Universidad de las Ciencias Informáticas
Facultad 5

Título: Interfaz asíncrona para la comunicación con los Controladores Lógicos Programables utilizando el Protocolo Industrial EtherNet/IP.

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

Autor(es): Yordanis Bridón Danger

Adrian Carlos Moreno Borges

Tutor: Dr. Rafael Arturo Trujillo Codorníu

Co-tutor: Ing. René López

Ciudad de La Habana, Julio, 2008.

"Año 49 de la Revolución"

DECLARACIÓN DE AUTORÍA

Por este medio declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 5 “Realidad Virtual y Automática” de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste se firma la presente a los 1 días del mes de Julio del año 2008.

Yordanis Bridón Danger

_____

Firma del Autor

Adrian Carlos Moreno Borges

_____

Firma del Autor

Dr. Rafael Arturo Trujillo Codorníu

_____

Firma del Tutor

Datos de Contacto

Nombre y Apellidos: Dr. Rafael Arturo Trujillo Codorníu.
Ciudadanía: Cubano.
Títulos: Máster en Ciencias Físico-matemáticas, 1979, Universidad de Odessa, URSS.
Doctor en Ciencias Físico-matemáticas (Ph.D), 1986, Universidad de Rostov del Don, URSS.
Profesión: Profesor.
Categoría Docente: Profesor Titular.
E-mail: Rtrujillo@ismm.edu.cu
Tiene 27 años de experiencia docente en la Educación Superior. Tutor de 2 Trabajos de Doctorado y 4 Trabajos de Maestrado. Presidente de la Comisión de Computación del Consejo Científico del Instituto Superior Minero Metalúrgico de Moa. Dirige la sección de Software del grupo “EROS” que diseña Sistemas de Supervisión y Control por Computadoras a las empresas del Níquel y a Centrales azucareros de Cuba y Brasil.

Nombre y Apellidos: Ing. René López Baracaldo.
Ciudadanía: Cubano.
Título: Ingeniero en Informática.
Profesión: Profesor.
E-mail: rene@uci.cu
Años de Graduado: 4
Años de Experiencia en el tema: 4

Agradecimientos

Muchas personas han contribuido, de una u otra forma, a la elaboración de este trabajo. A todas ellas queremos expresar nuestros más sinceros agradecimientos. Y en especial a:

A la revolución que nos dio la oportunidad a todos de tener educación.

A Fidel por tener esta maravillosa idea, que no deja de sorprendernos cada día que pasa.

A la decana Mayra, por poseer toda su confianza en nosotros.

Al profesor Fung, ese que fue nuestro primer guía y partícipe de nuestra formación profesional.

A nuestro Tutor que ha sabido con paciencia y sabiduría guiarnos por este reto que nos impusimos.

Al equipo de desarrolladores de la línea Drivers que nos acogieron entre ellos como parte de su equipo y nos brindaron todo su apoyo.

A Lidice (lizy), Dalia, Mariela y Saily por su dedicación.

A todos muchas gracias y estamos eternamente agradecidos.

Los autores.

No soy fruto de una sola persona sino de muchas que han puesto en mí un mar de amor y educación, para todas esas personas mi eterno agradecimiento pero en especial:

A todos mis profesores que me han hecho lo que soy hoy.

A la mi decana Mayra por darme una segunda oportunidad cuando la necesité.

A mis amigos y hermanos a esos que han sido fuente de inspiración que me han arrastrado con su ejemplo, en especial a Romanuel y al Mauro.

A Yordanis: ¡Lo logramos!

A Yisel por estos 10 años de amor y paciencia, que ha estado a mi lado en todo momento, dándome su apoyo ¡Ya tenemos fruto de nuestro amor!

A toda mi familia que me ha dado todo lo que necesita un hombre para vivir: amor.

A las mujeres a las cuales les dedico esta tesis que han sido para mí: familia, madres, tías, abuelas, amigas. Son todo lo que tengo y por lo que vivo. A María Eugenia, que ha tenido que sacar las fuerzas de donde no la tenía para continuar mi educación a pesar de su enfermedad. A ella que más que madre porque fue madre y padre a la vez. A Yolanda, que nunca creyó verme graduado, aquí estoy abuela querida, tú me ayudaste a dar los primeros pasos en esta vida.

Adrian Carlos.

Son muchas las personas especiales a las que les gustaría agradecer su amistad, apoyo, ánimo y compañía en las diferentes etapas de mi vida. Algunas están aquí conmigo, y otras en mis recuerdos y en mi corazón. Sin importar en donde estén o si alguna vez llegan a leer estás líneas quiero darle las gracias por formar parte de mí y por todo lo que me han brindado.

A todos mis profesores no solo de la carrera sino de toda la vida, mil gracias porque de alguna manera forma parte de lo que ahora soy. Especialmente a la profesora Lida, y al profesor Pedrito, mis últimos profesores, quienes me apoyaron y gracias a ellos estoy aquí, realizando mi sueño.

A mis amigos, que sin duda alguna es lo que más tengo en la vida, y sinceramente son todos ustedes porque son de la clase de personas que comprenden y dan lo mejor de sí mismo sin esperar nada a cambio, porque brindan ayuda cuando es necesario.

A mi compañero de tesis, que siempre lo tendré presente, porque este tipo de persona no son las que se olvidan de la noche a la mañana, para ti pues toda la felicidad del mundo.

Como testimonio de gratitud ilimitada, a mis hermanos, porque su presencia ha sido y será siempre el motivo más grande que ha impulsado para lograr esta meta.

No es fácil llegar siempre al final, se necesita ahínco, deseo y voluntad, pero sobre todo apoyo como el que he recibido de Eliades, ese padrastro a quién quisiera expresarle que mis esfuerzos y logros han sido también suyos y constituye el legado más grande que pudiera recibir.

A mi abuela: Una de mis razones de ser, partícipe de mi formación. Sirviéndome como madres su mirada, y su aliento, sirviéndome como padres su esfuerzo y su trabajo, sirviéndome como maestros sus palabras y sus sabios consejos. A ti te agradezco mi vida, gracias a ti, estoy aquí dedicándote este triunfo.

A mi padre: Por darme la oportunidad de existir, por su sacrificio en algún momento, por su confianza y amistad.

A mi madre: Quién me ha heredado el tesoro más valioso que puede dársele a un hijo: Amor. A quién sin escatimar esfuerzo alguno, ha sacrificado gran parte de su vida para formarme y educarme. A quién la ilusión de su vida ha sido convertirme en persona de provecho. A quién nunca podré pagar todos sus desvelos ni aún con las riquezas más grandes del mundo. Gracias por todo, te quiero mucho.

Por último quiero nuevamente darles las gracias a todos aquellos que me han devuelto una sonrisa, a todos aquellos que me ofrecieron su amistad en tiempos difíciles, a todos aquellos que han puesto de su parte para hacer posible este sueño y muy especial a la vida que como dijera Violeta Parra, me ha dado tanto……

Yordanis.

Dedicatoria

A mis madres: Yolanda, Marjorie,
Maria Eugenia y, Mariela.

Adrian Carlos

A mi madre Elena, a mi abuela Noris,
A Eliades, y a mis hermanos queridos
Yamanis y Noris.

Yordanis

Resumen

La comunicación de datos se ha convertido en una parte fundamental de la computación. Las redes globales reúnen datos sobre temas diversos, como las condiciones atmosféricas, la producción de cosechas y el tráfico aéreo. En los SCADA, el primer eslabón es la adquisición de datos, para esto se auxilian de los protocolos estandarizados para el intercambio de mensajes y el medio que estos especifican para enviar sus mensajes.

Los protocolos que están conveniados para el proyecto Guardián del ALBA son todos sincrónicos. Esto quiere decir que cuando se invoca una función de lectura de datos, el hilo que la invoca queda bloqueado hasta tanto no se reciba la respuesta del dispositivo. Para lograr la mayor concurrencia posible, este modelo exige un hilo por cada dispositivo que pueda ser encuestado en paralelo.

Este trabajo posee como objetivo implementar la capa protocolo EtherNet/IP basado en un modelo asincrónico para la conexión del SCADA Nacional a los Dispositivos de adquisición de Datos.

Esta capa de protocolo ahorrará los recursos del sistema al necesitar menos hilos por dispositivos, para encuestar en paralelo impactando de manera favorable en el rendimiento del sistema.

PALABRAS CLAVE

Protocolo, EtherNet/IP, transmisión de datos, CIP, TCP/IP, ASIO, PLC.

Índice

Índice de Figuras

Índice de Tablas

Introducción

Petróleos de Venezuela Sociedad Anónima (PDVSA) es la corporación estatal de la República Bolivariana de Venezuela dedicada a la exploración, producción, manufactura, transporte y mercadeo de los hidrocarburos.

PDVSA cuenta en sus instalaciones con Sistemas de Supervisión, Control y Adquisición de Datos (SCADA) suministrados por compañías extranjeras, que son imprescindibles para la confiabilidad y eficiencia de sus Procesos Tecnológicos, pero el costo de mantenerlos en funcionamiento es elevado e imponen una total dependencia de los proveedores. (1)

Algunas de estas compañías se plegaron y apoyaron el sabotaje petrolero acaecido durante diciembre del 2002 y enero del 2003. PDVSA sufrió pérdidas millonarias por concepto de las ventas no efectuadas. Estos hechos evidenciaron la necesidad de luchar por la independencia tecnológica de PDVSA, como empresa medular en la economía venezolana. (2)

Debido a la alta subordinación técnica inherente a los productos existentes en el mercado, no era posible contar con ellos como solución a la problemática. Surge entonces el proyecto SCADA Nacional (PSN), con el propósito de "Desarrollar un Sistema que supervise, controle, optimice y gestione los procesos industriales de PDVSA sirviendo de base para la implantación en otras industrias y organismos del país." (1)

En Cuba, existen experiencias previas, en el desarrollo, implementación y despliegue de SCADA en industrias nacionales, un ejemplo es EROS, desarrollado por ingenieros de la Unión del Níquel en la provincia de Holguín en el año 1991; utilizado en más de 100 plantas de diferentes tipos: industrias relacionadas con el sector niquelífero, centrales azucareros nacionales y 3 en Brasil. Es un sistema basado en tecnologías Windows que implementa muchas de las funcionalidades que exige un sistema distribuido y de control de este tipo.

En el 2006 el estado cubano, y en específico la UCI, se incorporan al proyecto conocido desde entonces en Cuba como SCADA NACIONAL PDVSA, pasándose a llamarse Guardián del ALBA. Se firmó un convenio de trabajo entre ambas partes y quedó planteado de manera escrita:

- Todos los componentes de las tareas propuestas debían ser desarrollados con software libre.
- Se deben utilizar estándares abiertos.

La comunicación de datos se ha convertido en una parte fundamental de la computación. Las redes globales reúnen datos sobre temas diversos, como las condiciones atmosféricas, la producción de cosechas y el tráfico aéreo.

La evolución de los sistemas de comunicación, permite ilustrar una dependencia del flujo de información oportuno, exacto y eficiente. Actualmente en los entornos industriales se aplica una distribución jerárquica para la interconexión de dispositivos, a través de diferentes soluciones de

comunicación; dicha distribución, conocida con el nombre de Pirámide de Automatización, define una serie de niveles caracterizados según prestaciones funcionales como la velocidad de transferencia, el grado de protección, el tipo de datos transmitidos, el volumen y el uso de los mismos.

Los primeros SCADA fueron diseñados para cumplir funciones específicas dentro de un proyecto determinado, es decir, se desarrollaban para una industria en particular. Las empresas que los producían se dieron cuenta que no estaban cumpliendo con varios de los principios de la programación como la adaptabilidad, escalabilidad y la reutilización del trabajo realizado, entonces empezaron a hacer módulos generales con capacidades requeridas comúnmente y otros para aspectos más específicos, que se adaptaran a cualquier ambiente con un mínimo de cambio, fue un paso significativo aunque previsible. Hoy día los SCADAs son parte integral de la estructura de las industrias y no son vistos como una simple herramienta operacional, sino como un recurso importante de información. Funcionan como centro de responsabilidad, al punto de poder controlar situaciones excepcionales como desastres ecológicos, pérdidas de vidas. (1)

En la cadena de tratamiento de la información en un SCADA el primer eslabón es la adquisición de los datos, estos provienen de disímiles equipos que pueden ser PLC, reguladores autónomos y sensores. Esta variedad está generalmente ligada al tipo de proceso y a consideraciones de orden económico.

Los drivers que están conveniados para el proyecto Guardián del ALBA son todos sincrónicos. Esto quiere decir que cuando se invoca una función de lectura de datos el hilo que la invoca queda bloqueado hasta tanto no se reciba la respuesta del dispositivo. Para lograr la mayor concurrencia posible, este modelo exige un hilo por cada dispositivo que pueda ser encuestado en paralelo.

Los modelos sincrónicos son simples y seguros ya que la lógica de programación es secuencial, sin embargo, consume muchos recursos del sistema cuando el número de dispositivos a atender es grande.

La situación polémica expuesta con anterioridad deviene en necesidad de respuesta al siguiente **problema científico**:

¿Cómo disminuir el consumo de recursos y atender eficientemente un número indeterminado de Controladores Lógicos Programables en un SCADA?

Para resolver este problema se plantea como **objeto de estudio** el Proceso de transmisión y recepción de datos sobre el protocolo de Red Industrial (EtherNet/IP) y como **campo de acción** el Proceso de transmisión y recepción de datos sobre el protocolo de EtherNet/IP con los Controladores Lógicos Programables (PLC) ControlLogix.

Este trabajo posee como objetivo general implementar la capa protocolo EtherNet/IP basado en un modelo asincrónico para la conexión del SCADA Nacional a los PLC ControlLogix.

La capa protocolo EtherNet/IP basado en un modelo asincrónico, al ahorrar considerablemente los recursos del sistema, impactará de manera favorable en el rendimiento general del SCADA Nacional de Venezuela.

Para el logro del objetivo de la investigación y teniendo en cuenta los elementos anteriores, se plantean las tareas científicas siguientes:

1. Estudiar la factibilidad de los modelos asíncronos basados en ASIO.
2. Analizar la posibilidad de usar las funciones que ofrece la librería Boost para crear los apuntadores a función necesarios para los eventos asíncronos.
3. Estudiar los métodos usados para encapsular datos del protocolo CIP sobre una red TCP/IP.
4. Diseñar la arquitectura jerárquica del protocolo EtherNet/IP en su variante asíncrona.
5. Implementar la arquitectura jerárquica del protocolo EtherNet/IP y someterla a pruebas de rendimiento.

Para el desarrollo de las tareas científicas se combinarán diferentes métodos y técnicas en la búsqueda y procesamiento de la información. Los que se utilizaron son:

Analítico – Sintético: Durante la definición de las funcionalidades básicas del Protocolo EtherNet/IP.

Inductivo – deductivo: En la revisión y justificación del modelo y la tecnología para desarrollar el Protocolo EtherNet/IP que se utilizará. Mediante el análisis de casos aislados se arribarán a proposiciones generales que luego se emplearán en la inferencia de la tecnología y modelo más adecuados a emplear en la implementación.

Análisis sistémico: Durante el desarrollo del diseño mediante la elaboración de los diagramas de clases.

Método de modelación: Durante el desarrollo del diseño mediante la elaboración de los diagramas de secuencia.

El presente documento está estructurado en cuatro capítulos:

En el **Capítulo 1** se describen todos los conceptos que desde el punto de vista teórico permitieron un mejor entendimiento de lo planteado en nuestra situación problemática. Definiciones y características de Protocolo, Modelo ISO/OSI, Interfaz Socket, fueron temas descritos exhaustivamente, siendo argumentadas cada una de ellas. Además de hacer un estudio de los modelos asincrónicos basados en la biblioteca ASIO, la cual da una claridad de las operaciones sincrónicas y asincrónicas ofrecidas por el mismo.

En el **Capítulo 2** se hace referencia a las tendencias y tecnologías existentes en la actualidad que se deben considerar para hacer la selección de aquellas que se van a utilizar en la implementación del protocolo EtherNet/IP.

En el **Capítulo 3** se describe el entorno de nuestro sistema, representado mediante diagramas. Se determinan las funcionalidades del sistema y se describen en detalle.

En el **Capítulo 4** se le da paso a la construcción de la solución propuesta determinando así, el tipo de arquitectura a emplear, y los diagramas de clases del diseño y diagramas de clases del sistema.

Capítulo 1. Fundamentación Teórica.

1.1 Introducción

A partir del desarrollo de redes se produjo un gran desorden en muchos sentidos: un enorme crecimiento en la cantidad y el tamaño de las redes. A medida que las empresas tomaron conciencia de las ventajas de usar tecnología de redes, estas se agregaban o expandían a casi la misma velocidad a la que se introducían las nuevas tecnologías de red. (3) Para mediados de la década de 1980, éstas empresas comenzaron a sufrir las consecuencias de la rápida expansión. De la misma forma en que las personas que no hablan un mismo idioma tienen dificultades para comunicarse, las redes que utilizaban diferentes especificaciones e implementaciones tenían dificultades para intercambiar información.

Para enfrentar el problema de incompatibilidad de redes, la Organización Internacional para la Estandarización (ISO) investigó modelos de redes a fin de encontrar un conjunto de reglas aplicables de forma general a todas las redes. Con base en esta investigación, la ISO desarrolló un modelo de red que ayuda a los fabricantes a crear redes que sean compatibles con otras redes: el Modelo de Referencia ISO/OSI para el empleo de procedimientos de intercambio de información entre redes del mismo tipo o tipos diferentes, de modo que la comunicación se establezca tan fácilmente por una combinación de redes como por una sola red. (4)

Los protocolos TCP/IP son la base de Internet, y sirve para enlazar computadoras que utilizan diferentes sistemas operativos, incluyendo PC, minicomputadoras y computadoras centrales sobre redes de área local (LAN) y área extensa (WAN). Hoy en día TCP/IP es el más difundido en Internet, aunque no define un API (interfaz con los programas de aplicación), los estándares sugieren cuál es la funcionalidad esperada y es cuando se desea crear una interfaz para permitir que las aplicaciones pudieran acceder a los servicios que brinda el software TCP/IP. (3) El resultado ha sido la Interfaz Socket, la cual es interfaz de comunicación o un sistema de comunicación entre ordenadores.

La API de sockets es amplia y extendida por lo que con el estudio de la ASIO darán paso algunas de las operaciones asíncronas, permitiendo así una comunicación simultánea.

1.2 Protocolo.

Para la comunicación entre dos entidades situadas en sistemas diferentes es necesario la definición y utilización de un protocolo. En general, una entidad es cualquier cosa capaz de enviar y recibir información, y un sistema es un objeto físico que contiene a una o más entidades. Para que dos entidades se comuniquen con éxito, se requiere que hablen el mismo idioma. Qué se comunica, cómo se comunica, y cuándo se comunica debe seguir una serie de convenciones mutuamente aceptadas por las entidades involucradas. Este conjunto de convenios se denominan protocolos, que se pueden

definir como el conjunto de reglas que gobiernan el intercambio de datos entre dos entidades. Los puntos clave que definen un protocolo son:

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

Las funciones de un protocolo se pueden agrupar en:

Encapsulamiento: Añadir datos de información de control. Los datos se aceptan o generan por una entidad y se encapsulan en la unidad de datos de protocolo (PDU) junto con la información de control.

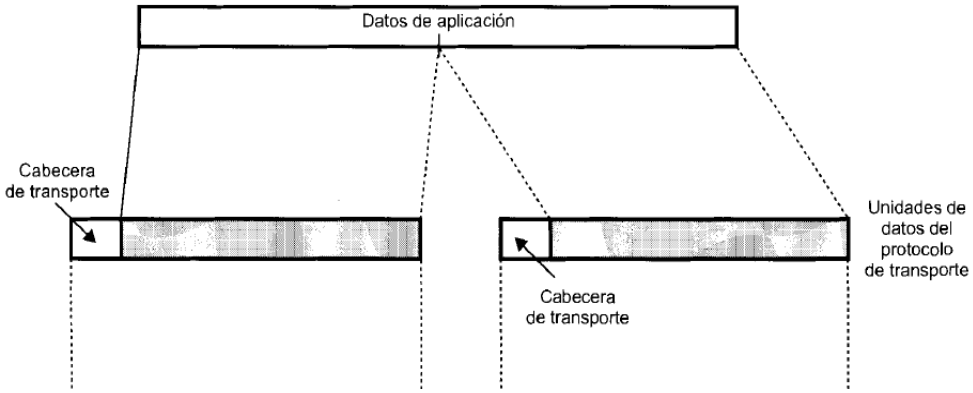


Figura 1. Encapsulamiento de un dato.

Segmentación y ensamblado: Cada Protocolo puede dividir los datos de un nivel superior en bloques más pequeños. Denominaremos unidad de datos del protocolo (PDU, *Protocol Data Unit*) al bloque de datos a intercambiar entre dos entidades. El procedimiento contrario a la segmentación se denomina ensamblado. Los datos tendrán que ensamblarse recuperando el formato de los mensajes originales para ser entregados a la entidad de destino.

Control de la conexión: La transferencia de datos puede ser no orientada a la conexión, cada PDU se tratará de forma independiente de los PDUs recibidos con anterioridad. La orientada a la conexión se establece una asociación lógica, o conexión, entre dos entidades.

Entrega en orden:

- Control del Flujo: El control de flujo es una operación realizada por la entidad receptora para limitar la velocidad y la cantidad de datos a enviar.
- Control de Errores: Recuperar pérdidas o deterioro de los datos.
- Direccionamiento: La dirección del nivel de red se utiliza para encaminar los PDUs a través de la red hasta el sistema destino.
- La multiplexación entre capas: Puede realizarse de dos formas distintas. La multiplexación ascendente o la descendente.
    - La multiplexación ascendente consiste en que varias conexiones del nivel superior compartan, o se multiplexen sobre una única conexión del nivel inferior.

- La multiplexación descendente consiste en establecer una única conexión del nivel superior utilizando varias conexiones del nivel inferior.

Servicios de transmisión:

- Prioridad.
- Calidad del Servicio.
- Seguridad.

No todos los protocolos deben proporcionar éstas funciones, pues en ocasiones implicaría una duplicación innecesaria de las mismas.

1.3 Modelo de Referencia OSI.

En 1977 la Organización Internacional de Estandarización (ISO, siglas en inglés) estableció un subcomité encargado de diseñar una arquitectura de comunicación. El resultado fue el modelo de referencia para la Interconexión de Sistemas Abiertos OSI, adoptado en los años 80, que establece bases que permiten conectar sistemas abiertos para procesamiento de aplicaciones distribuidas. Se trata de un marco de referencia para definir estándares que permitan comunicar ordenadores heterogéneos. (5)

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

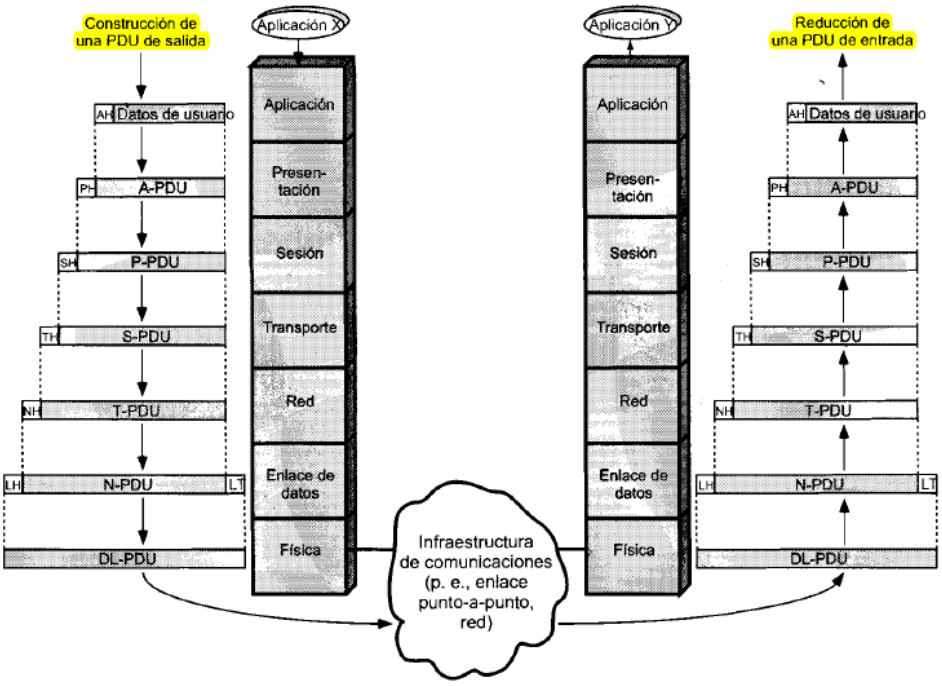


Figura 2. Intercambio de datos en el modelos OSI.

1.3.1 La arquitectura del modelo OSI (6)

Capa Física:

La capa física se encarga de la interfaz física entre los dispositivos. La capa física tiene cuatro características importantes:

Mecánicas: relacionadas con las propiedades físicas de la interfaz y con el medio de transmisión.

Eléctricas: especifican como se representan los bits, así como su velocidad de transmisión.

Funcionales: especifican las funciones que realizan cada uno de los circuitos de la interfaz física entre el sistema y el medio de transmisión.

De procedimiento: especifican la secuencia de eventos que se llevan a cabo en el intercambio del flujo de bits a través del medio físico.

Capa de enlace a datos:

Hace que el enlace físico sea seguro, además proporciona los medios para activar, mantener y desactivar el enlace.

Capa de Red:

Realiza la transferencia de información entre sistemas finales a través de algún tipo de red de comunicación.

Capa de Transporte:

La capa de transporte proporciona un mecanismo para intercambiar datos entre sistemas finales. El servicio de transporte orientado a conexión asegura que los datos se entregan libres de errores, en orden y sin pérdidas ni duplicaciones. La capa de transporte puede estar involucrada en la optimización del uso de los servicios de red, proporcionando la calidad del servicio solicitado.

Capa de Sesión:

Proporciona los mecanismos para controlar el diálogo entre las aplicaciones de los sistemas finales. La capa de sesión proporciona los siguientes servicios:

Control del diálogo: Este puede ser simultáneo en los dos sentidos (full dúplex) o alternado en ambos sentidos (half dúplex).

Agrupamiento: el flujo de datos se puede marcar para definir grupos de datos.

Recuperación: la capa de sesión puede proporcionar un procedimiento de puntos de comprobación, de forma que si ocurre algún tipo de fallo entre puntos de comprobación, la entidad de sesión puede retransmitir todos los datos desde el último punto de comprobación.

Capa de Presentación:

Define el formato de los datos que se van a intercambiar entre las aplicaciones y ofrece a los programas de aplicación un conjunto de servicios de transformación de datos.

Capa de Aplicación:

La capa de aplicación proporciona a los programas de aplicación un medio para que acceda al entorno OSI. Esta capa incluye a las funciones de administración y en general, a los mecanismos necesarios en la implementación de las aplicaciones distribuidas.

1.4 Arquitectura TCP/IP.

En 1969 la Agencia de Proyectos de Investigación Avanzada (*Defense Advanced Research Projects Agency* o DARPA) del Ejército de los EEUU desarrolla la *ARPAnet*. Se inició entonces una búsqueda de un conjunto de protocolos más fiables para la misma. Dicha búsqueda finalizó, a mediados de los 70, con el desarrollo de TCP/IP.

