UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 5

Centro de Informática Industrial

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

Simulador para el protocolo de comunicación industrial *Ethernet/IP*.

Autores: Reitel Ramos Macías.

Sergio Santos Hernández.

Tutores: Ing. Pedro Alberto Uriarte Rodríguez.

Ing. Yunaime Noda Cid.

La Habana, Cuba.

Junio, 2015.

"Año 57 de la Revolución"

“…creer que las cosas saldrán bien les dará la confianza necesaria para seguir lo que les dice el corazón…”

Steve Jobs.

DECLARACIÓN DE AUTORÍA

Declaración de autoría:

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

Para que así conste se firma la presente a los _____ días del mes de _____ del año _____.

Firma del Autor

_____

Sergio Santos Hernández

Firma del Autor

_____

Reitel Ramos Macías

Firma del Tutor

_____

Ing. Pedro Alberto Uriarte Rodríguez

Firma del Tutor

_____

Ing. Yunaime Noda Cid

Síntesis de los tutores:

- **Ing. Pedro Alberto Uriarte Rodríguez:** graduado de Ingeniero en Ciencias Informáticas en el año 2010 en la Universidad de las Ciencias Informáticas. Posee 6 años de experiencia en el tema.
  Correo Electrónico: *pauriarte@uci.cu*
- **Ing. Yunaime Noda Cid:** graduado de Ingeniero en Ciencias Informáticas, en el año 2010 en la Universidad de las Ciencias Informáticas. Posee 6 años de experiencia en el tema.
  Correo Electrónico: *ynoda@uci.cu*

Agradecimientos

Muchas personas han contribuido, de una u otra forma, a la elaboración de este trabajo. A todas ellas se les hace llegar los más sinceros agradecimientos. Y en especial a:

A mi mamá que sin ella hubiese sido imposible avanzar en la vida hasta llegar a este momento. Mi mamá siempre dispuesta a escuchar, siempre receptiva y presente, brindándome su apoyo incondicional. A mi papa que a pesar de no estar físicamente ha sido mi motor impulsor y mi mayor ejemplo a seguir. A mi novia por su entrega y ayuda incondicional durante los últimos años, sin ella hubiese sido imposible superar tantas metas. A mi tío Felipe por siempre brindarme su apoyo y demostrarme que todos los sueños son posibles si uno lucha por ellos. A mis suegros por acogerme en su hogar como uno más de la familia de la cual ya me siento parte. A mi compañero de tesis por ser como un hermano en estos años y compartir momentos malos y buenos en la carrera. A Lionys, Nora, Lienys y Julio por ser una segunda familia para mi desde mi niñez hasta estos días. A todos los profesores que fueron partícipes de nuestra formación profesional. A nuestros tutores que han sabido con paciencia y sabiduría guiarnos durante el desarrollo de todo el trabajo. A los compañeros, amigos y amigas que me han acompañado estos cinco años de la carrera, en especial a los de mi apartamento y grupo. A todos muchas gracias y estamos eternamente agradecidos. A la revolución que nos dio la oportunidad a todos de tener educación.

Reitel.

Igualmente quiero dar las gracias a todas esas personas que fueron y son muy importante tanto en la realización de este trabajo, como en mi vida.

Primeramente quiero agradecer a mis padres que a pesar de no estar físicamente siempre sus presencias se siente entre nosotros, sin ellos no estuviera hoy aquí.

A mi abuela quien fue la persona más constante y capaz que haya conocido.

A mis hermanos José, Ramonita, Carlos y Vismar por sus consejos y apoyo. Por ser además mis mejores amigos.

A mi abuelo Pancho y mi tío Yamil, por ser siempre constantes e incondicionales. Por apoyarme y guiarme siempre en cada camino que tomé en mi vida.

A mi novia por todo su amor y dedicación y a su familia por acógeme como uno más de los suyos.

AGRADECIMIENTOS

A Carlos y Xiomara por ser como unos padres para mí, por tener en ellos mi segunda familia. Por demostrarme que la familia no siempre está en la sangre.

A Reitel, la persona que desde el primer día en la universidad se hizo mi amigo, ya hoy es como mi hermano, juntos vencimos grandes y difíciles retos, hoy nos graduamos junto. A su novia y a su mamá.

A mis hermanitos Yader y Ale, por enseñarme que los que son verdaderos siempre estarán en tu vida.

A esas personas que siempre me brindaron su apoyo y en varias ocasiones pusieron mi trabajo antes que el suyo. A Dismey, Zahiro, las Yadiras del lab, Annierys, José y Pupi.

A Liuver (mi chama) por ser mi compa de cuarto estos cinco años, por ser como un hijo para mí.

A Orlando por su apoyo y amistad.

A los tutores, por permitirnos dar ese último paso en nuestra carrera.

A todos los profesores que nos brindaros sus conocimientos y apoyo para ir forjándonos personal y profesionalmente.

Al piquete de Ale, con los cuales la vida se vive mejor, con los que te basta conocer a uno para ser amigo de todo.

A toda la familia y amistades en general.

Sergio.

Resumen

En el presente trabajo se describe el proceso de desarrollo de un simulador para el protocolo de comunicación industrial *Ethernet/IP*. En la línea de Adquisición de Datos del Centro de Informática Industrial (CEDIN), donde se desarrollan manejadores (drivers en inglés) que implementan protocolos industriales, no se cuenta con Controladores Lógicos Programables (PLC, por sus siglas en inglés) o simuladores que implementen estos protocolos; estos son los que permiten realizar las pruebas de comunicación que garantizan el funcionamiento y liberación de los drivers. Ante esta imposibilidad fue necesario desarrollar un simulador que validara el correcto funcionamiento de las nuevas versiones del driver para el protocolo *Ethernet/IP*. Con este objetivo se realizó un estudio del estado del arte de la simulación de dispositivos industriales, así como, los principales conceptos utilizados en el proceso de adquisición y envío de datos a través del protocolo de comunicación industrial *Ethernet/IP*. Se representó la arquitectura definida para la aplicación, así como, los requisitos funcionales y no funcionales que rigieron el proceso de desarrollo de la misma, concluyendo con un proceso de pruebas que validó su funcionamiento. Como resultado se obtuvo un simulador para la comunicación mediante el protocolo *Ethernet/IP*, que permite generar y visualizar datos en diferentes formatos y activar situaciones excepcionales que pueden ocurrir en las comunicaciones.

Palabras claves: simulador, manejador, protocolo *Ethernet/IP*, PLC, SCADA.

Índice

Índice de tablas

Índice de figuras

Introducción

Al pasar de los años la humanidad ha sido testigo de un vertiginoso desarrollo tecnológico, que ha introducido cambios significativos en todas las esferas económicas y sociales. En el ámbito de la industria, ha provocado que los procesos industriales sean cada vez más complejos y difíciles de controlar de forma manual. Puesto que estos requieren la realización de una gran cantidad de operaciones que deben ser supervisadas a tiempo completo. Es por ello que el hombre ha creado sistemas de *software* capaces de controlar dichos procesos de forma automática. A partir de esto se crean los sistemas de supervisión, control y adquisición de datos (SCADA, por sus siglas en inglés). (1)

SCADA es un *software* especialmente diseñado para supervisar y controlar a distancia la actividad tecnológica en una instalación de cualquier tipo (Ej. Planta Biotecnológica, Central Termoeléctrica, Instalación Hotelera, etc.), por medio de la comunicación con Controladores Lógicos Programables (PLC, por sus siglas en inglés) u otros dispositivos de campo. (2)

En Cuba la industria del *software* ha aumentado su producción considerablemente, con el objetivo de satisfacer las necesidades de informatización de la sociedad cubana, alcanzando un nivel de competitividad acorde a los estándares internacionales y logrando potenciar las exportaciones de *software*. La UCI juega un importante papel en la producción nacional de *software*. Dicha universidad cuenta con varios centros productivos distribuidos entre las diferentes facultades que la conforman, entre los cuales se encuentra el Centro de Informática Industrial (CEDIN) perteneciente a la facultad 5. Este centro tiene como misión desarrollar productos y servicios informáticos de automatización industrial, con un alto valor agregado y que cumplan las necesidades y expectativas de los clientes, potenciando la formación especializada y la investigación.

El centro está conformado por varias líneas de desarrollo, incluyéndose entre estas la de Adquisición de Datos. La misma está distribuida por los módulos de recolección, procesamiento y manejadores de dispositivos (drivers, en inglés). Los manejadores, a través de un lenguaje de comunicación específico, conocido como protocolo de comunicación, contribuyen a que el SCADA intercambie datos con los dispositivos de campo encargados de monitorear los procesos que se desean controlar (3). En el centro se desarrollan varios manejadores de dispositivos entre los que se encuentra el manejador que implementa el protocolo de comunicación industrial *Ethernet/IP*, este ayuda a establecer la comunicación entre el SCADA y los dispositivos de campo, en este caso, PLC que utilizan este protocolo.

En el CEDIN el ciclo de desarrollo y de pruebas internas de las nuevas versiones del manejador que se comunica a través del protocolo industrial *Ethernet/IP* se ha visto afectado, debido a la carencia en la universidad de dispositivos PLC que utilicen este protocolo o la ausencia de herramientas que emulan su funcionamiento. Esta situación impide verificar que estas nuevas versiones del manejador cumplan con las especificaciones técnicas implementadas en el manejador del SCADA GALBA Miranda R2. Además que tengan un comportamiento correcto en la comunicación con los dispositivos y ante situaciones excepcionales que puedan presentarse en las comunicaciones, tales como: cortes y lentitud en las transmisiones y ruido en los datos recibidos o enviados. La no existencia de estos dispositivos y herramientas en el centro provoca que el desarrollo deba realizarse sin posibilidades de ejecutar el manejador para visualizar que los datos enviados sean correctos. Todo esto dificulta en gran medida comprobar que el envío de estos datos se corresponda con lo esperado por el dispositivo. Asimismo cuando las respuestas son recibidas verificar que se realicen las secuencias establecidas para concluir la recepción de información de forma correcta.

Por otra parte, el centro no cuenta con un laboratorio de automática donde pueda ser desplegado el SCADA, con el fin de simular el comportamiento del mismo en un entorno industrial. Esto impide que se pueda probar todo el sistema en un ambiente lo más cercano posible a la realidad, y a su vez adiestrar a todo el personal en el manejo del SCADA mediante un entorno simulado.

Debido a lo antes planteado surge el siguiente **problema de la investigación:** ¿Cómo comprobar que las nuevas versiones del manejador *Ethernet/IP* del SCADA GALBA Miranda R2 cumplan con las especificaciones técnicas que establece el protocolo y tengan un comportamiento correcto ante situaciones excepcionales en las comunicaciones?

Para solucionar el problema antes mencionado se define como **objetivo general**: desarrollar un simulador para el protocolo de comunicación industrial *Ethernet/IP* que permita probar nuevas versiones del manejador para el SCADA GALBA Miranda R2.

El problema planteado define el siguiente **objeto de estudio:** simuladores para protocolos de comunicación industrial.

Siendo el **Campo de acción**: la simulación de dispositivos de campo mediante el protocolo de comunicación industrial *Ethernet/IP*.

Para obtener como **posibles resultados:**

- Un simulador para las comunicaciones a través del protocolo industrial *Ethernet/IP* que cumpla con las especificaciones técnicas del protocolo implementadas en la versión del manejador para el SCADA GALBA Miranda R2.
- Generar y visualizar datos en diferentes formatos y activar situaciones excepcionales que pueden ocurrir en las comunicaciones, tales como: cortes y lentitud en las transmisiones y ruido en los datos recibidos o enviados.

En la investigación se utilizaron los siguientes métodos de investigación científica.

Métodos teóricos

- **Histórico-Lógico:** utilizado con el objetivo de analizar los simuladores de dispositivos desarrollados en el CEDIN, para identificar las funcionalidades y principales características comunes a la presente investigación.
- **Analítico-Sintético:** empleado para identificar los principales fundamentos teóricos relacionados con controladores lógicos programables, protocolos de comunicación industrial y manejadores de dispositivos.
- **Modelación:** utilizado en la creación y elaboración de los diagramas que modelan la estructura, el diseño y las relaciones internas de la solución.

Método empírico

- **Experimental:** se utiliza para llevar a cabo la ejecución de las pruebas, creando las condiciones necesarias que permiten validar el correcto funcionamiento de la herramienta desarrollada en la presente investigación.

Para dar cumplimiento al objetivo general se definen **las siguientes tareas de investigación:**

- Elaboración del marco teórico de la investigación a partir del estudio del estado del arte sobre simuladores de dispositivos industriales, manejadores de dispositivos y protocolos de comunicación industrial, para adquirir todo el conocimiento necesario para efectuar el desarrollo de la solución.
- Selección de la metodología, tecnologías y herramientas a utilizar para guiar y llevar a cabo el proceso de desarrollo de la investigación.
- Definición de requisitos funcionales y no funcionales del simulador.

- Estudio de las especificaciones técnicas del protocolo de comunicación industrial *Ethernet/IP* para implementar la capa de mensaje del simulador.
- Estudio de la arquitectura, funcionamiento y especificaciones técnicas, del manejador *Ethernet/IP* para determinar las respuestas del simulador ante las solicitudes provenientes del manejador.
- Diseño e implementación de un simulador para el protocolo de comunicación industrial *Ethernet/IP*.
- Ejecución de pruebas internas al simulador para verificar el cumplimiento de los requisitos y su correcto funcionamiento.

El presente documento está estructurado de la siguiente manera.
En el capítulo 1 se muestra la elaboración del marco teórico de la investigación. Este se desarrolla mediante la definición de conceptos y características fundamentales a cerca de la simulación de dispositivos y la comunicación a través del protocolo de comunicación industrial *Ethernet/IP.* Además se definen las tecnologías y herramientas que se utilizarán en el diseño y posterior implementación de la solución propuesta.

El capítulo 2 se centra fundamentalmente en el análisis y diseño de la aplicación. Basando sus principales objetivos en presentar los resultados de la fase de ejecución de la solución propuesta. Se obtienen las Historias de Usuarios (HU) y requisitos no funcionales que el sistema debe cumplir para satisfacer las necesidades del cliente, se define además la arquitectura del sistema y los patrones de diseño que se utilizan.

El capítulo 3 describe la fase de implementación del sistema, la cual sigue un orden iterativo e incremental, según la metodología propuesta. Se definen además las tareas de ingeniería, así como, la fase de pruebas a la cual se somete el producto para validar su correcto funcionamiento. Permitiendo de esta manera comprobar que el mismo cumple con todas las opciones requeridas por el cliente, por lo que dicho producto puede ser liberado.

