cambiado utilizando una fórmula matemática que devuelva un valor admisible para dicho registro.	

Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo TCP, y le asignará el puerto 10001. • Paso 2: El usuario agregará un dispositivo al canal de comunicación creado. • Paso 3: El usuario activa en el dispositivo la opción de mostrar el contenido de los registros 5888 de tipo *LongIntiger*. • Paso 4: El usuario agrega un evento de cambio de valor de un registro utilizando la fórmula matemática “2*x+10-(x/2+1)”, con intervalo de tiempo 60 segundos, al registro seleccionado en el Paso 3. • Paso 5: El usuario pone a correr el simulador. • Paso 6: Verificar que el valor inicial de cada registro sea 0 y luego cada vez que transcurra un minuto el valor de cada uno se incrementa según la fórmula entrada. • Paso 7: El usuario espera 15 minutos y detendrá el simulador.
Resultado esperado: Se observa que el valor del registro 5888 cambia acorde a la función entrada y el valor final será 30.
Evaluación de la prueba: Satisfactoria**.**

Tabla 17. Prueba de aceptación 5.

Caso de prueba de aceptación.	
Código: HU3_P2	**HU**: 3
Nombre: Modificar el contenido de un registro colocándole un valor demasiado grande.	
Descripción: El objetivo de esta prueba es verificar que el sistema no presente errores si el resultado de la fórmula matemática es un valor mayor que el máximo admisible para dicho registro.	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de	

comunicación de tipo ASCII, y le asignará el puerto “/dev/ttyUSB0”. • Paso 2: El usuario agregará un dispositivo al canal de comunicación creado. • Paso 3: El usuario activa en el dispositivo la opción de mostrar el contenido de los registros 3598 de tipo *ShortIntiger*. • Paso 4: El usuario agrega un evento de cambio de valor de un registro utilizando la fórmula matemática “x+30000”, con intervalo de tiempo 10 segundos, al registro seleccionado en el Paso 3. • Paso 5: El usuario pone a correr el simulador. • Paso 6: Verificar que el valor inicial del registro sea 0 • Paso 7: Verificar que luego 10 segundos el valor del registro sea 3000. • Paso 8: Verificar que luego 20 segundos el valor del registro sea 6000. • Paso 9: Verificar que luego 30 segundos el valor del registro vuelva a ser el inicial. • Paso 10: Detener el simulador.
Resultado esperado: El valor final del registro 3598 deberá ser 0.
Evaluación de la prueba: Satisfactoria**.**

Tabla 18. Prueba de aceptación 6.

Caso de prueba de aceptación.	
Código: HU3_P3	**HU**: 3
Nombre: Modificar el contenido de un registro colocándole un valor demasiado pequeño.	
Descripción: El objetivo de esta prueba es verificar que el sistema no presente errores si el resultado de la fórmula matemática es un valor menor que el mínimo admisible para dicho registro**.**	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo RTU, y le asignará el puerto “/dev/ttyUSB0”. • Paso 2: El usuario agregará un dispositivo al canal de comunicación creado. • Paso 3: El usuario activa en el dispositivo la opción de mostrar el contenido de los registros	

3698 de tipo *ShortIntiger*. • Paso 4: El usuario agrega un evento de cambio de valor de un registro utilizando la fórmula matemática “x-5”, con intervalo de tiempo 5 segundos, al registro seleccionado en el Paso 3. • Paso 5: El usuario ejecuta el simulador. • Paso 6: Luego 60 segundos detener el simulador.
Resultado esperado: El valor final de registro 3698 es 6.
Evaluación de la prueba: Satisfactoria**.**

Tabla 19. Prueba de aceptación 7.

Caso de prueba de aceptación.	
Código: HU456_P1	**HU**: 4, 5, 6
Nombre: Manejo del mapa de registros de los dispositivos de un canal de comunicación.	
Descripción: El objetivo de esta prueba es verificar que los canales de comunicaciones puedan cargar, salvar y cambiar la configuración del mapa de memoria de los dispositivos contenidos en ellos.	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo TCP, y le asignará el puerto 10007. • Paso 2: El usuario agregará un dispositivo al canal de comunicación creado. • Paso 3: El usuario ejecutará en una máquina remota una aplicación que se comunique a través del protocolo Modbus Omni vía TCP. • Paso 4: El usuario manda una solicitud de lectura de la referencia 1001 de tipo *Status*. • Paso 5: Verificar que la trama de respuesta contenga el valor del registro y no sea una trama de error. • Paso 6: El usuario cambiará el mapa de memoria del canal de comunicación, la referencia 1001 debe no debe existir en el nuevo mapa. • Paso 7: Ejecutar el Paso 4.	