En 1983, TCP/IP se integró en la versión 4.2 del sistema operativo UNIX de Berkeley y la integración en versiones comerciales de UNIX. (7)

1.4.1 La arquitectura TCP/IP (6)

TCP/IP se estructura en 4 capas verticales. Las capas superiores son los más cercanos al usuario y tratan con datos más abstractos, dejando a los niveles más bajos la labor de traducir los datos de forma que sean físicamente manipulables.

Capa de aplicación: proporciona la comunicación entre procesos o aplicaciones de computadores separados.

Capa de transporte: proporciona un servicio de transferencia de datos extremo-a-extremo. Esta capa puede incluir mecanismos de seguridad.

Capa Internet: relacionada con el encaminamiento de los datos del computador origen al destino a través de una o más redes conectadas por dispositivos de encaminamiento.

Capa Interfaz de red + Física: relacionada con la interfaz lógica entre un sistema final y una subred. Define las características del medio de transmisión, la tasa de señalización y el esquema de codificación de las señales.

1.5 Protocolos TCP y UDP

Para construir software cliente-servidor se necesita escoger entre dos filosofías de trabajo: orientada a la conexión y no orientada a la conexión. Estas dos filosofías se logran trabajando con los protocolos TCP y UDP de la arquitectura TCP/IP.

1.5.1 Protocolo UDP

El protocolo de datagrama de usuario (UDP, en inglés: User Datagram Protocol) proporciona el mecanismo primario que utilizan los programas de aplicación para enviar datagramas a otros programas de aplicación. El UDP proporciona puertos de protocolo utilizados para distinguir entre muchos programas que se ejecutan en la misma máquina. Esto es, además de los datos, cada mensaje UDP contiene tanto el número de puerto de destino como el número de puerto de origen,

haciendo posible que el software UDP en el destino entregue el mensaje al receptor correcto y que este envíe una respuesta.

UDP utiliza el Protocolo de Internet (IP) como protocolo subyacente para transportar un mensaje de una máquina a otra y al igual que este brinda servicio de entrega:

- No orientado a conexión: No emplea acuses de recibo para asegurarse de que llegan mensajes, no ordena los mensajes entrantes, ni proporciona retroalimentación para controlar la velocidad a la que fluye la información entre las máquinas. Por lo tanto, los mensajes UDP se pueden perder, duplicar o llegar sin orden.
- No fiable: Un programa de aplicación que utiliza el UDP acepta toda la responsabilidad por el manejo de problemas de confiabilidad, incluyendo la pérdida, duplicación y retraso de los mensajes, la entrega fuera de orden y la pérdida de conectividad.

1.5.2 Protocolo TCP

El protocolo de control de transmisión (TCP, en inglés: Transmission Control Protocol) constituye el segundo servicio más conocido y empleado de los protocolos TCP/IP y aunque está basado en el protocolo IP a diferencia de éste brinda servicio de entrega:

- Orientado a conexión: Es necesario establecer una conexión previa entre las dos máquinas antes de poder transmitir ningún dato. A través de esta conexión los datos llegarán siempre a la aplicación destino de forma ordenada y sin duplicados. Finalmente, es necesario cerrar la conexión.
- Fiable: La información que envía el emisor llega de forma correcta al destino. De esta forma, las aplicaciones que lo utilicen no tienen que preocuparse de la integridad de la información: dan por hecho que todo lo que reciben es correcto.

1.6 La interfaz socket

En el año 1980 la agencia ARPA (*Advanced Research Projects Agency*, Agencia para la Investigación de Proyectos Avanzados) fundó un grupo de investigación en la Universidad de Berkeley (California) para llevar a cabo la implementación del software TCP/IP dentro del propio núcleo del sistema operativo UNIX. Como parte del proyecto, los diseñadores crearon una interfaz para permitir que las aplicaciones pudieran acceder a los servicios que brinda el software TCP/IP. Decidieron hacer uso de las llamadas al sistema UNIX existentes siempre que fuera posible y añadir nuevas llamadas para dar soporte a las funciones TCP/IP que no se podían implementar adecuadamente con el conjunto de llamadas disponibles. Al fruto de este proyecto se le conoce como la interfaz socket, y el sistema operativo resultante se dio a conocer como UNIX Berckeley o BSD UNIX. La interfaz socket desarrollada en Berkeley ha sido tan ampliamente aceptada por los grandes distribuidores de computadoras que se ha convertido en un estándar de negocio.

Las llamadas al sistema de Entradas/Salidas en UNIX se basan en el proceso de *open-read-write-close* (abrir-leer-escribir-cerrar), y esto se utiliza tanto con archivos como con dispositivos hardware. En este tipo de comunicación se basó el diseño de la interface de sockets, con lo que para comunicarse con una red TCP/IP, se abre primero una conexión con la red, se leen y escriben datos a través de ella y una vez terminados los procesos se cierra la conexión.

Un socket es un punto de comunicación por el cual un proceso puede emitir o recibir información. En el interior de un proceso, un socket se identifica por un descriptor de la misma naturaleza que los que identifican los archivos, al igual que todos los procesos del sistema UNIX de Berkeley. (8) La diferencia principal entre los descriptores de archivos y los descriptores de socket es que el sistema operativo enlaza un descriptor de un archivo a un archivo o dispositivo específico cuando la aplicación llama a *open,* pero puede crear sockets sin enlazarlos a direcciones de destinos específicos. La aplicación puede elegir entre abastecer una dirección de destino cada vez que utliza el socket ó evadir la especificación de destino repetidamente.

Los sockets hacen ejecuciones exactamente iguales a los archivos o dispositivos de UNIX, de manera que pueden utilizarse con operaciones tradicionales como *read* y *write.* Por ejemplo, una vez que un programa de aplicación puede hacer uso de write para mandar un flujo de datos a través de la conexión y uso del *read* para recibirlo.

La comunicación mediante sockets es una interfaz (o servicio) con la capa de transporte (nivel 4) de la jerarquía OSI. La filosofía de la división por capas de un sistema es encapsular, dentro de cada una de ellas, detalles que conciernen sólo a cada capa, y presentársela al usuario de tal forma que este pueda trabajar con ella sin necesidad de conocer sus detalles de implementación.

Entre las principales características de la Interfaz Socket se destaca:

1. Interfaz estándar.
2. Portabilidad de aplicaciones, debido que se trabaja con las mismas primitivas con los mismos parámetros, los tipos y las estructuras de datos son idénticas, con algunas excepciones.
3. Define una librería de primitivas.

1.6.1 Tipos de sockets (9)

En cuanto a los tipos de sockets disponibles, se pueden considerar:

- SOCK_STREAM: Los sockets de éste tipo permiten comunicaciones fiables en modo conectado (propiedades a, b, c y d) y eventualmente autorizan, según el protocolo aplicado los mensajes fuera de flujo (propiedad f). El protocolo subyacente en el dominio Internet es TCP. Se establece un circuito virtual realizando una búsqueda de enlaces libres que unan los dos ordenadores a conectar. Una vez establecida la conexión, se puede proceder al envío secuencial de los datos, ya que la conexión es permanente. Son *streams* de bytes full-dúplex. Un socket *stream* debe estar en estado conectado antes de que se envíe o reciba en él.

- SOCK_DGRAM: Corresponde a los sockets destinados a la comunicación en modo no conectado para el envío de datagramas de tamaño limitado. Las comunicaciones correspondientes tienen la propiedad e. En el dominio Internet, el protocolo subyacente es el UDP. Los datagramas no trabajan con conexiones permanentes. La transmisión por los datagramas es a nivel de paquetes, donde cada paquete puede seguir una ruta distinta, no garantizándose una recepción secuencial de la información.
- SOCK_RAW: Permite el acceso a los protocolos de más bajo nivel (por ejemplo, el protocolo IP en el dominio Internet).
- SOCK_SEQPACKET: Corresponde a las comunicaciones que poseen las propiedades a, b, c, d y e. Estas comunicaciones se encuentran en el dominio XNS.

a) Cada tipo de socket va a definir una serie de propiedades en función de las comunicaciones en las cuales está implicado.
b) La fiabilidad de la transmisión. Ningún dato transmitido se pierde.
c) La conservación del orden de los datos. Los datos llegan en el orden en el que han sido emitidos.
d) La no duplicación de datos. Sólo llega a destino un ejemplar de cada dato emitido.
e) La comunicación en modo conectado. Se establece una conexión entre dos puntos antes del principio de la comunicación (es decir, se establece un circuito virtual). A partir de entonces, una emisión desde un extremo está implícitamente destinada al otro extremo conectado.
f) La conservación de los límites de los mensajes. Los límites de los mensajes emitidos se pueden encontrar en el destino.
g) El envío de mensajes (urgentes). Corresponde a la posibilidad de enviar datos fuera del flujo normal, y por consecuencia accesibles inmediatamente.

1.7 Surgimiento y desarrollo de las redes Industriales

En la década del 60 del pasado siglo, los fabricantes de Sistemas Automatizados comenzaron a desarrollar redes para el manejo de datos. Desde 1972 y por más de 10 años se empleó en la industria de procesos, de manera casi absoluta, un estándar de comunicación analógico de 4 -20 miliamperes, para establecer la conexión punto a punto entre los dispositivos de campo y los de control. Si bien es cierto que por su relativa simpleza ha sobrevivido hasta los días de hoy, también lo es que su capacidad de transmisión-recepción ha estado siempre limitada a una sola magnitud y que las inversiones en su instalación, por concepto de cableado, son elevadas. (10)

En la década del 80 surgieron los protocolos de comunicación digital y bidireccional (buses de campo), los cuales fueron diseñados para aprovechar las facilidades de la instrumentación inteligente. (10)

1.7.1 HART

Es el primer intento de enviar una señal digital. Se aprovechó los mismos cables utilizados en el lazo 4-20 miliamperes y de esa forma aprovechar la estructura del cableado ya instalada. El éxito de HART fue que permitía hacerlo coexistiendo incluso con dicha señal. (11)

1.7.2 RS-232, RS-485 y RS-422 (11)

Estándares de comunicación serie que por su gran versatilidad, robustez mecánica e inmunidad al ruido, han conseguido una enorme difusión y popularidad en la industria.

Definen las interfaces físicas, funcionales y lógicas para comunicaciones de datos digitales punto a punto o punto-multipunto. No especifica un protocolo en particular dado que esto estará facilitado por las capas superiores.

RS-232 es una interfaz desbalanceada punto a punto definida por la EIA/TIA. Soporta distancias hasta 15 m dependiendo de la velocidad. Su enorme popularidad se debe a haber sido incorporada desde las primeras PC's, con el objeto de interconectar la misma con un módem y así establecer comunicaciones vía línea telefónica. Hoy día resulta la interfaz por defecto para acceder a la programación de PLCs, controladores, variadores de frecuencia desde una PC o *notebook*.

RS-485 es un bus balanceado (diferencial) de 2 hilos, con una impedancia característica de 120 Ohms, *half dúplex*, al cual pueden conectarse hasta 32 Transmisores y 32 Receptores. La distancia máxima permitida también depende de la velocidad, típicamente se establece como límite 1200m sin repetidores.

RS-422 posee similares características pero es Simplex, con 1 solo transmisor y múltiples receptores. Por ende, para constituir un bus *full-duplex* se debe realizar un tendido de 4 hilos – 2 para Transmisión y 2 para Recepción.

1.8 Ethernet

En 1972 comenzó el desarrollo de una tecnología de redes conocida como Ethernet Experimental. El sistema Ethernet desarrollado, conocido en ese entonces como red *ALTO ALOHA*, fue la primera red de área local (LAN) para computadoras personales (PCs). Esta red funcionó por primera vez en mayo de 1973 a una velocidad de 2.94Mb/s con topología BUS. Empleo CSMA/CD (Acceso Múltiple con detección de colisiones). La idea de este protocolo MAC era muy simple, las estaciones antes de transmitir deberían detectar si el canal ya estaba en uso en cuyo caso esperarían a que la estación activa terminara. Además, cada estación mientras transmitiera estaría constantemente vigilando el medio físico por si se producía alguna colisión, en cuyo caso se pararía y retransmitiría más tarde. En el caso de Ethernet se le agregó un mecanismo denominado retroceso exponencial binario para reducir gradualmente la "agresividad" del emisor. Estas posibles colisiones se elevaban a medida que crecía el número de nodos en la red lo que lo hacía para ambientes industriales inseguro y poco confiable.

Las especificaciones formales de Ethernet de 10 Mb/s fueron desarrolladas en conjunto por las corporaciones Xerox, Digital (DEC) e Intel, y se publicó en el año 1980. Estas especificaciones son conocidas como el estándar DEC-Intel-Xerox (DIX), el libro azul de Ethernet. Este documento hizo de Ethernet experimental, operando a 10 Mb/s, un estándar abierto.

La tecnología Ethernet fue adoptada para su estandarización por el comité de redes locales (LAN) de la IEEE como IEEE 802.3. El estándar IEEE 802.3 fue publicado por primera vez en 1985.

El estándar IEEE 802.3 provee un sistema tipo Ethernet basado, pero no idéntico, al estándar DIX original. El nombre correcto para esta tecnología es IEEE 802.3 CSMA/CD, pero casi siempre es referido como Ethernet.

Ethernet continuó evolucionando en respuesta a los cambios en tecnología y necesidades de los usuarios. Desde 1985, el estándar IEEE 802.3 se actualizó para incluir nuevas tecnologías (ver tabla 1).

Por ejemplo, el estándar 10BASE-T fue aprobado en 1990, el estándar 100BASE-T fue aprobado en 1995 y Gigabit Ethernet sobre fibra fue aprobado en 1998.

Las tecnologías Ethernet que existen se diferencian en estos conceptos:

Velocidad de transmisión: Velocidad a la que transmite la tecnología.

Tipo de cable: Tecnología del nivel físico que usa la tecnología.

Longitud máxima: Distancia máxima que puede haber entre dos nodos adyacentes (sin estaciones repetidoras).

Topología: Determina la forma física de la red. Bus si se usan conectores T (hoy sólo usados con las tecnologías más antiguas) y estrella si se usan Hubs (estrella de difusión) o Switches (estrella conmutada).

A continuación se especifican los anteriores conceptos en las tecnologías más importantes:

Tecnología	Velocidad de Transmisión	Tipo de cable	Distancia Máxima	Topología
10Base2	10 Mbps	Coaxial	185 m	Bus (Conector T)
10BaseT	10 Mbps	Par Trenzado	100 m	Estrella (Hub o Switch)
10BaseF	10 Mbps	Fibra óptica	2000 m	Estrella (Hub o Switch)
100BaseT4	100 Mbps	Par Trenzado (categoría 3UTP)	100 m	Estrella. Half Duplex(hub) y Full Duplex(switch)
100BaseTX	100 Mbps	Par Trenzado (categoría 5UTP)	100 m	Estrella. Half Duplex(hub) y Full Duplex(switch)

100BaseFX	100 Mbps	Fibra óptica	2000 m	No permite el uso de hubs
1000BaseT	1000 Mbps	4 pares trenzado (categoría 5UTP)	100 m	Estrella. Full Dúplex (switch)
1000BaseSX	1000 Mbps	Fibra óptica (multimodo)	550 m	Estrella. Full Dúplex (switch)
1000BaseLX	1000 Mbps	Fibra óptica (monomodo)	5000 m	Estrella. Full Dúplex (switch)

Tabla 1. Tecnologías Ethernet (12)

1.8.1 Ethernet Industrial.

Ethernet es el estándar universal diseñado por *Xerox Corporation* y registrada posteriormente por Digital e Intel que trabaja en los niveles 1, 2 y 3 del modelo OSI. Su naturaleza inicial no la hacía apta para el ámbito industrial (13) .

Sin embargo, en los últimos años han sido desarrollados medios físicos, tecnologías y nuevos protocolos para adaptar su principio de funcionamiento, especialmente gracias a la mayor difusión de los switches, los cuales le confirieron una naturaleza "determinística", en vez de la "estadística" de los *hubs*, lo cual posibilitó su utilización industrial.

Varios fabricantes ya lo han adoptado en todo o en parte como plataforma de funcionamiento de sus redes. Los ejemplos más característicos de esto son Modbus/TCP de Schneider, Ethernet/IP de Rockwell, Profinet y Foundation Fieldbus HSE.

Prácticamente hay una total convergencia hacia Ethernet, ya que todos los fabricantes de Redes Industriales lo están adoptando como capa de Información de sus redes. Ethernet Industrial, hoy en día ha crecido de tal manera que tiene una evolución propia. (11)

1.8.2 EtherNet/IP

EtherNet/IP es un protocolo de red en niveles, apropiado al ambiente industrial (ver fig. 3). Es el producto acabado de cuatro organizaciones que aunaron esfuerzos en su desarrollo y divulgación para aplicaciones de automatización industrial: La Open DeviceNet Vendor Association (ODVA), la Industrial Open EtherNet Association (IOANA), la Control Net International (CI) y la Industrial EtherNet Association (IEA). Ese cometido común demuestra hasta qué punto EtherNet/IP puede significar todo un estándar tallado a la perfección para un vasto número de dispositivos de automatización. Éstas mismas organizaciones se están esforzando para atender a las demandas de conectividad física que el ambiente severo de pie de fábrica exige. (14)

EtherNet/IP™, se usa para configurar, acceder y controlar dispositivos de automatización industrial. Está basado en los protocolos estándar TCP/IP, y en el transporte EtherNet. EtherNet/IP clasifica los nodos de acuerdo a tipos de dispositivos preestablecidos, con sus actuaciones específicas. Da servicio al Protocolo Industrial Común (CIP- *Common Industrial Protocol*) tal como lo

hacen DeviceNet™ y ControlNet™. De esta manera ofrece un sistema integrado completo desde la planta industrial hasta la red empresarial.

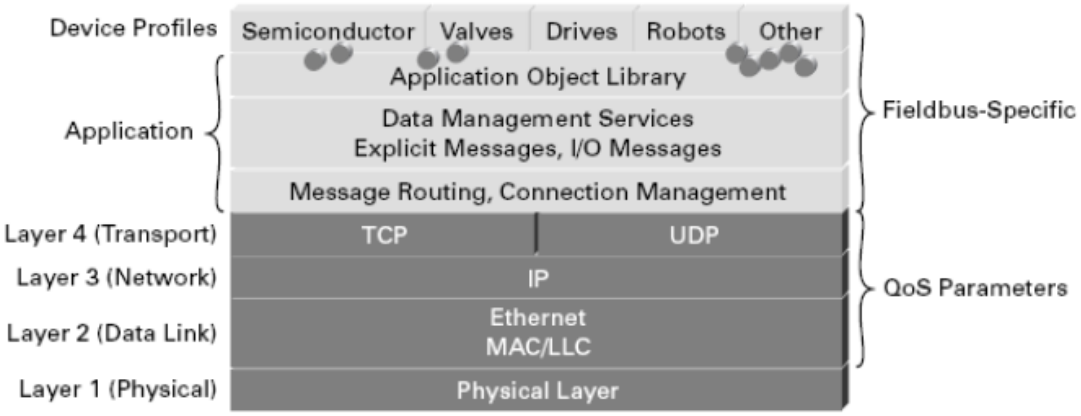


Figura 3. CIP y el modelo TCP.

Fue introducido en 2001 y tiene más de 1.125 millones de nodos instalados. EtherNet/IP y Modbus/TCP son los dos protocolos para Ethernet industrial más extendidos, representando más del 50% de cuota del mercado mundial, según el estudio de mercado más reciente de ARC *Advisory Group.* (15)

Está diseñado a partir de un estándar ampliamente implementado y utilizado en DeviceNet y ControlNet, denominado Protocolo de Control e Información (CIP). Este estándar, organiza 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. (15)

1.8.3 Protocolo Industrial Común. (CIP- *Common Industrial Protocol*)

El fundamento de la integración de verdaderas redes empresariales radica en la capa de aplicaciones. El protocolo común industrial se ha diseñado con este propósito. Esta basado en una sola plataforma independiente de los medios y protocolos de comunicación. Permite reducir los costos de ingeniería e instalación optimizando las utilidades.

Este protocolo abarca una amplia gama de mensajes y servicios para muchas aplicaciones de manufactura (control, seguridad, sincronización, movimiento, configuración y verificación). Permite a los usuarios integrar estas aplicaciones de manufactura con las redes empresariales y con Internet. Esto quiere decir, que se puede usar una arquitectura unificada de comunicación en las empresas, beneficiándolas con el uso de las redes abiertas.

El modelo del CIP es, en las capas superiores, un modelo estrictamente orientado a objetos. Cada objeto tiene atributos (datos), servicios (instrucciones), conexiones y comportamientos (relación entre los valores de los atributos y los servicios). En los objetos se implantan las funciones básicas de

- comunicaciones,

- transferencia de archivos,
- control de dispositivos.

1.8.3.1 Modelo de objetos CIP.

CIP usa un modelo de objeto abstracto para describir:

- La colección de servicios disponibles;
- El comportamiento externo y visible de un nodo CIP;
- Una manera común por la cual la información dentro de productos CIP se accedida e intercambiada.

Cada nodo CIP es modelado como una colección de objetos. Un objeto provee una representación abstracta de un componente en particular dentro de un producto. Cualquier cosa no descrita como un objeto no será visible a través de CIP. Los objetos de CIP están estructurados en clases, instancias y atributos.

Una clase es una colección de objetos que todos cumplen con características de un componente del sistema. Una instancia es la representación real de un objeto particular dentro de una clase. Cada instancia de una clase tiene los mismos atributos pero estos toman valores diferentes para cada una.

Además de los atributos de instancia, una clase del objeto también puede tener atributos de clase. Estos atributos describen propiedades de la clase. Además de esto, las instancias del objeto y la clase misma exhiben un comportamiento y dejan ciertos servicios que puedan aplicarse a los atributos, a las instancias o a todo.

Los objetos y sus componentes son direccionados por un esquema de direccionamiento que consiste en:

La dirección del nodo (*Node Address*): Es un valor entero asignado a cada nodo en la red de CIP. En EtherNet/IP es la dirección IP.

El identificador de clase (*Class ID*): Un valor entero de identificación, asignado a cada clase de objeto accesible desde la red.

El identificador de instancia (*Instance ID*): Un valor entero de identificación, asignado a una instancia del objeto que lo identifica entre todas las instancias de la misma clase.

El identificador de atributo (*Attribute ID*): Un valor entero de identificación, asignado a una clase o un atributo de instancia.

El Código de Servicio (Service Code): Un valor entero de identificación que denota una petición de acción que se dirigió en una instancia particular del objeto o una clase del objeto.

En la figura 4 se muestra un ejemplo de esquema de direccionamiento:

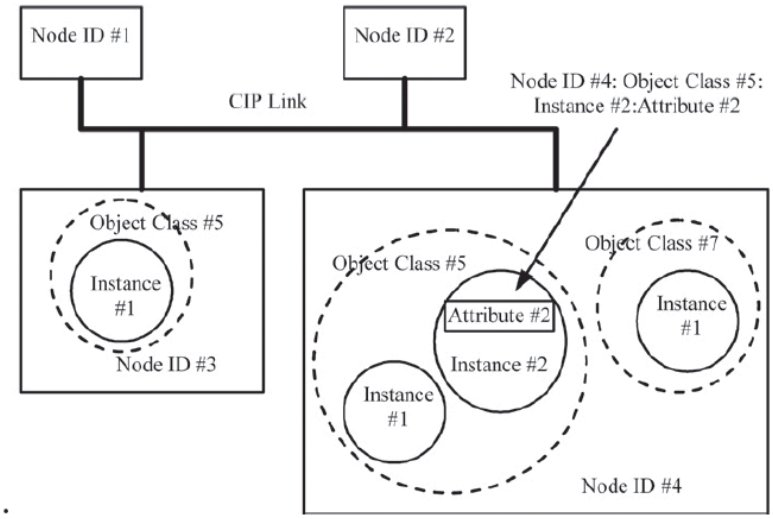


Figura 4. Ejemplo de un esquema de direccionamiento

1.8.3.2 Código de servicio

Son usados para definir la acción que se hará cuando un objeto o parte de un objeto, direccionado a través de un mensaje explícito, de su respuesta. Aparte de los simples servicios de leer y escribir, CIP define una colección de servicios que pueden ser usados en toda la red CIP y para diferentes objetos. Además los objetos pueden definir sus propios servicios.

1.8.3.3 Envió de mensajes.

CIP es un protocolo basado en conexiones. Una conexión CIP provee un camino entre objetos. Cuando una conexión es establecida, las transmisiones asociadas con la conexión se le asignada un ID de Conexión. CIP define un proceso que es el encargado de establecer las conexiones entre los dispositivos que no han estado conectados. Esto se hace a través del manejador de mensajes desconectados (Unconnected Message Manager, UCMM).

Para establecer una conexión CIP se envía un mensaje de petición de servicio UCMM, *Forward_Open*. Una petición *Forward_Open* contiene toda la información necesaria para crear una conexión entre el dispositivo solicita la conexión y el que recibe la solicitud, además de una petición para crearla entre el que la recibe y el que la origina. En particular una petición *Forward_Open* contiene:

- El tiempo que estará viva la conexión sin mandar un nuevo mensaje.
- El ID de conexión desde el que crea la conexión hacia el que recibe la petición.
- El ID de conexión del que recibe la conexión hacia el que solicitó crearla.
- Información de la identidad del dispositivo.
- El tamaño máximo del mensaje que se puede enviar.
- El camino de conexión para los datos que se envíen al objeto en el nodo.

Todas las conexiones en CIP se pueden clasificar como conexiones Entrada/Salida o conexiones explícitas.

Las conexiones Entrada/Salida: Provee caminos de comunicación dedicados entre una aplicación productora y uno o más aplicaciones receptoras. Proceso que es a menudo llamado envío de mensajes implícito. Estos mensajes a menudo se envían en *multicast*.

Las conexiones explícitas de envío de mensajes proveen caminos genéricos de comunicación, multipropósito entre dos dispositivos. Los mensajes explícitos proveen petición-respuesta típica de las comunicaciones en la red. Estos mensajes son enviados punto a punto.

La familia de protocolos CIP contiene una colección de objetos. Esta colección puede estar subdividida en tres tipos:

- De uso general;
- Específico para la aplicación;
- Específico para la red.

Los objetos más comunes en CIP son:

Assembly, Connection, Connection Configuration, Connection Manager, Identity, Message Router, Parameter, Port.