Capítulo 1. Fundamentación teórica

1.1. Introducción

En la actualidad, el uso de la simulación es muy común en diferentes áreas debido a las ventajas que proporciona. En el ámbito industrial, es de vital importancia, ya que permite prevenir eventos indeseables mediante su apoyo a la toma de decisiones y corrección de errores.

En este capítulo se describen características, conceptos y definiciones de los elementos relacionados con la simulación y la transferencia de datos utilizando el protocolo de comunicación industrial *Ethernet/IP*. Así como, el análisis de simuladores de dispositivos existentes que son afines con la investigación. Además, se mencionan las herramientas y tecnologías utilizadas para el desarrollo de la solución propuesta.

1.2. Simulación por ordenador

La simulación es una técnica para conducir experimentos en una computadora digital. Comprende ciertos tipos de relaciones matemáticas y lógicas, necesarias para describir el comportamiento y la estructura de sistemas complejos del mundo real. Otros estudios indican que simulación es el proceso de diseñar y desarrollar un modelo computarizado de un sistema o proceso. Dicho modelo permite conducir experimentos con el propósito de entender el comportamiento del sistema o evaluar varias estrategias con las cuales se puede operar el sistema. (4)

1.2.1. Etapas para realizar un estudio de simulación

Para llevar a cabo un estudio de simulación, se deben tener en cuenta una serie de pasos, los cuales se describen a continuación. (4)

1. Definición del sistema.
2. Formulación del modelo.
3. Colección de datos.
4. Implementación del modelo.
5. Validación.
6. Experimentación.

1.2.2. ¿Cuándo simular?

La simulación se usa generalmente cuando desarrollar un modelo real es muy difícil, costoso o quizás imposible. 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. (3)

1.2.3. Ventajas de la simulación

La simulación es una técnica universalmente aceptada por diversas razones. (5)

- Es un proceso relativamente eficiente y flexible.
- Puede ser usada para analizar y sintetizar una compleja y extensa situación real.
- Los modelos de simulación se estructuran para resolver generalmente problemas trascendentes.
- No interfiere en sistemas del mundo real.
- Permite estudiar los efectos de componentes individuales o variables para determinar las más importantes.
- Permite la inclusión en complicaciones del mundo real.

1.2.4. Desventajas de la simulación

- Un buen modelo de simulación puede resultar bastante costoso, debido a que el proceso es bastante largo y complicado a la hora de desarrollarlo.
- El modelo de simulación no produce respuestas por sí mismo.
- Cada modelo de simulación es único. Las soluciones e inferencias no son usualmente transferibles a otros problemas.

1.2.5. Simulador de dispositivos

Los simuladores son herramientas desarrolladas con el objetivo de simular determinado proceso real mediante la construcción de modelos virtuales, con la posibilidad de interactuar con elementos externos. La mayoría son sistemas informáticos capaces de manejar un gran número de variables y tienen gran importancia en las predicciones, el entrenamiento y la educación. Su importancia está dada por la capacidad de representar procesos reales, reducir costos y en ocasiones prevenir pérdidas materiales y de vidas

humanas. Los simuladores de dispositivos son aquellos que su modelo representa las características y comportamientos en distintos escenarios, de un dispositivo real. Se pueden clasificar como simuladores contra-producto puesto que son muy ventajosos para asegurar una correcta funcionalidad de un determinado producto en fase de diseño, y sirven de ayuda durante las pruebas del mismo. El simulador actúa, frente al producto a verificar, figurando el entorno real en el que se va a encontrar, creando incluso situaciones de “máxima carga” que garanticen su correcta respuesta en casos muy desfavorables. (6)

1.3. Controladores Lógicos Programables (PLC)

En la presente investigación se realiza un estudio de los dispositivos que se van a simular. Para un mayor entendimiento se muestran a continuación algunas de sus principales características y funcionalidades. (7)

- Un PLC es una máquina electrónica programable, diseñada para ser utilizada en un entorno industrial. Utiliza una memoria para el almacenamiento interno de instrucciones orientadas al usuario, para implantar soluciones específicas tales como funciones lógicas, secuenciales, temporizaciones, recuentos y funciones aritméticas, con el fin de controlar mediante entradas y salidas, digitales y analógicas diversos tipos de máquinas o procesos.
- Posee un procesador de comunicaciones provisto de uno o más puertos de interfaz. Estas interfaces permiten al controlador la comunicación con los dispositivos de programación, computadoras de monitoreo, herramientas de mano para el diagnóstico y otros dispositivos maestros.
- Un PLC está equipado con un número determinado de entradas y salidas que lo conectan a los sensores y los actuadores. El programa memorizado en el equipo de control se compone de instrucciones que activan o desactivan las respectivas entradas y salidas. Se necesitan direcciones para distinguir cada entrada o salida en las instrucciones.

Fig. 1. Controlador Lógico Programable.

1.3.1. PLC *ControlLogix*

El *ControlLogix* es un tipo de PLC o Autómata Programable (AP) creado por *Allen Bradley*[1] de *Rockwell Automation*[2]. Ofrecen la memoria, velocidad y capacidad de procesamiento para cumplir con las demandas de aplicaciones de plantas básicas, así como, procesos de alta demanda y aplicaciones de movimiento.

Para aplicaciones de procesos, estos controladores: (8)

- Ofrecen una capacidad mejorada, lo que le permite ejecutar más estrategias de control en cada tarea.
- Mejoran significativamente la cantidad de información que se puede intercambiar entre las capas de control y de supervisión.
- Ofrecen una mejora importante en el rendimiento de las aplicaciones de control de redundancia.

A continuación, en la figura 2, se muestra el esquema de configuración modular de los dispositivos *ControlLogix*.

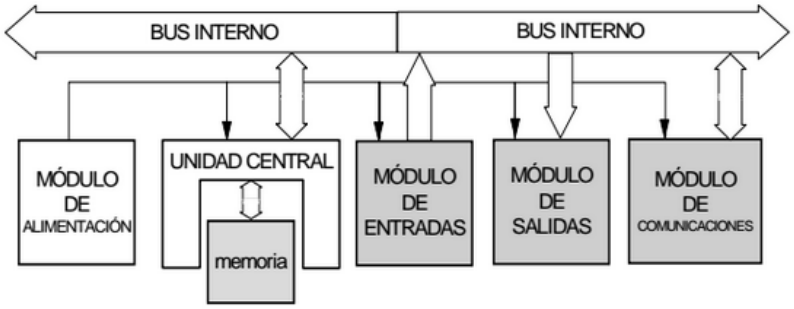


Fig. 2. Constitución Modular de un PLC *ControlLogix*.

El AP está constituido por un conjunto de bloques funcionales, organizados de forma modular y conectados a un bus interno de comunicaciones. Cada bloque está formado por un determinado módulo, diseñado para desarrollar a su vez tareas específicas, como son: gestión de entradas y salidas, gestión de la memoria, unidad de control de procesos, comunicaciones, etc. Esta organización modular permite una gran flexibilidad de configuración para las necesidades de control de cada proceso industrial, que pueden llegar a ser muy dispares y persigue un diagnóstico y mantenimiento sencillo del propio PLC. (9)

[1] Allen Bradley: es la marca de una línea de equipos de automatización de fábricas, creada por la compañía Rockwell Automation.

[2] Rockwell Automation: es un proveedor global de soluciones de automatización e información industrial.

De los módulos antes expuestos, solo intervendrá en el desarrollo de la solución propuesta el Módulo de Comunicaciones. Este le brinda la posibilidad al PLC de comunicarse con otros dispositivos, como son: los equipos de programación, ordenadores personales o con otros autómatas, con el fin de realizar controles más complejos.

1.4. Modelo maestro/esclavo

El modelo maestro/esclavo es una tecnología que proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios de cómputo o cualquier otro recurso de la organización y/o grupo de trabajo, en múltiples plataformas. El modelo soporta un medio ambiente distribuido, en el cual los requerimientos de servicio hechos por estaciones de trabajos inteligentes o maestros, resultan realizado por otros computadores llamados esclavos. El maestro es el que inicia un requerimiento de servicio. Dicho requerimiento 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. Los esclavos pueden estar conectados a los maestros para proveerlos de múltiples servicios, tales como: impresión, acceso a bases de datos, fax, procesamiento de imágenes, etc. (3)

En el entorno de la aplicación, el maestro está representado por el manejador, mientras que el esclavo hace referencia al simulador. El maestro realiza encuestas al esclavo, mediante el envío de paquetes a través de un canal de comunicación. El esclavo le responde uno o varios paquetes, en función de los datos que necesite transmitir a partir de las solicitudes hechas por el maestro.

Fig. 3. Modelo maestro/esclavo.

1.5. Protocolo de comunicación

Constituye un conjunto de normas estándar que especifican el método para enviar y recibir datos entre varios ordenadores. Es una convención que controla o permite la conexión, comunicación y transferencia de datos entre dos puntos finales. (10)

En su forma más simple, un protocolo puede ser definido como las reglas que dominan la sintaxis, semántica y sincronización de la comunicación. Los protocolos pueden ser implementados por *hardware*, *software* o una combinación de ambos. (10)

1.6. Protocolo de comunicación TCP/IP

TCP/IP (Protocolo de Control de Trasmisiones/ Protocolo de Internet) es un grupo de protocolos diseñados para la comunicación, suministrando, a su vez, servicios de red como: registro de entrada remoto, transferencia remota de archivos, correo electrónico, etc. Un protocolo de comunicación debe manejar los errores en la trasmisión, administrar el enrutamiento y entregar los datos, así como, controlar la trasmisión real mediante el uso de señales de estado predeterminadas. TPC/IP se ocupa de todo esto.

TCP/IP está compuesto por dos de las partes del *software* de internet particularmente importantes e innovadoras. El *software* de IP proporciona la comunicación básica, en tanto que el *software* TCP suministra las facilidades adicionales que necesitan las aplicaciones. Aunque estos protocolos se pueden utilizar por separados, se diseñaron al mismo tiempo para trabajar como parte de un sistema unificado y también para cooperar entre sí y complementarse. Una computadora conectada con internet necesita tanto del *software* IP como del TCP. IP proporciona una forma para transferir un paquete desde su origen hasta su destino, pero no soluciona problemas como la pérdida de datagramas o fallas en la entrega. TCP resuelve problemas que IP no puede. Juntos proporcionan una forma confiable de enviar datos a través de la red. (11)

1.7. Protocolo de comunicación industrial *Ethernet/IP*

Ethernet/IP es un protocolo de red en niveles para aplicaciones de automatización industrial. Es una red abierta que utiliza tecnologías como: (12)

- El estándar de vínculo físico y de datos IEEE[3] 802.3.

[3] IEEE: *Institute of Electrical and Electronics Engineers* (Instituto de Ingeniería Eléctrica y Electrónica).

- El conjunto de protocolos TCP/IP, estándar del sector para *Ethernet*.
- El Protocolo de Control e Información (CIP, por sus siglas en inglés), para la transmisión de mensajes de entrada/salida en tiempo real e información/transmisión de mensajes entre dispositivos similares.

Basado en los protocolos estándar TCP/IP, utiliza los ya bastante conocidos *hardware* y *software Ethernet* para establecer un nivel de protocolo para configurar, acceder y controlar dispositivos de automatización industrial. *Ethernet/IP* clasifica los nodos de acuerdo a los tipos de dispositivos preestablecidos, con sus actuaciones específicas. Brinda soporte al protocolo CIP utilizado en *DeviceNet*[4] y *ControlNet*[5]. Apoyado en esos protocolos, *Ethernet/IP* ofrece un sistema integrado completo, enterizo, desde la planta industrial hasta la red central de la empresa. (13)

La utilización del protocolo CIP le permite a *Ethernet/IP* organizar los mecanismos en red como una colección de objetos (o elementos) y define los accesos, atribuciones y extensiones con los cuales se puede acceder a una gama muy vasta de mecanismos mediante la utilización de un protocolo en común. *Ethernet/IP* está basado en un estándar ampliamente conocido y probado. (13)

1.7.1. Encapsulación del protocolo *Ethernet/IP*

Todos los mensajes CIP son enviados en el campo de datos de un paquete que define *Ethernet* para enviar cualquier dato a otro punto de la red. (14)

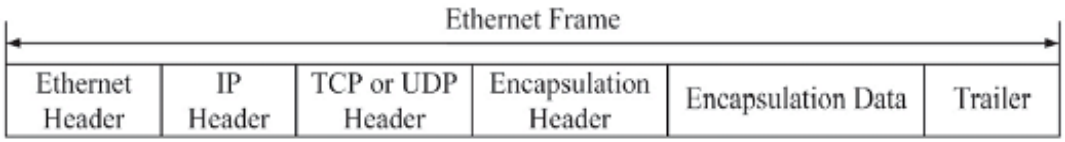


Fig. 4. Muestra la relación entre *Ethernet/IP* y el paquete de *Ethernet*.

El protocolo de encapsulación define un puerto TCP, que debe ser soportado por todos los dispositivos *Ethernet/IP*. Se debe aceptar solo una conexión TCP por el puerto 0xAF12 (44818).

[4] DeviceNet: es una red digital, multi-punto para conexión entre sensores, actuadores y sistemas de automatización industrial en general.

[5] ControlNet: es un protocolo de red abierto para aplicaciones de automatismos industriales.

De igual forma el protocolo de encapsulación define un puerto UDP[6] reservado que debe ser soportado por todos los dispositivos *Ethernet/IP*, los cuales deben aceptar paquetes UDP por el puerto UDP 0xAF12. Como este, a diferencia de TCP, no tiene la posibilidad de reordenar los paquetes, siempre que se use para enviar mensajes encapsulados, el mensaje completo debe contenerse en un paquete simple.

Todos los mensajes encapsulados, enviados vía TCP, o vía UDP al puerto 0xAF12, deben estar compuestos por una cabecera de longitud fija de 24 *bytes* seguida de una porción opcional de datos. La longitud total del mensaje encapsulado, incluyendo la cabecera, está limitada a 65535 *bytes*.
Su estructura es la siguiente:

[6] UDP (*Protocolo de datagrama de usuario*), está orientado a conexión de la capa de transporte del modelo TCP/IP.

Tabla 1. Estructura de un mensaje encapsulado.

Estructura	Nombre del campo	Tipo de dato	Valor del campo
Cabecera del mensaje encapsulado.	*Command*	*UNIT*	Comando encapsulado.
	length	*UNIT*	Longitud, en *byte*, de la porción de datos del mensaje, o sea el número de *bytes* que sigue a la cabecera.
	Session handle	*UDINT*	Identificador de la sesión. (Dependiente de la aplicación).
	Status	*UDINT*	Código de estado.
	Sender Contex	*ARRAY of byte*	Arreglo de 8 *bytes* que contiene información pertinente solo al emisor del mensaje.
	Options	*UDINT*	Bandera de opciones.
Datos específicos del comando.	*Encapsulat data*	*ARRAY of* 0 *to* 65511 *USINT*.	Parte de datos del mensaje encapsulado. Requerida solo en ciertos comandos.