• Paso 8: Verificar que la trama de respuesta sea una trama de error de lectura. • Paso 9: El usuario guardará la configuración del canal de comunicación actual. • Paso 10: El usuario creará un nuevo canal de comunicación a partir del archivo de configuración generado en el paso 9. • Paso 11: El usuario mandará a leer la referencia 1001 del nuevo canal de comunicación. • Paso 12: Verificar que la trama de respuesta sea una trama de error de lectura.
Resultado esperado: Se observa si el sistema cambia, carga y salva la configuración del mapa de memoria sin cerrarse.
Evaluación de la prueba: Satisfactoria**.**

Tabla 20. Prueba de aceptación 8.

Caso de prueba de aceptación.	
Código: HU7_P1	**HU**: 7
Nombre: Simular fallas de caída de dispositivos sincrónicas.	
Descripción: El objetivo de esta prueba es verificar que la opción de caída sincrónica de dispositivos funcione correctamente.	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo ASCII, y le asignará el puerto “dev/ttyUSB0”. • Paso 2: El usuario agregará varios dispositivos al canal de comunicación creado. • Paso 3: El usuario seleccionará un subgrupo de dispositivos y creara un evento de caída de dispositivos sincrónica con un intervalo de tiempo de 10 segundos. • Paso 4: Verificar que cada 10 segundos todos los dispositivos asociados al evento cambien su estado. • Paso 5: Luego de 1 minuto detener el simulador.	
Resultado esperado: Se observa si los dispositivos seleccionados cambian su estado de encendido	

apagado y viceversa y su estado final es encendido.
Evaluación de la prueba: Satisfactoria**.**

Tabla 21. Prueba de aceptación 9.

Caso de prueba de aceptación.	
Código: HU8_P1	**HU**: 8
Nombre: Simular fallas de caída de dispositivos aleatorias.	
Descripción: El objetivo de esta prueba es verificar que la opción de caída aleatoria de dispositivos funcione correctamente.	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo ASCII, y le asignará el puerto “dev/ttyUSB0”. • Paso 2: El usuario agregará varios dispositivos al canal de comunicación creado. • Paso 3: El usuario seleccionará un subgrupo de dispositivos y creará un evento de caída de dispositivos aleatoria con un intervalo de tiempo de 10 segundos. • Paso 4: Verificar que cada 10 segundos uno de los dispositivos asociados al evento cambien su estado.	
Resultado esperado: Se observa si uno de los dispositivos seleccionados cambian su estado de encendido-apagado cada vez que vez que transcurran 10 segundos.	
Evaluación de la prueba: Satisfactoria**.**	

Tabla 22. Prueba de aceptación 10.

Caso de prueba de aceptación.	
Código: HU9_P1	**HU**: 9
Nombre: Simular ruido en el canal.	
Descripción: El objetivo de esta prueba es verificar que la opción de simular ruido en el canal funcione	

correctamente.
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará un canal de comunicación de tipo RTU, y le asignará el puerto “dev/ttyUSB0”. • Paso 2: El usuario agregará varios dispositivos al canal de comunicación creado. • Paso 3: El usuario seleccionará un subgrupo de dispositivos y creará un evento de simulación de ruido en el canal con un intervalo de tiempo de 10 segundos. • Paso 4: Verificar que cada 10 segundos todos los dispositivos asociados al evento cambien su estado y envíen tramas correctas o incorrectas según el mismo.
Resultado esperado: Se observa si los dispositivos seleccionados alternan tramas correctas y tramas incorrectas (producto del ruido en el canal), cada vez que se cumpla 10 segundos.
Evaluación de la prueba: Satisfactoria**.**

Tabla 23. Prueba de aceptación 11.