La figura 5 muestra el modelo de objeto de un dispositivo:

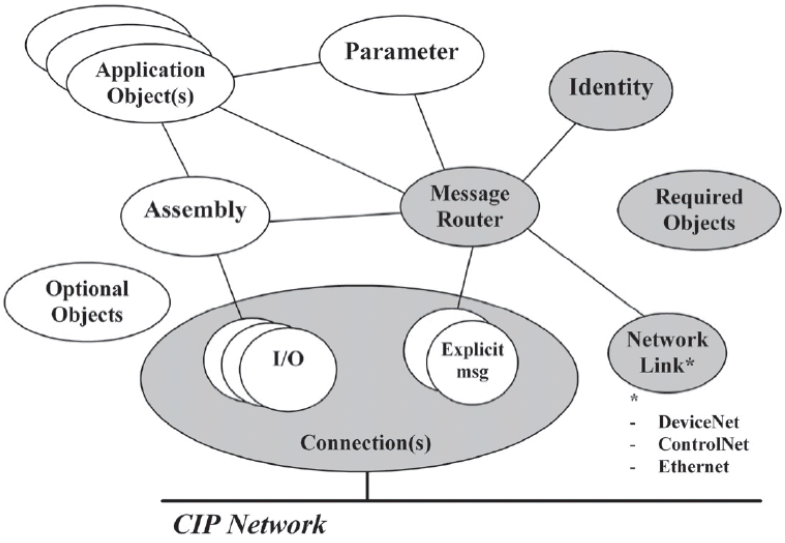


Figura 5. El modelo de objeto típico para dispositivos.

1.8.3.4 Ruteo.

CIP define mecanismos que permiten la transmisión de mensajes a través de múltiples redes, proveen los dispositivos de ruteo entre las diversas redes. El mensaje será reenviado de *router* a *router* hasta logre alcanzar su nodo de destino.

1.8.3.5 Protocolos que se han derivado del CIP

- ***CIPSync*:** que permite sincronizar las aplicaciones en sistemas distribuidos usando cronómetros de precisión en todos los dispositivos, sincronizados en tiempo real con un reloj maestro.
- ***CIPMotion*:** que usa esta sincronización para monitorear y controlar movimientos.

- ***CIPSafety*:** que permite el intercambio de mensajes relacionados con la seguridad. Estos mensajes se rigen por mecanismos adicionales de sincronización e integridad que garantizan la detección de fallas obedeciendo los estándares. Si algo falla el sistema se para en un estado seguro. Este protocolo abierto mejora con el funcionamiento de las redes que funcionan en tiempo real a nivel industrial basadas en CIP. Ofrece una colección de servicios de seguridad integrados que se apoyan en la pila de comunicación estándar de CIP para transportar datos de un origen a un destino. (16)

1.8.4 Protocolos que utilizan CIP™

EtherNet/IP — Red basada en CIP, usando Ethernet estándar sin modificación, para comunicarse desde las E/S hasta el área de tecnología de la información.

ControlNet — Red de control basada en CIP que proporciona transmisión de alta velocidad de datos de E/S en tiempo real con tiempos críticos y envío de mensajes, incluida la carga y descarga de datos de programación y configuración, y el envío de mensajes entre dispositivos similares a través de un solo vínculo físico.

DeviceNet — Red a nivel de dispositivos basada en CIP y una de las redes líderes del mundo para automatización industrial, DeviceNet ofrece un manejo de datos robusto y eficiente. Los usuarios se benefician de las políticas de pruebas de conformidad con las normas de ODVA, las cuales aseguran que los productos tienen capacidad de interoperación, lo que significa que los usuarios pueden combinar productos de una variedad de suministradores e integrarlos de manera transparente.

1.9 Librerías que implementan CIP.

Hoy en día existen muchas librerías que implementan el protocolo CIP, entre las libres, la que más se destaca es TuxEip. En las siguientes líneas se exponen las principales características de las librerías tomadas en consideración como una posible solución al problema.

TuxEip es una librería de comunicación que facilita el intercambio de información con los Controladores Allen Bradley. Esta librería permite a los programadores crear aplicaciones que se comuniquen con los siguientes Controladores: ControlLogix, PLC5, SLC500, Micrologix, FlexLogix. Ésta proporciona funciones para interactuar con los objetos definidos por el protocolo CIP y además, permite agregar fácilmente sus propias funciones. Fue escrita en C y se rige bajo la licencia GPL. Está escrita para el sistema operativo GNU\Linux. (17)

PLCIO es una librería diseñada para leer y escribir datos en un PLC. Esta permite a los programadores acceder a las direcciones de memoria del PLC mediante el nombre de los *tag* para diferentes tipos de datos, independientemente de la arquitectura de la computadora en la que se encuentra y de si está conectado directamente o remoto. Incluye un ejemplo donde se programa una interfaz de CGI para acceder desde la Web a un PLC. La librería funciona en Linux, HP-UX, Solaris y QNX. PLCIO soporta los protocolos Ethernet de la Allen-Bradley PLC-5, SLC 500, ControlLogix,

CompactLogix, y MicroLogix PLC, el PLC Modicon Quantum, la Wago 750-842 PLC, y el Siemens *Step* 7 300 - y 400-serie CPUs. También apoya el Siemens Step 5 AS511 protocolo en serie o bien a través de Ethernet utilizando el INAT Echolink. (18)

RSLinx™ es un servidor de comunicación completo que proporciona la conectividad de los dispositivos de la planta con los de campo para una gran variedad de dispositivos de *Rockwell* tales como *RSLogix*™ 5/500/5000. Además, proporciona varias interfaces abiertas para tercera persona, los paquetes de la colección de datos y de análisis. *RSLinx* puede soportar múltiples dispositivos simultáneamente, comunicando a una variedad inmensa de dispositivos en muchas redes diferentes.

Con *RSLinx se* puede comunicar a cualquier lugar desde cualquier lugar. Este provee una interfaz gráfica amistosa para navegar a través de la red. Esto incluye enrutamiento sobre una red de Ethernet de oficina a través de un *Gateway* de *ControlLogix*® para obtener las redes y dispositivos de control que están en la planta. Brinda la opción de seleccionar un dispositivo y mostrar toda una gama de elementos de configuración e instrumento de vigilancia. (19)

Todas estas librerías están creadas en el lenguaje C. La mayoría de la información, se encuentra en forma de estructuras y las funciones no se encuentran directamente relacionadas con estas como están relacionados los métodos con sus clases. Podemos decir que todo esto conlleva a que la línea de aprendizaje de un programador medio al tratar de dominar esta tecnología, sea elevada. No obstante al estar programadas en lenguaje C hace que sean mucho más portables a la mayoría de los sistemas operativos y más rápidos en la ejecución de sus funciones. En el caso de *RSLinx* es propiedad de *Rockwell Automation* ya esto le impide que sea utilizada como posible solución al problema. *TuxEip* y *PLCIO* solo define el protocolo y no el transporte por tanto no se puede referirse a si son asíncrono o no. Esto deja a mitad la solución al problema que no solo requiere el protocolo sino que sea asíncrono el transporte.

1.10 ASIO (20)

La biblioteca ASIO está destinada para programadores que usan C++ para la programación de software, donde el acceso a la funcionalidad del sistema operativo como el establecimiento de conexiones es necesario.

En particular, ASIO trata de abordar los siguientes objetivos:

- Portabilidad: La biblioteca debe apoyar y ofrecer un comportamiento coherente en todo, una gama de sistemas operativos de uso común.
- Escalabilidad: La biblioteca debe permitir el desarrollo de aplicaciones de red que escala a cientos o miles de conexiones simultáneas.
- Eficiencia: La biblioteca debe apoyar técnicas por ejemplo: dispersar-recolectar la entrada/salida, y permite las aplicaciones de protocolo que reducen al mínimo el copiado de los datos.

- Modelo de sockets de Berkeley: La API de sockets de Berkeley es ampliamente conocido y aplicado, además de ser tratados en muchas literaturas.
- Facilidad de uso: Trata de minimizar la inversión del aprendizaje con sólo unas pocas reglas y directrices.
- Base para más abstracción: La biblioteca debe permitir el desarrollo de otras bibliotecas que ofrecen mayores niveles de abstracción.

1.10.1 ASIO y Proactor.

La biblioteca ASIO ofrece operaciones sincrónicas y asincrónicas. El apoyo asíncrono se basa en el patrón de diseño: *Proactor.* Examinemos cómo el patrón del diseño de Proactor se ejecuta en Boost.ASIO sin hacer referencia a una plataforma específica.

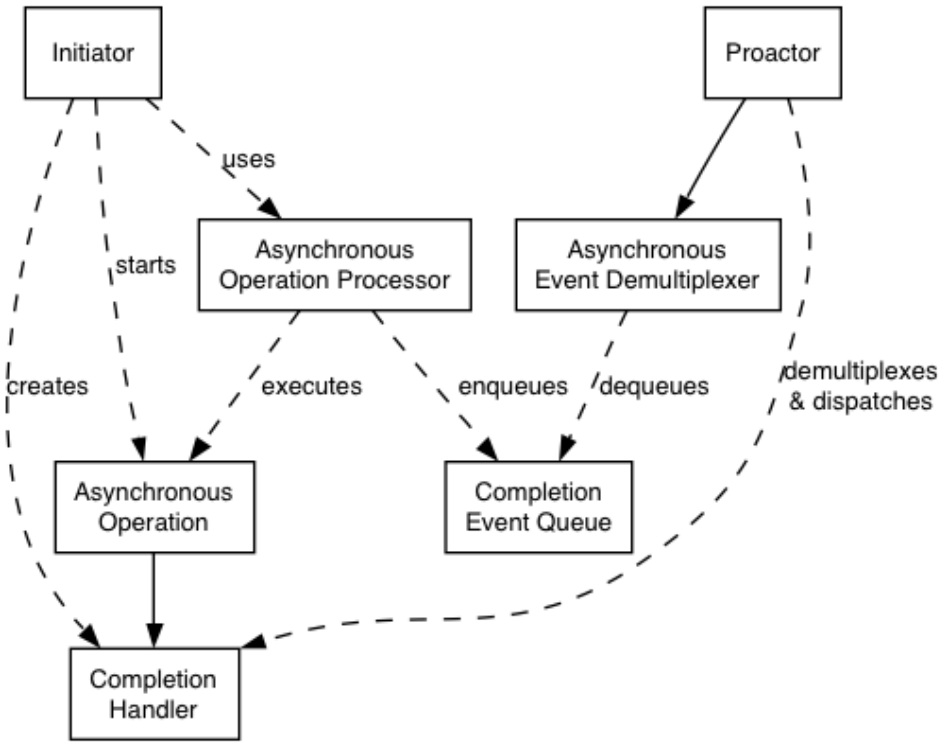


Figura 6. Diseño del patrón Proactor.

El Iniciador (Initiator), es el que comienza las operaciones asincrónicas, mandándola a ejecutarse a través de una interfaz de alto nivel como ASIO::basic_stream_socket, que a su vez delega un servicio como ASIO::stream_socket_service. Esta operación asincrónica es procesada por el Procesador de la operación asincrónica (Asynchronous Operation Processor), la cual ejecuta las operaciones asincrónicas y los eventos que están en colas, después de haber terminado dichos eventos, es colocado en una cola de eventos terminados (Completion Event Queue), lugar donde se tendrá el buffer de estos eventos terminados hasta que no se quiten de la cola por un evento asíncrono multiplexor. Este evento es llamado por el patrón de diseño Proactor, que es quién dispara el manejador de eventos terminados, asociado al evento, esta abstracción está representada por

boost::asio::io_service, siendo retornado por el Completion Handle, el cual lo tendría en espera hasta ser llamado.

1.10.2 ASIO y Reactor.

En muchas plataformas, *Boost.ASIO* ejecuta el patrón del diseño *Proactor* en términos de *Reactor*, tal como *select, epoll* o *kqueue*. Este acercamiento al patrón de diseño *Proactor* corresponde como sigue:

Procesador de la operación asincrónica: Reactor se implementó usando *select*, *epoll* o *kqueue*. Cuando el reactor indica que el recurso está listo para realizar la operación, el procesador ejecuta la operación asincrónica y envía a la cola de eventos terminados el manejador asociado.

Cola de eventos terminados: Una lista de manejadores de eventos terminados.

Demultiplexor asíncrono de eventos: Es implementada por la espera de un evento o una variable de condición hasta que un manejador esté disponible en la cola de eventos terminados.

1.10.3 ASIO y Windows.

En Windows NT, 2000 y XP, *Boost.ASIO* toma las ventajas de superponer la entrada/salida para proporcionar una implementación eficaz del patrón de diseño *Proactor*. Este acercamiento al patrón de diseño *Proactor* corresponde como sigue:

Procesador de la operación asincrónica: Es implementado por el sistema operativo. Las operaciones son iniciadas llamando una función superpuesta como AcceptEx.

Cola de eventos terminados: Esto es ejecutado por el sistema operativo y asociado a un puerto de completamiento. Hay un puerto de entrada-salida para cada instancia de *asio::io_service*.

Demultiplexor asíncrono de eventos: Llamado por *Boost.ASIO* para quitar eventos de la cola y los manejadores asociado a ellos.

1.10.4 Ventajas.

Portabilidad: Muchos sistemas operativos ofrecen una entrada/salida asincrónica nativa, como la opción preferida para desarrollar redes del alto rendimiento. La biblioteca propuesta se puede ejecutar en términos de entrada/salida asincrónica nativa. Sin embargo, si la ayuda nativa no está disponible, la biblioteca se puede también ejecutar usando los demultiplexores de eventos síncronos que caracterizan el patrón del *Reactor*.

Creación de hilos para la concurrencia: Las operaciones de largas duraciones son realizadas asincrónicamente por la biblioteca. Consecuentemente las aplicaciones no necesitan crear muchos hilos para aumentar a la concurrencia.

Rendimiento y escalabilidad: Las estrategias de implementación como un hilo por conexión pueden degradar la función de un sistema, debido al cambio contexto, sincronización y movimiento de datos entre CPUs. Con operaciones asincrónicas se reduce el costo de cambio de contexto, minimizando el número de hilos del sistema, solo activando los hilos que procesan eventos.

Simplifica la sincronización de la aplicación: Los manipuladores asíncronos pueden ser escritos en un solo hilo y se puede desarrollar la lógica de aplicación sin ninguna preocupación por las cuestiones de sincronización.

Funciones de alto nivel: Se refiere a las funciones que implementa la biblioteca para facilitar el trabajo, como el envío de mensajes con determinado formato. Cada función se lleva a cabo llamando a varias funciones de bajo nivel, como *read* o *write*.

1.10.5 Desventajas.

Alta complejidad de los programas: Es más difícil desarrollar aplicaciones utilizando mecanismos asíncronos debido a la separación en el tiempo y el espacio entre el inicio de la operación y la finalización de esta. Las aplicaciones también pueden ser más difíciles de depurar debido al flujo de control.

Uso de la memoria: El espacio de un buffer se bloquea durante las operaciones de escritura y lectura, estas se pueden ejecutar indefinidamente. Se requiere un buffer por cada operación concurrente. Por otro lado con el uso del patrón Reactor no se requiere el uso del buffer hasta que no esté listo el socket para leer o escribir.

1.11 Boost.

Boost es un conjunto de bibliotecas de clases, de código abierto y revisión por pares preparadas para extender las capacidades del lenguaje de programación C++. Su licencia permite que sea utilizada en cualquier tipo de proyectos, ya sean comerciales o no. Varios fundadores de Boost pertenecen al Comité ISO de Estándares C++. La próxima versión estándar de C++ incorporará varias de estas librerías.

Su diseño e implementación permiten que sea utilizada en un amplio espectro de aplicaciones y plataformas. Abarca desde librerías de propósito general hasta abstracciones del sistema operativo. Con el objetivo de alcanzar el mayor rendimiento y flexibilidad se hace un uso intensivo de plantillas. Boost ha representado una fuente de trabajo e investigación en programación genérica y meta programación en C++. (21)

La última versión de la misma, salió el 29 de marzo del 2008. Las bibliotecas que usamos son la Boost.Function, la Boost.Bind. A continuación se dará una breve explicación de las bibliotecas utilizadas.

1.11.1 Boost.Function + Boost.Bind.

Son librerías que encapsulan funciones y las dos se usan a la par. Boost.Bind soporta objeto de funciones, funciones, punteros a función y miembros de punteros a función. Es capaz de asignar valores específicos a cualquier argumento o intercambiar los argumentos de entradas en posiciones arbitrarias. Boost.Function es una librería que contiene una familia de plantillas de clases que son los contenedores de los objetos de función. En general, cualquier lugar en el que una función puntero se

utiliza para aplazar una llamada o hacer una llamada, Boost.Function puede utilizarse. (21) Se muestra un ejemplo sencillo que ayudará a comprender lo anterior:

```
void foo();

class Bar
{
public:
      void baz();
};


int main()
{
       boost::function<void ()> func = foo;
       func();

       Bar bar;
       func = boost::bind(&Bar::baz, &bar);
       func();
}
```

Como se observa, se puede usar el mismo objeto para guardar una función normal y una función miembro. Esto es muy útil para cuando se crean los "*callbacks*".

Capítulo 2. Tendencias y tecnologías actuales a desarrollar

La búsqueda de la eficiencia, la productividad y la competitividad por parte de las empresas están contribuyendo a la búsqueda de metodologías de apoyo a la toma de decisiones complejas en escenarios de múltiples criterios de selección. En el mundo se han desarrollado muchas metodologías de desarrollo, escoger una de éstas es un problema para muchos desarrolladores de software.

En este capítulo se presentan varias de éstas metodologías y sus principales características. Además se expondrán las principales herramientas usadas para el desarrollo de la solución y otras que se consideraron como posibles a utilizar en el desarrollo de la misma. En el capítulo anterior se dio una pequeña descripción del protocolo CIP, en este se incluye un resumen de la especificación del protocolo.

2.1 Metodologías de Desarrollo de Software

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Debido a la diversidad de propuestas es un problema decidir qué metodología de desarrollo utilizar.

El objetivo de una metodología de desarrollo es elevar la calidad del software, en todas las fases por las que pasa, a través de una mayor transparencia y control sobre el proceso. Es labor de la metodología de desarrollo hacer que esas medidas, para aumentar la calidad, sean reproducibles en cada desarrollo.

En los últimos años se han desarrollado dos corrientes en cuanto a metodologías de desarrollo de software se refiere, las llamadas metodologías ágiles y las pesadas. A continuación se analizan tres de las más usadas actualmente, dentro de las dos corrientes antes mencionadas.

2.1.1 Proceso Unificado de Desarrollo de Software (RUP)

RUP es uno de los procesos más generales de los existentes actualmente, ya que en realidad está pensado para adaptarse a cualquier proyecto, y no tan solo de software. (22)

El proceso unificado de desarrollo, es el resultado de la evolución e integración de diferentes metodologías de desarrollo de software, es el producto final de tres décadas de desarrollo y uso práctico. Como producto sigue un camino (ver la figura 5) desde el Proceso *Objectory* (primera publicación en 1987) pasando por el Proceso *Objectory* de *Rational* (publicado en 1997) hasta el Proceso Unificado de *Rational* (publicado en 1998).

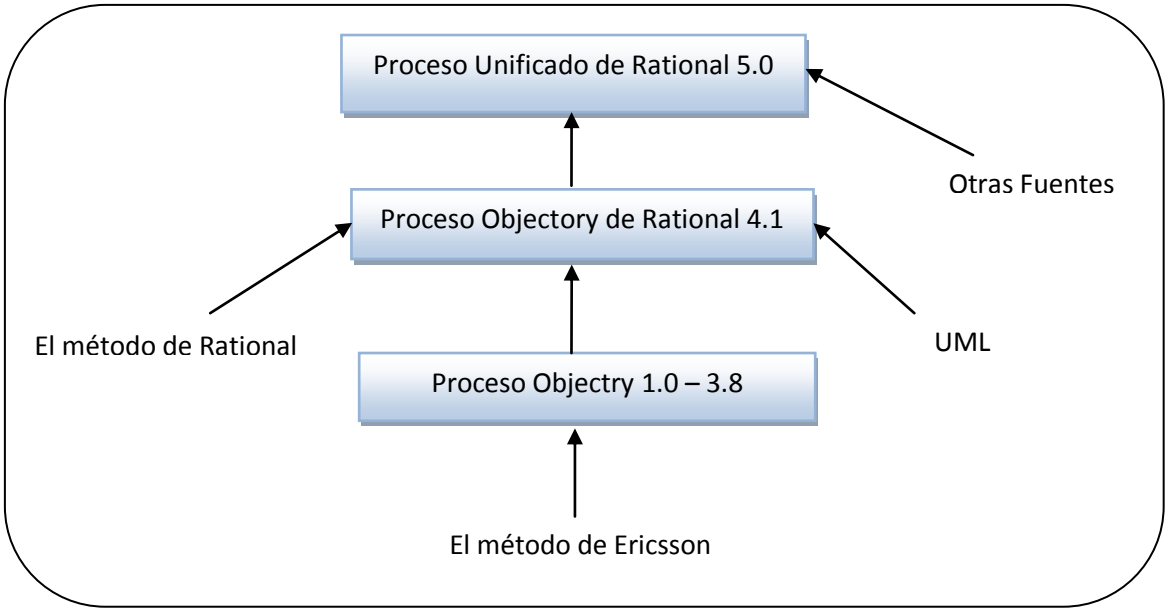


Figura 7. Evolución del Proceso Unificado.

Permite sacar el máximo provecho de los conceptos asociados a la orientación a objetos y al modelado visual. Cuenta con las mejoras prácticas del modelo de desarrollo de un software en particular.

1. Desarrollo de software de forma iterativa.
2. Manejo de requerimientos.
3. Utiliza arquitectura basada en componentes.
4. Modela el software de forma visual, usando el lenguaje unificado de modelado UML.
5. Verifica la calidad del software.
6. Controla los cambios.
7. Dirige las tareas de cada desarrollador por separado y del equipo como un todo.
8. Especifica los artefactos que deben desarrollarse en cada fase de desarrollo del software.

Un proyecto realizado siguiendo RUP se divide en cuatro fases:

1. Inicio.
2. Elaboración.
3. Construcción.
4. Transición.

En cada fase se ejecutarán una o varias iteraciones, de tamaño variable según el proyecto, y dentro de cada una de ellas seguirá un modelo de cascada para los flujos de trabajo que requieren las nuevas actividades anteriormente citadas.

El proceso define una serie de roles que se distribuyen entre los miembros del proyecto y que definen las tareas de cada uno y el resultado, artefactos en la jerga de RUP, que se espera de ellos.

RUP se basa en casos de uso para describir lo que se espera del software y está muy orientado a la arquitectura del sistema, documentándose lo mejor posible, basándose en UML (*Unified Modeling Language*) como herramienta principal.

2.1.2 Programación Extrema (XP)

Es una metodología ágil, intenta reducir la complejidad del software por medio de un trabajo orientado al objeto, basado en las relaciones interpersonales y la velocidad de reacción. Intenta minimizar el riesgo de fallo del proceso por medio de la disposición permanente de un representante competente del cliente a disposición del equipo de desarrollo. Este representante debe estar en condiciones de contestar rápida y correctamente a cualquier pregunta del equipo de desarrollo de forma que no se retrase la toma de decisiones.

La codificación del software en XP se produce siempre en parejas (dos programadores, un ordenador), por lo que se espera que la calidad del mismo suba en el mismo momento de escribirlo. Al contrario que muchos otros métodos, el código pertenece al equipo en completo, no a un programador o pareja, de forma que cada programador puede cambiar cualquier parte del código en cualquier momento si así lo necesita, dejándose en todo caso las mejoras orientadas al rendimiento para el final.

En XP se programará solo la funcionalidad que es requerida para el *release* actual. Es decir, una gran flexibilidad y capacidad de configuración solo será implementada cuando sea necesaria para cumplir los requerimientos del *release*. Se sigue un diseño evolutivo con la siguiente premisa: conseguir la funcionalidad deseada de la forma más sencilla posible. (23)

2.1.3 Desarrollo Guiado por Funcionalidad (FDD)

FDD es un proceso diseñado por Peter Coad, Erich Lefebvre y Jeff De Luca y se podría considerar a medio camino entre RUP y XP, aunque al seguir siendo un proceso ligero es más similar a este último.

Está pensado para proyectos con tiempo de desarrollo relativamente cortos, menos de un año. Se basa en un proceso iterativo con iteraciones cortas (~2 semanas). Las iteraciones se deciden en base a las funcionalidades, de ahí el nombre del proceso, que son pequeñas partes del software con significado para el cliente.

Un proyecto que sigue FDD se divide en 5 fases:

1. Desarrollo de un modelo general.
2. Construcción de la lista de funcionalidades.
3. Plan de *releases* en base a las funcionalidades a implementar.
4. Diseñar en base a las funcionalidades.
5. Implementar en base a las funcionalidades.

Las primeras tres fases ocupan gran parte del tiempo en las primeras iteraciones, siendo las dos últimas las que absorben la mayor parte del tiempo según va avanzando el proyecto, limitándose las primeras a un proceso de refinamiento.

FDD también define métricas para seguir el proceso de desarrollo de la aplicación, útiles para el cliente y la dirección de la empresa, y que pueden ayudar, además de para conocer el estado actual del desarrollo, a realizar mejores estimaciones en proyectos futuros. (23)

2.1.4 Selección de la metodología a utilizar

Luego de hacer una exhaustiva investigación de las características de las metodologías expuestas anteriormente, decidimos utilizar a RUP por caracterizarse como proceso iterativo e incremental, centrada en la arquitectura y dirigida por los casos de uso, que sirve para guiar el proceso de desarrollo de software en el proyecto que se lleva a cabo. Además, con esta metodología se genera una serie de artefactos que facilita que se elabore una documentación detallada del proyecto.

Sin embargo XP es ligera, genera poca documentación y requiere de una presencia constante del cliente en el proceso de desarrollo y en cuanto a FDD aunque su documentación es aceptable, necesita rigurosamente de personal especializado en el tema, además de pequeñas debilidades potenciales para una buena obtención de requisitos del sistema.

2.2 Lenguaje de Programación

El C++ (pronunciado "ce más más" o "ce plus plus") es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C. (24)

C++ es un lenguaje muy versátil, potente y general. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. (25)

Las principales características del C++ son las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica, templates. (24)

Este lenguaje de alto nivel posee una serie de propiedades, las cuales se muestran a continuación:

- **Uniformidad.** Ya que la representación de los objetos lleva consigo tanto el análisis como el diseño y la codificación de los mismos.
- **Comprensión.** Tanto los datos que componen los objetos, como los procedimientos que los manipulan, están agrupados en clases, que se corresponden con las estructuras de información que el programa trata.
- **Flexibilidad.** Al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan.
- **Estabilidad.** Dado que permite un tratamiento diferenciado de aquellos objetos que permanecen constantes en el tiempo sobre aquellos que cambian con frecuencia permite aislar las partes del programa que permanecen inalterables en el tiempo.

- **Reusabilidad.** La noción de objeto permite que programas que traten las mismas estructuras de información reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de objetos ya definidos donde estos están relacionados de una manera particular.

2.3 Lenguaje de Modelado

Entre los lenguajes de modelado que define OMG (*Object Management Group*) el más conocido y usado es UML (*Unified Modelling Language*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar los componentes de un sistema de software, permite tanto la especificación conceptual de un sistema como la especificación de elementos concretos, como pueden ser las clases o un diseño de base de datos.

UML es apropiado para modelar desde sistemas de información en empresas hasta aplicaciones distribuidas basadas en la web, e incluso para sistemas empotrados de tiempo real. Es un lenguaje muy expresivo, que cubre todas las vistas necesarias para desarrollar y luego desplegar tales sistemas. (22)

Representa una colección de las mejores prácticas de ingeniería que han sido probadas con éxito en el modelado de sistemas grandes y complejos. (26) UML es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos.