Los campos enteros de más de un *byte* deben ser trasmitidos en el formato *big-endian*[7]. Aunque la cabecera no contiene información explícita que permita distinguir entre una solicitud y una respuesta, el carácter del mensaje puede ser determinado:

1. Implícitamente, por el comando y el contexto en el cual el comando es generado.

[7] ***big-endian***: consiste en representar los bytes en el orden "natural": así el valor hexadecimal 0x4A3B2C1D se codificaría en memoria en la secuencia {4A, 3B, 2C, 1D}. En un sistema de *big endian*, el valor más significativo de la secuencia se almacena en la dirección de almacenamiento más bajo (es decir, primero).

2. Explícitamente por el contenido de la parte de datos del mensaje.

1.7.2. Formato de direcciones de variables de los dispositivos *Ethernet/IP*

Los *tags* o nombres simbólicos por los cuales se accede a la información son identificadores con no más de 40 caracteres de longitud. De acuerdo a su alcance se clasifican en *tags* del controlador (o globales) a los cuales se pueden acceder directamente y *tags* del programa (o locales) a los cuales no se puede acceder desde un dispositivo externo. Cada uno de estos tiene un tipo de dato que define la organización interna del dato y las posibles operaciones sobre el mismo. Se soportan tipos atómicos tales como *bit*, *byte*, palabras de 16 *bits*, de 32 *bits* y tipos estructurados. (15)

Las estructuras son agrupaciones de diferentes tipos de datos que funcionan como una unidad simple y sirven a un propósito específico. A continuación se muestra un ejemplo de la sintaxis de una estructura:

Motor [4].eje.temperatura

Pozo.muestras [7]

Lo cual debe interpretarse de la siguiente manera; en el primer caso tenemos un *tag* Motor de tipo arreglo unidimensional. El tipo de cada elemento del arreglo es una estructura que tiene un miembro cuyo identificador es eje, que a su vez es de tipo estructura también y tiene un miembro llamado temperatura. Por tanto Motor [4].eje.temperatura se refiere al miembro temperatura de la estructura referida por el miembro eje de la estructura Motor [4].
En el segundo caso el arreglo muestras es un miembro de la estructura referenciada por el *tag* Pozo (15).

1.8. Análisis de simuladores de dispositivos existentes

Durante la presente investigación se analizaron varias herramientas de simulación existentes en la línea de Adquisición de Datos del CEDIN. Para determinar si cumplen con las condiciones necesarias y reúnen los requisitos para realizar las pruebas a las nuevas versiones del manejador *Ethernet/IP*. A continuación se describen estas herramientas y por qué no se utilizaron en la solución del problema.

***Modbus Omni*:** esta aplicación funciona como un esclavo del protocolo *Modbus*. Además, sólo puede comunicarse a través de la variante *Modbus* TCP/IP. Por tanto, se descartó la idea de utilizar esta herramienta debido a que no implementa el protocolo requerido para realizarle las pruebas a las nuevas versiones del manejador *Ethernet/IP.*

***Modsim*:** esta herramienta está concebida para comportarse como un esclavo dentro de una red de comunicación *Modbus*, esto significa que el protocolo de comunicación que utiliza no es *Ethernet/IP*, que será el que utilicen las nuevas implementaciones del manejador. Por tanto, no es posible utilizarlo como solución del problema.

1.9. Tecnologías y herramientas

Para el desarrollo del simulador se hace necesario la utilización de un marco de trabajo (*framework,* en inglés). Un *framework* sirve de base para la organización y desarrollo del *software*. Puede incluir soporte de programas, bibliotecas, un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. También se requiere de un entorno de desarrollo integrado (IDE, por sus siglas en inglés). Un IDE puede denominarse como un entorno de programación que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI, por sus siglas en inglés). (16) Además se utiliza un lenguaje de programación, que es una herramienta que permite comunicarse e instruir a la computadora para que realice una tarea específica. (17) Para modelar el sistema se utiliza una herramienta que permite representar sus principales características mediante diagramas. Asimismo se utiliza una metodología de desarrollo de *software* para estructurar, planificar y controlar todo el proceso de desarrollo del proyecto.

1.9.1. *Framework* QT

Se utiliza el *framework* Qt[8] en su versión 4.8.2. El *framework* QT es multiplataforma, además es ampliamente usado para desarrollar aplicaciones con interfaz gráfica de usuario, así como, en el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores. El marco de Qt se compone de módulos de plataforma C ++, bibliotecas Qt y un entorno de desarrollo integrado con Qt *Creator*. Además cuenta con herramientas que permiten una elección del enfoque de interfaz de usuario, ya sea Qt *Quick*, C++, HTML5 o un híbrido de estos.

1.9.2. IDE Qt *Creator*

El IDE utilizado para el desarrollo de la aplicación es Qt *Creator* en su versión 2.5.0. Qt *Creator* es un IDE multiplataforma que se ajusta a las necesidades de los desarrolladores. Este se centra en proporcionar

[8] Quasar Technologies.

características que ayudan a los nuevos usuarios del IDE a aprender y comenzar a desarrollar rápidamente, también aumenta la productividad de los desarrolladores con experiencia en Qt. (18)

Qt *Creator* cuenta con:

- Un editor de código con soporte para C++.
- Herramientas para la rápida navegación por el código.
- Resaltado de sintaxis y auto-completado de código.
- Control estático de código y estilo a medida.
- Soporte para refactorización de código.
- Paréntesis coincidentes y modos de selección.

1.9.3. Lenguaje de programación C++

Como lenguaje de programación se utiliza C++ versión 98. Este es un lenguaje imperativo[9] orientado a objetos derivado de C. Es un súper conjunto de C, que se creó para añadirle cualidades y características de las que carecía. El resultado es que como su ancestro, sigue ligado al *hardware* subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se le han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. (19)

1.9.4. Biblioteca *TransportProvider*

Biblioteca de comunicación desarrollada en el CEDIN para realizar de forma eficiente la conexión al medio físico. La misma permite la lectura y escritura de datos a través de redes *Ethernet*, sobre protocolos de nivel de transporte TCP/IP, UDP/IP o buses seriales RS232. Contiene un conjunto de funcionalidades que permiten la abstracción en el intercambio de información entre diferentes nodos dentro de una red. (20) Posibilita el intercambio asíncrono, dándole una mayor eficiencia en tiempo y recursos a los manejadores que la emplean. Puede ser utilizada por cualquier sistema que necesite comunicarse con dispositivos de campo. Es multiplataforma, ya que puede ser compilada en cualquiera de los sistemas operativos más conocidos y utilizados.

[9] Los lenguajes imperativos son aquellos en los que se especifica cómo conseguir los objetivos que se persiguen. C; C++; JavaScript y Perl, entre otros muchos, pertenecen a esta categoría.

1.9.5. *Visual Paradigm*

La herramienta seleccionada para realizar el modelado de la solución fue el *Visual Paradigm* en su versión 8.0. *Visual Paradigm* es una herramienta de Ingeniería de *Software* Asistida por Computación (CASE, por sus siglas en inglés). La misma propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. (21) Fue utilizada para crear el modelo del dominio y diseñar los diagramas de clases y despliegue del sistema, haciendo uso del lenguaje de modelado unificado (UML, por sus siglas en inglés).

1.9.6. Lenguaje de Modelado Unificado (UML)

UML es el lenguaje de modelado de sistemas de *software* más conocido y manejado en la actualidad. Se utiliza para visualizar, especificar, construir y documentar el sistema. Ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. (22)

El lenguaje de modelado es una notación gráfica que indica los pasos que se deben seguir para llegar a un diseño. (22) Se utilizó para modelar los diagramas del sistema, describiendo la semántica, los objetos y símbolos utilizados en los mismos.

1.10. Metodología de desarrollo de *software*

Un proceso de *software* detallado y completo suele denominarse “Metodología”. Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental, espiral, entre otros). Adicionalmente una metodología debería definir con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto, guías para uso de herramientas de apoyo, etc. Habitualmente se utiliza el término “método” para referirse a técnicas, notaciones y guías asociadas, que son aplicables a las actividades del proceso de desarrollo, por ejemplo, suele hablarse de métodos de análisis y/o diseño. (23)

1.10.1. Metodología AUP-UCI

El Proceso Unificado Ágil o *Agile Unified Process* (AUP, por sus siglas en inglés) de *Scott Ambler* es una versión simplificada del Proceso Racional Unificado (RUP, por sus siglas en inglés). Este describe de una

manera simple y fácil de entender la forma de desarrollar aplicaciones de *software* de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP.

Al no existir una metodología de *software* universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable. Se decide hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI.

Dentro de las técnicas ágiles que implementa esta nueva variación y que se mantienen válidas en su antecesor, se pueden mencionar. (24)

- Desarrollo dirigido por pruebas.
- Modelado ágil.
- Gestión de cambios ágil.
- Refactorización de base de datos.

El uso de la técnica de modelado ágil permite encapsular los requisitos funcionales en HU o descripción de requisitos por procesos.

1.10.2. Fases de AUP-UCI:

Tabla 2. Fases de AUP-UCI.

Fases	Objetivos
Inicio	Planeación del proyecto. Estudio inicial de la organización cliente. Alcance del proyecto. Estimaciones de tiempo, esfuerzo y costo.
Ejecución	Ejecutan las actividades requeridas para desarrollar el *software*. Ajuste de los planes del proyecto. Modelado del negocio. Obtención de requisitos. Se elaboran la arquitectura y el diseño. Se implementa y se libera el producto.

	El producto es transferido al ambiente de los usuarios finales. Capacitar a los usuarios finales.
Cierre	Resultados del proyecto. Cierre del proyecto.

1.10.3. Disciplinas de APU-UCI

Tabla 3. Disciplinas de APU-UCI.

Disciplinas	Objetivos
Modelado de negocio (opcional)	Comprender los procesos de negocio.
Requisitos	Administración y gestión de los requisitos funcionales y no funcionales.
Análisis y diseño	Modelar el sistema.
Implementación	Construcción del sistema.
Pruebas internas	Verificar el resultado de la implementación.
Pruebas de liberación	Diseñar y ejecutar pruebas por una entidad externa certificadora de la calidad.
Pruebas de Aceptación	Verificar que el *software* está listo.
Despliegue (Opcional)	Instalación, configuración, adecuación, y puesta en marcha.
Gestión y soporte	-

1.10.4. Roles AUP-UCI

- Jefe de proyecto.
- Planificador.
- Analista.
- Arquitecto de información (Opcional).
- Desarrollador.

- Administrador de la configuración.
- *Stakeholder.*
- (Cliente/Proveedor de requisitos).
- Administrador de calidad.
- Probador.
- Arquitecto de *software* (Sistema).
- Administrador de BD.

1.11. Conclusiones parciales

Como resultado de la fundamentación teórica de la presente investigación, se puede concluir que la selección de las principales tecnologías, herramientas y metodología permitió sentar las bases para el desarrollo del producto. Así mismo los conceptos plasmados brindaron el pie de apoyo para el entendimiento de estas. Por otro lado se puede expresar que el estudio de simuladores existentes permitió adquirir un amplio conocimiento acerca de estas herramientas, lo cual es de vital importancia para el desarrollo del tema de la investigación.

Capítulo 2. Análisis y diseño.

2.1. Introducción

En presente capítulo se explicará el análisis y diseño de la aplicación. Basando sus principales objetivos en presentar los resultados de la fase de ejecución de la solución propuesta. Se especifican los requisitos funcionales y requisitos no funcionales que el sistema debe cumplir para satisfacer las necesidades del cliente. Se define la arquitectura del sistema y los patrones de diseño utilizados.

2.2. Fase de Inicio

En esta fase se realizó un análisis y estudio inicial de la organización cliente, donde se identificaron las necesidades del proyecto y se obtuvo toda la información necesaria acerca del alcance del mismo, para decidir si se ejecuta o no, siendo el resultado final positivo en este caso.

2.3. Fase de ejecución.

En esta fase se ejecutan las actividades requeridas para desarrollar el *software*, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Este es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del *software*. (24)

Con el resultado obtenido en esta fase y siguiendo el ciclo de vida de desarrollo de *software* que propone la metodología seleccionada AUP-UCI, se encapsularon los requisitos funcionales del sistema en HU. Las HU describen de manera sencilla y en lenguaje natural las necesidades del usuario final.

2.3.1. Requisitos funcionales del sistema.

Tabla 4. Historia de Usuario 1.

Historia de Usuario	
Número: 1	**Nombre del requisito:** Configurar la propiedad tiempo de lectura del dispositivo.
Programador: Reitel Ramos Macías. Sergio Santos Hernández.	**Iteración Asignada:** 1.

Prioridad: Alta.	**Tiempo Estimado:** 10 días.
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 10 días.
Descripción: Se modifica el valor del campo Tiempo de espera, se puede hacer entrando valores manualmente o simplemente a través del elemento visual *SpinBox* aumentando o disminuyendo secuencialmente el valor. Solo se aceptan números comprendidos en el rango de 10000 a 50000. Posteriormente se presiona el botón Aplicar cambios para que se introduzcan los nuevos valores en el sistema.	
Observaciones: Cuando se ejecuta esta función, el simulador se mantiene en espera de alguna señal del maestro durante el tiempo definido, luego de este tiempo si no llega ninguna trama se desconecta.	
Prototipo de interfaz: 	

Tabla 5. Historia de Usuario 2.