Caso de prueba de aceptación.	
Código: HU10_P1	**HU**: 10
Nombre: Simular redes seriales comunicándose con un mismo maestro.	
Descripción: El objetivo de esta prueba es verificar que la opción de crear diversos canales de comunicaciones, con diversos dispositivos funcione correctamente.	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará varios canales de comunicaciones de tipo RTU, y le asignará el puerto “dev/ttyUSB0”. • Paso 2: El usuario agregará varios dispositivos a cada canal de comunicación creado. • Paso 3: El usuario ejecutará en una computadora remota una aplicación que se comunique utilizando el protocolo Modbus Omni.	

• Paso 4: El usuario manda a leer registros de todos los dispositivos creados en el distinto canal de comunicaciones creados. • Paso 5: Verificar que cada pedido reciba una respuesta correcta.
Resultado esperado: Se observa si la aplicación remota está recibiendo respuestas procedentes de cada uno de los dispositivos creados
Evaluación de la prueba: Satisfactoria**.**

Tabla 24. Prueba de aceptación 12.

Caso de prueba de aceptación.	
Código: HU10_P2	**HU**: 10
Nombre: Simular redes seriales con diversos maestros*.*	
Descripción: El objetivo de esta prueba es verificar que la opción de crear diversos canales de comunicaciones, con diversos dispositivos funcione correctamente si cada uno se comunica con un maestro diferente.	
Condiciones de ejecución: • En la computadora se encuentra instalada la biblioteca TransportProvider.	
Procedimiento: • Paso 1: El usuario mientras el simulador no está corriendo, creará varios canales de comunicación de tipo TCP, y le asignará el puerto puertos diferentes. • Paso 2: El usuario agregara varios dispositivos a cada canal de comunicación creado. • Paso 3: El usuario ejecuta en varias computadoras remota aplicaciones que se comuniquen utilizando el protocolo Modbus Omni. • Paso 4: El usuario manda a leer registros de todos los dispositivos creados en los distintos canales de comunicaciones creados. • Paso 5: Verificar que cada pedido reciba una respuesta correcta.	
Resultado esperado: Se observa si las aplicaciones remotas están recibiendo respuestas procedentes de cada uno de los dispositivos creados.	
Evaluación de la prueba: Satisfactoria**.**	

Tabla 25. Prueba de aceptación 13.

4.3.2 Conclusiones Parciales.

Luego de realizar todas las pruebas al simulador y corroborar que la aplicación se ha comportado según lo previsto; se puede afirmar que se cuenta con un *software* que posee las funcionalidades solicitadas por el cliente y que por tanto el mismo está listo para ser utilizado en la etapa de pruebas del manejador Modbus Omni.

Conclusiones.

Como producto final de esta investigación se obtuvo una aplicación que simula parte del comportamiento de los dispositivos de campo que utilizan el protocolo Modbus Omni. Esta aplicación fue diseñada e implementada utilizando el estilo arquitectónico basado en capas, donde se definieron las siguientes capas: capa de aplicación, capa de protocolo y capa de transporte. La capa de aplicación tiene la responsabilidad de servir como intermediario entre el sistema y el usuario. En la capa de protocolo se implementó la lógica de acceso a los dispositivos esclavos y la semántica de las tramas, según las especificaciones del protocolo, además se adicionaron 4 tipos de eventos que pueden cambiar esta lógica. Por último, en la capa de transporte se utilizó la biblioteca TransportProvider con el objetivo de permitir a la aplicación comunicarse a través de una red. Además, dicha arquitectura es útil para la implementación de aplicaciones donde se ejecute un proceso y se desee crear alteraciones complejas en dicho proceso, sin tener que redefinir la lógica de su funcionamiento.

Dada las características funcionales del Simulador Modbus Omni se puede concluir que este producto puede ser empleado para el proceso de detección de errores del manejador Modbus Omni. Además, esta variante posee la ventaja de ser más flexible que la opción de adquirir *software* de simulación. Esto se debe a que al utilizar aplicaciones ya concebidas, el equipo de prueba tiene que utilizar las funcionalidades que éste brinda y puede darse el caso que una sola aplicación no posea todas las características requeridas. Este problema no afecta a una aplicación desarrollada en la línea, debido a que la misma puede ser moldeada al problema específico que originó su necesidad.

Recomendaciones:

- Dada las similitudes entre el protocolo Modbus y el Modbus Omni, se recomienda a la línea de Adquisición agregar en las clases de la capa de protocolo los métodos necesarios para poder utilizar ambos protocolos.