Fue diseñado para ser un lenguaje de modelado de propósito general, por lo que puede utilizarse para especificar la mayoría de los sistemas basados en objetos o en componentes, y para modelar aplicaciones de muy diversos dominios de aplicación y plataformas de objetos. Es un lenguaje gráfico, que puede ser usado en todas las fases de desarrollo de software y que permite representar los sistemas con varios modelos parciales, lo que facilita su entendimiento y la comunicación.

2.4 Herramientas de Desarrollo

Un entorno de desarrollo integrado o en inglés *Integrated Development Environment* (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. El lenguaje Visual Basic por ejemplo puede ser usado dentro de las aplicaciones de Microsoft Office, lo que hace posible escribir sentencias Visual Basic en forma de macros para Microsoft Word.

Los IDEs proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, *Delphi*, Visual Basic, *Object* Pascal, Velneo, etc. En algunos

lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto, como es el caso de Smalltalk u Objective-C.

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, que mediante *plugins* se le puede añadir soporte para otros lenguajes. (27)

2.4.1 KDevelop

KDevelop es un entorno de desarrollo integrado para sistemas Linux y otros sistemas Unix, publicado bajo licencia GPL.

A diferencia de otros entornos de desarrollo integrados, KDevelop no cuenta con un compilador propio, por lo que depende de GCC para producir código binario.

Su última versión se encuentra actualmente bajo desarrollo y funciona con distintos lenguajes de programación como C, C++, Java, Ada, SQL, Python, Perl y Pascal, así como guiones para el intérprete de comandos Bash.

KDevelop ofrece las siguientes características:

- *KAppWizard*, que genera aplicaciones de ejemplo completas.
- Administración de archivos para archivos fuente, archivos de encabezado, documentación del proyecto entre otras.
- Soporte internacional para su aplicación, lo cual le permite agregar fácilmente sus nuevos lenguajes a un proyecto.
- Administración de proyectos mediante CVS y la provisión de una interfaz fácil de usar para las funciones más necesarias.
- Depuración de programas por medio de *KDbg*.
- La inclusión de cualquier otro programa que necesite para el desarrollo, agregándolo al menú herramientas de acuerdo con sus necesidades individuales.

KDevelop facilita el trabajo con todos los programas en un mismo lugar y ahorra tiempo al automatizar los procesos estándar de desarrollo, proporciona un acceso directo y transparente a toda la información necesaria para el control de un proyecto. Los mecanismos de explotación integrados están diseñados para soportar solicitudes de documentación que tengan los desarrolladores junto con su proyecto. (28)

2.4.2 Eclipse

Esta plataforma, típicamente ha sido usada para desarrollar IDE, como el llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como *BitTorrent Azureus*.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para *VisualAge*. Hoy día, es desarrollado por la Fundación Eclipse, una organización independiente sin

ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

La versión actual de Eclipse dispone de las siguientes características:

- Editor de texto.
- Resaltado de sintaxis.
- Compilación en tiempo real.
- Pruebas unitarias con *JUnit*.
- Control de versiones con CVS.
- Asistentes (*wizards*): para creación de proyectos, clases, test, etc.
- Refactorización.

Asimismo, a través de "*plugins*" libremente disponibles es posible añadir:

- Control de versiones como *Subversion*, vía *Subclipse*.
- Integración con *Hibernate*, vía *Hibernate Tools*. (29)

2.4.3 Visual Studio

Visual Studio es una sencilla, pero poderosa herramienta de desarrollo, es ligera, fácil de aprender y de usar.

Con la herramienta, los desarrolladores pueden:

- Crear aplicaciones de línea de negocio usando varios lenguajes de programación.
- Construir aplicaciones para Windows, la Web y dispositivos móviles todo desde el mismo entorno unificado de desarrollo.
- Desarrollar aplicaciones cliente-servidor usando servicios Web y herramientas integradas de diseño para acceder a datos remotos.

Sin embargo los usuarios pueden:

- Aprender a programar utilizando un entorno de desarrollo sencillo y directo con tutoriales integrados.
- Evaluar el .NET Framework para el desarrollo para Windows y la Web.
- Crear aplicaciones interesantes y divertidas para disfrute personal o para compartir con amigos.

Visual Studio, presenta características que a continuación les brindamos que la hacen única de las demás herramientas de desarrollo:

Código administrado: El CLR realiza un control automático del código para que este sea seguro, es decir, controla los recursos del sistema para que la aplicación se ejecute correctamente.

Interoperabilidad multilenguaje: El código puede ser escrito en cualquier lenguaje compatible con .Net ya que siempre se compila en código intermedio (MSIL).

Compilación *just-in-time*: El compilador JIT incluido en el Framework compila el código intermedio (MSIL) generando el código máquina propio de la plataforma. Se aumenta así el rendimiento de la aplicación al ser específico para cada plataforma.

Garbage collector: El CLR proporciona un sistema automático de administración de memoria denominado recolector de basura (*garbage collector*). El CLR detecta cuándo el programa deja de utilizar la memoria y la libera automáticamente. De esta forma el programador no tiene porque liberar la memoria de forma explícita aunque también sea posible hacerlo manualmente (mediante el método *disponse*()) liberamos el objeto para que el recolector de basura lo elimine de la memoria).

Seguridad de acceso al código: Se puede especificar que una pieza de código tenga permisos de lectura de archivos pero no de escritura. Es posible aplicar distintos niveles de seguridad al código, de forma que se puede ejecutar código procedente del Web sin tener que preocuparse si esto va a estropear el sistema.

Despliegue: Por medio de los ensamblados resulta mucho más fácil el desarrollo de aplicaciones distribuidas y el mantenimiento de las mismas. El Framework realiza esta tarea de forma automática mejorando el rendimiento y asegurando el funcionamiento correcto de todas las aplicaciones. (30)

2.4.4 Selección de la herramienta de desarrollo a utilizar

Luego de hacer una exhaustiva investigación de las características de las herramientas de desarrollo expuestas anteriormente, se ha decidido utilizar las herramientas: Visual Studio y el Eclipse.

2.5 Herramientas CASE

Las Herramientas CASE (*Computer Aided Software Engineering*,) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Estas tienen como objetivos:

1. Mejorar la productividad en el desarrollo y mantenimiento del software.
2. Aumentar la calidad del software.
3. Mejorar el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
4. Mejorar la planificación de un proyecto
5. Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
6. Automatizar el desarrollo del software, la documentación, la generación de código, las pruebas de errores y la gestión del proyecto.
7. Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
8. Gestión global en todas las fases de desarrollo de software con una misma herramienta.
9. Facilitar el uso de las distintas metodologías propias de la ingeniería del software. (27)

2.6 Rational Software Architect (RSA)

Rational Software Architect proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad más rápidamente. Rational Software Architect es una herramienta de desarrollo y diseño integrada que fortalece el desarrollo dirigido por modelos con UML para la creación de servicios y aplicaciones con arquitecturas sólidas. Proporciona un soporte de desarrollo y diseño integrado para el desarrollo dirigido por el modelo con UML.

Con Rational Software Architect, puede unificar todos los aspectos del desarrollo y el diseño de software. Desarrolla aplicaciones de forma más productiva que nunca:

- Utilice lo último de la tecnología de lenguajes de modelado.
- Refuerce la plataforma de modelado ampliable y abierto.
- Simplifique su solución de herramienta de desarrollo y diseño.
- Integre la solución con otros aspectos del ciclo de vida.

Permite modelo flexible de gestión para el desarrollo paralelo de arquitectura y re-factoring, por ejemplo, dividir, combinar, fusionar y comparar los modelos y los modelos de fragmentos. Simplificar la creación de vínculos de trazabilidad de requisitos a través de diseño. Mejora la calidad del código. Tienen como Sistemas Operativos: Linux y Windows. (31)

2.7 Especificaciones de Protocolos

2.7.1 EtherNet/IP

Hoy día, EtherNet/IP es uno de los temas de debate más recurrentes de la ingeniería de automatización y procesos. La aceptación mundial de Ethernet en los entornos industriales y de oficina ha generado el deseo de expandir su aplicación. Es posible que con los avances de Ethernet se pueda aplicar también al manejo de aplicaciones críticas de control actualmente implementadas con otras redes específicamente industriales. EtherNet/IP es un sistema de comunicación diseñado para entornos industriales. Permite que los dispositivos de control intercambien información crítica con requerimientos estrictos de tiempo. Entre los dispositivos que pueden interconectarse con EtherNet/IP se encuentran sensores, de forma general, dispositivos simples de entrada/salida, dispositivos de alta complejidad tales como robots, controladores lógicos programables y otros. Esta particularidad lo hace muy atractivo para enlazar, bajo un mismo protocolo, todos los eslabones del proceso, desde el SCADA hasta los dispositivos más simples.

EtherNet/IP es un protocolo de red en niveles para aplicaciones de automatización industrial. 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 es una red abierta debido a que está basada en la normativa IEEE 802.3 es compatible con la popular familia de protocolos TCP/IP, permite el uso de aplicaciones de control con

el protocolo de control e información CIP, utilizada como protocolo de aplicación para entrada/salida en tiempo real.

EtherNet/IP proporciona un modelo productor-consumidor para el intercambio de información de control crítica en el tiempo. El modelo productor-consumidor permite el intercambio de información entre un dispositivo emisor y varios dispositivos receptores (o consumidores) sin la necesidad de que se envíe el mismo bloque de datos varias veces a diferentes destinos. Para EtherNet/IP, esto se logra haciendo uso de la capa de red y de transporte CIP conjuntamente con la tecnología IP "*multicast*". De esta manera muchos dispositivos EtherNet/IP pueden recibir la misma información producida por otro dispositivo.

2.7.2 Encapsulación (32)

Todos los mensajes CIP son enviados en el campo de datos de un paquete que define Ethernet (ver Figura 6) para el envió de cualquier dato a otro punto de la red.

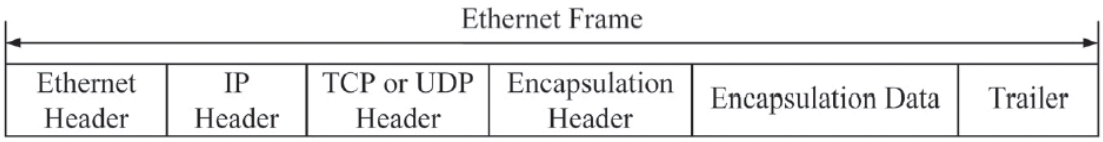


Figura 8. Muestra la relación entre EtherNet/IP y el paquete de Ethernet

El protocolo de encapsulación define un puerto TCP, que debe ser soportado por todos los dispositivos EtherNet/IP. Todos los dispositivos EtherNet/IP deben aceptar, como mínimo, dos conexiones TCP en el puerto 0xAF12.

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

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:

Tabla 2. Estructura de un mensaje encapsulado.

Estructura	Nombre del campo	Tipo de dato	Valor del campo
Cabecera del mensaje encapsulado	Command	UINT	Comando encapsulado.
	Length	UINT	Longitud, en bytes, de la porción de datos del mensaje, o sea el número de bytes que sigue a la cabecera.
	Session handle	UDINT	Identificador de sesión. (Dependiente de la aplicación).
	Status	UDINT	Código de estado.
	Sender Context	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	La parte de datos del mensaje encapsulado. Requerida sólo en ciertos comandos.

Los campos enteros de más de un byte deben ser trasmitidos en el formato "*little-endian*". 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.
2. Explícitamente por el contenido de la parte de datos del mensaje.

Los principales comandos posibles de encapsulación son los siguientes:

Tabla 3. Comandos de encapsulación.

Código	Nombre	Comentario
0x0000	NOP	Este comando no dispara una acción en particular y no requiere respuesta alguna. Es usado como un latido para señalizar periódicamente al receptor la necesidad de mantener la conexión viva. Puede ser enviado sólo a través de TCP.
0x0001–0x0003	Reservados	
0x0004	*ListServices*	Permite obtener los servicios que ofrece el dispositivo. Puede ser enviado tanto a través de TCP como de UDP.

0x0005–0x0062	Reservados	
0x0063	*ListIdentity*	El que origina una conexión puede usar el comando *ListIdentity* para localizar e identificar posibles destinos. El comando puede ser enviado como un *broadcast* usando UDP y no requiere que la sesión esté establecida. La porción de datos de la respuesta a este mensaje proporciona información acerca del dispositivo, tal como fabricante, número de serie, nombre del producto, entre otros.
0x0064	*ListInterfaces*	El comando opcional *ListInterfaces* puede ser usado por el que origina una conexión para identificar interfaces de comunicación que respondan a CIP asociadas con el destino. No requiere que la sesión esté establecida.
0x0065	*RegisterSession*	Este comando le solicita al destino iniciar una sesión. El destino debe enviar en la respuesta el identificador de sesión que estará presente en los mensajes subsiguientes entre ambos puntos. Puede ser enviado sólo a través de TCP.
0x0066	*UnRegisterSession*	El receptor de este comando debe comenzar el cierre de la conexión TCP correspondiente. Puede ser enviado sólo a través de TCP.
0x0067-0x006E	Reservados	
0x006F	*SendRRData*	El comando *SendRRData* transfiere un paquete de solicitud o respuesta hacia el destino encapsulado en la porción de datos del comando. Cuando el protocolo que se encapsula es CIP, el comando *SendRRData* se usa para transferir mensajes explícitos. Puede ser enviado sólo a través de TCP.
0x0070	*SendUnitData*	El comando *SendUnitData* transfiere un paquete de solicitud o respuesta hacia el destino encapsulado en la porción de datos del comando Cuando el protocolo que se encapsula es CIP, el comando *SendUnitData* se usa para transferir datos conectados en mensajes de tipo I/O. Puede ser enviado sólo a través de TCP.

2.7.3 Trama EtherNet

A continuación una muestra de cómo esta estructurado una trama EtherNet, viendo de forma independiente cada uno de las partes que las contienen: (33)

Tabla 4. Estructura de la trama EtherNet.

Preámbulo	SOF	Destino	Origen	Tipo	Datos	FCS
7 bytes	1 byte	6 bytes	6 bytes	2 byte	46 a 1500 bytes	4 bytes

Preámbulo:

Un campo de 7 bytes (56 bits) con una secuencia de bits usada para sincronizar y estabilizar el medio físico antes de iniciar la transmisión de datos. El patrón del preámbulo es:

10101010 10101010 10101010 10101010 10101010 10101010 10101010

Estos bits se transmiten en orden, de izquierda a derecha y en la codificación Manchester representan una forma de onda periódica.

SOF (*Start Of Frame*) Inicio de Trama:

Campo de 1 byte (8 bits) con un patrón de 1s y 0s alternados y que termina con dos 1s consecutivos. El patrón del SOF es: 10101011. Indica que el siguiente bit será el bit más significativo del campo de dirección MAC de destino.

Aunque se detecte una colisión durante la emisión del preámbulo o del SOF, el emisor debe continuar enviando todos los bits de ambos hasta el fin del SOF.

Dirección de destino:

Campo de 6 bytes (48 bits) que especifica la dirección MAC de tipo EUI-48 hacia la que se envía la trama. Esta dirección de destino puede ser de una estación, de un grupo multicast o la dirección de broadcast de la red. Cada estación examina este campo para determinar si debe aceptar el paquete.

Dirección de Origen:

Campo de 6 bytes (48 bits) que especifica la dirección MAC de tipo EUI-48 desde la que se envía la trama. La estación que deba aceptar el paquete conoce por este campo la dirección de la estación origen con la cual intercambiará datos.

Tipo:

Campo de 2 bytes (16 bits) que identifica el protocolo de red de alto nivel asociado con el paquete o, en su defecto, la longitud del campo de datos. La capa de enlace de datos interpreta este campo.

Datos:

Campo de 46 a 1500 Bytes de longitud. Cada byte contiene una secuencia arbitraria de valores. El campo de datos es la información recibida del nivel de red (la carga útil). Este campo, también incluye los H3 y H4 (cabeceras de los niveles 3 y 4), provenientes de niveles superiores.

FCS (*Frame Check Sequence* - Secuencia de Verificación de Trama):

Campo de 32 bits (4 bytes) que contiene un valor de verificación CRC (Control de Redundancia Cíclica). El emisor calcula este CRC usando todo el contenido de la trama y el receptor lo recalcula y lo compara con el recibido a fin de verificar la integridad de la trama.

Métodos para calcular el número de secuencia de verificación de trama

- Verificación por redundancia cíclica.

- Paridad bidimensional: Coloca a cada uno de los bytes en un arreglo bidimensional y realiza chequeos verticales y horizontales de redundancia sobre el mismo creando así un byte extra con un número par o impar de 1s binarios.

- Checksum (suma de verificación) de Internet: Agrega los valores de todos los bits de datos para obtener una suma.

2.7.4 Formato Común del Paquete.

El Formato Común del Paquete (CPF, siglas en ingles) es una construcción que provee una forma para estructurar el campo *Encapsulation Data*. Si la definición del comando que esta en *Encapsulation Header* requiere más artículos, el CPF le permite hacer un paquete de múltiples ítems en un marco de encapsulación, como se muestra en Figura 7.

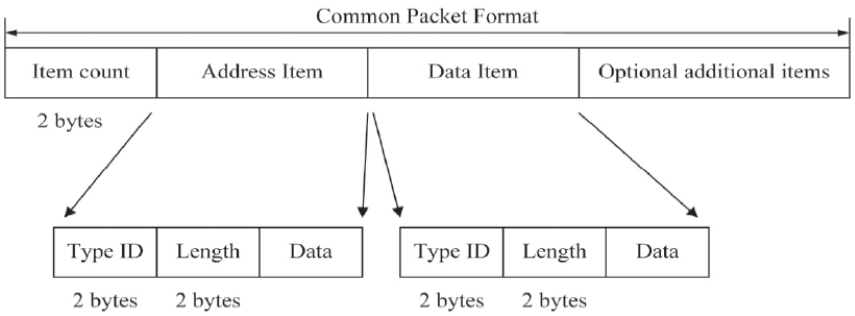


Figura 9. Ejemplo de un CPF.

2.7.5 Dispositivo Logix

Logix es un dispositivo que ha sido diseñado para que sea posible acceder a los datos, las cuales se puedan escribir y leer.

2.7.6 Factores diferenciadores de Logix

1. Basado en información: Logix ha sido diseñado para que sea posible acceder a los datos en toda la arquitectura, minimizando así las tareas de programación y manteniendo el "*glueware*" (software de unión).
2. Multidisciplinario: Logix la única arquitectura del mercado que puede emplearse para control multidisciplinario integrado (secuencia, proceso, movimiento, variadores y seguridad).
3. Escalable: Logix ha sido diseñado para ser útil desde el control de maquinaria de pequeño tamaño hasta toda una empresa. (34)

2.7.7 Logix 5000

El controlador Logix5000 almacena datos en *tags*, donde esos *tags* contienen las siguientes propiedades:

1. **Nombre**, que identifica a los datos:

1.1 Hasta un máximo de 40 caracteres de longitud.

1.2 No incluyen un número de archivo.

2. **Alcance**:

2.1 Un controlador que se pueda acceder directamente.

2.2 Un programa con la que no se pueda acceder por vía de un dispositivo externo.

3. **Tipo de datos**, que define la organización de los datos.

El controlador Logix5000 soporta varios tipos de datos:

- **Atómico**: un bit, byte, 16 bit o 32 bit, de las cuales cada una almacena un único valor.
- **Estructura**: son un grupo de tipos de datos que funcionan como única unidad. Dependiendo de las necesidades de la aplicación, puede crear estructuras adicionales referenciadas por el usuario.
- **Arreglo**: una secuencia de elementos, siendo cada uno de ellos un tipo de datos:
  - I. Se puede definir el dato en 0, 1, 2 ó 3 dimensiones como sea requerido (Generalmente 0 ó 1 dimensión son los más comunes).
  - II. Se puede usar otro tipo de datos atómicos o estructuras según sea conveniente. (35)

2.7.8 Identificador del Objeto Interno (IOI)

Un segmento CIP, es un flujo de codificación de objetos, utilizados para referenciar, describir y/o configurar una entidad específica de CIP. Los segmentos son usualmente agrupados con el fin de proporcionar la información completa. La codificación que se utiliza para los segmentos de CIP ha sido relacionada con la codificación que se utiliza para los reportes de los tipos de datos de CIP, permitiendo así que ambos se entremezclen. Un segmento contiene la información que indica el tipo de segmento y los datos del segmento. Están estructurados de la siguiente forma: [Class ID] [Instance ID] [Attribute ID, if required], ver figura 8. Cada elemento de esta estructura permite formatos diversos (1 byte, 2 el byte y 4 el byte).

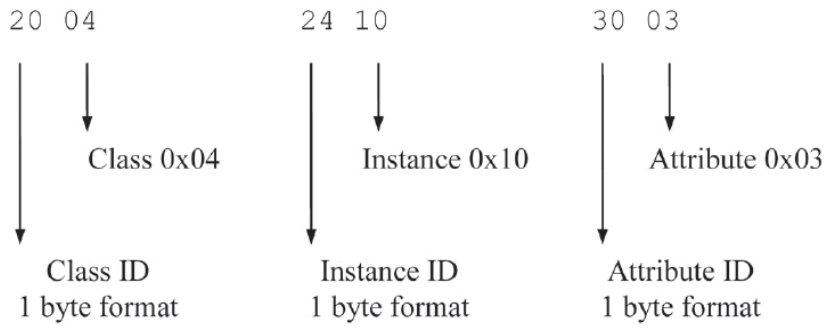


Figura 10. Ejemplo de codificación de un segmento lógico.

Se han definido los siguientes tipos de segmento:

- ***Port segment*** es utilizado para el enrutamiento de una subred a otra.
- ***Logical segment*** refiere a la lógica de la información (tales como clases, instancias y atributos).

- ***Network segment*** se utiliza para especificar los parámetros de red que requiere un nodo para transmitir un mensaje a través de una red.
- ***Symbolic segment*** contiene un símbolo de cadenas que debe ser interpretado por el dispositivo.
- ***Data segment*** contiene los datos para la aplicación de destino.

Los mensajes en CIP pueden ser enviados conectados o desconectados. Los mensajes que son enviados conectados usan el ID de conexión (dirección implícita del objeto), donde los mensajes desconectados solo usan el IOI para especificar la ruta y el objeto.

Un IOI (*EPATH*) es una dirección explícita en un mensaje que es decodificado por el *Message Router*. Un IOI se compone de segmentos. De los segmentos definidos anteriormente, en la familia de dispositivo Logix, se realizan algunas modificaciones las cuales se especifican a continuación:

1) Versiones de clases de segmentos:

   Versión de 1 byte

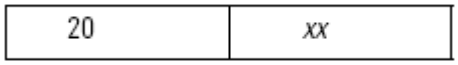


Figura 11. Clases de segmento de 1 byte.

Donde: xx es el valor código de la clase.

Versión 2 bytes:

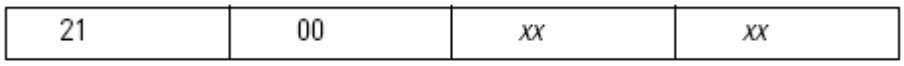


Figura 12. Clases de segmento de 2 bytes.

Donde: xxxx es el valor código de la clase.

2) Versión instancia de segmento:

   Versión de 1 byte

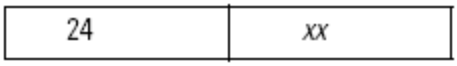


Figura 13. Instancia de segmento de 1 byte.

Donde: xx es el valor código de la instancia.

Versión 2 bytes

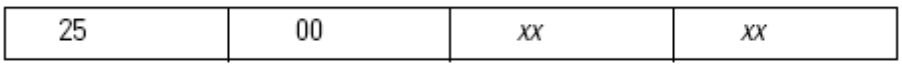


Figura 14. Instancia de segmento de 2 bytes.

Donde: xx es el valor código de la instancia.

3) Versiones de segmentos de elementos:

Versión de 1 byte

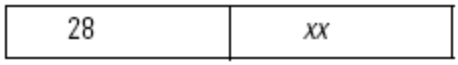


Figura 15. Segmento de elementos de 1 byte.

Donde: xx es el número del elemento.

Versión 2 bytes

Figura 16. Segmento de elementos de 2 bytes.

Donde: xx es el número del elemento, byte bajo primero.

Versión 4 bytes

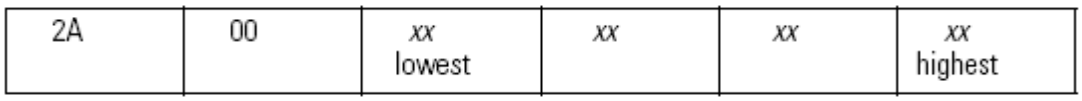


Figura 17. Segmento de elementos de 4 bytes.

Donde: xx es el número del elemento, byte más bajo al byte mas alto.

4) Versión de segmento simbólico:

Versión de caracteres de 1 byte

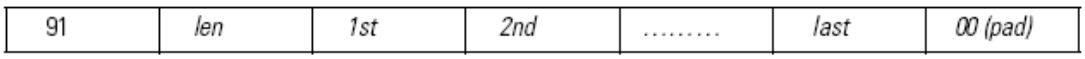


Figura 18. Segmento simbólico de 1 byte.

Donde:

len es la longitud de la palabra simbólica en bytes, no incluye cabecera.

Primero, Segundo y Tercero son los caracteres en orden.

El *pad* es necesario cuando el número de caracteres es desconocido. (35)

Capítulo 3. Presentación de la solución propuesta.

3.1 Introducción

En este capítulo se hará una descripción de la propuesta a defender. Se describe el entorno del sistema, representado mediante un diagrama de clases, con los principales conceptos que lo rodean. A partir de este se establecen los requisitos funcionales y no funcionales con los que debe cumplir el sistema. Estos requisitos se reúnen y se conforman los casos de uso. Estos representan una funcionalidad del sistema y de los cuales se propone, además del diagrama, una descripción textual.

3.2 Modelo conceptual

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, 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). (36)

Luego de un exhaustivo análisis y debido a no tener una estructura definida de los procesos de negocio, se plantea un modelo de dominio. Se realizará a través de un diagrama de clases de UML e identificando los conceptos representados en el diagrama en un glosario de términos; logrando que todo el interesado adquiera un entendimiento mínimo del contexto en que se emplaza el Protocolo a implementar.

3.2.1 Diagrama de clases del Modelo de Dominio

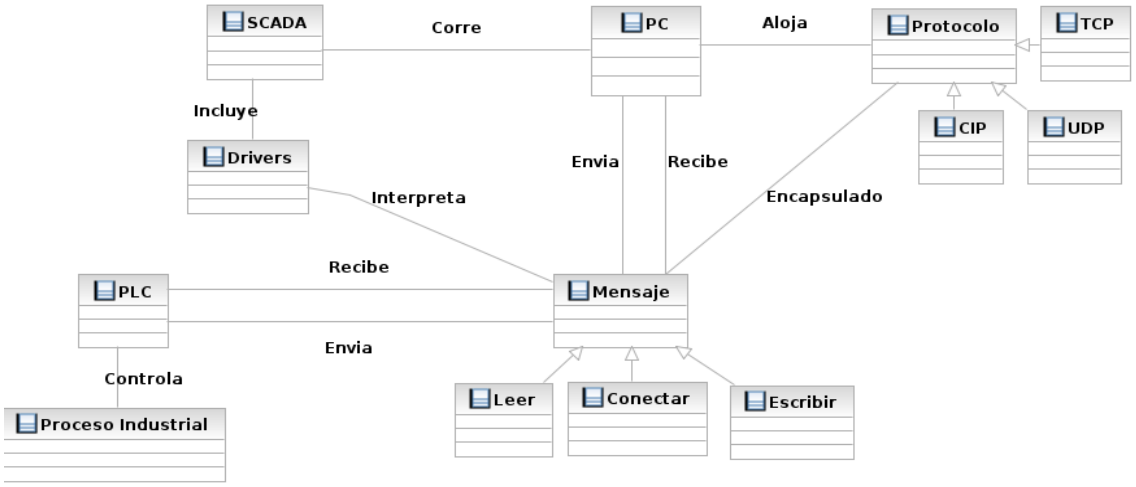


Figura 19. Modelo de Dominio.