Historia de Usuario	
Número: 2	**Nombre del requisito:** Administrar mapa de memoria.
Programador: Reitel Ramos Macías. Sergio Santos Hernández.	**Iteración Asignada:** 1.
Prioridad: Alta.	**Tiempo Estimado:** 15 días.
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 20 días.
Descripción: Se presiona *clic* derecho sobre el elemento visual *TreeView*, se despliega un menú contextual con la opción Crear nuevo *tag.* Presionando *clic* sobre esta opción aparece una	

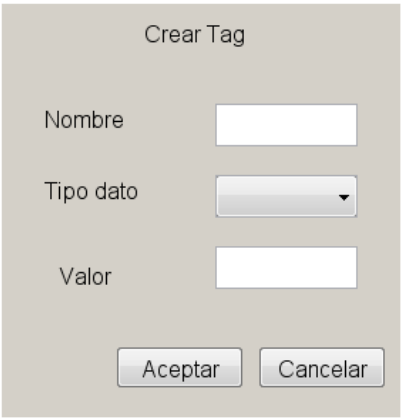
ventana donde se asignan los nuevos valores Nombre, Valor y Tipo de dato al *tag* creado. Se selecciona la opción Aceptar o Cancelar. Si la opción es Aceptar, el sistema verifica que los campos no estén vacíos y de ser así se crea el nuevo *tag*, en caso de tener campos vacíos se muestra un mensaje de alerta. Si la opción es Cancelar la ventana de diálogo se cierra. En caso de que no se quiera visualizar la ventana de determinado *tag* se cierra la misma dando *clic* en el botón cerrar, para el caso de querer restaurar dicha ventana se da *clic* derecho sobre el *tag en el TreeView*. Se despliega un menú contextual con las opciones de Mostrar y Eliminar, se selecciona la primera. Mostrando así la ventana con los valores asociados ha dicho *tag*. Para eliminar el *tag* se presiona *clic* derecho sobre él en el *TreeView*. Se despliega un menú contextual con las opciones de Mostrar y Eliminar, se selecciona la segunda. Se elimina el *tag* del sistema.
Observaciones: Se crea o se elimina un tag de memoria.
Prototipo de interfaz: 

Tabla 6. Historia de Usuario 3.

Historia de Usuario	
Número: 3	**Nombre del requisito:** Simular error en trama.
Programador: Reitel Ramos Macías. Sergio Santos Hernández.	**Iteración Asignada:** 2.
Prioridad: Alta.	**Tiempo Estimado:** 10 días.

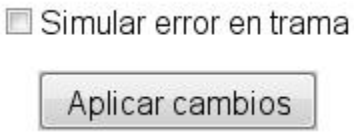
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 10 días.
Descripción: Se marca el *CheckButton* Simular error en trama, seguidamente se oprime el botón Aplicar cambios para que el sistema comience a simular error en trama.	
Observaciones:	
Prototipo de interfaz: 	

Tabla 7. Historia de Usuario 4.

Historia de Usuario	
Número: 4	**Nombre del requisito:** Configurar valores a simular en el mapa de memoria.
Programador: Reitel Ramos Macías. Sergio Santos Hernández.	**Iteración Asignada:** 3.
Prioridad: Alta.	**Tiempo Estimado:** 10 días.
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 10 días.
Descripción: Luego de creado el *tag,* se da doble *clic* en cualquiera de los campos Nombre, Valor o Tipo de dato de la ventana asociada. El sistema muestra una nueva ventana de diálogo en la cual se pueden modificar los valores del *tag* a ser simulados, Valor, Auto Simulación, Tipo (donde se despliega un *ComboBox* con los diferentes tipos Aleatorio, Ascendente y Descendente), Saltos (se habilita en caso de que no se haya seleccionado anteriormente el Tipo Aleatorio), Frecuencia, Valor mínimo y Valor máximo. Se presiona la opción Aceptar o Cancelar. Si la opción es Aceptar, el sistema comprueba que los campos no estén vacíos, de ser así modifica los valores, de otra forma muestra un mensaje de alerta. Si la opción es Cancelar la ventana de diálogo se cierra.	
Observaciones: Se muestra una ventana de diálogo con los valores del tag a simular.	

Prototipo de interfaz:

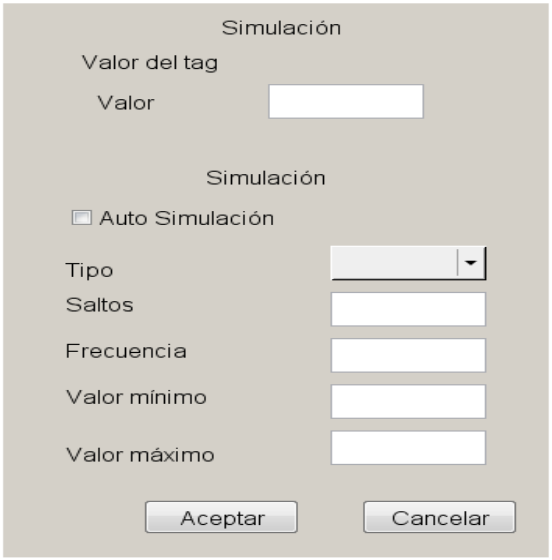


Tabla 8. Historia de Usuario 5.

Historia de Usuario	
Número: 5	**Nombre del requisito:** Administrar mensajes de lectura provenientes del manejador.
Programador: Reitel Ramos Macías. Sergio Santos Hernández.	**Iteración Asignada:** 3.
Prioridad: Alta.	**Tiempo Estimado:** 10 días.
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 10 días.
Descripción: Para administrar los mensajes de lectura provenientes del manejador es necesario haber iniciado el simulador presionando *clic* izquierdo sobre el botón iniciar, esta acción habilita el canal de comunicación y pasa a un estado operativo, luego comienza la comunicación y se recibe el mensaje proveniente del manejador. El simulador desensambla el mensaje recibido y verifica los datos que contiene en cada campo, para determinar la respuesta que debe dar al manejador.	

Observaciones:
Prototipo de interfaz:

Tabla 9. Historia de Usuario 6.

Historia de Usuario	
Número: 6	**Nombre del requisito:** Administrar mensajes de escritura a enviar al manejador.
Programador: Reitel Ramos Macías. Sergio Santos Hernández.	**Iteración Asignada:** 2.
Prioridad: Alta.	**Tiempo Estimado:** 10 días.
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 10 días.
Descripción: Para administrar los mensajes de escritura a enviar al manejador es necesario haber iniciado el simulador presionando *clic* izquierdo sobre el botón iniciar, esta acción habilita el canal de comunicación y pasa a un estado operativo. Luego el simulador ensambla el mensaje de escritura y se lo envía al manejador.	
Observaciones:	
Prototipo de interfaz:	

2.3.2. Requisitos no funcionales del sistema

Son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema. (25)

Requerimientos de *software*

- Debe funcionar en Sistemas Operativos basados en GNU/Linux.

Requerimientos de Hardware

Debe ser ejecutado en computadoras que tengan como requerimientos mínimos.

- Microprocesador: *dual core* a 1.5 GHz.
- RAM: 512 Mb.
- Tarjeta de Red: que soporte los estándares establecidos para la transmisión de datos sobre redes *Ethernet*.

Restricciones en el diseño y la implementación

- Se implementará utilizando el lenguaje de programación C++.
- Se utilizará la biblioteca *transportProvider* del CEDIN.
- Se utilizará el IDE Qt *Creator*.
- Requiere de un compilador de C++.
- Se utilizará del *framework* Qt.

Requerimientos de usabilidad

- El sistema debe proporcionar una interfaz gráfica sencilla, intuitiva y fácil de entender.

Rendimiento

- El tiempo de respuesta de la aplicación dependerá de las prestaciones de la computadora donde sea ejecutada.

Requerimiento de soporte

- Se publicará toda la documentación relacionada con la aplicación.
- Se realizará el mantenimiento al producto siempre que se necesite.

2.4. Arquitectura del sistema

La arquitectura del *software* alude a la estructura general del *software* y las formas en que proporciona una integridad conceptual para un sistema. En su forma más simple, la arquitectura estructura y organiza los componentes (o módulos) del programa, define la manera en que estos interactúan y la estructura de datos que utilizan. En un sentido más amplio, los componentes pueden generalizarse para representar elementos importantes del sistema y sus interacciones. (26)

2.4.1. Patrón de arquitectura de *software*

Para el desarrollo de la aplicación se propone utilizar el patrón de arquitectura de *software* Modelo-Vista-Controlador (MVC, por sus siglas en inglés). Ver figura 5.

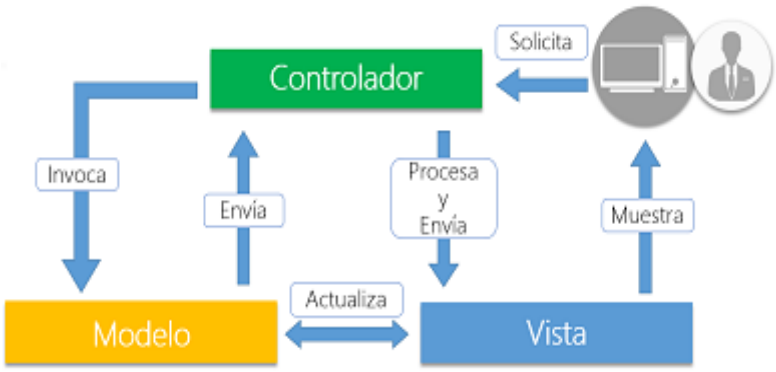


Fig. 5. Patrón de arquitectura de *software* Modelo-Vista-Controlador.

La arquitectura de *software* basada en el patrón arquitectónico MVC permite separar de manera clara y precisa los tres componentes de una aplicación: el modelo, la vista y el controlador. Su objetivo principal es el de separar la lógica del negocio de la lógica de la presentación para darle estructura a la implementación y facilitar con esto su posterior mantenimiento. (27)

Como se mencionó anteriormente, el sistema consta de tres partes de acuerdo con la siguiente descripción.

- El modelo representa la lógica de la aplicación, estará compuesto por un grupo de clases que permitirán administrar el mapa de memoria. Garantizando manejar los datos de la aplicación.
- La vista hace referencia a la interfaz de usuario y a la presentación de la información, la cual tendrá de forma amigable e intuitiva las opciones requeridas por el cliente. Además permite conocer el estado de las comunicaciones y los eventos que ocurren en el sistema mediante elementos visuales.
- El controlador actúa como mediador entre la solicitud del usuario y los modelos y vistas involucrados en la ejecución. La lógica de control del sistema radica principalmente en el canal de comunicación, donde se ejecutan los comandos recibidos.

2.4.2. Patrones de diseño

Un patrón de diseño nomina, abstrae e identifica los aspectos claves de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizables. Este identifica las clases e instancias participantes, sus roles y colaboraciones, y la distribución de responsabilidades. Cada uno se

centra en un problema concreto, describiendo cuando aplicarlo y si tiene sentido hacerlo teniendo en cuenta otras restricciones de diseño, así como, las consecuencias y las ventajas e inconvenientes de su uso. Por otro lado, como normalmente se tienen que implementar los diseños, un patrón proporciona código de ejemplo en C++ y a veces en *Smalltalk*[10] para ilustrar una implementación. (28)

El uso de los patrones de diseño en la implementación del sistema brinda la posibilidad de ahorrar gran cantidad de tiempo y recursos, además permite comprender, mantener y extender el *software* construido, así como, centrarse en desarrollar sólo las funcionalidades específicas requeridas por la aplicación.

2.4.2.1. Patrones grupo de los cuatro (GoF, Gang of Four)

Patrón Fachada: proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. (29)

Este patrón es utilizado en la aplicación, específicamente en las clases *EncoderFacade* y *DecoderFacade*, estas clases delegan en sus clases hijas, la responsabilidad de creación de objetos de las clases *Ethernet/IP*, *EIPLogix*, *EIPMessageRouter* y *EIPConnectionMessage*, como se puede apreciar en la figura 7 del diagrama que representa las relaciones entre las clases del controlador.

Patrón *Singleton*: el patrón de diseño *Singleton* (instancia única), está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. No se encarga de la creación de objetos en sí, sino que se enfoca en la restricción de la creación de un objeto (30).

Se utiliza este patrón en la clase *MemoryManager* garantizando que el mapa de memoria en el simulador sea único. Esta clase contiene un constructor privado que solo será accedido desde la misma clase, además se crea una instancia única de ella, mediante el método estático, el cual crea una instancia de la clase solo en el caso de que no exista. Luego devuelve la instancia creada permitiendo el acceso global a ella.

2.4.2.2. Patrones de Software Generales para la Asignación de Responsabilidades (GRASP)

[10] Lenguaje reflexivo de programación, orientado a objetos y con tipado dinámico.

En la siguiente tabla se muestran los patrones GRASP utilizados, además se describe cada uno y las principales clases donde estos se evidencian.

Tabla 10. GRASP.

Patrón	Descripción	Clases
Creador	Asigna a la clase B la responsabilidad de crear una instancia de la clase A si se cumple alguno de los puntos siguientes: 1. B contiene a A 2. B agrega a A 3. B tiene los datos de inicialización de A 4. B registra a A 5. B utiliza estrechamente a A	*EIPServer* *MainWindow* *TagTreeView* *EIPClientSession* *TagWidget* *MemoryManager*
Controlador	Asigna la responsabilidad de gestionar un mensaje de un evento del sistema a una clase que represente una de estas opciones. 1. Representa el sistema global, dispositivo o un subsistema. 2. Representa un escenario de caso de uso en el que tiene lugar el evento del sistema.	*EIPServer* *EIPClientSession* *MemoryManager* *MainWindow*
Alta Cohesión **&** **Bajo Acoplamiento**	Asigne responsabilidades de manera que la cohesión permanezca alta.	*EIPClientSession* *EncoderFacade* *DecoderFacade*

2.5. Modelo de dominio

El modelo de dominio (o modelo conceptual) es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio

del problema y conceptos del mundo real, no de los componentes de *software*. Una clase conceptual puede ser una idea o un objeto físico (símbolo, definición y extensión). (14)

2.5.1. Diagrama de clases del dominio

Luego del análisis y estudio de los procesos del negocio se diseña el diagrama de clases del dominio. Se realiza a través de un diagrama de clases UML, identificando los conceptos y objetos significativos, asociados al área de interés de la investigación. Tiene como objetivo lograr que se alcance un entendimiento del contexto en que se emplea el simulador.

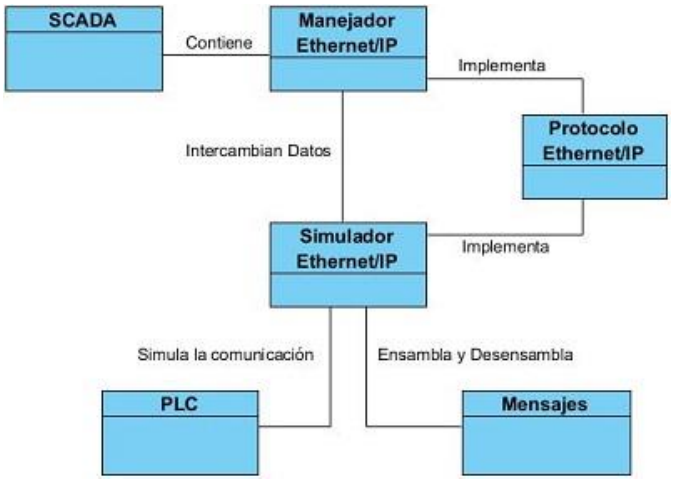


Fig. 6. Diagrama de clases del dominio.