- La línea de Adquisición podría adicionar a la capa de protocolo nuevos eventos, con el objetivo de ampliar el número de situaciones que la aplicación podría simular.

Bibliografía referenciada.

1. MODBUS ORGANIZATION, I. *Modbus Messaging Implementations Guide* Disponible en: http://www.modbus.org.

2. COMER, D. E. *Redes globales de información con Internet y TCP/IP*. 2005, Disponible en: http://www.docstoc.com/docs/34796937/Redes-Globales-de-Informaci%C3%B3n-con-Internet-y-TCPIP. .

3. ZIMMERANN, H. *OSI Reference Model-The ISO Model of Architecture for Open System Interconnections.* 1980,

4. MONGE, A. L. Instrumentación y control básicos de planta de refrigeración solar térmica en Caseta Integral Compact. nº Disponible en: http://bibing.us.es/proyectos/abreproy/5035/fichero/01_Memoria.pdf. .

5. OMNI FLOW COMPUTERS, I. OMNI 3000/6000 Flow Computer User Manual. 1999, vol. 1, nº Disponible en: http://www.omniflow.com

6. BANKS, J. *Handbook of Simulation.* Editado por: Wiley, E. J. 1998, Disponible en: http://www.4shared.com/office/8qJaY7M3/Jerry_Banks_-_Handbook_of_Simu.html.

7. LAW, A. M. *Simulation Modeling and Analysis.* Editado por: Mcgraw-Hill, E. 2000,

8. TRUJILLO, D. R. *Especificación de la interfaz genérica con el SCADA.* Ciudad de la Habana 2007,

9. BJARNE, S. *El lenguaje de programacion C++*. segunda ed. Addison-Wesley, 1993,

10. NOKIA_CORPORATION. *Sitio Oficial de Qt* Disponible en: http://qt.nokia.com/.

11. VISUAL_PARADIGM_INTERNATIONAL. *Sitio Oficial de Visual Paradigm* [Consultado el: 2012 Disponible en: https://www.visual-paradigm.com/product/vpuml/.

12. MOLPECERES, A. *java Hispano* Disponible en: http://www.javahispano.org/contenidos/es/procesos_de_desarrollo/.

13. BECK, K. *Extreme Programming Explained.* firt ed. 1999, Disponible en: http://www.ittelkom.ac.id/staf/apk/ngajar/RPL%20OOT/papers/RUP%20dll/extreme%20programming%20explained.pdf.

14. *RATIONAL ROSE ENTERPRISE EDITION* Disponible en: http://www.ecured.cu/index.php/Rational_Rose_Enterprise_Edition].

15. JOSUTTIS, N. M. *The C++ Standart Library a tutorial and reference.* 1999,

16. *INTRODUCCIÓN A LA DISCIPLINA DE ANÁLISIS Y DISEÑO*. Universidad de Ciencias Informáticas, Disponible en: http://eva.uci.cu/.

17. MAILEN EDITH ESCOBAR POMPA, L. A. O. A. *Análisis y Diseño de un Nodo Virtual de Procesos.* Universidad de las Ciencias Informáticas, 2008.

18. POZO, A. C. *Módulos de adquisición y análisis para la interacción con dispositivos de campo en un SCADA.* Ciudad de la Habana 2009,

19. CORTÉS, C. R. *Controladores Lógicos Programables.* Santiago de Chile 2011, Disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=Controladores+L%C3%B3gicos+Programables.+Ram%C3%ADrez+Cort%C3%A9s%2C+Christian.+&source=web&cd=2&ved=0CCsQFjAB&url=http%3A%2F%2Finfoplc.net%2Ffiles%2Fdocumentacion%2Fautomatas%2FinfoPLC_net_apunte_plc.pdf&ei=osZXT9ymOsLl0QGeq7nXDw&usg=AFQjCNETqnIIIdzsABBowWyR4hgfDB74IQ&cad=rja.

Glosario de términos.

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

Autómatas programables, los Controladores Lógicos Programables (PLC): Dispositivo electrónico operado digitalmente que utiliza la memoria programable para el almacenamiento interno de instrucciones a fin de implementar funciones específicas, tales como lógicas, secuenciales, tiempo y aritméticas y así controlar varios tipos de máquinas o procesos a través de módulos de entrada / salida analógica o digital. (19)

Bindings: En el campo de la programación, un binding es una adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquel en el que ha sido escrita.