3.2.2 Análisis de los conceptos del Dominio

Se define:

PC: Equipo electrónico con capacidad de cálculo que soporta el sistema operativo, la misma soportando así el resto de los programas.

PLC: Dispositivo de Adquisición de Datos el cual mediante mensajes se comunica con los programas que residen en una computadora.

Proceso Industrial: Aquel en el cual se producen transformaciones físicas o químicas.

SCADA: Este concepto engloba a todos los programas que conforman al Sistema de Adquisición y Supervisión de Datos exceptuando los protocolos que él utiliza.

Drivers: Parte de los programas que conforman al SCADA, pero que su función es interpretar los mensajes que llegan a través de distintos modos de adquisición de datos.

Protocolo: Conjunto de reglas o leyes que permiten la comunicación entre dos dispositivos de diversa índole. Estos permiten encapsular los mensajes para ser enviados o recibidos con un formato requerido.

TCP: Conjunto de protocolo de la familia de TCP/IP que permite una comunicación orientada a la conexión.

UDP: Conjunto de protocolo de la familia UDP/IP que permite una comunicación no orientada a la conexión.

Mensaje: Representa cualquier tipo de datos que se pueda enviar o recibir entre una PC y un PLC.

3.3 Solución propuesta.

Denominemos al PLC ControlLogix, servidor y a la computadora que se conecta a éste, cliente. Esta denominación se hace porque la computadora nunca esperará que el PLC ControlLogix se conecta a ella, siempre esta iniciará la comunicación (ver fig. 20). Este cliente debe conectarse con varios servidores a la vez, creando para cada uno un canal de comunicación. Cuando se decide conectar con un servidor, el cliente, crea un hilo para atender todas las operaciones con el mismo. Para lograr la concurrencia este modelo exige un hilo para cada servidor.

Un hilo es un simple flujo de control secuencial dentro de un proceso. Un proceso consta de uno o más hilos, la memoria de trabajo (compartida por todos los hilos) y la información de planificación.

Cada hilo tiene su propio contador de programa y estado del procesador. Esto quiere decir que los hilos progresan en forma independiente a través del código. Cada hilo tiene, a su vez, su propia pila, de modo que las variables locales son intrínsecamente locales para cada hilo y no poseen formas de sincronizarse por sí, estas variables. (37) (38)

Un hilo puede estar en uno de seis estados posibles en cualquier momento, sólo uno de los cuales hace al hilo susceptible de ser elegido para su ejecución. Una vez inicializado, el hilo progresa a través de los siguientes estados:

Listo: Cuando el despachador de Windows busca un hilo para que sea ejecutado, sólo considera los hilos que se encuentran en el estado listo. Estos hilos simplemente están esperando por el acceso al CPU para ser ejecutados.

Standby: Un hilo en estado standby ha sido seleccionado para su próxima ejecución en un procesador particular. Cuando se dan las condiciones correctas, el despachador realiza una selección de contexto para este hilo. Sólo un hilo puede estar en estado standby para cada procesador del sistema.

Ejecución: Una vez que el despachador realiza una selección de contexto a un hilo, el hilo entra en el estado de ejecución, y se ejecuta. La ejecución del hilo continúa hasta que el núcleo del Sistema Operativo lo desplaza para ejecutar un hilo con una prioridad más alta o su quantum de tiempo finaliza, o termina, o voluntariamente entra en estado de espera.

Espera: Un hilo entra en el estado de espera cuando espera voluntariamente a un objeto (asociado a entrada/salida, por ejemplo) para sincronizar su ejecución. Cuando finaliza la espera del hilo, el hilo vuelve al estado listo para volver a entrar en la planificación.

Transición: Un hilo entra en el estado de transición si está listo para la ejecución pero los recursos que necesita no están disponibles.

Terminado: Cuando un hilo finaliza la ejecución, entra en el estado terminado. Una vez terminado, un objeto hilo podría o no ser eliminado.

Se dice que en un hilo es limitado por CPU si emplea la mayor parte del tiempo asignado a él, manteniendo el CPU ocupado. Estos hilos están predominantemente en el estado Listo, Standby o en el de Ejecución. Ejemplos típicos de hilos limitados por CPU son aquellos que se encargan de cálculos matemáticos complejos.

Se dice que un hilo es limitado por entrada/salida si emplea la mayor parte del tiempo asignado a él, en esperas por la conclusión de una operación de entrada/salida. En los Sistemas Operativos Modernos las solicitudes de entrada/salida se satisfacen de manera asíncrona, bien porque para ellas existen un procesador dedicado, o porque se servician a través de interrupciones. Por esta razón los hilos limitados por entrada/salida no compiten por el CPU y prácticamente no degradan el rendimiento de los demás hilos. Estos hilos están predominantemente en el estado de Espera. (39)

Por Rendimiento entenderemos simplemente una magnitud inversa el tiempo total que necesita el Sistema en realizar un cálculo determinado. Si un modelo ejecuta un cálculo en menor tiempo que otro se dice que tiene un Rendimiento mayor.

El modelo multihilo raras veces mejora el rendimiento total del cliente y las razones de esto son obvias: A menos que el cliente se ejecute en un ordenador con múltiples procesadores este tiene que simular el paralelismo asignándole pequeñas cuotas de acceso al CPU a cada hilo. Esto provoca muchos cambios de contexto que son los cambios que ocurren cuando se promueve un hilo del estado Listo al estado de Ejecución. El hilo desplazado debe poder volver a ejecutarse más tarde y por tanto tiene que salvar cierta información (que se denomina información de contexto o simplemente contexto)

para poder reiniciar su trabajo cuando se le asigne nuevamente el CPU. Un cambio de contexto típico requiere salvar y volver a cargar los siguientes datos:

- Contador de programa.
- Registro de estado del procesador.
- Contenidos de otros registros (incluidos los del coprocesador).
- Punteros de la pila del núcleo y del usuario.
- Un puntero al espacio de direcciones en el cual se ejecuta el hilo (directorio de la tabla de páginas del proceso).

Como puede apreciarse los cambios de contexto no son gratuitos, y el CPU debe gastar cierto tiempo en realizarlos. Si la cantidad de cambios de contexto a realizar por el procesador es muy grande entonces podría ser mayor el tiempo empleado por el CPU en ellos, que el tiempo útil de procesamiento, con lo que se degradaría de manera significativa el rendimiento general. Esto ocurre, como es fácil ver, si en el sistema existen muchos hilos en el estado Listo. Podemos decir que la atención a servidores son tareas limitadas por CPU el modelo de servidor multihilo no es eficiente desde el punto de vista del rendimiento. En efecto, en este caso los hilos de atención a los clientes serían limitados por CPU y estarían la mayor parte del tiempo en estado Listo.

Debe señalarse que si las tareas de atención al Cliente son limitadas por entrada/salida el modelo de servidor multihilo no degrada significativamente el rendimiento de servidores con pocos clientes. En este caso los hilos de Clientes están predominantemente en estado de espera y habrá por tanto relativamente pocos hilos listos y consecuentemente pocos cambios de contexto. Por el contrario si el número de servidores es muy grande, aún cuando la probabilidad de que un hilo de Servidor esté listo sea baja, habrá muchos hilos listos y se degradará el Rendimiento. que atienden muchas conexiones simultáneas. (39)

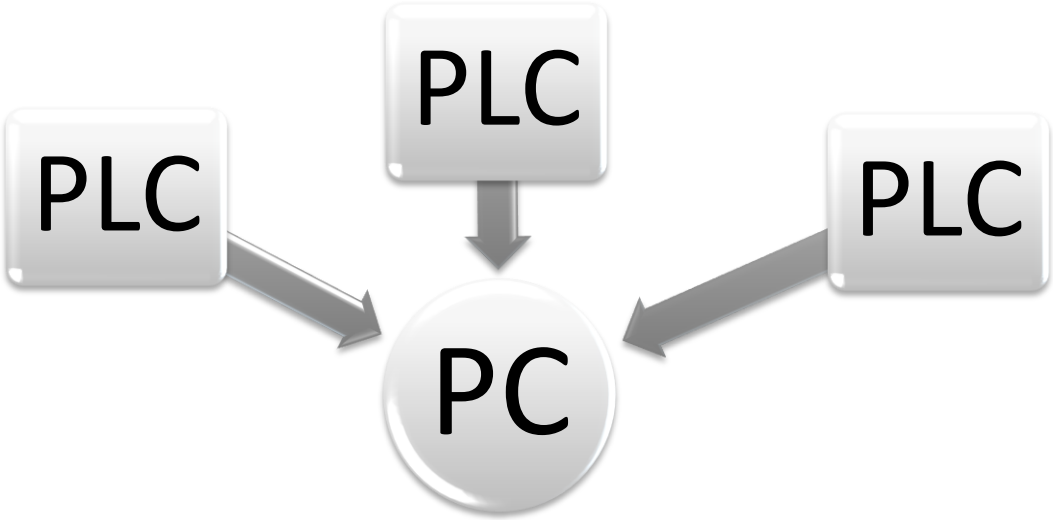


Figura 20. Comunicación de una PC con varios PLC.

Las operaciones que realiza el cliente son conectar, leer, escribir y desconectar estas se ejecutan de forma síncrona. Cuando una función se ejecuta de manera sincrónica ella no retorna hasta que la

operación haya sido completada. Eso significa que el hilo que llama a ese procedimiento cae en estado de espera por un tiempo indefinido. Las funciones que se ejecutan de manera asíncrona retornan inmediatamente incluso aunque la operación no haya concluido aún, lo que permite que la operación de entrada/salida se ejecute en el fondo mientras el hilo que efectuó la llamada ejecuta otras tareas. Por ejemplo un solo hilo puede ejecutar operaciones simultáneas de entrada/salida sobre diferentes objetos e incluso sobre un mismo objeto.

En efecto este modelo minimiza los cambios de contexto por lo que los recursos del CPU se aprovechan de manera más eficiente. Pero tener un solo hilo para el sistema completo desaprovecharía los recursos en las computadoras actuales que en su mayoría ya portan dos núcleos donde cada uno puede atender un hilo. Además muchas máquinas poseen varios procesadores y en este caso también desaprovecharíamos los recursos. Por otra parte si el número de respuesta que provienen de los servidores es grande este único hilo demorará mucho en atender a la última respuesta que entró a la cola.

Por lo tanto se propone la creación de una interfaz asíncrona para la comunicación con los Controladores Lógicos Programables de la familia Logix. Que brinde la posibilidad de crear tantos hilos como el usuario determine pero que esto no tenga que ser proporcional a la cantidad de PLC que existan. La cual será desarrollada en lenguaje C++, con apoyo en la biblioteca ASIO y Boost. Esta contendrá las funciones de conectar, desconectar, escribir, leer del PLC entre otras funciones auxiliares. Todas las funciones mencionadas anteriormente serán asíncronas y permitirán la entrada de una función por parámetros a la cual se le avisará cuando termine una de esas funciones.

3.4 Descripción de las funcionalidades del sistema propuesto

Una especificación de requisitos del software es una descripción completa del comportamiento del sistema a desarrollar. Incluye un conjunto de casos de uso que describen todas las interacciones que se prevén que los usuarios tendrán con el software. También contiene requisitos no funcionales.

3.4.1 Requisitos funcionales del sistema

Un requisito funcional es: condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente. (40)

Ahora que hemos definido que son los requisitos y conocidos los conceptos que rodean el entorno de nuestro problema, se puede comenzar a analizar ¿Qué debe hacer el sistema para que se cumplan los objetivos planteados al inicio de este trabajo? Para ello se enumeran a través de requerimientos funcionales las funciones principales con las que debe cumplir el sistema.

Se puede definir que este sistema debe ser capaz de:

1. Conectar con el PLC Logix.
    1.1. Conectar con el PLC mediante una conexión TCP con el uso de los *socket*.

2. Desconectar del PLC Logix.
   2.1. Cerrar la conexión establecida sobre el protocolo TCP.
3. Registrar sesión en el PLC Logix.
   3.1. Crear el encabezado del mensaje CIP.
   3.2. Interpretar la respuesta del mensaje.
4. Cerrar la sesión en el PLC Logix.
   4.1. Crear el encabezado del mensaje CIP.
5. Crear el objeto conexión en el PLC (*Forward_Open*).
   5.1. Crear *Port Segment*.
   5.2. Crear los *Conection Path* y los *Request Path*.
   5.3. Crear la cabecera del mensaje CIP.
   5.4. Crear el cuerpo de este mensaje CIP.
6. Destruir el objeto conexión en el PLC *(Forward_Close*).
   6.1. Crear el *Request Path*.
   6.2. Crear la cabecera del mensaje CIP.
   6.3. Crear el cuerpo del mensaje CIP.
   6.4. Interpretar la respuesta del mensaje.
7. Leer Tag del PLC Logix.
   7.1. Crear el Identificador Interno del Objeto (IOI).
   7.2. Crear el cuerpo del mensaje CIP.
   7.3. Crear el encabezado del mensaje.
   7.4. Crear el cuerpo de este mensaje.
   7.5. Interpretar la respuesta del mensaje.
8. Escribir los datos en el PLC Logix.
   8.1. Crear el Identificador Interno del Objeto (IOI).
   8.2. Crear el cuerpo del mensaje CIP.
   8.3. Crear el encabezado.
   8.4. Crear el cuerpo del mensaje.
   8.5. Interpretar la respuesta del mensaje.
9. Escribir Bit en el PLC Logix.
10. Enviar y recibir mensajes de forma asincrónica.

3.4.2 Requisitos no funcionales del sistema

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. (40)

Para este trabajo se basaron para encontrar los requisitos funcionales en la descripción del problema y de las entrevistas con el tutor.

- **Requerimientos de Software**
  1. Debe funcionar en los sistemas operativos siguientes: sistema operativo Windows 98 o superior, distribuciones de Unix y distribuciones de Mac OS.
  2. Requiere de un compilador de C++ como GCC.
- **Requerimientos de Hardware**
  1. Requiere una tarjeta de red que soporte transmisión de datos sobre Ethernet con el estándar 802.3 establecido por la IEEE.
  2. Un PLC Logix.
- **Restricciones en el diseño y la implementación**
  1. Se utilizará los estándares de codificación establecidos en el Proyecto Guardián del ALBA.
  2. Se implementará con el lenguaje C++.
  3. Se utilizará las Bibliotecas ASIO y Boost.
- **Requerimientos de Usabilidad**
  1. El sistema debe proporcionar una interfaz sencilla y fácil de entender.
- **Requerimientos de Soporte**
  1. Se publicará toda la documentación relacionada con la aplicación y se dispondrá de asesoramiento por parte de los desarrolladores durante la instalación de la aplicación.

3.5 Modelo de Caso de Uso del Sistema

El modelo de casos de uso es un modelo del sistema que contiene actores, casos de uso y sus relaciones. Los actores representan los usuarios del sistema y otras aplicaciones que interactúan con él, es decir, representan terceros fuera del sistema que se relacionan con él. Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario. (40)

3.5.1 Determinación y justificación de los actores del sistema.

En este caso con el sistema interactúa un único actor que se define a continuación:

Tabla 5. Actores del sistema.

Actores	Justificación
Usuario	Cualquier aplicación que haga uso de la interfaz es un usuario porque se beneficia de las funcionalidades de esta.

3.5.2 Diagrama de caso de uso del sistema

Un diagrama de casos de uso del sistema representa gráficamente a los procesos y su interacción con los actores. Cada caso de uso debe comunicarse con al menos un actor u otro caso de uso que se relacione con un actor. (40)

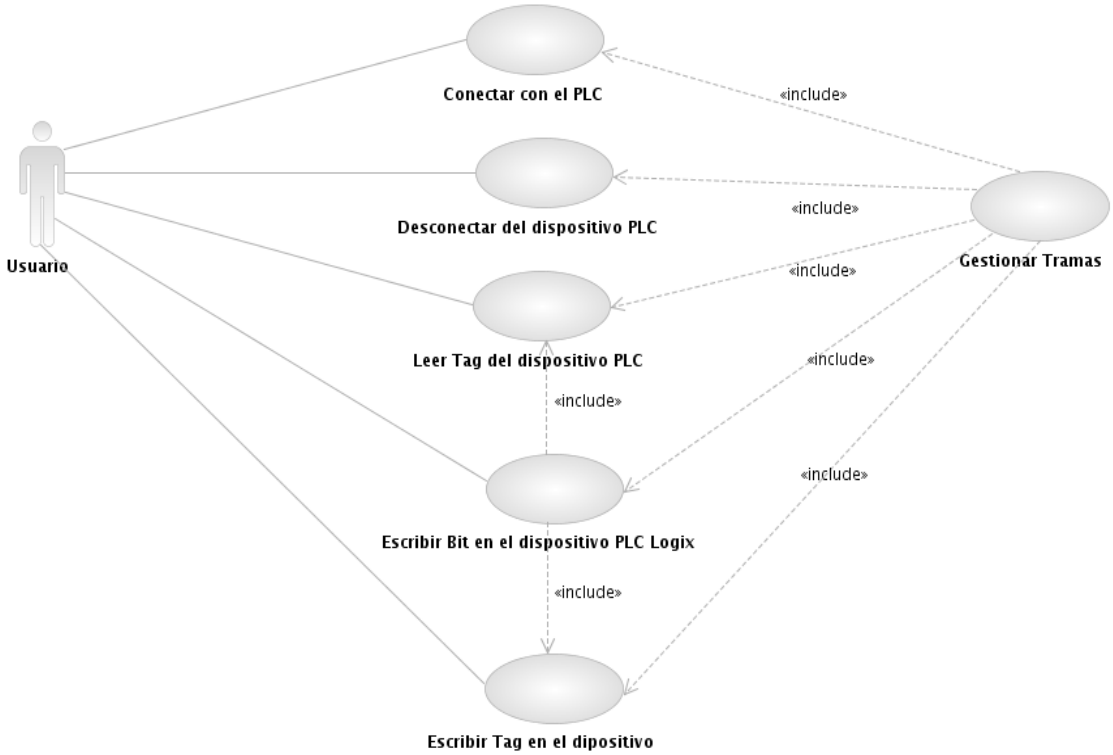


Figura 21. Diagrama de Casos de Uso del sistema.

3.5.3 Descripción textual de los casos de uso del sistema.

Para entender la funcionalidad asociada a cada caso de uso no es suficiente con la representación gráfica del diagrama de casos de uso. La descripción de estos es fundamental para lograr una mejor comprensión de los mismos. A continuación se le mostrará cada una de las descripciones de cada caso de uso utilizado en el sistema.

Tabla 6. Descripción del Caso de Uso “Conectar con el PLC”.

Descripción del Caso de Uso “Conectar con el PLC”	
CU1	Conectar con el PLC.
Propósito	Que la conexión quede establecida y se registre la sesión.
Actores	Usuario
Resumen:	
El caso de uso se inicia cuando el actor solicita conectarse con el PLC. El actor, debe establecer el numero IP al que se va a conectar. El sistema crea el socket e intenta establecer la conexión. Si esta es exitosa continúa con registrar una sesión en el PLC, crea el objeto de conexión que es el encargado de atender las solicitudes a ejecutar a través de esta conexión.	

Referencias	R3, R3.1, R3.2, R5, R5.1, R5.2, R5.3, R5.4
Precondición	
Poscondición	Se ha creado la conexión con el PLC.

Acción del actor	**Respuesta del sistema**
1. El actor especifica un número de IP y una función.	
	1. El sistema crea un puntero a función. 2. Se intenta conectar con el dispositivo ControlLogix, ver el CU “Gestionar Tramas”. 3. Si se establece la conexión se intenta registrar la sesión. Para esto se crea el encabezado donde se especifica el comando de registrar sesión y se envía asíncronamente, ver el CU “Gestionar Trama”. 4. Si se registra la sesión el sistema procede a crear el objeto conexión en el PLC. Crea el *conection Path* y el *Port Segment*. Se crea un mensaje CIP con el servicio 0x54. Se encapsula y se envía este mensaje, ver el CU “Gestionar Tramas”, utilizando el comando *sendRRData*. Se establece un apuntador a función en espera de la respuesta. 5. Al recibo de la respuesta se decodifica el mensaje y se deja establecida la conexión con el objeto creado. 6. El sistema devuelve el control a la función especificada por el usuario con un código de error en caso de que existiera.
1. Flujo alternativo	
Acción del actor	**Respuesta del sistema**
	5., 6., 8. Se le devuelve el control a la función especificada por el usuario con el código de error.
2. Puntos de extensión.	
CU6	

Tabla 7. Descripción del Caso de Uso “Desconectar del Dispositivo PLC”.

Descripción del Caso de Uso “Desconectar del dispositivo PLC”	
CU2	Desconectar del Dispositivo PLC
Propósito	Cerrar la sesión y la conexión con el PLC.
Actores	Usuario
Resumen:	
El caso de uso se inicia cuando el actor solicita desconectarse. Este especifica la función a la cual se llamará cuando termine la operación. Libera el objeto conexión en el dispositivo, elimina el registro de la sesión y cierra la conexión TCP.	
Referencias	R4, R4.1, R6, R6.1, R6.2, R6.3, R6.4
Precondición	Establecida la conexión, este registrada la sesión y este creado el objeto conexión.
Poscondición	El sistema queda completamente desconectado.
Acción del actor	**Respuesta del sistema**
1. El actor solicita desconectarse y especifica una función que se llamará cuando termine la operación.	
	1. EL sistema crea el un puntero a función que se llamará al finalizar la operación. 2. Se crea un mensaje CIP con el servicio 0x4E. Se encapsula con el comando de cabecera *sendRRdata* y se envía, ver CU “Gestionar Trama”. Se establece un puntero a función para esperar la respuesta. 3. Al recibo de este se crea una cabecera con el comando *unregisterSession.* 4. Cerrar la conexión, ver el CU “Gestionar Trama”.
3. Flujo alternativo (8)	
Acción del actor	**Respuesta del sistema**
4. Puntos de extensión.	
CU6	

Tabla 8. Descripción del Caso de Uso “Leer tag del Dispositivo PLC”.

Descripción del Caso de Uso “Leer tag del dispositivo PLC”

CU3	Leer *tag* del dispositivo PLC
Propósito	Leer un *tag* simple o un conjunto de valores a partir de ese *tag* desde el dispositivo.
Actores	Usuario
Resumen:	
El caso de uso se inicia cuando el usuario solicita leer un *tag* o un conjunto de valores a partir de este, al finalizar se le deposita por parámetros los datos leídos en la función que el usuario establece para devolver el control de la aplicación.	
Referencias	R7, R7.1, R7.2, R7.3, R7.4, R7.5
Precondición	Debe estar establecida la conexión y creado el objeto conexión en el PLC.
Poscondición	

Acción del actor	**Respuesta del sistema**
1. El actor solicita leer un *tag* del cual especifica el nombre del *tag*, si es o no un arreglo de *tag*, de donde comenzará a leer en caso de arreglo, y la función a la cual se le devolverá el control cuando el sistema termine esta operación. La signatura de esta función debe ser una variable donde se depositará el tipo del valor leído, un arreglo con los valores y un entero para el error.	
	2. El sistema crea un puntero a función, de la función especificada por parámetro por el usuario. 3. El sistema crea una copia de los datos. 4. Con el nombre del *tag* construye un *tag*. 5. Especifica las funciones internas que se llamarán cuando se reciba la respuesta del PLC. 6. El sistema crea el Identificador Interno del Objeto con el *tag* ya creado. 7. Se crea el mensaje CIP, se encapsula y se envía con el comando *sendUnitData*, ver el CU Gestionar Trama". 8. Se establece un puntero a función a la

	espera asíncronamente. 9. Se recibe la respuesta, ver el CU “Gestionar Trama”. 10. Si no existe ningún error se interpreta. 11. Llama al puntero a función creado en el paso 1.
5. Flujo alternativo (8)	
Acción del actor	**Respuesta del sistema**
	7. ; 10. Llama al puntero a función creado en el paso 1.
6. Puntos de extensión.	
CU6	

Tabla 9. Descripción del Caso de Uso “Escribir tag en el dispositivo PLC”.

Descripción del Caso de Uso “Escribir tag en el dispositivo PLC”	
CU4	Escribir *tag* del dispositivo PLC.
Propósito	Este método permite modificar un *tag* simple o un conjunto de valores a partir de ese *tag* en el dispositivo.
Actores	Usuario
Resumen:	
El caso de uso comienza cuando el actor solicita escribir un *tag* en el dispositivo PLC Logix o un conjunto de valores a partir de este. Se le devuelve cero al usuario si se realizó correctamente la operación.	
Referencias	R7, R7.1, R7.2, R7.3, R7.4, R7.5
Precondición	Debe estar establecida la conexión y creado el objeto conexión en el PLC.
Poscondición	
Acción del actor	**Respuesta del sistema**
1. El usuario solicita escribir un *tag* o grupo de valores a partir de este, especifica el nombre del *tag*, marca desde donde se comenzará a escribir, tipo de dato de la variable a escribir, vector con los valores a establecer y la función a la cual se le devolverá el control al terminar la operación.	
	2. Crea el puntero a función que se llama al

	terminar la operación. 3. Con el nombre del *tag* construye un *tag.* 4. Comprueba el tipo de datos y la cantidad de datos a escribir. 5. Establece las funciones internas que se llamarán cuando se reciba el mensaje del PLC. 6. El sistema crea el Identificador Interno del Objeto con el *tag* ya creado. 7. Crea el mensaje CIP se encapsula y se envía con el comando *sendUnitData*. 8. Se envía asíncronamente ver caso de uso "Gestionar Trama". 9. Se establece un puntero a función a la espera asíncronamente. 10. Se recibe la respuesta, ver el CU "Gestionar Trama". 11. Se devuelve el código de error si ocurrió o un cero en caso de ser satisfactoria la operación. 12. Llama al puntero a función creado en el paso 1.
7. Flujo alternativo (8)	
Acción del actor	**Respuesta del sistema**
	9., 10. Se llama al puntero a función creado en el paso 1
8. Puntos de extensión.	
CU6	

Tabla 10. Descripción del Caso de Uso "Escribir Bit en el Dispositivo PLC Logix".