2.5.2. Descripción del diagrama de clases del dominio

Los sistemas SCADA son *software* que se conectan por medio de redes de telecomunicaciones a los dispositivos de campo que intervienen en los procesos industriales como es el caso de los PLC. Estos sistemas contienen varios manejadores de dispositivos, que son módulos de *software* que contribuyen a que el sistema se comunique con los dispositivos de campo mediante un protocolo de comunicación determinado, en este caso *Ethernet/IP*.

Basándose en la descripción anterior, se utiliza el Simulador *Ethernet/IP* para simular la comunicación de un PLC en el intercambio de datos con el manejador *Ethernet/IP* que contiene el SCADA.

2.6. Diagramas de clases del sistema

Los diagramas de clases representan un conjunto de interfaces, colaboraciones y sus relaciones. Gráficamente son una colección de nodos y arcos. (31) Muestran una serie de clases, elementos y contenidos, representados a través de las relaciones entre ellos, conformando de esta manera la estructura el sistema.

En la investigación se modelaron los diagramas de clases del sistema teniendo en cuenta la arquitectura definida por el patrón Modelo-Vista-Controlador.

2.6.1. Relación entre clases del controlador

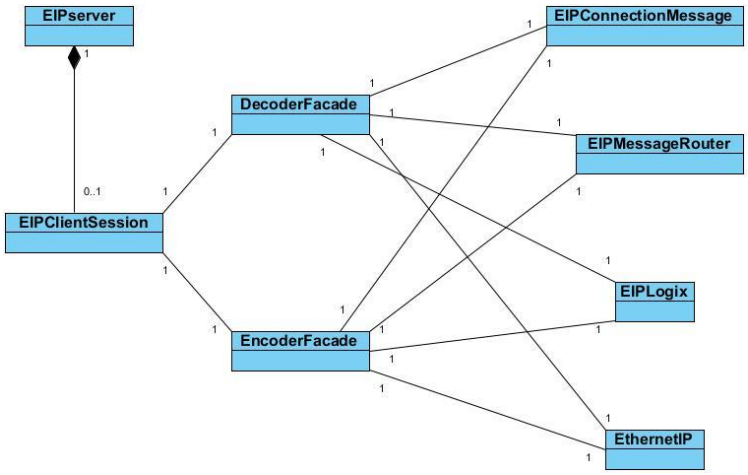


Fig. 7. Diagramas de clases del sistema.

2.6.2. Relación entre clases del modelo

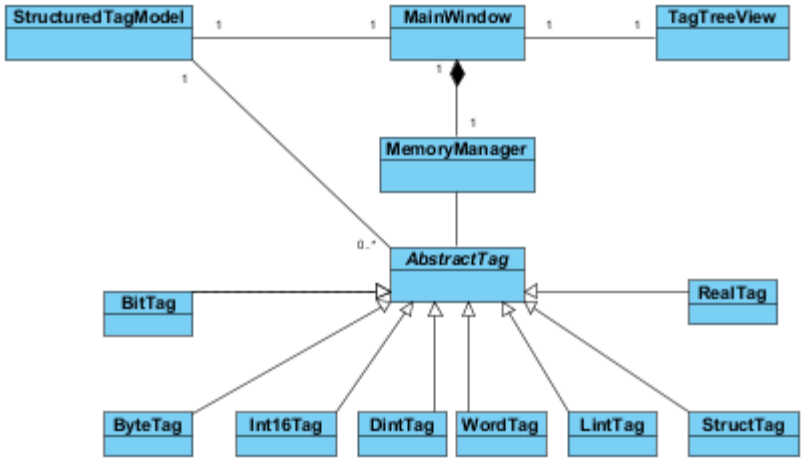


Fig. 8. Diagrama de clases del modelo.

2.6.3. Relación entre clases de la vista

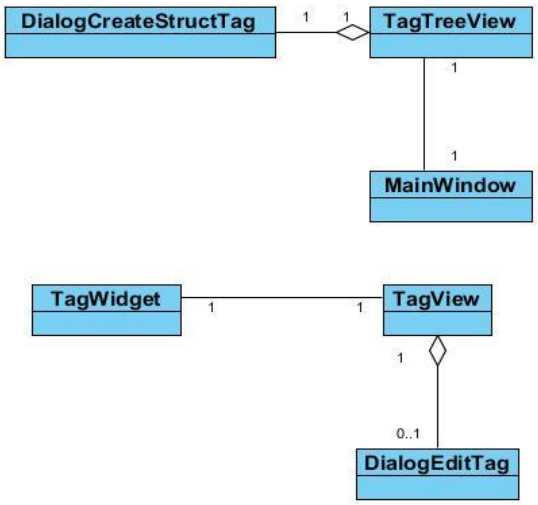


Fig. 9. Diagrama de clases de la vista.

2.7. Conclusiones parciales

Como resultado del desarrollo del presente capítulo se obtuvo el diseño de la aplicación a desarrollar. Donde se logró identificar los requerimientos funcionales a implementar, los cuales, según la metodología seleccionada se describen como HU. Así mismo se identificaron los requisitos no funcionales que permiten que el sistema funcione de manera correcta. La arquitectura del *software* permitió estructurar y organizar los componentes del programa, además generar el diagrama de dominio del sistema, así como, los diagramas de clases con sus respectivas relaciones. Todo este proceso llevado a cabo durante el concluyente capítulo permitió obtener el modelado del negocio, listo para ser implementado.

Capítulo 3. Implementación y Pruebas.

3.1. Introducción

Partiendo de los resultados del análisis y diseño obtenidos en el capítulo anterior se procede a iniciar la fase de implementación del sistema, la cual se debe realizar de forma iterativa e incremental, según la metodología AUP-UCI. De esta manera se obtiene un producto al final de cada iteración, que posteriormente pasa a la fase de pruebas. En esta fase la aplicación desarrollada se somete a pruebas internas y finalmente se despliega en los sistemas de producción. El presente capítulo describe las iteraciones y las tareas de ingeniería asociadas, así como, el diseño de casos de pruebas.

3.2. Fase de implementación

El objetivo de esta disciplina es transformar el modelo del sistema en código ejecutable. Se definen las iteraciones que permiten alcanzar una meta deseada, objetivo o resultado. Además se implementan las HU que se describen mediante las tareas de ingeniería. Estas tareas permiten organizar el proceso de implementación además de posibilitar que sea conocido el grado de complejidad de cada HU, teniendo en cuenta la cantidad de tareas asociadas a ella.

3.2.1. Iteración No.1

En la iteración No.1 la construcción del sistema se centra en darle cumplimiento a las HU 1 y 2. Se determinaron un total de 4 tareas de ingeniería para la implementación de las funcionalidades asociadas a cada HU.

3.2.1.1. *Tareas de ingeniería asociadas a la iteración No.1*

Tabla 11. Iteración No.1. Tarea de Ingeniería No.1.

Tarea de Ingeniería	
Número TI: 1	**Número HU:** 1
Nombre TI: Configurar la propiedad tiempo de lectura del dispositivo.	
Tipo de TI: Desarrollo.	
Tiempo dedicado: 10 días.	

Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías.
Descripción: • En el área de configuración del simulador se establece el valor deseado en el campo Tiempo de espera. Este valor va a permitir que el dispositivo realice la lectura durante el tiempo introducido en el campo, luego de ese tiempo, si no hay respuesta del cliente se termina la comunicación y se cierra la sesión. Los valores del campo se representan en milisegundos.

Tabla 12. Iteración No.1. Tarea de Ingeniería No.2.

Tarea de Ingeniería	
Número TI: 2	**Número HU:** 2
Nombre TI: Diseño e implementación del mapa de memoria.	
Tipo de TI: Desarrollo.	
Tiempo dedicado: 5 días.	
Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías	
Descripción: • El mapa de memoria del dispositivo se representa mediante el elemento visual *QTreeView*, que muestra en forma de árbol los nodos que contiene la memoria, cada nodo va a estar asociado a un *tag* del dispositivo. Se debe permitir crear y eliminar un nodo del mapa de memoria.	

Tabla 13. Iteración No.1. Tarea de Ingeniería No.3.

Tarea de Ingeniería	
Número TI: 3	**Número HU:** 2
Nombre TI: Crear nodo en el mapa de memoria.	
Tipo de TI: Desarrollo.	
Tiempo dedicado: 5 días.	
Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías.	

Descripción: • Cuando se inicia la aplicación se crea un *tag* de tipo estructura que constituye el nodo *root* o nodo inicial del árbol. • Los nodos representan los *tag* del dispositivo. Cada *tag* contiene un Nombre, Valor y Tipo de dato. • Para crear un nuevo *tag* se presiona *clic* derecho sobre el elemento visual *QTreeView* que representa la memoria, se selecciona la opción crear *tag.* • Cada vez que se crea un nuevo *tag* se añade un nuevo nodo a la memoria del dispositivo. • Se crean ventanas independientes para visualizar cada nodo de la memoria que representa un *tag* del dispositivo y se muestran en el elemento visual *QTreeView* que representa el mapa de memoria.

Tabla 14. Iteración No.1. Tarea de Ingeniería No.4.

Tarea de Ingeniería	
Número TI: 4	**Número HU:** 2
Nombre TI: Eliminar nodo del mapa de memoria.	
Tipo de TI: Desarrollo.	
Tiempo dedicado: 5 días.	
Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías.	
Descripción: • Haciendo *clic* derecho sobre el nodo que se desea eliminar en el elemento visual *QTreeView* que representa el mapa de memoria, se muestra un menú donde se selecciona la opción Eliminar. • Al eliminar un nodo de la memoria, en caso de ser de tipo estructura y ser nodo padre se destruyen todos sus hijos. • Solo se destruye el nodo inicial o nodo *root* cuando se cierra la aplicación.	

3.2.2. Iteración No.2

En la iteración No.2 la construcción del sistema se centra en darle cumplimiento a las HU 3 y 4. Se determinaron un total de 5 tareas de ingeniería para la implementación de las funcionalidades asociadas a cada HU.

3.2.2.1. *Tareas de ingeniería asociadas a la iteración No.2*

Tabla 15. Iteración No.2. Tarea de Ingeniería No.5.

Tarea de Ingeniería	
Número TI: 5	**Número HU:** 3
Nombre TI: Enviar trama válida con errores en los campos.	
Tipo de TI: Desarrollo.	
Tiempo dedicado: 10 días.	
Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías.	
Descripción: • Se configuran todas las opciones para establecer la comunicación. • Se selecciona la opción de Simular error en trama en el área de configuración de la aplicación. • Las tramas de respuestas a las peticiones del cliente se envían con un código de error en el campo *status*.	

Tabla 16. Iteración No.2. Tarea de Ingeniería No.6.

Tarea de Ingeniería	
Número TI: 6	**Número HU:** 4
Nombre TI: Diseño e implementación de la configuración de los valores a simular en el mapa de memoria.	
Tipo de TI: Desarrollo.	
Tiempo dedicado: 5 días.	

Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías.
Descripción: • Para simular valores en los nodos asociados a los *tag* del dispositivo en el mapa de memoria, se muestra un elemento visual que permite modificar los parámetros siguientes: el Valor, Auto Simulación, Tipo (puede ser Aleatorio, Ascendente o Descendente), Salto (para el caso que no se seleccione el Tipo Aleatorio), Frecuencia, Valor máximo y Valor mínimo.

Tabla 17. Iteración No.2. Tarea de Ingeniería No.7.

Tarea de Ingeniería	
Número TI: 7	**Número HU:** 4
Nombre TI: Simular valores aleatorios en el mapa de memoria.	
Tipo de TI: Desarrollo.	
Tiempo dedicado: 5 días.	
Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías	
Descripción: • Haciendo doble *clic* sobre las ventanas que representan de forma independiente cada *tag* del dispositivo se muestra un formulario que permite seleccionar Auto Simulación. • Se selecciona el tipo de simulación Aleatoria y se indica el rango de valores representados por Valor máximo y Valor mínimo entre los cuales se desea realizar la simulación. • Se generan valores aleatorios entre el rango definido.	

Tabla 18.Iteración No.2. Tarea de Ingeniería No.8.

Tarea de Ingeniería	
Número TI: 8	**Número HU:** 4
Nombre TI: Simular valores Ascendentes en el mapa de memoria.	

Tipo de TI: Desarrollo.
Tiempo dedicado: 5 días.
Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías.
Descripción: • Haciendo doble *clic* sobre las ventanas que representan de forma independiente cada *tag* del dispositivo se muestra un formulario que permite seleccionar Auto Simulación. • Se selecciona la simulación Ascendente y se indican los saltos, la frecuencia y el rango de valores representados por Valor máximo y Valor mínimo entre los cuales se desea realizar la simulación. • Se generan valores ascendentes con el salto definido entre el rango de valores seleccionado.

Tabla 19. Iteración No.2. Tarea de Ingeniería No.9.

Tarea de Ingeniería	
Número TI: 9	**Número HU:** 4
Nombre TI: Simular valores Descendentes en el mapa de memoria.	
Tipo de TI: Desarrollo.	
Tiempo dedicado: 5 días.	
Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías.	
Descripción: • Haciendo doble *clic* sobre las ventanas que representan de forma independiente cada *tag* del dispositivo se muestra un formulario que permite seleccionar Auto Simulación. • Se selecciona la simulación Descendente y se indica el Saltos, la Frecuencia y rango de valores representados por Valor máximo y Valor mínimo entre los cuales se desea realizar la simulación.	

• Se generan valores descendentes con el salto definido entre el rango de valores seleccionado.

3.2.3. Iteración No.3

En la iteración No.3 la construcción del sistema se centra en darle cumplimiento a las HU 5 y 6. Se determinaron un total de 2 tareas de ingeniería para la implementación de las funcionalidades asociadas a cada HU.

3.2.2.2. *Tareas de ingeniería asociadas a la iteración No.3*

Tabla 20. Iteración No.3. Tarea de Ingeniería No.10.

Tarea de Ingeniería	
Número TI: 10	**Número HU:** 5
Nombre TI: Desensamblar mensajes de petición de lectura provenientes del manejador.	
Tipo de TI: Desarrollo.	
Tiempo dedicado: 10 días.	
Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías.	
Descripción: • Se realiza la lectura de la cabecera del mensaje recibido que contiene 24 *bytes*. • Se leen posterior a la cabecera los *bytes* del cuerpo del mensaje. • Dependiendo del comando en el mensaje recibido se hace la codificación del mensaje de respuesta.	

Tabla 21. Iteración No.3. Tarea de Ingeniería No.11.