CASE: Las herramientas CASE (Computer Aided Software Engineering, o en español Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de *software* reduciendo el costo de las mismas en término de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del *software* en tareas como el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras, que analizaba la relación existente entre los requisitos de un problema y las necesidades que éstos generaban, el lenguaje en cuestión se denominaba PSL (Problem Statement Language) y la aplicación que ayudaba a buscar las necesidades de los diseñadores PSA (Problem Statement Analyzer). (17)

Dispositivo de campo: Son los elementos físicos que miden, monitorean y, en algunos casos, almacenan los datos de las variables del proceso. Estos dispositivos no se conectan directamente al SCADA.

Framework: En los sistemas orientados a objeto un *framework* es un conjunto de clases que encapsulan diseños abstractos de soluciones a un determinado número de problemas en relación. Los objetivos

principales que persigue un *framework* son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

GNU/Linux: El núcleo Linux se complementa con una serie de aplicaciones desarrolladas por el grupo GNU para conformar el sistema operativo *software* libre GNU/Linux. Linux/GNU es además multiusuario, multitarea, multiprocesador, multiplataforma y multilingüe, nacido en la red de redes Internet.

IBM: International Business Machines es una empresa multinacional estadounidense de tecnología y consultoría. IBM fabrica y comercializa *hardware* y *software* para computadoras, además ofrece servicios de infraestructura, alojamiento de Internet y consultoría en una amplia gama de áreas relacionadas con la informática, desde computadoras centrales hasta nanotecnología.

Interfaz: Zona de contacto o conexión entre dos componentes de *hardware*, entre dos aplicaciones o entre un usuario y una aplicación. Apariencia externa de una aplicación informática.

LAN: siglas de Local Área Network o Red de Área Local.

Manejadores de Dispositivos (Drivers): Estos son módulos independientes en forma de bibliotecas dinámicas, que implementan un protocolo, permiten el intercambio de datos que provienen de disímiles equipos, que pueden ser autómatas, PLC, reguladores autónomos, sensores inteligentes, controladores, etc. (18)

Multiplataforma: Término usado para referirse a los programas, sistemas operativos, lenguajes de programación u otra clase de *software*, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en Windows en un procesador x86, en GNU/Linux en un procesador x86, y en Mac OS X en uno x86.

Protocolos de comunicación: Podemos definir un protocolo como el conjunto de normas que regulan la comunicación entre los distintos componentes de una red informática.

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

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

Tarjetas CRC: Tarjetas Clase, Responsabilidad y Colaboración permiten a los desarrolladores ver el sistema en término de la programación orientado a objetos, ya que cada una de las tarjetas representa una clase en el sistema. En ellas se escriben brevemente las responsabilidades de la clase y una lista de los objetos con los que colabora para llevar a cabo esas responsabilidades. La información recopilada se puede enriquecer utilizando diagramas de clases y de interacción. Lo importante no son las tarjetas o los diagramas, sino tener presente la asignación de responsabilidades. (13)

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

WAN: siglas de Wide Área Network o Red de Área Amplia.

Anexo 1.

Historia de Usuario	
Número: 1	**Usuario:** Línea de Adquisición.
Nombre historia: Mostrar el contenido de los registros de un dispositivo.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Se deberá brindar la opción al usuario de ver el valor y el tipo de todos los registros de un dispositivo en tiempo real.	
Observaciones:	

Tabla 26. Historia de usuario 1.

Historia de Usuario	
Número: 2	**Usuario:** Línea de Adquisición
Nombre historia: Modificar el contenido de un registro.	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Se deberá brindar la opción al usuario de seleccionar un registro y asignarle un valor.	
Observaciones:	

Tabla 27. Historia de usuario 2.

Historia de Usuario	
Número: 3	**Usuario:** Línea de Adquisición
Nombre historia: Modificar el contenido de un registro de tipo entero colocándole un valor obtenido a través de una fórmula matemática.	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Bajo
Puntos estimados: 2	**Iteración asignada:** 3
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Se deberá brindar la opción al usuario de seleccionar un registro y asignarle una fórmula matemática (la fórmula sólo debe contener una variable), un período de tiempo y un valor inicial. A partir de ese momento cada vez que se cumpla dicho período el sistema tomará el valor inicial, lo sustituirá en fórmula matemática, calculará un nuevo valor y asignará el resultado al registro, luego incrementará el valor inicial.	
Observaciones:	

Tabla 28. Historia de usuario 3.