Descripción del Caso de Uso "Escribir Bit en el dispositivo PLC Logix"	
CU5	Escribir Bit en el dispositivo PLC
Propósito	Modificar un conjunto de Bit en el dispositivo PLC Logix.
Actores	usuario
Resumen:	
Escribir un conjunto de bits hacia el dispositivo PLC Logix. Es decir, las variables que se desean establecer serán variables de tipo BOOLEAN, para las cuales el Ethernet/IP utiliza registros de 2	

ó 4 bytes para su almacenamiento, y lo hace utilizando la composición en bits de los mencionados registros.	
Referencias	R9
Precondición	Debe estar establecida la conexión y creado el objeto conexión en el PLC.
Poscondición	
Acción del actor	**Respuesta del sistema**
1. El usuario solicita escribir un *tag* o grupo de valores a partir de este, especifica el nombre del *tag*, vector con los índices que se de los bit que se desea modificar en el *tag*, vector que contiene los valores a establecer y la función a la cual se le devolverá el control al terminar la operación.	
	2. Crea el puntero a función que se llama al terminar la operación. 3. El sistema guarda las variables locales. 4. El sistema construye el *tag*. 5. Ver CU3. Se llama al puntero a función con los datos de esta operación. 6. Ver CU4. Se llama al puntero a función con los resultados de esta operación. 7. Se devuelve al usuario un código, que indica el estado de la operación.
9. Flujo alternativo	
Acción del actor	**Respuesta del sistema**
	5. Se devuelve al usuario un código, que indica el estado de la operación.
10. Puntos de extensión.	
CU3, CU4	

Tabla 11. Descripción del Caso de Uso “Gestionar Tramas”.

Descripción del Caso de Uso “Gestionar Tramas”	
CU6	Gestionar Tramas
Propósito	Conectar, enviar, recibir tramas de la red Ethernet asíncronamente.
Actores	

Resumen:	
Referencias	R1, R1.1, R2, R2.1, R10
Precondición	
Poscondición	
Sección "Conectar"	
Acción del actor	**Respuesta del sistema**
	1. Recibe el puerto y el número IP y una función a la cual se le devolverá el control al terminar la operación. 2. Inicializa el socket y establece el protocolo a utilizar en este caso siempre será TCP. 3. Intenta conectarse con el dispositivo. 4. Devuelve el control a la función especificada por el usuario.
11. Flujo alternativo	
Acción del actor	**Respuesta del sistema**
	3. Devuelve el control de la función especificada por el usuario con el código de error correspondiente.
Sección "Enviar Mensaje"	
Acción del actor	**Respuesta del sistema**
	1. Recibe de entrada el mensaje, su tamaño y una función a la cual se le devolverá el control al terminar la operación. 2. Envía el mensaje asíncronamente. 3. Llama al puntero a función creado en el paso 1.
12. Flujo alternativo	
Acción del actor	**Respuesta del sistema**
Sección "Recibir Mensaje"	
Acción del actor	**Respuesta del sistema**
	1. Recibe de entrada el tamaño del mensaje a

	recibir, el tiempo de espera y una función a la cual se le devolverá el control al terminar la operación. 2. Se queda en espera asíncrona por la llegada de ese mensaje. 3. Llama al puntero a función creada en el paso 1.
Flujo alternativo	
Acción del actor	**Respuesta del sistema**
	1. Si vence el tiempo se llama al puntero a función creada en el paso 1.
Sección "Desconectar"	
Acción del actor	**Respuesta del sistema**
	1. Parar todas las operaciones de enviar y recibir. 2. Parar los controladores de tiempo. 3. Cerrar la conexión establecida. 4. Notificar el cierre de la conexión.
13. Puntos de extensión.	

Capítulo 4. Construcción de la solución propuesta.

UML define el diseño de clases como un lenguaje natural para comunicarse los programadores con los diseñadores, además con él también pueden interactuar los analistas y otros involucrados en la evolución del mismo. En este capítulo se presenta el diagrama de clases por casos de uso y los diagramas de colaboración. Estos últimos muestran como interactúan las clases y el orden en que son invocados los métodos de cada clase. Se muestra una vista estructural de alto nivel, denominada arquitectura con el caso específico de la arquitectura de dos capas.

4.1 Patrones de Arquitectura

El concepto de arquitectura de software incluye los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura surge de las necesidades de la empresa, como las perciben los usuarios y los inversores, y se refleja en los casos de uso. Sin embargo, también se ve influida por muchos otros factores, como la plataforma en la que tiene que funcionar el software (arquitectura hardware, sistema operativo, sistema de gestión de base de datos, protocolos para comunicaciones en red), los bloques de construcción reutilizables de que se dispone. (41)

4.1.1 ¿Qué es la arquitectura de software?

La arquitectura de software es el conjunto de decisiones significativas sobre la organización de un sistema, la selección de los elementos estructurales y sus interfaces, de los cuales el sistema está compuesto junto con su comportamiento. Describe los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. La misma se representa en 4+1 vistas arquitectónicas. (42)

4.1.2 Arquitecturas en Capas

Este patrón define cómo organizar el modelo de diseño en capas, que pueden estar físicamente distribuidas, lo cual quiere decir que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Este patrón es importante porque simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores.

Además, nos ayuda a identificar qué se puede reutilizar y proporciona una estructura que nos ayuda a tomar decisiones sobre qué partes comprar y qué partes construir.

Principales estilos de arquitecturas estratificadas de las aplicaciones distribuidas contemporáneas (43):

- Arquitecturas de dos capas.
- Arquitecturas de tres capas.
- Arquitecturas de N capas.

El protocolo Industrial EtherNet/IP es uno de los casos representativos de este patrón arquitectónico. Según la concepción escrita anteriormente, pues la arquitectura a utilizar es la de 2-

Capas, ya que define cómo organizar el modelo de diseño a través de la capa de protocolo y la capa de transporte, distribuidas, como a continuación se muestra:

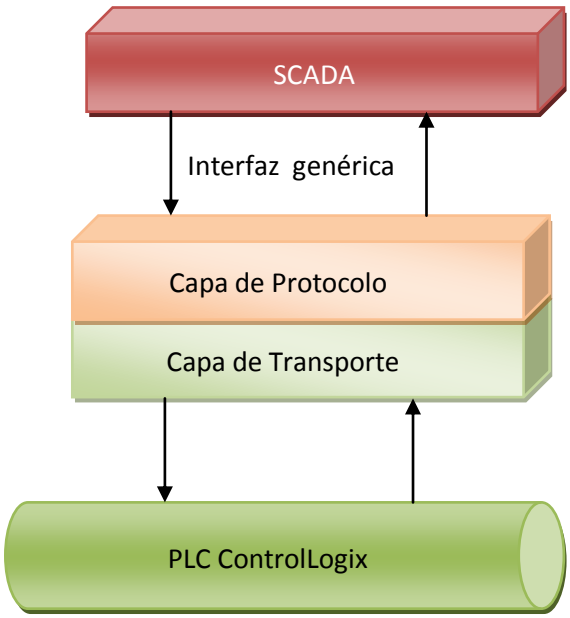


Figura 22. Arquitectura de la aplicación.

Cada capa tiene una funcionalidad lógica propia. La capa de protocolo construye e interpreta los mensajes necesarios para la comunicación. La capa de transporte se encarga de enviar y recibir los mensajes. Ambas son capas que son reutilizables y tienen valor por sí mismas.

La figura 17 muestra la arquitectura de la aplicación de una forma muy general, en el siguiente diagrama mostramos la arquitectura real de nuestro sistema.

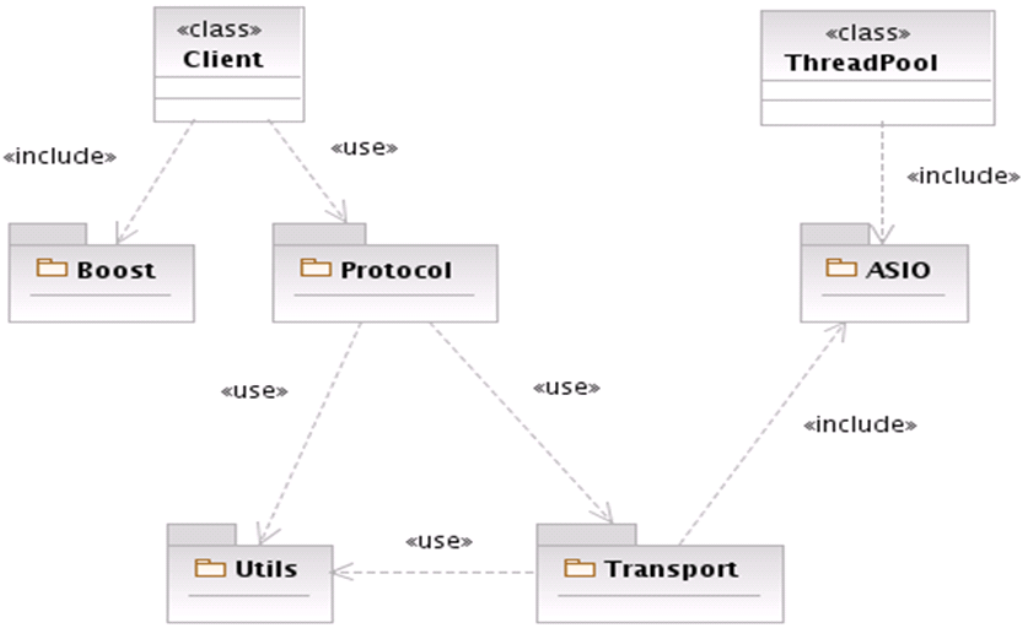


Figura 23. Arquitectura del sistema.

La clase *Client* es la interfaz del protocolo donde se encuentran las principales funciones que brinda el sistema. Estas son conectar, desconectar, leer *tag*, escribir *tag*, escribir bit. La clase *ThreadPool* es la encargada de gestionar los hilos, crear el objeto *io_service* encargado de asignar los hilos existentes a los eventos asíncronos. Boost es la biblioteca de clases que nos permite crear los apuntadores a función. ASIO es la biblioteca de clases que nos permite crear los *socket* y brinda las funciones para la transmisión asíncrona sobre le protocolo TCP/IP. La capa protocolo y la capa de transporte son el núcleo fundamental de nuestro sistema.

4.2 Diagrama de clases del diseño

El diseño es el centro de atención al final de la fase de elaboración, capaz de dar una solución que satisfaga a los requerimientos significativos de la arquitectura. Esto contribuye a una arquitectura estable y sólida, y crear un plano del modelo de implementación.

Además impone una estructura del sistema que se debe conservar lo más fielmente posible cuando den forma al sistema. El modelo de diseño es un modelo de objetos que describe la realización de los CU, y sirve como una abstracción del modelo de implementación y el código fuente. Identifica todas las clases posibles, que participarán en la solución del software.

4.3 Diagramas de clases del sistema propuesto.

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. A continuación se mostrará los diagramas de clases del diseño distribuidos por los casos de usos del sistema. Las clases contendrán los atributos y métodos vinculados a cada uno de los casos de usos (CU) definidos en el capítulo anterior.

4.3.1 Diagrama de clase referente al CU ” Conectar con el dispositivo PLC”

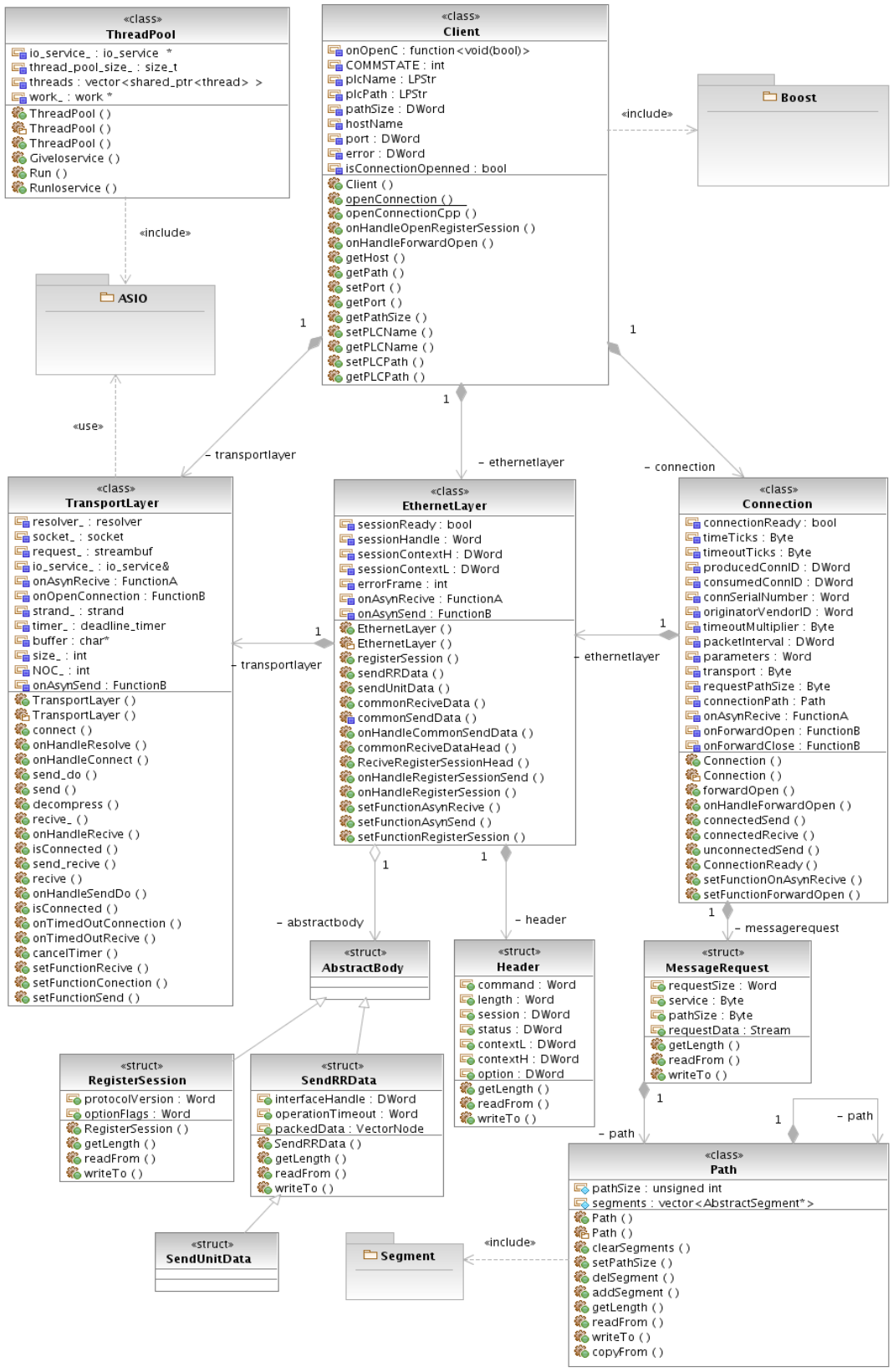


Figura 24. Diagrama de clase referente al CU ” Conectar con el dispositivo PLC”.

4.3.2 Diagrama de clase referente al CU ” Desconectar del dispositivo PLC”

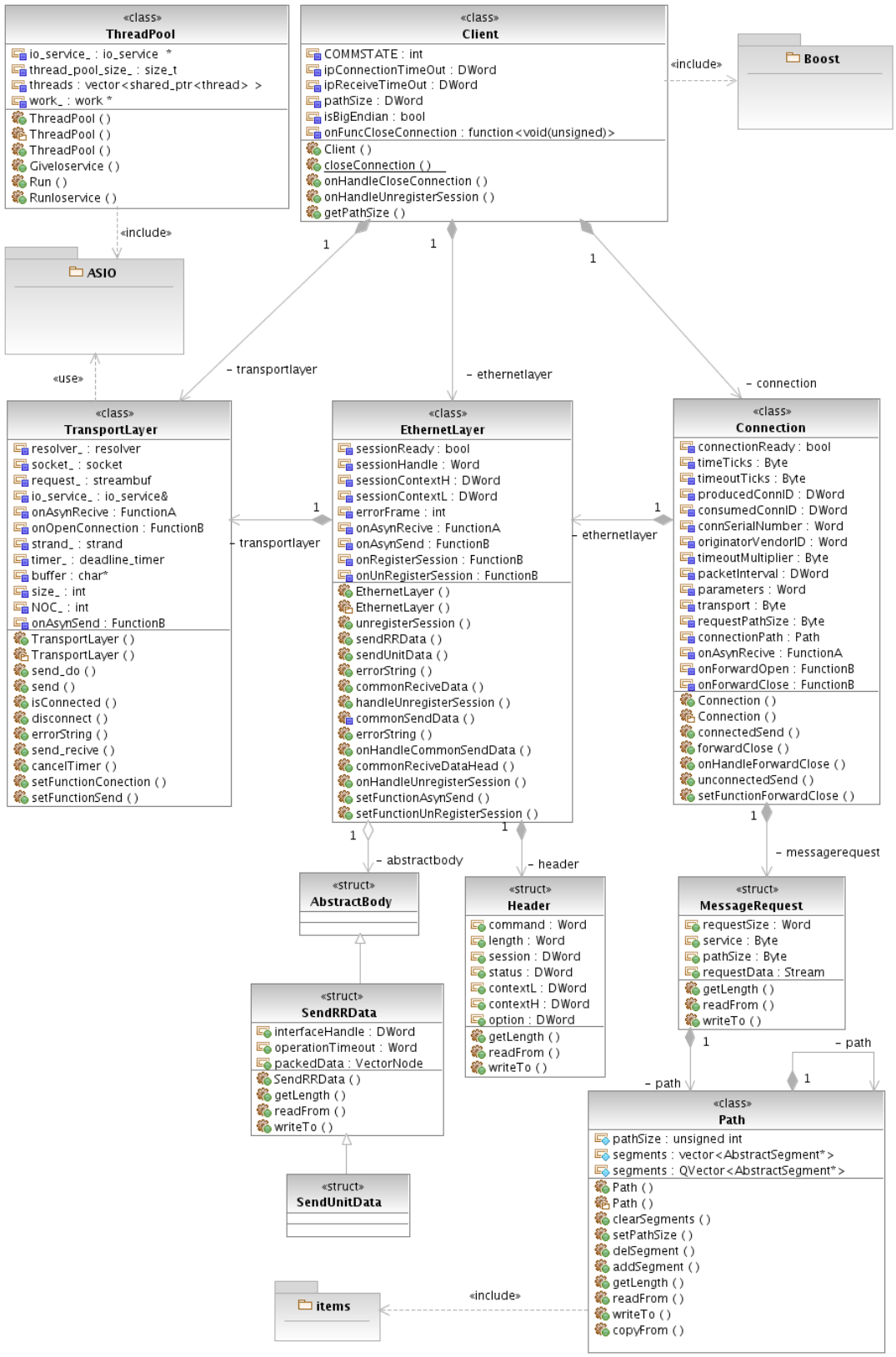


Figura 25. Diagrama de clase referente al CU ” Desconectar del dispositivo PLC”.

4.3.3 Diagrama de clase referente al CU ” Leer tag del dispositivo PLC”

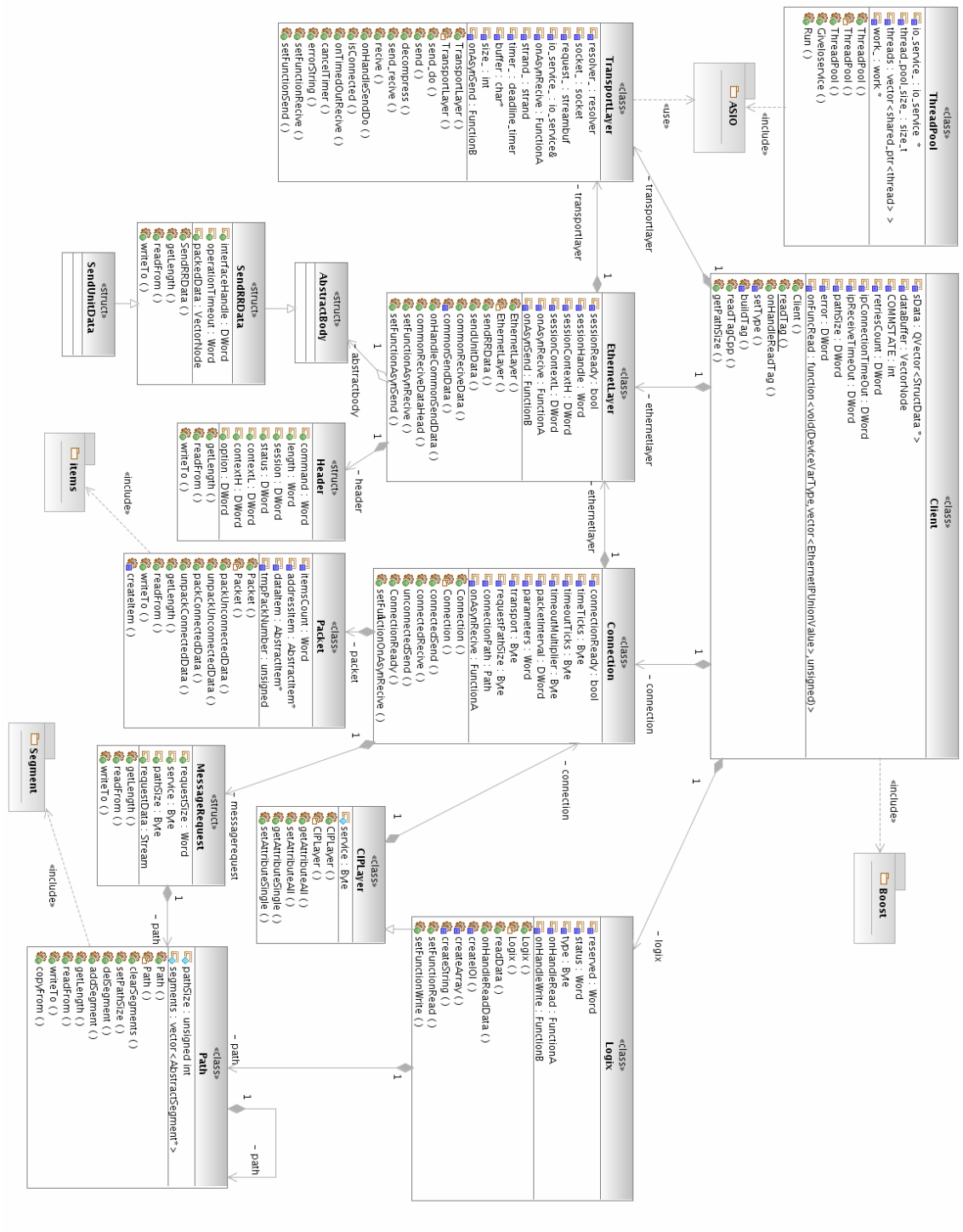


Figura 26. Diagrama de clase referente al CU ”Leer tag del dispositivo PLC”.

4.3.4 Diagrama de clase referente al CU ” Escribir Bit en el dispositivo PLC Logix”

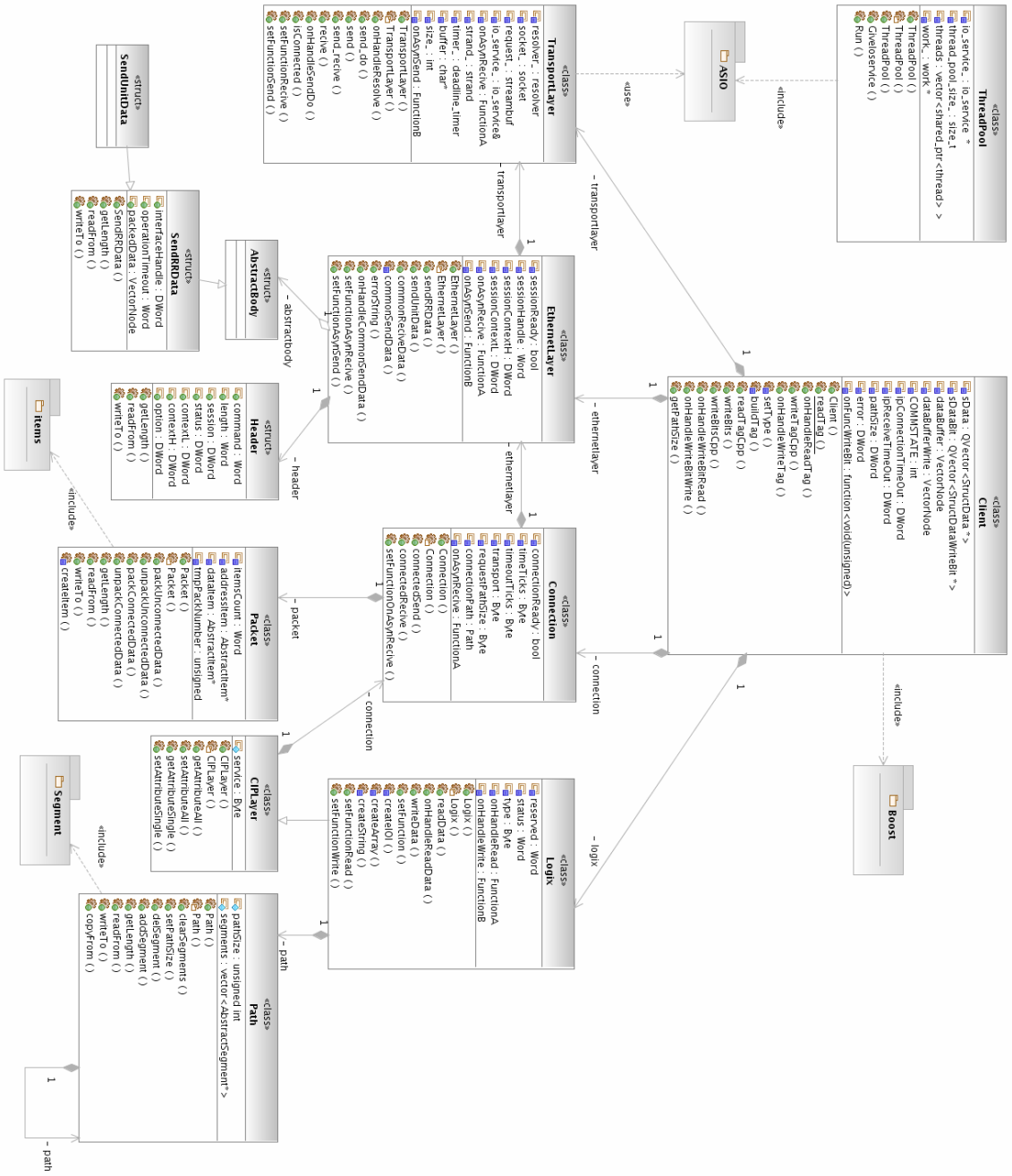


Figura 27. Diagrama de clase referente al CU ” Escribir Bit en el dispositivo PLC Logix”.