Tarea de Ingeniería	
Número TI: 11	**Número HU:** 6
Nombre TI: Ensamblar mensajes de petición de escritura provenientes del manejador.	

Tipo de TI: Desarrollo.
Tiempo dedicado: 10 días.
Programador Responsable: Sergio Santos Hernández, Reitel Ramos Macías.
Descripción: • Se realiza la codificación de la trama dependiendo del tipo de mensaje de escritura. • Se envía la trama codificada al cliente.

3.3. Diagrama de despliegue

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria. Los estereotipos permiten precisar la naturaleza del equipo: (32)

- Dispositivos.
- Procesadores.
- Memoria.

En la siguiente figura se muestra el diagrama de desligue del sistema:

Fig. 10. Diagrama de Despliegue.

3.3.1. Descripción del diagrama de despliegue del sistema

A continuación se muestra una representación real del despliegue del sistema, así como, su descripción para lograr una mayor comprensión.

Fig. 11. Despliegue del Sistema.

EL nodo PC-Cliente, representa el ordenador donde se ejecuta el manejador *Ethernet/IP*. El nodo Simulador *Ethernet/IP*, representa el ordenador donde se ejecuta la aplicación desarrollada. La comunicación entre ambos nodos se realiza a través del protocolo de comunicación industrial *Ethernet/IP* y el protocolo TCP/IP.

3.4. Fase de pruebas

El único instrumento adecuado para determinar el estatus de la calidad de un producto de *software* es el proceso de pruebas. Estas se ejecutan dirigidas a componentes del producto o al sistema en su totalidad, con el objetivo de medir el grado en que el *software* cumple con los requerimientos. Para ejecutarlas se utilizan los casos de prueba, especificados de forma estructurada mediante técnicas de prueba. Este proceso, sus objetivos y los métodos y técnicas usados se describen en el plan de prueba. (33)

La metodología de desarrollo de *software* AUP-UCI desagrega la fase de pruebas en tres disciplinas: Pruebas Internas, de Liberación y Aceptación. Para verificar el resultado de la implementación y comprobar la calidad del producto se realizaron Pruebas Internas, utilizando como método de prueba, las Pruebas de Caja Negra y como tipo, las Pruebas Funcionales. Esto incluye la búsqueda de defectos, validar que el sistema funciona tal como está establecido y verificando que se cumplan los requisitos.

3.2.1. Pruebas internas

En esta disciplina se verifica el resultado de la implementación probando cada etapa de la construcción del *software*, así como, las versiones finales a ser liberadas. Los artefactos generados en este caso son los diseños de casos de prueba.

3.2.2. Diseño de casos de pruebas

Un caso de prueba cubre el *software* más a fondo y con más detalle que un caso de uso. Los mismos incluyen todas las funciones que el programa es capaz de realizar. Estos deben tener en cuenta el uso de

todo tipo de datos de entrada/salida, cada comportamiento esperado, todos los elementos de diseño, y cada clase de defecto. Todos los requisitos deberán ser cubiertos por los casos de prueba. (34)

A continuación se muestran los casos de pruebas funcionales realizados al sistema.

Tabla 22. Caso de Prueba No.1

Prueba Funcional	
Caso de Prueba: 1	**Número de HU:** 1
Nombre de Prueba: Configurar la propiedad tiempo de lectura del dispositivo.	
Descripción: Comprueba que la aplicación permita configurar las propiedades del dispositivo.	
Condiciones de Ejecución: El simulador debe estar iniciado. En la computadora se encuentra instalada la biblioteca *TransportProvider*.	
Entrada/Pasos de Ejecución: Se modifica el valor del campo Tiempo de espera, se puede hacer entrando valores manualmente o simplemente a través del *SpinBox* aumentando o disminuyendo secuencialmente cada valor. Solo se aceptan números comprendidos en el rango de 10000 a 50000. Posteriormente se presiona el botón Aplicar cambios para que se introduzcan los nuevos valores en el sistema.	
Resultado Esperado: Se modifica el tiempo de espera a recibir una trama de lectura del maestro.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 23. Caso de Prueba No.2

Prueba Funcional	
Caso de Prueba: 2	**Número de HU:** 2
Nombre de Prueba: Administrar mapa de memoria.	
Descripción: Comprueba que la aplicación permita crear el mapa de memoria.	
Condiciones de Ejecución: En la computadora se encuentra instalada la biblioteca *TransportProvider*.	

Entrada/Pasos de Ejecución: Se presiona *clic* derecho sobre el *QTreeView*, se despliega un menú contextual con la opción Crear nuevo *tag,* se da *clic* en él. Aparece una ventana donde se asignan nuevos valores al *tag* creado como Nombre, Valor y Tipo de dato. Se presiona la opción Aceptar o Cancelar. Si la opción es Aceptar, el sistema verifica que los campos no estén vacíos y de ser así se crea el nuevo *tag*, de tener campos vacíos muestra un mensaje. Si la opción es Cancelar la ventana de diálogo se cierra. En caso de que no se quiera visualizar la ventana de determinado *tag* se cierra la misma dando *clic* en el botón cerrar, para el caso de querer restaurar dicha ventana se da *clic* derecho sobre el *tag en el QTreeView*. Se despliega un menú contextual con las opciones de Mostrar y Eliminar, se selecciona la primera. Mostrando así la ventana con los valores asociados ha dicho *tag*. Para eliminar el *tag* se presiona *clic* derecho sobre él. Se despliega un menú contextual con las opciones de Mostrar y Eliminar, se selecciona la segunda. Se elimina el *tag* del *QTreeView*.
Resultado Esperado: Se agrega/elimina un componente visual que representa la memoria con sus respectivas propiedades.
Evaluación de la Prueba: Satisfactoria.

Tabla 24. Caso de Prueba No.3

Prueba Funcional	
Caso de Prueba: 3	**Número de HU:** 3
Nombre de Prueba: Simular error en trama.	
Descripción: Comprobar que la aplicación simule errores en las tramas.	
Condiciones de Ejecución: El simulador debe estar iniciado. En la computadora se encuentra instalada la biblioteca *TransportProvider*.	
Entrada/Pasos de Ejecución: Se marca el *CheckButton* Simular error en trama, seguidamente se oprime el botón Aplicar cambios para que el sistema comience a simular error en trama.	
Resultado Esperado: Se envía una trama al maestro con un valor de error en al campo *status*.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 25. Caso de Prueba No.4

Prueba Funcional	
Caso de Prueba: 4	**Número de HU:** 4
Nombre de Prueba: Configurar valores a simular en el mapa de memoria.	
Descripción: Comprobar que la aplicación simule valores de diferentes tipos.	
Condiciones de Ejecución: El simulador debe estar iniciado. Se debe haber creado al menos un *tag* del dispositivo. En la computadora se encuentra instalada la biblioteca *TransportProvider*.	
Entrada/Pasos de Ejecución: Luego de creado el *tag,* se da doble *clic* en cualquiera de los campos Nombre, Valor o Tipo de dato de la ventana asociada. El sistema muestra una nueva ventana de diálogo en la cual se pueden modificar los valores del *tag* a ser simulados, Valor, Auto Simulación, Tipo (donde se despliega un *ComboBox* con los diferentes tipos Aleatorio, Ascendente y Descendente), Saltos (se muestra en caso de que no se haya seleccionado anteriormente el Tipo Aleatorio), Frecuencia, Valor mínimo y Valor máximo. Presiona la opción Aceptar o Cancelar. Si la opción es Aceptar, el sistema verifica que no existan campos vacíos, de ser así modifica los valores, de otra forma muestra un mensaje. Si la opción es Cancelar la ventana de diálogo se cierra.	
Resultado Esperado: Se observa cómo se van simulando los valores en los *tag* de memorias.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 26. Caso de Prueba No.5

Prueba Funcional	
Caso de Prueba: 5	**Número de HU:** 5
Nombre de Prueba: Administrar mensajes de lectura provenientes del manejador.	
Descripción: Comprueba que la aplicación administre correctamente los mensajes de lectura provenientes del manejador.	
Condiciones de Ejecución: El simulador debe estar iniciado. En la computadora se encuentra instalada la biblioteca *TransportProvider*.	

Entrada/Pasos de Ejecución: Para administrar los mensajes de lectura provenientes del manejador es necesario haber iniciado el simulador presionando *clic* izquierdo sobre el botón iniciar, esta acción habilita el canal de comunicación y pasa a un estado operativo, luego comienza la comunicación y se recibe el mensaje proveniente del manejador. El simulador desensambla el mensaje recibido y verifica los datos que contiene en cada campo, para determinar la respuesta que debe dar al manejador.
Resultado Esperado: Se administra correctamente en el simulador los mensajes de lectura provenientes del manejador.
Evaluación de la Prueba: Satisfactoria.

Tabla 27. Caso de Prueba No.6

Prueba Funcional	
Caso de Prueba: 4	**Número de HU:** 6
Nombre de Prueba: Administrar mensajes de escritura a enviar al manejador	
Descripción: Comprueba que la aplicación administre correctamente los mensajes de escritura a enviar al manejador.	
Condiciones de Ejecución: El simulador debe estar iniciado. En la computadora se encuentra instalada la biblioteca *TransportProvider*.	
Entrada/Pasos de Ejecución: Para administrar los mensajes de escritura a enviar al manejador es necesario haber iniciado el simulador. Se presionando *clic* izquierdo sobre el botón iniciar, esta acción habilita el canal de comunicación y pasa a un estado operativo. Luego el simulador ensambla el mensaje de escritura y se lo envía al manejador.	
Resultado Esperado: Se administra correctamente en el simulador los mensajes de escritura a enviar al manejador.	
Evaluación de la Prueba: Satisfactoria.	

3.5. Conclusiones parciales

Al finalizar la fase de implementación del sistema descrita a través de las tareas de ingeniería, se logró desarrollar la aplicación. La misma cumple con los requisitos funcionales encapsulados en las HU definidos

anteriormente. Además culminó la fase de pruebas de manera satisfactoria, donde a partir del diseño de casos de prueba se pudo verificar que el *software* desarrollado está listo y que puede ser usado por los usuarios finales para ejecutar aquellas funciones y tareas para las cuales fue diseñado y construido.

Conclusiones

Como resultado de este trabajo se obtiene una herramienta que simula la comunicación de un dispositivo de campo con el manejador, para el protocolo *Ethernet/IP.* Dicha herramienta permite probar el funcionamiento de las nuevas implementaciones del manejador *Ethernet/IP*, desarrollado para la versión Miranda R2 del SCADA GALBA. Además permite generar y visualizar datos en diferentes formatos, así como, recrear situaciones excepcionales que pueden presentarse en las comunicaciones.

Para el desarrollo de este trabajo se obtuvo además:

- Un diseño del modelo a desarrollar a través del uso de la metodología, tecnologías y herramientas seleccionadas.
- Los requisitos que permitieron que la implementación cumpla con todas las exigencias y necesidades requeridas por el cliente.
- Un programa que funciona correctamente, garantizado por la ejecución de las pruebas, las cuales permitieron detectar y corregir los errores cometidos durante la implementación.

De manera general el desarrollo de este trabajo le brinda a la línea de Adquisición de Datos del centro CEDIN la posibilidad de contar con una herramienta que permite realizar el proceso de pruebas a las nuevas versiones de los manejadores a desarrollar.

Recomendaciones

- Adicionar nuevas funcionalidades que le permitan a la aplicación contar con un mayor número de fallas a simular.
- Que se implemente el tipo de dato arreglo, definido en las especificaciones técnicas del protocolo.
- Incorporar la capacidad para simular dispositivos con diferentes características en cuanto al manejo del mapa de memoria que utilizan.

Referencias bibliográficas

1. SCADA. *TheFreeDictionary.com.* [En línea] [Citado el: 17 de febrero de 2015.] http://encyclopedia2.thefreedictionary.com/supervisory+control+and+data+acquisition.

2. Scielo. *Impacto de algunas tecnologías en el desarrollo de los sistemas SCADA.* [En línea] [Citado el: 11 de mayo de 2015.] http://www.scielo.org.ve/scielo.php?script=sci_arttext&pid=S1316-48212005000400008&.

3. Abraham Ravelo, José Carlos. Desarrollo de un simulador para realizar las pruebas del manejador Modbus Omni. *biblioteca2.uci.cu.* [En línea] 2010. [Citado el: 22 de febrero de 2015.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_05721_12.

4. Bú, Raúl Coss. Simulación: Un enfoque práctico. Editorial Limusa. [En línea] 1998. [Citado el: 20 de febrero de 2015.] http://books.google.es/books?id=iY6dI3E0FNUC&pg=PA9&hl=es&source=gbs_selected_pages&cad=2#v=onepage&q&f=false.

5. *SEDE MANIZALES.* [En línea] [Citado el: 19 de mayo de 2015.] http://www.virtual.unal.edu.co/cursos/sedes/manizales/4060015/Lecciones/Capitulo%20VI/ventajas.htm.

6. Albuerne Rivero, Miguel. Simulador de dispositivos PLC-5 para realizar el proceso de pruebas al manejador DF1 del SCADA SAINUX. [En línea] 2010. [Citado el: 10 de mayo de 2015.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/8195/1/TD_06604_13.pdf.

7. Control lógico programable 2.0 UTP - Control lógico programable. [En línea] http://electricidad.utpuebla.edu.mx/Manuales%20de%20asignatura/5to%20cuatrimestre/Control%20logico%20programable.p.

8. Controladores ControlLogix estándar. [En línea] [Citado el: 16 de marzo de 2015.] http://ab.rockwellautomation.com/es/Programmable-Controllers/ControlLogix-Standard-Controllers#applications.

9. AUTÓMATAS PROGRAMABLES. [En línea] [Citado el: 23 de marzo de 2015.] http://www.infoplc.net/files/descargas/rockwell/infoPLC_net_Ejemplo_Programacion_Control_Logix.pdf.

10. Protocolos de red. *EcuRed.* [En línea] [Citado el: 10 de abril de 2015.] http://www.ecured.cu/index.php/Protocolos_de_red.

11. Tecnología y redes de transmisión de datos. [En línea] [Citado el: 23 de marzo de 2015.] https://books.google.es/books?id=2zzUqp-Jp-

oC&pg=PA266&dq=Protocolo+de+comunicaci%C3%B3n+TCP/IP&hl=es&sa=X&ei=H7JUVfO0IIiAsQSg8YHQDA&ved=0CCAQ6AEwAA#v=onepage&q&f=false.

12. Rockwell Automation. *Descripción general del sistema Ethernet/IP de Rockwell Automation.* [En línea] [Citado el: 23 de marzo de 2015.] http://www.etitudela.com/entrenadorcomunicaciones/downloads/ethertnetipdescripciondelsistema.pdf.