Historia de Usuario	
Número: 4	**Usuario:** Línea de Adquisición
Nombre historia: Cambiar la configuración del mapa de memoria de los dispositivos de un canal de comunicación.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Se deberá brindar la opción al usuario de seleccionar un canal de comunicación y redefinir la cantidad y	

el tipo de registros que los dispositivos contenidos en el poseerán.
Observaciones:

Tabla 29. Historia de usuario 4.

	Historia de Usuario
Número: 5	**Usuario:** Línea de Adquisición
Nombre historia: Cargar la configuración del mapa de memoria de los dispositivos de un canal de comunicación.	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Bajo
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Se deberá brindar la opción al usuario de poder crear un canal de comunicación con la configuración del mapa de memoria existente en un fichero de configuración determinado.	
Observaciones:	

Tabla 30. Historia de usuario 5.

	Historia de Usuario
Número: 6	**Usuario:** Línea de Adquisición
Nombre historia: Salvar la configuración del mapa de memoria de los dispositivos de un canal de comunicación.	
Prioridad en negocio: Media.	**Riesgo en desarrollo:** Medio

Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Se deberá brindar la opción al usuario de poder salvar la configuración del mapa de memoria de los dispositivos de un canal de comunicación en un fichero de configuración determinado.	
Observaciones:	

Tabla 31. Historia de usuario 6.

	Historia de Usuario
Número: 7	**Usuario:** Línea de Adquisición
Nombre historia: Simular fallas de caída de dispositivos sincrónicas.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 3
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Se deberá brindar la opción al usuario de seleccionar un grupo de dispositivos y colocarle un intervalo de tiempo. A partir de ese momento cada vez que se cumpla dicho período de todos tiempo los dispositivos alternaran su estado de encendido y apagado, con todo lo que esto implica.	
Observaciones:	

Tabla 32. Historia de usuario 7.

	Historia de Usuario
Número: 8	**Usuario:** Línea de Adquisición

Nombre historia: Simular fallas de caída de dispositivos aleatorias**.**	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 3
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Se deberá brindar la opción al usuario de seleccionar un grupo de dispositivos y colocarle un intervalo de tiempo. A partir de ese momento cada vez que se cumpla dicho período de tiempo uno de los dispositivos alternarán su estado de encendido y apagado, con todo lo que esto implica	
Observaciones: .	

Tabla 33. Historia de usuario 8.

		Historia de Usuario
Número: 9	**Usuario:** Línea de Adquisición	
Nombre historia: Simular ruido en el canal.		
Prioridad en negocio: Alta		**Riesgo en desarrollo:** Medio
Puntos estimados: 2		**Iteración asignada:** 3
Programador responsable: José Carlos Abraham Ravelo		
Descripción: Se deberá brindar al usuario la opción de seleccionar un grupo de dispositivos, y un intervalo de tiempo. A partir de ese momento cada dispositivo alternará su estado de ruido (si el estado de ruido está activado el dispositivo cambiará, agregará o eliminará valores aleatorios de cada una de las tramas que éste envíe, sino se enviarán tramas correctas).		
Observaciones:		

Tabla 34. Historia de usuario 9.

Historia de Usuario	
Número: 10	**Usuario:** Línea de Adquisición
Nombre historia: Simular redes seriales.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 1	**Iteración asignada:** 2
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Se deberá brindar al usuario la opción de crear canal de comunicaciones que albergaran entre 1 y 247 dispositivos de un mismo tipo, donde cada uno de ellos será independiente y en conjunto conformarán una red virtual.	
Observaciones:	

Tabla 35. Historia de usuario 10.

Anexo 2.

	Tarea ingeniería
Número tarea: 1	**Número historia:** 1
Nombre tarea: Obtener los registros de un dispositivo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Los dispositivos contarán con los métodos necesarios para tener acceso a cualquiera de sus registros.	

Tabla 36. Tarea de ingeniería 1.