4.3.5 Diagrama de clase referente al CU ” Escribir tag en el dispositivo PLC”

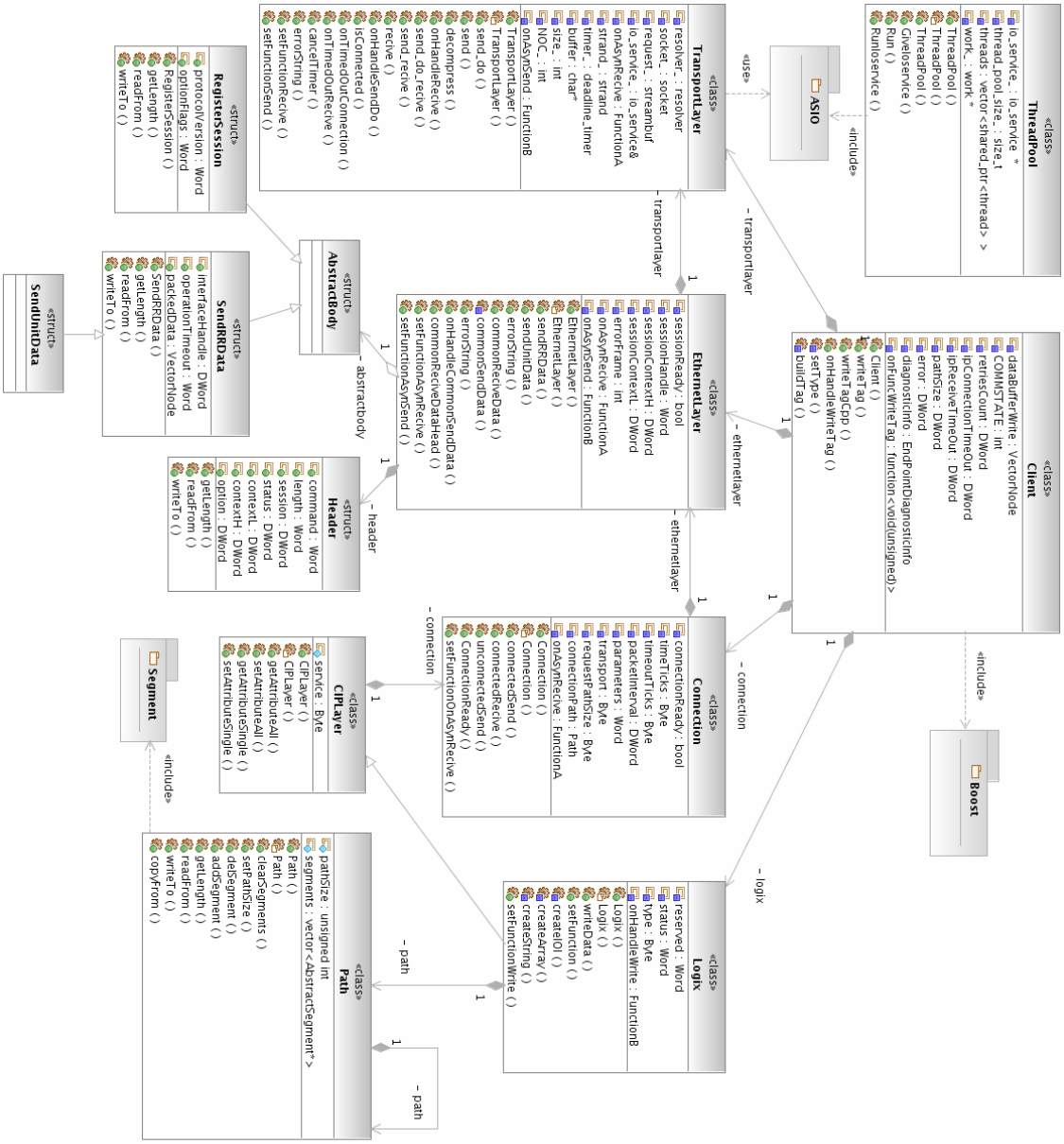


Figura 28. Diagrama de clase referente al CU” Escribir tag en el dispositivo PLC”.

4.4 Descripción de las Clases del Diseño

Las clases que se describen a continuación, poseen un papel importante dentro del desarrollo de la aplicación, las otras pueden ser consultadas en los anexos.

Tabla 12. Descripción de la clase ThreadPool.

Nombre	ThreadPool
Tipo de clase	Controladora
Atributo	**Tipo**

io_service_	boost::asio::io_service *
threads	std::vector<boost::shared_ptr<boost::thread>>
work_	boost::asio::io_service::work *
thread_pool_size_	std::size_t
Para cada responsabilidad:	
Nombre:	ThreadPool(void)
Descripción:	Constructor vacío de la clase.
Nombre:	~ThreadPool(void)
Descripción:	Destructor de la clase.
Nombre:	ThreadPool(std::size_t thread_pool_size)
Descripción:	Constructor de la clase.
Nombre:	GiveIoservice()
Descripción:	Se encarga de devolver un servicio.
Nombre:	Run()
Descripción:	Se encarga de crear un pool de hilos para que funcionen todos los servicios.
Nombre:	RunIoservice()
Descripción:	Se encarga de poner a funcionar al io_service.

Tabla 13. Descripción de la clase Client.

Nombre	Client
Tipo de clase	Controladora
Atributo	**Tipo**
transp	TransportLayer
encap	EthernetLayer
manager	Connection
message	Logix
sData	QVector<StructData *>
sDataBit	QVector<StructDataWriteBit *>
dataBuffer	VectorNode
dataBufferWrite	VectorNode
plcPath	Utils::LPStr
ipConnectionTimeOut	Utils::DWord
path[maxPathSize]	Utils::Byte
pathSize	Utils::DWord

hostName[168]	Utils::Byte
port	Utils::DWord
error	Utils::DWord
isBigEndian	bool
getBoolArray(Utils::DWord start, std::vector<EthernetIP:: EthernetIPUnionValue> &data);	Utils::DWord
onOpenC	boost::function<void(bool)>
onForwardO	boost::function<void(bool)>
onFuncRead	boost::function<void(Utils::DeviceVarType,vector<EthernetIP:: EthernetIPUnionValue>, unsigned)>
onFuncWriteTag	boost::function<void(unsigned)>
onFuncWriteBit	boost::function<void(unsigned)>
onFuncCloseConnection	boost::function<void(unsigned)>

Para cada responsabilidad:

Nombre:	Client(boost::asio::io_service &io_service): transp(io_service),encap(transp),manager(encap),message(manager)
Descripción:	Constructor de la clase.
Nombre:	openConnectionCpp()
Descripción:	Se encarga de llamar al método connect((const char*) (hostName), port1, 20) de la clase TransportLayer pasándole como parámetro un número de IP, el puerto y un timeout, y deja que el manejador le avise.
Nombre:	openRegisterSession(unsigned result)
Descripción:	Se encarga a partir del valor dado por parámetro, llamar al método registerSession() de la clase EthernetLayer, y deja que el manejador le avise de lo contrario lanzaría un error .
Nombre:	onHandleOpenRegisterSession (unsigned result)
Descripción:	Se encarga de llamar al método forwardOpen (mrouter), pasándole como parámetro un objeto creado de tipo Path, y deja que el manejador le avise, sino lanzaría un error.
Nombre:	onHandleForwardOpen(unsigned result)
Descripción:	Se encarga a partir de un valor dado por parámetro de llamar a la función onOpenC (true).

Nombre:	readTagCpp(Utils::LPStr tagBase, bool isBitArray, Utils::DWord start, Utils::DeviceVarType &dataType, vector<EthernetIP::EthernetIPUnionValue> &data)
Descripción:	Se encarga de llamar al método readData ((const char*)(tag), count, logixType) de la clase Logix, y deja que el manejador le avise.
Nombre:	onHandleReadTag(VectorNode dataBuffer, unsigned result, Word logixType)
Descripción:	Se encarga de leer tag del manejador llamando la función onFuncRead (dataType, data, result), pasándole como parámetro, el tipo de dato a leer, el dato, y el error existente.
Nombre:	writeTagCpp (Utils::LPStr tagName,Utils::DWord start, Utils::DeviceVarType dataType, vector<EthernetIP::EthernetIPUnionValue> data)
Descripción:	Se encarga de llamar al método writeData ((Utils::LPStr) tag, count, logixType, dataBufferWrite) de la clase Logix, y deja que el manejador le avise.
Nombre:	onHandleWriteTag(unsigned result)
Descripción:	Se encarga de escribir tag en un manejador llamando la función onFuncWriteTag(result), pasándole como parámetro un error existente.
Nombre:	writeBitsCpp(Utils::LPStr tagName, vector<Utils::DWord> address, vector<EthernetIP::EthernetIPUnionValue> data)
Descripción:	Se encarga de escribir Bit en el manejador.
Nombre:	onHandleWriteBitRead(Utils::DeviceVarType dataType, vector<EthernetIP::EthernetIPUnionValue> toRead, unsigned result)
Descripción:	Se encarga de escribir Tag , llamando la función onHandleWriteBitWrite (unsigned result).
Nombre:	onHandleWriteBitWrite(unsigned result)
Descripción:	Se encarga de llamar a la función onFuncWriteTag(result), pasándole como parámetro la variable result.
Nombre:	onHandleCloseConnection(unsigned result)
Descripción:	Se encarga de llamar al método unregisterSession() de la clase EthernetLayer, y

	deja que el manejador le avise.
Nombre:	onHandleUnregisterSession(unsigned result)
Descripción:	Se encarga de llamar al método disconnect() de la clase TransportLayer para desconectar el socket, luego llama a la función onFuncCloseConnection(result), pasándole por parámetro un error existente.
Nombre:	setType(Utils::DeviceVarType &dataType, Word &logixType, bool forRead)
Descripción:	Se encarga a partir del booleano que se pasa por parámetro cambiarle el tipo de dato al mensaje.
Nombre:	setType(Utils::DeviceVarType &dataType)
Descripción:	Se encarga de llamar al método setType(Utils::DeviceVarType &dataType, Word &logixType, false).
Nombre:	getValueAsInteger(unsigned index, Word logixType)
Descripción:	Se encarga de llamar el método getValueAsFloat(index, logixType).
Nombre:	getValueAsFloat(unsigned index, Word logixType)
Descripción:	Se encarga de devolver la longitud del dato.
Nombre:	getHost()
Descripción:	Se encarga de devolver la dirección del manejador.
Nombre:	setPort(Utils::DWord newPort)
Descripción:	Se encarga de establecer un nuevo puerto, a partir del nuevo número de puerto que se desea establecer, y que es pasado por parámetro.
Nombre:	getPath()
Descripción:	Se encarga de devolver el resto del path, exceptuando la dirección del manejador.
Nombre:	getPathSize()
Descripción:	Se encarga de devolver el tamaño del path.
Nombre:	getPLCPath()
Descripción:	Se encarga de devolver el path del PLC.
Nombre:	parsePath(Utils::LPStr strpath)
Descripción:	Se encarga de establecer el valor de los atributos path y hostName.
Nombre:	getSerial(void)
Descripción:	Se encarga de devolver el número del pid del proceso que esté en ejecución.
Nombre:	setRetriesCount(Utils::DWord newRetriesCount)

Nombre:	buildTag(Utils::LPStr tagBase, Utils::DWord start, Utils::LPStr tag)
Descripción:	Se encarga de construir el tag.
Nombre:	iToA(unsigned int number)
Descripción:	Se encarga de convertir un valor entero en la cadena correspondiente.
Nombre:	getDWordArray(std::vector<Utils::DWord>&address, std::vector<EthernetIP::EthernetIPUnionValue>& readData, const std::vector<EthernetIP::EthernetIPUnionValue>& data)
Descripción:	Se encarga de devolver la longitud del dato.
Nombre:	getBoolArray(Utils::DWord start, std::vector<EthernetIP:: EthernetIPUnionValue> &data);
Descripción:	Se encargará de devolver la longitud del dato expandido.

Tabla 14. Descripción de la clase TransportLayer.

Nombre	TransportLayer
Tipo de clase	Entidad
Atributo	**Tipo**
resolver_	boost::asio::ip::tcp::resolver
socket_	boost::asio::ip::tcp::socket
request_	boost::asio::streambuf
&io_service_	boost::asio::io_service
m_HiddenDelegate	FunctionA
openConnection	OpenC
Para cada responsabilidad:	
Nombre:	TransportLayer(boost::asio::io_service &io_service): io_service_(io_service), timer_(io_service),resolver_(io_service), socket_(io_service),strand_(io_service)
Descripción:	Constructor de la clase.
Nombre:	~TransportLayer()
Descripción:	Destructor de la clase.

Nombre:	(std::string host, std::string port, int timeout)
Descripción:	Se encarga de conectar el socket a un endpoint específico a partir de un puerto y un número de IP.
Nombre:	onHandleResolve(const boost::system::error_code& err, tcp::resolver::iterator endpoint_iterator, int timeout)
Descripción:	Se encarga de que si el socket no está conectado, abrir una nueva conexión, teniendo en cuenta su timeout pasado por parámetro, de lo contrario, se lanzaría un error.
Nombre:	onHandleConnect(const boost::system::error_code& err)
Descripción:	Se encarga a partir del error pasado por parámetro de devolver un número de operaciones canceladas, junto con un error.
Nombre:	send(Stream& data)
Descripción:	Se encarga de enviar el dato pasado por parámetro llamando al método: io_service_.post(boost::bind(&TransportLayer::send_do,this,bytes,len)), pasándole la longitud y los bytes de dicho dato.
Nombre:	send_recive(Stream& data,int timeout)
Descripción:	Se encarga de enviar lo que recibió por parámetro, llamando al método: io_service_.post(boost::bind(&TransportLayer::send_do,this,bytes,len)), pasándole la longitud ,los bytes de dicho dato, y el timeout para esta operación.
Nombre:	onHandleRecive(const boost::system::error_code& err)
Descripción:	Se encarga a partir del error pasado por parámetro de llamar a la función onAsynRecive(buffer, size_,success), pasándole a la misma el buffer, y el tamaño del mensaje, junto al error correspondiente. De lo contrario se pasaría el buffer y el tamaño inicializado en cero, junto al error correspondiente.
Nombre:	onHandleSendDo(const boost::system::error_code& err)
Descripción:	Se encarga de llamar a la función onAsynSend(success), pasándole por parámetro un error existente.
Nombre:	isConnected()
Descripción:	Se encarga de preguntarte si está conectado.
Nombre:	disconnect()

Descripción:	Se encarga de cerrar el socket.
Nombre:	onTimedOutConnection(const boost::system::error_code& err)
Descripción:	Se encarga de que mientras la operación no esté cancelada, se cierre el socket y llame a la función onOpenConnection(success+33), pasándole así el error 33, relacionada al timeout.
Nombre:	onTimedOutRecive(const boost::system::error_code& err)
Descripción:	Se encarga de que mientras la operación no esté cancelada, se cierre el socket y llame a la función onAsynRecive(buffer,length,success+34), pasándole por parámetro al buffer y el tamaño del mensaje inicializado en cero, junto al error 34 generado.
Nombre:	cancelTimer()
Descripción:	Se encarga de guardar en la variable NOC_ la cantidad de operaciones asincrónicas que se cancelan.

Tabla 15. Descripción de la clase EthernetLayer.

Nombre	EthernetLayer
Tipo de clase	Entidad
Atributo	**Tipo**
head	Header
sessionReady	bool
sessionHandle	Word
sessionContextH	DWord
sessionContextL	DWord
& transp	TransportLayer
errorFrame	int
onAsynRecive	FunctionA
onAsynSend	FunctionB
onRegisterSession	FunctionB
onUnRegisterSession	FunctionB
Para cada responsabilidad:	
Nombre:	EthernetLayer (TransportLayer& clien)
Descripción:	Constructor de la clase.
Nombre:	~EthernetLayer()
Descripción:	Destructor de la clase.

Nombre:	registerSession()
Descripción:	Se encarga de llamar al método send(request) de la clase TransportLayer, y deja que el manejador le avise.
Nombre:	onHandleRegisterSessionSend(unsigned code)
Descripción:	Se encarga de llamar al método recive() de la clase TransportLayer, y deja que el manejador le avise.
Nombre:	ReciveRegisterSessionHead(char * buffer, int size,unsigned code)
Descripción:	Se encarga de llamar al método recive(head.length,timeout) de la clase TransportLayer, y deja que el manejador le avise.
Nombre:	onHandleRegisterSession(char * buffer, int size,unsigned code)
Descripción:	Se encarga de de llamar a la función onRegisterSession(error), pasándole como parámetro el error correspondiente al estado de la cabecera.
Nombre:	unregisterSession()
Descripción:	Se encarga de llamar al método recive(int size,int timeout), cerrando así la sesión leída, devolviendo el error correspondiente, y deja que el manejador le avise.
Nombre:	onHandleUnregisterSession(char * buffer, int size, unsigned code)
Descripción:	Se encarga de llamar a la función onUnRegisterSession(error), pasándole el error correspondiente que se le pasa por parámetro.
Nombre:	sendRRData (Stream& data, DWord timeout)
Descripción:	Se encarga de llamar al método commonSendData(data, timeout) , pasándole por parámetro el dato, y el timeout correspondiente.
Nombre:	sendUnitData (Stream& data, DWord timeout)
Descripción:	Se encarga de llamar al método commonSendData(data, timeout) , pasándole por parámetro el dato, y el timeout correspondiente.
Nombre:	commonSendData (Stream& data, DWord timeout)
Descripción:	Se encarga de llamar al método de la send de la clase TransportLayer para que el mensaje se envíe.
Nombre:	onHandleCommonSendData (unsigned code)
Descripción:	Se encarga de llamar al método recive(int size,int timeout) de la clase TransportLayer, y deja que el manejador le avise.
Nombre:	commonReciveDataHead(char * buffer, int size,unsigned code)
Descripción:	Se encarga de llamar al método recive(head.length,timeout) de la clase TransportLayer, y deja que el manejador le avise.

Nombre:	commonReciveData(char * buffer, int size,unsigned code)
Descripción:	Se encarga de llamar a la función onAsynRecive(aux_,data.size(),error) Pasándole por parámetro una variable aux, el tamaño del dato, y el error correspondiente.

Tabla 16. Descripción de la clase CIPLayer.

Nombre	CIPLayer	
Tipo de clase	Entidad	
Atributo		**Tipo**
service		Byte
manager		Connection
Para cada responsabilidad:		
Nombre:	CIPLayer(Connection& conn)	
Descripción:	Constructor de la clase.	
Nombre:	~CIPLayer()	
Descripción:	Destructor de la clase.	
Nombre:	getAttributeAll (Path objOrClass, Stream& data)	
Descripción:	Se encarga de llamar al método unconnectedSend(getAttributeAllService, objOrClass, data) de la clase Connection.	
Nombre:	setAttributeAll (Path objOrClass, Stream& data)	
Descripción:	Se encarga de llamar al método unconnectedSend(getAttributeAllService, objOrClass, data) de la clase Connection y establece los atributos.	
Nombre:	getAttributeSingle (Path attribute, Stream& data)	
Descripción:	Se encarga de llamar al método unconnectedSend(getAttributeSingleService, attribute, data) de la clase Connection.	
Nombre:	setAttributeSingle (Path attribute, Stream& data)	
Descripción:	Se encarga de llamar al método unconnectedSend(getAttributeSingleService, attribute, data) de la clase Connection y establece el atributo.	

Tabla 17. Descripción de la clase Logix.

Nombre	Logix
Tipo de clase	Entidad
Atributo	**Tipo**
reserved	Word

status	Word
type	Byte
onHandleRead	FunctionA
onHandleWrite	FunctionB

Para cada responsabilidad:	
Nombre:	Logix(Connection& conn)
Descripción:	Constructor de la clase.
Nombre:	~Logix()
Descripción:	Destructor de la clase.
Nombre:	readData (std::string tag, Word count, Byte typ)
Descripción:	Se encarga de leer Dato, devuelve un error correspondiente.
Nombre:	onHandleReadData(char * mess, int size, unsigned code)
Descripción:	Se encarga de llamar el método onHandleRead(data,error,type), de lo contrario devuelve un error.
Nombre:	writeData (std::string tag, Word count, Byte type, VectorNode& data)
Descripción:	Se encarga de escribir datos, devuelve un error correspondiente.
Nombre:	createIOI (std::string tag, Path & createdPath)
Descripción:	Se encarga de crear los segmentos IOI.
Nombre:	createArray (std::string tag, Path & createdPath)
Descripción:	Se encarga de crear path llamando el método addSegment(new Logical(memberIDSegment, logicalFormat, value)) de la clase Path.
Nombre:	createString (std::string tag, Path & createdPath)
Descripción:	Se encarga de llamar el método addSegment(new Logical(memberIDSegment, logicalFormat, value)) de la clase Path.

Conclusiones

Se realizó un estudio profundo del protocolo EtherNet/IP y su origen. Además se abundó en el protocolo que este encapsula, CIP.

Se analizaron las soluciones existentes y se expusieron las ventajas y desventajas que presentaban ante la polémica presentada en el Proyecto Guardián del ALBA.

Se estudiaron las principales tecnologías que presentaban fuertes características para el desarrollo de nuestra aplicación dentro de estas se escogieron:

- C++ como lenguaje de desarrollo.
- RUP como metodología de desarrollo.
- UML como lenguaje de modelado.
- Eclipse y Visual Studio como IDEs de desarrollo.
- RSA como herramienta de modelado.

Se examinó la biblioteca Boost.ASIO la cual proporciona funciones asíncronas sobre el protocolo TCP/IP. Boost.Bind y Boost.Function estas permiten crear los punteros a funciones miembros. Estas tres bibliotecas en un trabajo conjunto permitieron realizar la interfaz asíncrona.

Se demuestra la viabilidad de usar las llamadas asíncronas de ASIO, y los apuntadores a función de la Boost.Bind y la Boost.Function para la implementación de protocolos asíncronos.

Basado en la metodología RUP se capturaron los requisitos y se modeló el diagrama de caso de uso. Se diseñó una arquitectura 2-capas estable y escalable. Esta consta de la capa de Protocolo y la capa de Trasporte. Se modelaron las clases y se realizó una descripción de cada una de ellas.

Se implementaron las principales funcionalidades que requiere una aplicación de este tipo. Con el desarrollo de esta interfaz asíncrona se cumple con los objetivos propuestos para este trabajo. Es multiplataforma, se basa en estándares abiertos y presenta una documentación abundante. Además que la incorporación de la misma al SCADA o cualquier otro software que necesite comunicarse con un PLC Logix, ahorrará los recursos del sistema de cómputo donde este se encuentre.

Recomendaciones

Se exhorta:

- Implementar los servicios que faltan a la interfaz asíncrona, tales como: obtener los atributos de un PLC Logix.
- Diversificar los dispositivos con los cuales la interfaz asíncrona pueda comunicarse (PLC-5, SLC500, Micrologix, FlexLogix).
- Proponer la capa de transporte o su filosofía de trabajo a los proyectos de realidad virtual.
- Completar un conjunto de pruebas exhaustivas de rendimiento bajo condiciones de stress.

Bibliografía Citada

1. **NACIONAL, SCADA.** *Ingenería Conceptual Proyecto Desarrollo SCADA Nacional.* 2005.

2. *Soberanía plena en soluciones AIT para el sector energético aportando valor social.* **PDVSA.SA.**

3. **Stallings, William.** *Comunicaciones y Redes de Computadoras.* Hardcover : Prentice Hall, 1999.

4. **Wikipedia.** Modelo OSI. [En línea] 2 de Noviembre de 2007. [Citado el: 4 de Noviembre de 2007.] http://es.wikipedia.org/wiki/Modelo OSI.

5. **Meijomence, Javier.** Modelo de Referencia OSI. *"Teleinformática y Redes".* 1998.

6. *Redes Computadoras.* La Habana : Publiaciones Felix Varela, 2004.

7. **Automation GmbH.** EtherNet/IP Scanner Software. [En línea] 2004. [Citado el: 4 de Noviembre de 2007.] http://www.ixxat.de/eip_scanner_en,20402,5873.html.

8. El interfaz socket. [En línea] 2006. [Citado el: 4 de Diciembre de 2007.] http://www.lcc.uma.es/~eat/services/i_socket/i_socket.html#link1.

9. **Douglas.** *"Internetworking with TCP/IP".* Prentice Hall : s.n., 1995. Vols. I: Principles, Protocol, and Architecture.

10. **DeGaspari, Jhon.** Memagazine. [En línea] 2001. [Citado el: 10 de Enero de 2008.] http://www.memagazine.org/backissues/sept01/features/twobuses/twobuses.html.

11. **FIGUEROLA LÓPEZ, Ing. Mariano.** TECNOLOGIAS DE TELECOMUNICACIONES APLICADAS A TRANSMISIÓN DE DATOS DE PROCESOS INDUSTRIALES. *ITBA. Instituto tecnologico de Buenos Aires.* [En línea] 2005/2006. [Citado el: 15 de Junio de 2008.] http://www.itba.edu.ar/capis/epg-tesis-y-tf/lopezfiguerola-tfe.pdf.

12. Wikipedia. [En línea] 10 de Junio de 2008. [Citado el: 21 de Junio de 2008.] http://es.wikipedia.org/wiki/Ethernet.

13. **CISCO.** Ethernet. [En línea] 12 de Octubre de 2006. [Citado el: 6 de Noviembre de 2007.] http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ethernet.htm#wp1026438.

14. **Siemon.** Industrial application layer protocol for industrial automation applications. *Siemon.* [En línea] 3 de Marzo de 2005. [Citado el: 4 de Noviembre de 2007.] http://www.siemon.com/us/white_papers/03-03-25-ethernet-ip.asp.

15. **Meyer, Adrienne, Jackson, Jhon y MOine, Veronique.** *SCHNEIDER ELECTRIC se convierte en un miembro principal de ODVA.* Ann Arbor : s.n., 10 de abril del 2007. págs. 1-3.

16. Blogger. *Blogger.* [En línea] 4 de Enero de 2008. [Citado el: 22 de Marzo de 2008.] http://enriqued-infosec.blogspot.com/.

17. TuxPLC. *TuxPLC.* [En línea] [Citado el: 24 de Mayo de 2008.] http://www.tuxplc.net/.

18. Freshmeat. *Freshmeat.* [En línea] 2008. [Citado el: 24 de Mayo de 2008.] http://freshmeat.net/projects/plcio/.

19. Rockwell Automation. *Rockwell Automation.* [En línea] 2008. [Citado el: 1 de Junio de 2008.] http://www.rockwellautomation.com/rockwellsoftware/design/rslinx/.

20. **Kohlhoff, Christopher M.** Boost.Asio. [En línea] 17 de Mayo de 2007. [Citado el: 23 de Mayo de 2007.] http://www.boost.org.

21. Boost. *Boost.* [En línea] 23 de Abril de 2008. [Citado el: 4 de Mayo de 2008.] http://www.boost.org/.

22. **Pressman, Roger S.** *Ingenería del Software: "Un enfoque Práctico".* Quinta edición. Ciudad Habana : s.n., 2005. págs. 1-340. Vol. I.

23. WillyDev. *WillyDev.* [En línea] 15 de Febrero de 2003. [Citado el: 25 de Marzo de 2008.] http://www.willydev.net/InsiteCreation/v1.0/descargas/Articulos/General/cualxpfddrup.PDF.

24. Zator Systems: Tecnología de la información para el conocimiento. *Zator Systems: Tecnología de la información para el conocimiento.* [En línea] 2008. [Citado el: 20 de Mayo de 2008.] http://www.zator.com/Cpp/E1.htm.

25. **Gonzalez, Carlos.** Introducción a C++ y a la resolución de problemas. [En línea] Febrero de 2008. [Citado el: 15 de Mayo de 2008.] http://tecni.com/es/c.htm .