13. *Ethernet/IP*. Protocolo de red en niveles para aplicaciones de automatización industrial. *SIEMON.* [En línea] [Citado el: 15 de mayo de 2015.] https://www.siemon.com/la/white_papers/03-10-13-ethernet-ip.asp.

14. Bridón Danger, Yordanis. Interfaz asíncrona para la comunicación con los Controladores Lógicos Programables utilizando el Protocolo Industrial *Ethernet/IP*. [En línea] [Citado el: 14 de mayo de 2015.] http://repositorio_institucional.uci.cu/jspui/bitstre am/ident/TD_1625_08/1/TD_1625_08.pdf.

15. Enrique García, Luis, y otros. *Desarrollo SCADA Nacional. Sintaxis de Protocolo Implementados por los Manejadores.* La Habana : s.n., 2009.

16. Programación Desarrollo. *¿Qué es un Entorno de Desarrollo Integrado, IDE?* [En línea] [Citado el: 10 de mayo de 2015.] http://programaciondesarrollo.es/que-es-un-entorno-de-desarrollo-integrado-ide/# .

17. Programación en C++. [En línea] http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Introducci%C3%B3n.

18. QT Creator. [En línea] [Citado el: 16 de marzo de 2015.] https://wiki.qt.io/Category:Tools::QtCreator_Spanish.

19. El lenguaje C++. [En línea] [Citado el: 15 de mayo de 2015.] http://www.zator.com/Cpp/E1_2.htm.

20. Ing. Lorié Guerra, Javier. *Curso de entrenamiento para el desarrollo de drivers. Conferencia #2 Transporte de Datos.* La Habna : s.n., 2010.

21. Visual Paradigm. *EcuRed.* [En línea] [Citado el: 14 de mayo de 2015.] ttp://www.ecured.cu/index.php/Visual_Paradigm.

22. Castillejos Aragon, Jesús Alberto. Lenguaje de modelado unificado: Diagrama de clases. [En línea] http://jesusalbertocastillejosaragon.blogspot.com/2011/10/lenguaje-de-modelado- unificado-diagrama.html.

23. METODOLOGIAS PARA DESARROLLO DE *SOFTWARE*. *Procesos de Software.* [En línea] [Citado el: 10 de mayo de 2015.] http://procesosde*software*.wikispaces.com/METODOLOGIAS+PARA+*SOFTWARE*.

24. Rodríguez Sánchez, Tamara. PROGRAMA DE MEJORA. Metodología de desarrollo para la Actividad productiva de la UCI. [En línea] 2014. excriba.prod.uci.cu/documentos públicos/metodología uci.pdf.

25. Requerimientos Funcionales y No Funcionales (RF/RNF). [En línea] [Citado el: 15 de mayo de 2015.] http://ingenieriade*software*.bligoo.com.mx/requerimientos-funcionales-y-no-funcionales-rf-rnf.

26. Pressman, Roger S. Cap_09_Ingeniería_del_Diseño, Epíg_9.3.2. *bibliodoc.uci.cu.* [En línea] [Citado el: 15 de mayo de 2015.] http://bibliodoc.uci.cu/pdf/8448111869.pdf.

27. INTRODUCCIÓN AL MVC DE YII – PARTE I. [En línea] [Citado el: 15 de mayo de 2015.] http://blog.jorgeivanmeza.com/2009/03/introduccion-al-mvc-de-yii-parte-i.

28. Patrones de diseño. [En línea] [Citado el: 20 de mayo de 2015.] http://www.utnianos.com.ar/foro/attachment.php?aid=3577.

29. Patrón fachada. [En línea] [Citado el: 20 de mayo de 2015.] https://msdn.microsoft.com/es-es/library/bb972240.aspx.

30. Singleton. *Marco de Desarrollo de la Junta de Andalucía.* [En línea] febrero de 2014. [Citado el: 19 de junio de 2015.] http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/202.

31. Diagramas de clases del sistema. *Manografias.com.* [En línea] [Citado el: 15 de mayo de 2015.] http://www.monografias.com/trabajos88/diagramas-clases/diagramas-clases.shtml.

32. Diagrama de Despliegue. *ANALISIS Y DISEÑO DE SISTEMAS II.* [En línea] [Citado el: 15 de mayo de 2015.] virtual.usalesiana.edu.bo/web/practica/archiv/despliegue.doc.

33. Fase de pruebas del sistema. *Pruebas de Software.* [En línea] [Citado el: 15 de mayo de 2015.] http://pruebasde*software*.com/laspruebasde*software*.htm.

34. Definición de caso de prueba. *Testeando el Software.* [En línea] [Citado el: 15 de mayo de 2015.] http://testeando*software*.com/casos-de-uso-vs-casos-de-prueba/.

Glosario

Modbus: protocolo de comunicaciones, basado en la arquitectura cliente-servidor, diseñado en 1979 por *Modicon* para su gama de controladores lógicos programables.

OMNI: se refiere al protocolo de comunicación, *Modbus Omni,* desarrollado por *Enron Corporations*. 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.

OSI: interconexión de sistemas abiertos. Son estándares de redes de ordenador desarrollados con el fin de crear estándares comunes de comunicación entre programas y ordenadores creados por distintos fabricantes. Se estructura en siete niveles (Presentación, Aplicación, Sesión, Transporte, Red, Nivel físico y Enlace de datos) que definen normas en cada uno de ellos desde las conexiones puramente físicas hasta las relaciones entre las aplicaciones.

***Ethernet*:** es un estándar de redes de área local para computadores con acceso al medio por detección de la onda portadora y con detección de colisiones (CSMA/CD). *Ethernet* define las características de cableado y señalización de nivel físico y los formatos de tramas de datos del nivel de enlace de datos del modelo OSI. *Ethernet* se tomó como base para la redacción del estándar internacional IEEE 802.3, siendo usualmente tomados como sinónimos. Se diferencian en uno de los campos de la trama de datos. Sin embargo, las tramas *Ethernet* e IEEE 802.3 pueden coexistir en la misma red.

TCP (*Transmission Control Protocol*): protocolo de nivel de transporte TCP/IP estándar que proporciona el servicio de flujo confiable *full duplex* y del cual dependen muchas aplicaciones. TCP/IP permite que el proceso en una máquina envíe un flujo de datos hacia el proceso de otra. Está orientado a la conexión en el sentido de que, antes de transmitir datos, los participantes deben establecer la conexión. Todos los datos viajan en segmentos TCP, en donde cada viaje se rechaza a través de Internet en un datagrama IP. El conjunto de protocolos completo se conoce frecuentemente como TCP/IP debido a que el TCP y el IP son los dos protocolos más importantes.

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

Controlador Lógico Programable (PLC): dispositivo electrónico programable utilizado en la automatización industrial para realizar estrategias de control básicas. Por su robustez y características permite ejecutar tareas básicas del control, aún cuando no tenga conexión a las capas superiores del control.

IEEE 802.3: es una asociación mundial de técnicos e ingenieros dedicada a la estandarización. El comité de la IEEE definió un estándar que incluye el formato del paquete de datos para redes *Ethernet*. Describe el cableado a usar, el máximo de distancia alcanzable, la topología, el método de acceso al medio y la velocidad para manejar los datos, establecida en 10 Mb/s.

Anexo 1. Descripción de las Clases del Sistema.

1. Descripción de las Clases del Controlador

Tabla 28. Descripción de la clase EIPServer.

Nombre	*EIPServer*				
Descripción	La clase EIPServer se encarga de la gestión de sesiones y del mecanismo para aceptar conexiones.				
Atributos					
Nombre		**Tipo**		**Descripción**	
atbProvider		*ITransportProvider*		Apuntador a la instancia del *TransportProvider*	
atbAceptor		*IAcceptor*		Apuntador a la instancia del Aceptor	
atbClientSession		*EIPClientSession*		Apuntador a la instancia de la sesión activa.	
atbMemory		*Abstracttag*		Apuntador a la memoria *ControlLogix*	
atbSessionTimeExpire		*unsigned int*		Tiempo de límite de espera	
atbErrFrameSimulation		*bool*		Bandera que indica la simulación de falla de error en trama.	
Métodos					
Nombre	EIPServer()	**Tipo**		**Parámetros**	
Descripción	Constructor de la clase, inicializa los parámetros del servidor.				
Nombre	*start()*	**Tipo**	*void*	**Parámetros**	
Descripción	Se encarga de iniciar el proceso de aceptar conexiones por el puerto configurado.				
Nombre	*stop()*	**Tipo**	*void*	**Parámetros**	
Descripción	Se encarga de cerrar la sesión activa en caso de que exista y para el proceso de aceptar conexiones.				
Nombre	*freeInstance()*	**Tipo**	*void*	**Parámetros**	

Descripción	Se encarga de liberar la memoria de una sesión activa.				
Nombre	*setMemory()*	**Tipo**	*void*	**Parámetros**	*Abstracttag * newMemory*
Descripción	Modifica el valor del apuntador de la memoria *ControlLogix* asignándole la pasada por parámetro.				
Nombre	*setTimeExpire()*	**Tipo**	*void*	**Parámetros**	*unsigned int time*
Descripción	Modifica el valor del parámetro Tiempo de espera asignándole el pasado por parámetro.				
Nombre	*getTimeExpire()*	**Tipo**	*unsigned int*	**Parámetros**	
Descripción	Brinda el valor del parámetro Tiempo de espera.				
Nombre	*managerDataValue()*	**Tipo**	*void*	**Parámetros**	*vector<LGXAddress*> &address, vector<bool> listOperation*
Descripción	Se encarga según una lista de direcciones y la operación a ejecutar para cada una de estas variables de buscar la variable en la memoria y establecer u obtener el valor según corresponda.				
Nombre	*setErrFrameSimulation()*	**Tipo**	*void*	**Parámetros**	*bool flag*
Descripción	Modifica el valor del parámetro Simular falla error en trama asignándole el pasado por parámetro				
Nombre	*setErrFrameSimulation()*	**Tipo**	*bool*	**Parámetros**	
Descripción	Brinda el valor del parámetro Simular falla error en trama.				

Tabla 29. Descripción de la clase EIPClientSession

Nombre	*EIPClientSession*
Descripción	La clase *EIPClientSession* se encarga de manejar el proceso de intercambio de información con los clientes. Desarrolla el mecanismo para el control de la comunicación, la codificación y decodificación de los mensajes.
Atributos	

Nombre	Tipo	Descripción
atbTransport	*ITCPTransport*	Apuntador a la instancia del transporte de tipo TCP encargado de ejecutar las acciones de envío y recepción de los mensajes.
atbBufferManager	*ByteBuffer*	Contenedor de la información del mensaje con que se está trabajando.
atbIsHeader	*bool*	Bandera que indica si la lectura actual es la de la cabecera del mensaje o del cuerpo.
atbDecoder	*DecoderFacade*	Instancia del objeto que se encarga de decodificar los mensajes enviados por los clientes.
atbEncoder	*EncoderFacade*	Instancia del objeto que se encarga de codificar los mensajes de respuesta del servidor al cliente.
atbCurrentState	*EtherNetIPState*	Parámetro que indica el estado en que se encuentra la comunicación entre el cliente y el servidor.
atbServer	*EIPServer*	Apuntador a la instancia del servidor.
atbSessionTimeOut	*unsigned int*	Tiempo límite para que la sesión este activa sin recibir mensajes del cliente.
atbCurrentHeader	*Encap_Header*	Instancia del objeto cabecera de los mensajes *EthernetIP*. En esta variable se guarda la cabecera del mensaje que se trabaja.
atbSenderContext	*unsigned char*	Contexto enviado por el cliente.
atbSessionHandle	*unsigned int*	Identificador de la sesión.

Métodos					
Nombre	*EIPClientSession()*	**Tipo**		**Parámetros**	*ITCPTransport*transport, EIPServer * server*

Descripción	Constructor de la Clase, encargado de inicializar los parámetros.				
Nombre	*startListen()*	**Tipo**	*void*	**Parámetros**	
Descripción	Este método permite poner a esperar solicitudes de los clientes.				
Nombre	*stopListen()*	**Tipo**	*void*	**Parámetros**	
Descripción	Método que cierra la sesión.				
Nombre	*registerSession()*	**Tipo**	*void*	**Parámetros**	
Descripción	Método encargado de la gestión del mensaje de tipo *REGISTER SESSION*				
Nombre	*forwardOpen()*	**Tipo**	*void*	**Parámetros**	
Descripción	Método encargado de la gestión del mensaje de tipo *FORWARD OPEN*				
Nombre	*unRegisterSession()*	**Tipo**	*void*	**Parámetros**	
Descripción	Método encargado de la gestión del mensaje de tipo *UNREGISTER SESSION*				
Nombre	*forwardClose()*	**Tipo**	*void*	**Parámetros**	
Descripción	Método encargado de la gestión del mensaje de tipo *FORWARD OPEN*				
Nombre	*operateReadWrite()*	**Tipo**	*void*	**Parámetros**	
Descripción	Método encargado de la gestión de los mensajes de tipo *READ* y *WRITE*				
Nombre	*disconnectHandler()*	**Tipo**	*void*	**Parámetros**	unsigned long error
Descripción	Método redefinido, es invocado tras desconectar el transporte del canal.				
Nombre	*readHandler()*	**Tipo**	*void*	**Parámetros**	*unsigned char * buffer,* *unsigned long size,*

					unsigned long error
Descripción	Método redefinido, es invocado tras realizar una operación de lectura de forma asíncrona.				
Nombre	*writeHandler()*	**Tipo**	*void*	**Parámetros**	*unsigned long size, unsigned long error*
Descripción	Método redefinido, es invocado tras realizar una operación de escritura de forma asíncrona.				

Tabla 30. Descripción de la Clase EncoderFacade.