	Tarea ingeniería
Número tarea: 2	**Número historia:** 1
Nombre tarea: Mostrar los registros.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: En la interfaz de usuario se mostrará la lista de dispositivos creados ordenados según los canales de comunicación a que pertenezcan. Al dar click izquierdo sobre uno de los dispositivos el sistema mostrará su características básicas (tipo de dispositivo: TCT, ASCII o RTU, su identificador, si está encendido o apagado y si este dispositivo está mandando tramas correctas o incorrectas, o sea si está simulando o no algún tipo de ruido en su canal, estos dos últimos valores serán cambiables) y la opción de mostrar su memoria. Si el usuario da click izquierdo sobre la opción de memoria el sistema le pedirá que seleccione un tipo de registro y un intervalo. Una vez hecho esto el sistema accederá a los métodos pertinentes de dicho dispositivo y mostrará el valor de todos los registros que cumplan con los criterios escogidos.	

Tabla 37. Tarea de ingeniería 2.

Tarea ingeniería

Número tarea: 1	**Número historia:** 2
Nombre tarea: Modificar el valor de un registro en memoria.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: La memoria contará con los métodos necesarios para cambiar el valor de uno o varios de sus registros.	

Tabla 38. Tarea de ingeniería 3.

	Tarea ingeniería
Número tarea: 2	**Número historia:** 2
Nombre tarea: Cambiar el valor de un registro**.**	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Cuando en la interfaz de usuario se muestren los registros de un dispositivo, se brindará la opción de cambiarle su valor actual por uno que el usuario entrará. El sistema legará la tarea al dispositivo seleccionado y éste a su vez accederá a su memoria y cambiará el valor actual del registro seleccionado por el que el usuario entró.	

Tabla 39. Tarea de ingeniería 4.

	Tarea ingeniería
Número tarea: 1	**Número historia:** 3
Nombre tarea: Adicionar un evento.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: El sistema brindará la opción de agregar eventos con sus respectivos intervalos de tiempo que puedan alterar el valor de un registro cuando se cumpla dicho intervalo de tiempo.	

Tabla 40. Tarea de ingeniería 5.

Tarea ingeniería	
Número tarea: 2	**Número historia:** 3
Nombre tarea: Crear un evento que modifique el valor de un registro.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Cuando en la interfaz de usuario se brinde la opción de añadir un evento que altere el valor de un registro, para ello se pedirá al usuario que entre una fórmula matemática correcta y un intervalo de tiempo. El sistema creará un evento a partir de dichos valores.	

Tabla 41. Tarea de ingeniería 6.

Tarea ingeniería	
Número tarea: 1	**Número historia:** 4
Nombre tarea: Cambiar el mapa de memoria.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: La memoria de cada dispositivo constará de los métodos necesarios para poder cambiar la configuración actual que posea.	

Tabla 42. Tarea de ingeniería 7.

Tarea ingeniería	
Número tarea: 2	**Número historia:** 4
Nombre tarea: Cambiar el mapa de memoria de un canal de comunicación**.**	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1

Programador responsable: José Carlos Abraham Ravelo
Descripción: Cada canal de comunicación contará con los métodos necesarios para poder cambiar la configuración del mapa de memoria que posea los dispositivos lógicos asociados a él.

Tabla 43. Tarea de ingeniería 8.

	Tarea ingeniería
Número tarea: 3	**Número historia:** 4
Nombre tarea: Crear un nuevo mapa de memoria para un canal de comunicación.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Cuando en la interfaz de usuario se muestre un canal de comunicación se brindará la opción de cambiar el mapa de memoria de los dispositivos asociados a él. Si el usuario da clic izquierdo sobre esta opción el sistema le exigirá que entre una serie de valores con los que se conformará el nuevo mapa de memoria. Una vez hecho esto el sistema accederá a los métodos pertinentes del canal de comunicación seleccionado, para cambiar la configuración del mapa de memoria de los dispositivos asociados a él.	

Tabla 44. Tarea de ingeniería 9.

	Tarea ingeniería
Número tarea: 1	**Número historia:** 5
Nombre tarea: Crear un dispositivo a partir de una configuración de memoria guardada.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: En la interfaz de usuario se deberá brindar la opción de crear un canal de comunicación. Cuando el usuario da clic izquierdo sobre esta opción se le exigirá entrar la serie de parámetros necesarios para la creación del canal de comunicación. Dentro de estos parámetros se encuentra la opción de crear el canal con la configuración del mapa de memoria por default o uno seleccionado por él, si el usuario escoge	

esta opción el canal de comunicación creado tendrá la configuración del archivo de configuración entrado. Luego se le brindará la opción de añadir un dispositivo a este canal de comunicación, si el realiza esta operación, el dispositivo creado poseerá la configuración asociada al canal de comunicación.