Nombre	*EncoderFacade*				
Descripción	La clase *EncoderFacade* es la encargada de codificar los datagramas del protocolo.				
Atributos					
Nombre	**Tipo**	**Descripción**			
atbConnMessage	*EIPConnectionMessage*	Instancia del objeto Conexión encargado de codificar los segmentos de los mensajes referentes a la conexión.			
atbMsgRouter	*EIPMessageRouter*	Instancia del objeto *Router* encargado de codificar los segmentos de los mensajes referentes a la ruta del dispositivo			
atbEip	*EthernetIP*	Instancia del objeto *EthernetIP* encargado de codificar los segmentos del mensaje referentes a la cabecera *EthernetIP*			
atbLgx	*EIPLogix*	Instancia del objeto *Control Logix* encargado de codificar los segmentos del mensaje referentes al Dispositivo *Control Logix.*			
Métodos					
Nombre	*EncoderFacade()*	**Tipo**		**Parámetros**	
Descripción	Constructor de la Clase				
Nombre	*registerSessionReplyEncoder()*	**Tipo**	*void*	**Parámetros**	*ByteBuffer * writeBuffer, int sessionHandle,*

					short status, *unsigned char*senderContext,* *int MRStatus,* *int FOStatus,* *CIP_UDINT OtoTConnID,* *CIP_UDINT TtoOConnID,* *CIP_UINT connSerialNumber,* *CIP_UINT origVendorID,* *CIP_UDINT origSerialID,* *unsigned int msgRecive,* *unsigned int msgSend*
Descripción	Se encarga de codificar el mensaje Forward Open enviado por el servidor al cliente. Para esta tarea utiliza el identificador de la sesión, el contexto enviado por el cliente y los estados de cada una de las capas implicada en el mensaje.				
Nombre	*forwardCloseReplyEncoder()*	**Tipo**	*void*	**Parámetros**	*ByteBuffer * writeBuffer,* *int sessionHandle,* *short status,* *unsigned char*senderContext,* *int MRStatus,* *int FCStatus,* *CIP_UINT connSerialNumber,* *CIP_UINT origVendorID,* *CIP_UDINT origSerialID*
Descripción	Se encarga de codificar el mensaje *Forward Close* enviado por el servidor al cliente. Para esta tarea utiliza el identificador de la sesión y los estados de cada uno de los segmentos del mensaje implícitos.				
Nombre	*operationReadWriteReplyEncoder()*	**Tipo**	*void*	**Parámetros**	*ByteBuffer* writeBuffer,* *int sessionHandle,* *short status,int MRStatus,*

					*unsigned char * senderContext,* *vector<LGXAddress *> listVars,* *vector<bool> listService,* *int isRead,* *int packetNumber*
Descripción	Se encarga de codificar el mensaje *Read/Write* enviado por el servidor al cliente. Para esta tarea utiliza el identificador de la sesión, la lista de variables con los resultados de las operaciones y el número del paquete.				

Tabla 31: Descripción de la Clase DecoderFacade

Nombre	*DecoderFacade*	
Descripción	La clase *DecoderFacade* es la encargada de decodificar los datagramas provenientes del cliente.	
Atributos		
Nombre	**Tipo**	**Descripción**
atbEip	*EthernetIP*	Instancia del objeto *EthernetIP* encargado de decodificar las cabeceras *EthernetIP* del mensaje.
atbMR	*EIPMessageRouter*	Instancia del objeto *Router* encargado de decodificar los segmentos del mensaje referentes a la ruta del dispositivo.
atbConnMessage	*EIPConnectionMessage*	Instancia del objeto Conexión encargado de decodificar los segmentos del mensaje referentes a la conexión.
atbLogix	*EIPLogix*	Instancia del objeto Control *Logix* encargado de decodificar los segmentos del mensaje referentes a el dispositivo Control *Logix*.

Métodos					
Nombre	*DecoderFacade()*	**Tipo**		**Parámetros**	
Descripción	Constructor de la Clase				
Nombre	*headerRequestDecode()*	**Tipo**	*void*	**Parámetros**	*ByteBuffer * payload,* *Encap_Header * header*
Descripción	Se encarga de decodificar las cabeceras de los datagramas.				
Nombre	*registerSessionRequestDecode()*	**Tipo**	*void*	**Parámetros**	*ByteBuffer * payload,* *short &status,* *unsigned char*senderContex*
Descripción	Se encarga de decodificar los mensajes de tipo *Register Session* enviados por el cliente. De este se obtiene el contexto y el estado del mensaje.				
Nombre	*fowardOpenRequestDecode()*	**Tipo**	*int*	**Parámetros**	*ByteBuffer*payload,* *short &status,* *unsigned char*senderContex,* *unsigned int sessionHandle,* *int &MRStatus,* *int &FOStatus,* *CIP_UDINT OtoTConnID,* *CIP_UDINT TtoOConnID,* *CIP_UINT*

					connSerialNumber, *CIP_UINT origVendorID,* *CIP_UDINT &origSerialID*
Descripción	Se encarga de decodificar los mensajes de tipo Forward Open enviados por el cliente. De este se salva los estados de cada uno de los segmentos del mensaje, el contexto enviado por el cliente y los parámetros de la conexión enviados desde el cliente.				
Nombre	*forwardCloseRequestDecode()*	**Tipo**	*int*	**Parámetros**	*ByteBuffer * payload,* *short &status,* *unsigned char*senderCon-tex,* *unsigned int sessionHan-dle,* *int &MRStatus,* *int &FCStatus,* *CIP_UINT&connSerial-Number,* *CIP_UINT &origVendorID,* *CIP_UDINT &origSerialID*
Descripción	Se encarga de decodificar los mensajes de tipo *Forward Close* enviados por el cliente. De este mensaje se salvan los estados de cada uno de los segmentos presentes en el datagrama, el contexto enviado por el cliente y los parámetros de la conexión.				
Nombre	*operationReadWrite()*	**Tipo**	*int*	**Parámetros**	*ByteBuffer* payload,* *short &status,* *unsigned char*senderContex,*

					unsigned int sessionHandle, *int &MRStatus,* *int &LGXStatus,* *vector<LGXAddress*>&listVars,* *vector<bool>&listService,* *int &isRead,* *int &packetNumber*
Descripción	Se encarga de decodificar los mensajes de tipo *Read/Write* enviados por el cliente. De este se salvan el contexto enviado por el cliente , los estados de cada uno de los segmentos de los mensajes ,las direcciones de las variables sobres las que se operara , el tipo de operación y en caso de escritura el nuevo valor a establecer.				

2. Descripción de las Clases del Modelo

Tabla 32. Descripción de la Clase MemoryManager.

Nombre	*MemoryManager*				
Descripción	La clase MemoryManager es la encargada de controlar las operaciones sobre la memoria.				
Atributos					
Nombre	**Tipo**	**Descripción**			
memoryMap	*QMap<QString,tagWidget*>*	Mapa que contiene indexado por nombre de variable el *tagWidget* asociado.			
root	*tagWidget*	*tagWidget* raíz del árbol.			
Métodos					
Nombre	*MemoryManager*()	**Tipo**		**Parámetros**	

Descripción	Constructor de la Clase.				
Nombre	*addtagWidget()*	**Tipo**	*tagWidget*	**Parámetros**	*QString _name, tagType _type, QVariant _value = 0, Abstracttag* parent = 0*
Descripción	Permite construir un nuevo *tagWidget* según tipo, nombre y valor del tag.				
Nombre	deletetagWidget()	**Tipo**	bool	**Parámetros**	Abstracttag* _tag
Descripción	Permite eliminar un *tagWidget* mediante su referencia.				
Nombre	*gettagWidget()*	**Tipo**	*tagWidget*	**Parámetros**	*QString _name, bool& findtag*
Descripción	Permite obtener un *tagWidget* según el nombre de la variable que contiene.				
Nombre	*roottagWidget()*	**Tipo**	*tagWidget*	**Parámetros**	
Descripción	Permite obtener el *tagWidget* raíz del árbol.				

Tabla 33. Descripción de la Clase Abstracttag.

Nombre	*Abstracttag*				
Descripción	La clase representa las características genéricas de los *tag* del protocolo.				
Atributos					
Nombre	**Tipo**	**Descripción**			
tagName	*Qstring*	Nombre del *tag*			
atbtimer	*Qtimer*	*Timer* para la simulación del nodo			
tagType	*tagType*	Tipo de dato del valor del *tag*			
Métodos					
Nombre	*Abstracttag()*	**Tipo**		**Parámetros**	*QString _name*

					*Abstracttag *parent*
Descripción	Constructor de la Clase				
Nombre	*appendChild()*	**Tipo**	*void*	**Parámetros**	*Abstracttag* item*
Descripción	Permite adicionar un *Abstracttag* hijo.				
Nombre	removeChild()	**Tipo**	void	**Parámetros**	*Abstracttag *child*
Descripción	Permite eliminar un *Abstracttag* hijo.				
Nombre	*child()*	**Tipo**	*Abstracttag*	**Parámetros**	*int row*
Descripción	Permite según el índice en la lista de subárboles obtener el *Abstracttag* correspondiente				
Nombre	*getChild()*	**Tipo**	*Abstracttag*	**Parámetros**	*QString name*
Descripción	Permite obtener un *Abstracttag* mediante el nombre del *tag*.				
Nombre	*onTimeOut()*	**Tipo**	*void*	**Parámetros**	
Descripción	Acción que se realiza tras expirar el *timer*, en ella se simula el valor de los datos.				

3. Descripción de las Clases de la Vista

Tabla 34. Descripción de la Clase MainWindow.

Nombre	*MainWindow*	
Descripción	Ventana principal del sistema donde se gestionan los componentes visuales del sistema.	
Atributos		
Nombre	**Tipo**	**Descripción**
manager	*MemoryManager*	Instancia del Manejador de Memoria.
atbConfigure	*ConfigureWidget*	Instancia del *Widget* encargado de configurar los parámetros del servidor.
atbStatus	*StatusWidget*	Instancia del *Widget* encargado de mostrar el estado del Servidor.
treeView	*tagTreeView*	Instancia de la vista del Árbol de *tags*

model;	*StructuredtagModel*	Instancia del modelo utilizado en la vista del Árbol de *tags*.			
atbServer;	*EIPServer*	Instancia del servidor *EthernetIP*.			
Métodos					
Nombre	*setWidget()*	**Tipo**	*void*	**Parámetros**	*QMdiSubWindow* w*
Descripción	Adiciona al *mdiArea* un nuevo *QMdiSubWindow* en caso de que ya no exista dentro del *mdiArea*.				
Nombre	*removeWidget()*	**Tipo**	*void*	**Parámetros**	*QMdiSubWindow* w*
Descripción	Elimina el *QMdiSubWindow* del *mdiArea*.				
Nombre	*updateModelView()*	**Tipo**	*void*	**Parámetros**	
Descripción	Actualiza el modelo del Árbol de *tags* tras alguna operación sobre el mismo.				
Nombre	*onStart()*	**Tipo**	*void*	**Parámetros**	
Descripción	Ejecuta la acción de iniciar a esperar conexiones.				
Nombre	*onStop()*	**Tipo**	*void*	**Parámetros**	
Descripción	Ejecuta la acción de detener la comunicación del servidor.				
Nombre	*onOpen()*	**Tipo**	*void*	**Parámetros**	
Descripción	Ejecuta la acción de cargar una configuración de memoria salvada anteriormente.				
Nombre	*onSave()*	**Tipo**	*void*	**Parámetros**	
Descripción	Ejecuta la acción de salvar una configuración de memoria.				
Nombre	*createActions()*	**Tipo**	*void*	**Parámetros**	
Descripción	Crea internamente las acciones de los menús y de la barra de tareas.				
Nombre	*createMenu()*	**Tipo**	*void*	**Parámetros**	
Descripción	Crea internamente los menús del sistema.				

Anexo 2. Interfaz del sistema.

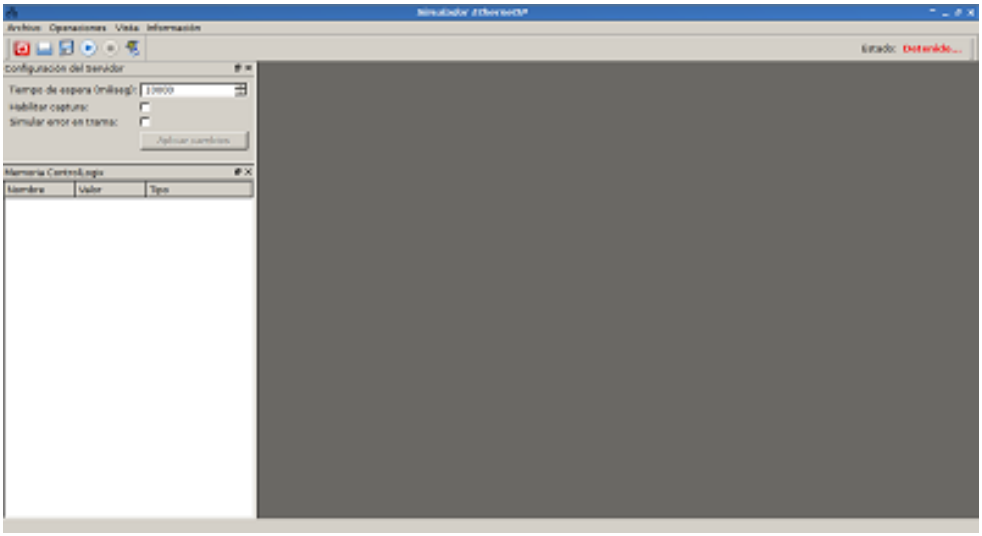


Fig. 12. Interfaz principal del Simulador *Ethernet/IP*.

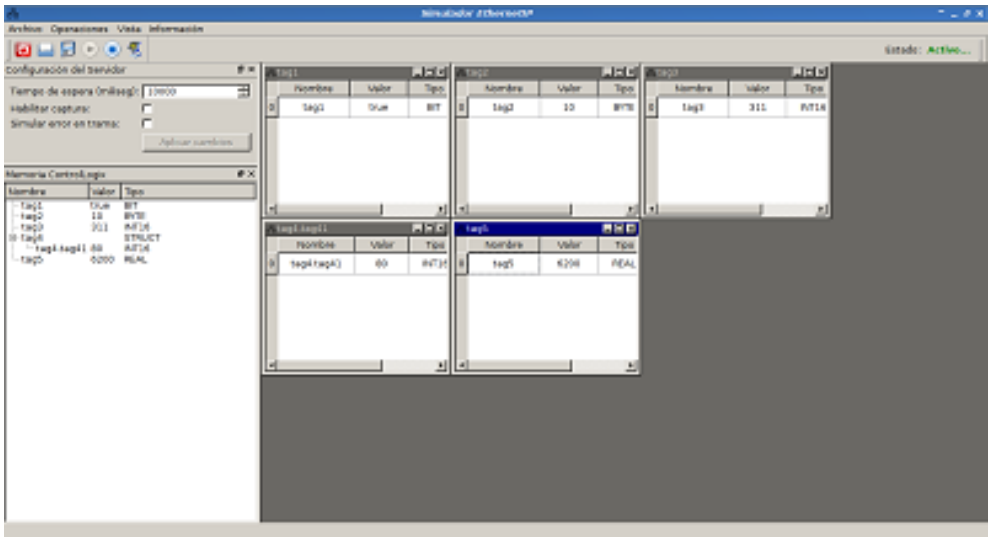


Fig. 13. Proceso de simulación.