Tabla 45. Tarea de ingeniería 10.

	Tarea ingeniería
Número tarea: 1	**Número historia:** 6
Nombre tarea: Guardar la configuración de un canal de comunicación para un fichero.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: En la interfaz de usuario se deberá brindar la opción de salvar la configuración actual del mapa de memoria de un canal de comunicación. Cuando el usuario da clic izquierdo sobre esta opción se le exigirá al usuario entrar el nombre del nuevo fichero de configuración, y luego el sistema ejecutará los métodos pertinentes para ejecutar dicha acción.	

Tabla 46. Tarea de ingeniería 11.

	Tarea ingeniería
Número tarea: 1	**Número historia:** 7
Nombre tarea: Cambiar el estado de encendido-apagado de un dispositivo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Cada dispositivo contará con los métodos necesarios para cambiar su estado de apagado a encendido y viceversa.	

Tabla 47. Tarea de ingeniería 12.

	Tarea ingeniería
Número tarea: 2	**Número historia:** 7
Nombre tarea: Adicionar un evento al sistema.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: El sistema brindará la opción de agregar eventos con sus respectivos intervalos de tiempo que puedan apagar un grupo de dispositivo, una vez que se cumpla dicho intervalo de tiempo el sistema deberá ejecutar el evento. En la interfaz de usuario se brindará la opción de crear este tipo de eventos.	

Tabla 48. Tarea de ingeniería 13.

	Tarea ingeniería
Número tarea: 1	**Número historia:** 8
Nombre tarea: Cambiar el estado de encendido-apagado de un dispositivo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Cada dispositivo contará con los métodos necesarios para cambiar su estado de apagado a encendido y viceversa.	

Tabla 49. Tarea de ingeniería 14.

	Tarea ingeniería
Número tarea: 2	**Número historia:** 8
Nombre tarea: Adicionar un evento al sistema.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5

Programador responsable: José Carlos Abraham Ravelo
Descripción: El sistema brindará la opción de agregar eventos que cada vez que se cumpla un intervalo de tiempo puedan alternar el estado de encendido-apagar de uno de los dispositivos asociados al él, una vez que se cumpla dicho intervalo de tiempo el sistema deberá ejecutar el evento. En la interfaz de usuario se brindará la opción de crear este tipo de eventos.

Tabla 50. Tarea de ingeniería 15.

	Tarea ingeniería
Número tarea: 1	**Número historia:** 9
Nombre tarea: Crear ruido en un canal.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: Cada dispositivo contará con los métodos necesarios para que a cada trama que salga de él se le agregue, quite o cambie un valor aleatorio.	

Tabla 51. Tarea de ingeniería 16.

	Tarea ingeniería
Número tarea: 2	**Número historia:** 9
Nombre tarea: Crear ruido en el canal de un dispositivo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Programador responsable: José Carlos Abraham Ravelo	
Descripción: El sistema brindará la opción de agregar eventos que cada vez que se cumpla un intervalo de tiempo puedan obligar a los dispositivos asociados a él a enviar tramas incorrectas. En la interfaz de usuario se brindará la opción de crear este tipo de evento.	

Tabla 52. Tarea de ingeniería 17.

	Tarea ingeniería
Número tarea: 1	**Número historia:** 10
Nombre tarea: Crear un canal de comunicación a partir de un tipo de comunicación (TCP, ASCII o RTU)	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Programador responsable: José Carlos Abraham Ravelo	
Descripción: El sistema poseerá los métodos necesarios para dado un tipo de comunicación crear un canal de comunicación.	

Tabla 53. Tarea de ingeniería 18.

	Tarea ingeniería
Número tarea: 2	**Número historia:** 10
Nombre tarea: Crear un nuevo canal de comunicación.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Programador responsable: José Carlos Abraham Ravelo	
Descripción: En la interfaz de usuario se mostrará la opción de crear un nuevo canal de comunicación. El usuario seleccionará un tipo de comunicación y el sistema creará el canal de comunicación según el tipo. A partir de ese momento todos los canales de comunicaciones se crearán de ese tipo.	

Tabla 54. Tarea de ingeniería 19.