26. Modelo Paracurricular-Desarrollo de Software. [En línea] 1, Febrero de 2004. [Citado el: 10 de Mayo de 2008.] http://www.capacinet.gob.mx/wb2/eMex/eMex_Programacion_Orientada_a_Objetos.

27. Lenguajes y Ciencias de la Computacion. *Universidad de Malaga.* [En línea] [Citado el: 12 de 11 de 2007.] http://www.lcc.uma.es/~eat/services/i_socket/i_socket.html#link1.

28. Kdevelop. [En línea] http://www.kdevelop.org/index.html?filename=main.html&set_lang=es.

29. **Eclipse, F. D.** Eclipse (Software). [En línea] Diciembre de 2006. [Citado el: 13 de Abril de 2007.] http://es.wikipedia.org.

30. Ventajas de .Net. *Ventajas de .Net.* [En línea] 21 de Octubre de 2007. [Citado el: 12 de Noviembre de 2007.] www.DesarrolloWeb.com/Manuales/Manualsobrelaplataforma.Net.

31. **IBM.** Rational Architect Software. [En línea] 26 de 04 de 08. [Citado el: 07 de 06 de 08.] ftp://fttp.software.ibm.com/software/rational_es/Rational Architect 07esp. pdf.

32. **International, Open DeviceNet Vendor Assoc. & ControlNet.** Encapsulation Protocol. 2001, Vol. 2: EtherNet/IP Adaptation of CIP, págs. 1-22.

33. **Schiffer, Viktor y Automation, Rockwell.** *The Common Industrial Protocol (CIP™) and the Family of CIP Networks.* 2006.

34. **Martinez, Luis.** *Logix: El controlador de automatizacion programable mas avanzado del mundo.* 2005. págs. 1-55.

35. **Automation, Rockwell.** *Reference Manual: Logix5000 Data Access.* EE.UU : s.n., 2000. 1770-6.5.16.

36. **Jacobson, Ivar y Booch, Grady.** *El proceso unificado de desarrollo de software.* Ciudad Habana : s.n., 2004. págs. 1-279. Vol. I.

37. DriverOP. *DriverOP.* [En línea] 2007. [Citado el: 1 de Junio de 2008.] http://www.driverop.com.ar/tutorialesdelphi/threads/capitulo1.htm.

38. Computer Architecture and Networks. *Computer Architecture and Networks.* [En línea] 25 de Marzo de 2007. [Citado el: 1 de Junio de 2008.] http://arco.inf-cr.uclm.es/~david.villa/pensar_en_C++/products/vol2/C11.html.

39. *Structure of Data Server in Supervision and Distributed Control System.* **Trujillo Codorníu, Rafael, y otros.** Santiago de Cuba : 2da International Conference on Automatic Control, 2002. ISBN 84-699-9025-x.

40. **(UCI), Universidad de las Ciencias Informaticas.** Flujo de trabajo de requerimientos. [aut. libro] Ivar JACOBSON y Grady, RUMBAUGH, James BOOCH. *"El Proceso Unificado de Desarrollo de Software".* Ciudad Habana : s.n., 2002, págs. 1-19.

41. Wikipedia. La Enciclopedia Libre. *Wikipedia. La Enciclopedia Libre.* [En línea] 29 de Mayo de 2008. [Citado el: 21 de Junio de 2008.] http://es.wikipedia.org/wiki/Arquitectura_de_software.

42. Teleformacion. *Teleformacion.* [En línea] [Citado el: 25 de Mayo de 2008.] http://teleformacion.uci.cu/Ingeneria de Sotware1/Arquitecturas.

43. **Pressman, Roger S.** *Ingeneria de Software. Un enfoque practico.* La Habana : Felix Varela, 2005. págs. 237-258. Vol. 1.

Bibliografías Consultadas

1. **Wkipedia.** Capa enlace de datos. [En línea] 27 de Octubre de 2007. [Citado el: 4 de Noviembre de 2007.] http://es.wikipedia.org/wiki/Capa_enlace_de_datos.

2. **Systems, Zator.** Programación C++. [En línea] http://www.zator.com/Cpp/E1.htm.

3. Wikipedia. *Hilos en ejecucion.* [En línea] 1 de Junio de 2008. [Citado el: 06 de Junio de 2008.] http://es.wikipedia.org/wiki/Hilo_de_ejecuci%C3%B3n.

4. **Wikipedia.** EtherNet/IP. [En línea] 11 de Octubre de 2007. [Citado el: 4 de Noviembre de 2007.] http://en.wikipedia.org/wiki/EtherNet/IP.

5. —. Common Industrial Protocol. [En línea] 11 de Octubre de 2007. [Citado el: 4 de Noviembre de 2007.] http://en.wikipedia.org/wiki/Common_Industrial_Protocol.

6. —. Transmission Control Protocol. [En línea] 24 de Octubre de 2007. [Citado el: 4 de Noviembre de 2007.] http://es.wikipedia.org/wiki/Transmission_Control_Protocol.

7. —. Wikipedia. La enciclopedia Libre. [En línea] 30 de Octubre de 2007. [Citado el: 4 de Noviembre de 2007.] http://es.wikipedia.org/wiki/User_Datagram_Protocol.

8. Wikipedia, La enciclopedia libre. [En línea] 24 de 10 de 2007. [Citado el: 12 de 11 de 2007.] http://es.wikipedia.org/wiki/Protocolo.

9. The Industrial Era. [En línea] 12 de Noviembre de 2006. [Citado el: 4 de Noviembre de 2007.] http://www.thocp.net/timeline/1972.htm.

10. **Tanembaum, Andrew S.** *Computer Networks.* Cuarta edición. New Jersey : Pearson Education, Inc., 2002. 0-1 3-0661 02-3.

11. **Stallings, William.** *Comunicaciones y Redes de Computadoras.* Hardcover : Prentice Hall, 1999.

12. **Sommerville, I.** *Ingeniería de software.*

13. *DeviceNet Development Considerations.* **Schiffer, Viktor y Romito, Ray.** págs. 1-7.

14. **Sastre Fernández, Sergio M.** *Fundamentos del diseño y la programación orientada a objetos.*

15. **Automation, Rockwell.** Networks and Communications:EtherNet/IP. [En línea] 2006. [Citado el: 4 de Noviembre de 2007.] http://www.ab.com/en/epub/catalogs/12762/2181376/214372/1810894/.

16. **RESSMAN, R. S.** Ingienería del software. Un enfoque práctico. [En línea] 2002. [Citado el: 15 de Mayo de 2007.] http://bibliodoc.uci.cu/pdf/reg02689.pdf.

17. **Trujillo, Rafael.** *Guía de Implementación del Protocolo y del Manejador Ethernet/IP.* Universidad de las Ciencias Informaticas. 2007.

18. —. *Ingenería del Software: "Un enfoque Práctico".* Quinta Edición. Ciudad Habana : s.n., 2005. págs. 341-601. Vol. II.

19. —. *Ingenería del Software: "Un enfoque Práctico".* Quinta edición. Ciudad Habana : s.n., 2005. págs. 1-340. Vol. I.

20. *File Transfer Protocol.* **Postel, J. y Reynolds, J.** 1985.

21. *Simple Mail Transfer Protocol.* **Postel, J.** 1982.

22. *Transmission Control Protocol.* **Postel, J.** 1981.

23. *Internet Control Message Protocol.* **Postel, J.** 1981.

24. *Internet Protocol.* **Postel, J.** 1981.

25. *User Datagram Protocol.* **Postel, J.** 1980.

26. *Network Infrastruture for Ethernet/IP: Introduction and Considerations.* **ODVA.** 2007.

27. **ODVA.** The CIP Advantage™ Technology Overview Series. págs. 3-8.

28. **Optimization, Networking and Emerging.** *Transmisión de datos en OSI.* 2006.

29. **Moldovansky, Anatoly y Automation, Rockwell.** *Utilization of modern switching tecnology in Ethernet/IP Network.*

30. *EtherNet/IP EtherNet/IP -The answer for The answer for Interoperability in Industrial Interoperability in Industrial Automation.* **Mark, Daniels.** págs. 1-36.

31. Introducción a C++ y a la resolución de problemas. *Introducción a C++ y a la resolución de problemas.* [En línea] Febrero de 2008. [Citado el: 12 de Febrero de 2008.] http://www.usabilidadweb.com.ar/.

32. La librería socket.h en Linux. . [En línea] 2006. [Citado el: 4 de Noviembre de 2007.] http://gsyc.es/~juaner/investigacion/pfc2/node21.html.

33. **Urrea, Julio César Chavez.** Protocolos de Red: Protocolo TCP/IP. [En línea] [Citado el: 4 de Noviembre de 2007.] http://www.monografias.com/Computacion/Redes.

34. **JSIG, Enabler, Knake, K y Green, P.** *General Recommendations for EtherNet/IP Developers.* 2005. págs. 1-37. PUB00100.

35. *Ethernet/IP, la red ethernet industrial abierta y estandar.* **Paredes, José.** s.l. : Cypsela S.L.

36. **Jiménez, Luis Miguel y Puerto, Rafael.** *PRÁCTICAS DE TCP-IP . Comunicación mediante SOCKETS.* págs. 1-17.

37. **Jacoboson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de software.* Ciudad Habana : s.n., 2004. págs. 279-438. Vol. II.

38. IPv4 Address Conflict Detection for EtherNet/IP Devices. 24 de julio de 2004, págs. 1-12.

39. **International, Open DeviceNet Vendor Assoc. & ControlNet.** Electronic Data Sheets. *EtherNet/IP Adaptation of CIP Specification and CIP Common Specification.* Vols. 1: CIP Common Specification, 2: EtherNet/IP Adaptation of CIP.

40. —. Introduction to EtherNet/IP. *EtherNet/IP Adaptation of CIP Specification.* 2001, Vol. 2: EtherNet/IP Adaptation of CIP, págs. 1-8.

41. —. Indicators and Middle Layers. *EtherNet/IP Adaptation of CIP Specification and CIP Common Specification.* 2001, Vols. 1: CIP Common Specification, 2: EtherNet/IP Adaptation of CIP.

42. —. Physical Layer. *CIP Common Specification and EtherNet/IP Adaptation of CIP Specification.* 2001, Vols. 1: CIP Common Specification, 2: EtherNet/IP Adaptation of CIP.

43. —. How to Read Specifications in the Object Library. *EtherNet/IP Adaptation of CIP Specification and CIP Common Specification.* 2001, Vol. 1: CIP Common Specification.

44. —. Communication Object Classes. *CIP Common Specification.* 2001, Vol. 1: CIP Common Specification.

45. —. Messaging Protocol. *CIP Common Specification.* 2001, Vol. 1: CIP Common Specification.

46. —. Introduction to CIP. *CIP Common Specification.* 2001, Vol. 1: CIP Common Specification.

47. —. Engineering Units. *EtherNet/IP Adaptation of CIP Specification and CIP Common Specification.* 2001, Vol. 1: CIP Common Specification.

48. **International, Open DeviceNet Vendor Assoc & ControlNet.** Explicit Messaging Services. *EtherNet/IP Adaptation of CIP Specification and CIP Common Specification.* 2001, Vols. 1: CIP Common Specification,2: EtherNet/IP Adaptation of CIP.

49. **International, Open DeviceNet Vendor Assoc. & ControlNet.** Device Profiles. *CIP Common Specification and EtherNet/IP Adaptation of CIP Specification.* 2001, Vols. 1: CIP Common Specification, pag 1-4, 2: EtherNet/IP Adaptation of CIP,pag 1-12.

50. —. Object Library. *EtherNet/IP Adaptation of CIP Specification and CIP Common Specification.* 2001, Vols. 1: CIP Common Specification, 2: EtherNet/IP Adaptation of CIP, 5, págs. 1-18.

51. —. Object Model. *CIP Common Specification and EtherNet/IP Adaptation of CIP Specification.* 2001, Vol. 2: EtherNet/IP Adaptation of CIP, págs. 1-4.

52. —. Mapping of Explicit and I/O Messaging to TCP/IP. *EtherNet/IP Adaptation of CIP Specification.* 2001, Vol. 2: EtherNet/IP Adaptation of CIP.

53. —. Encapsulation Protocol. 2001, Vol. 2: EtherNet/IP Adaptation of CIP, págs. 1-22.

54. —. Data Management. *EtherNet/IP Adaptation of CIP Specification, CIP Common Specification.* 2001, Vols. 1: CIP Common Specification, 2: EtherNet/IP Adaptation of CIP.

55. —. Status Codes. *CIP Common Specification;EtherNet/IP Adaptation of CIP Specification.* 2001, Vol. 1: CIP Common Specification.

56. —. Bridging and Routing. *EtherNet/IP Adaptation of CIP Specification, CIP Common Specification.* Vols. 1: CIP Common Specification, 2: EtherNet/IP Adaptation of CIP.

57. Ethernet/IP. [En línea] 13 de Octubre de 2003. [Citado el: 9 de Febrero de 2008.] http://www.siemon.com/es/white_papers/03-10-13-ethernet-ip.asp.

58. **Hinden, Robert M.** *IP Next Generation Overview.* 1995.

59. *Un configurador de buses de campo "universal".* **Heijma, René.** 4 de Noviembre de 2007.

60. *ATM Networks: Concepts, Protocols, Applications. Reading,MA.* **Handel, R., Huber, N. y and Schroder, S.** 1994.

61. *Recommended Functionality for EtherNet/IP Devices.* **Green Perry, Schiffer Viktor.** 16 de Febrero de 2006, págs. 1-15.

62. *A Primer On Internet and TCP/IP Tools and Utilities.* **G. Kessler, S. Shepard.** 1997.

63. *Performance Test Terminology for EtherNet/IP Devices.* **Gilsinn, James.** 14 de Mayo de 2005.

64. *Gigabit Ethernet: From 100 to 1,000 Mbps.. IEEE Internet Computing.* **Frazier, H., and Johnson, H.** January-February 1999.

65. *Performance Test Terminology for EtherNet/IP Devices.* **EtherNet/IP Performance Workgroup, James Gilsinn.** 2005 : EtherNet/IP Implementors Workshop, 14 de marzo del 2005.

66. **EtherNet/IP Implementors Workshop, Open DeviceNet Vendor Association (ODVA),ControlNet International (CI),.** *Recommended IP Addressing Methods for EtherNet/IP™ Devices.* 10-June-2003.

67. **GmbH, Automation.** EtherNet/IP Adapter Module. [En línea] 2004. [Citado el: 4 de NOviembre de 2007.] http://www.ixxat.de/introduction_ethernet_ip_en,7492,5873.html.

68. **JSIG, Enabler, Knake, K. y Green, P.** *General Recommendations for EtherNet/IP Developers.* 2005. págs. 2-37.

69. **Della Gaspera, Jorge, Navarro, Mario y Rey, Daniel.** Interconexión de Sistemas Abiertos. [En línea] [Citado el: 4 de Noviembre de 2007.] http://web.frm.utn.edu.ar/comunicaciones/default.html.

70. **Plummer, David C.** *An Ethernet Address Resolution Protocol.* 1982.

71. CTV Jet Internet. *http://www.ctv.es/.* [En línea] [Citado el: 12 de 11 de 2007.] http://www.ctv.es/USERS/carles/PROYECTO/cap4/cap4.html.

72. **Craig, Larman.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Primera edición. Ciudad Habana : s.n., 2004. págs. 295-507. Vol. II.

73. **Conner, D. E.** *Internetworking with TCP/IP.* Vols. I, Capitulo 5.

74. **Comer, Douglas E.** *Redes Globales de INformacion con INternet y TCP/IP. Principios basicos, protocolos y arquitectura.* [ed.] Inc Prentice-Hall. 3. págs. 1-628. Vol. 1. 968-880-541-6.

75. **P, Coad.** *Object-Oriented Programming.* s.l. : Yourdon Press.

76. **Catalunya, Universidad Politècnica de, Cuba, Universidad de Oriente de Santiago de y de, Universidad Web.** *Curso de automatización industrial con prácticas de autómatas programables por Internet.* 2004. págs. 1-57.

77. **Duffy, Carolyn Marsan.** Industrial Ethernet standards proliferate. [En línea] 15 de Abril de 2002. [Citado el: 4 de Noviembre de 2007.] http://www.networkworld.com/research/2002/0415spotside.html.

78. *EtherNet/IP: Industrial Protocol White Paper.* **Brooks, Paul.** octubre del 2001, págs. 1-12.

79. *Requirements for Internet Hosts -- Communication Layers.* **Braden, R.** 1989.

80. **Black, U.** *Data Link Protocols.* Englewood Cliffs, NJ : s.n., 1993.

81. *ETHERNET/IP OVERVIEW.* **Automation, Rockwell.** Agosto de 2005, págs. 1-12.

82. **GmbH, Automation.** TCP/IP Protocol Software Introduction. [En línea] 2004. [Citado el: 4 de Noviembre de 2007.] http://www.ixxat.de/introduction_tcp_ip_en,7467,5873.html.

83. *EtherNet/IP™ Deploy Ethernet Technology In Industrial Control Applications.* **Automation, Rockwell.** 2004, págs. 1-12.

84. *Communicating with RA Products Using EtherNet/IP Explicit Messaging.* **Automation, Rockwell.** 7 de Junio de 2001.

85. **Armitage, G y Adams, K.** *Packet Reassembly During Cell Loss. IEEE Network.* September 1993.

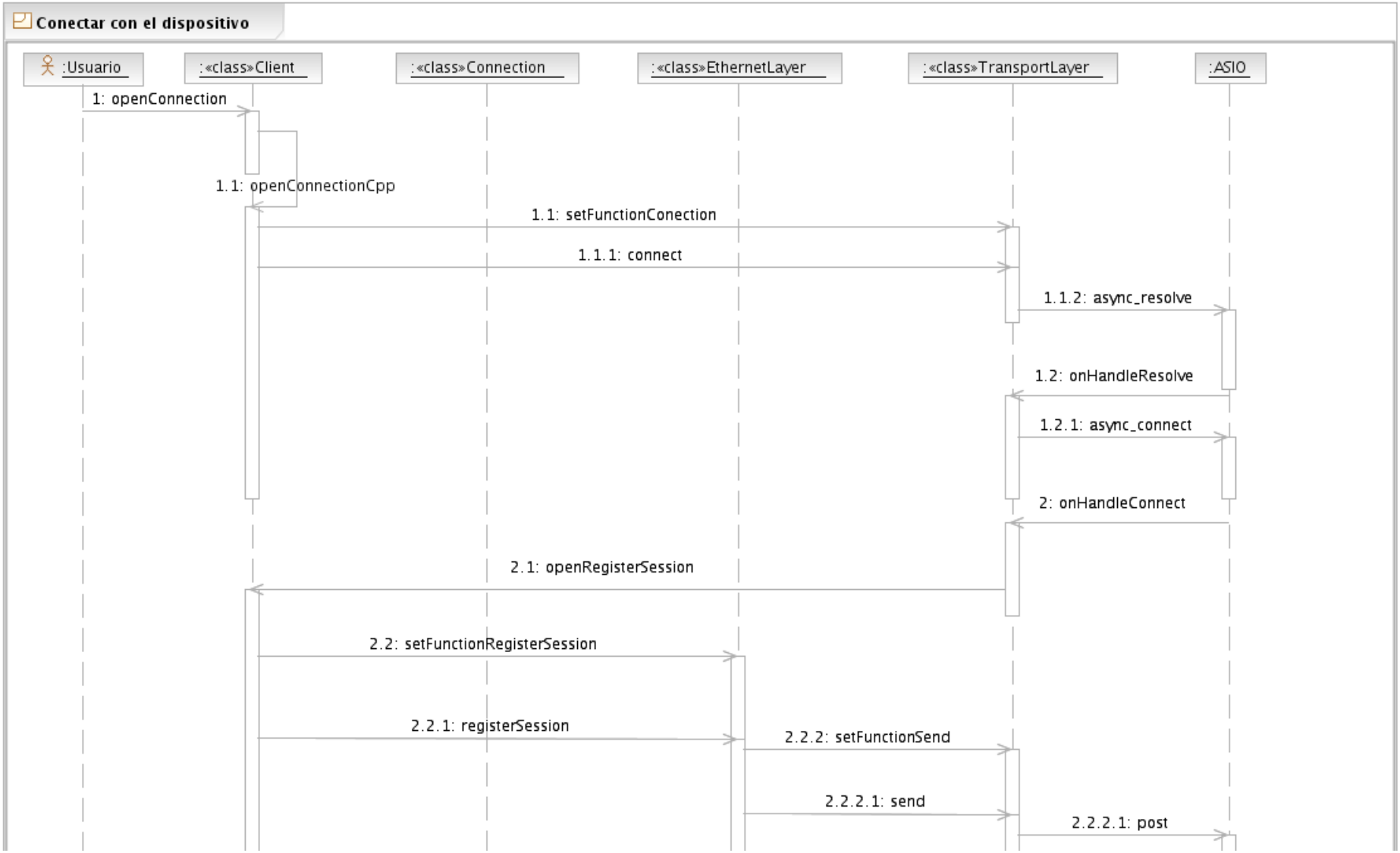
86. *Apuntes de cátedra Programación 3 lenguaje C++.* Universidad Nacional de la Matanza : s.n.

Anexo 1. Diagramas de Secuencia.

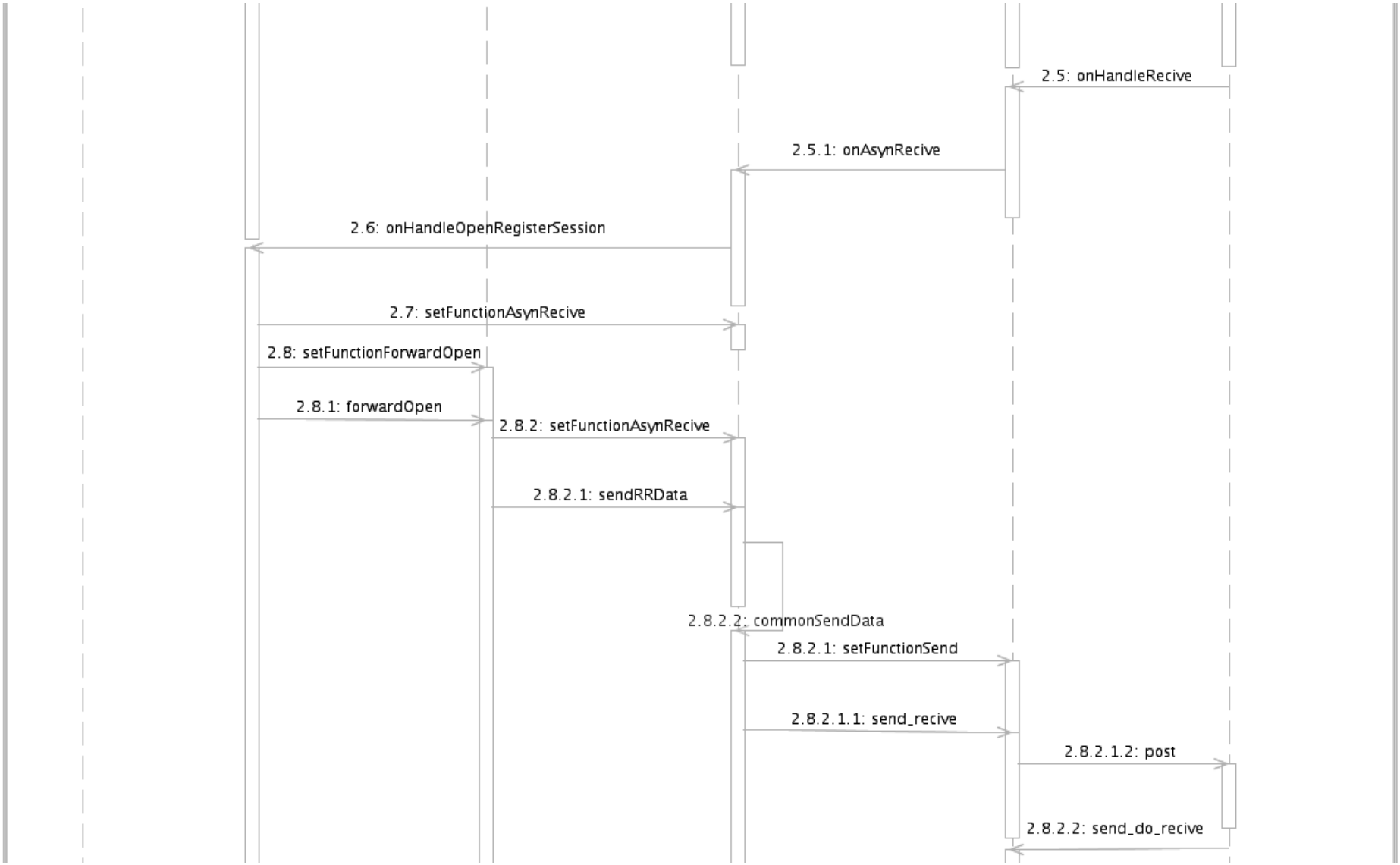
Un diagrama de secuencia muestra las interacciones entre objetos ordenadas en secuencia temporal. Muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por el escenario. En aplicaciones grandes además de los objetos se muestran también los componentes y casos de uso. El mostrar los componentes tiene sentido ya que se trata de objetos reutilizables, en cuanto a los casos de uso hay que recordar que se implementan como objetos cuyo rol es encapsular lo definido en el caso de uso.

Los diagramas de secuencia, formalmente diagramas de traza de eventos o de interacción de objetos, se utilizan con frecuencia para validar los casos de uso. Documentan el diseño desde el punto de vista de los casos de uso. Observando qué mensajes se envían a los objetos, componentes o casos de uso y viendo a grosso modo cuanto tiempo consume el método invocado, los diagramas de secuencia nos ayudan a comprender los cuellos de botella potenciales, para así poder eliminarlos. A la hora de documentar un diagrama de secuencia resulta importante mantener los enlaces de los mensajes a los métodos apropiados del diagrama de clases.

Después de argumentar la importancia de este tipo de diagrama, se muestra los diagramas por casos de uso.


Conectar con el dispositivo
:Usuario
:«class»Client
:«class»Connection
:«class»EthernetLayer
:«class»TransportLayer
:ASIO
1: openConnection
1.1: openConnectionCpp
1.1: setFunctionConection
1.1.1: connect
1.1.2: async_resolve
1.2: onHandleResolve
1.2.1: async_connect
2: onHandleConnect
2.1: openRegisterSession
2.2: setFunctionRegisterSession
2.2.1: registerSession
2.2.2: setFunctionSend
2.2.2.1: send
2.2.2.1: post


2.2.3: send_do
2.2.3.1: async_write
2.3: onHandleSendDo
2.3.1: onHandleRegisterSessionSend
2.3.2: setFunctionRecive
2.3.2.1: recive
2.3.2.2: async_read
2.4: onHandleRecive
2.4.1: ReciveRegisterSessionHead
2.4.2: setFunctionRecive
2.4.2.1: recive_
2.4.2.2: async_read


2.5: onHandleRecive
2.5.1: onAsynRecive
2.6: onHandleOpenRegisterSession
2.7: setFunctionAsynRecive
2.8: setFunctionForwardOpen
2.8.1: forwardOpen
2.8.2: setFunctionAsynRecive
2.8.2.1: sendRRData
2.8.2.2: commonSendData
2.8.2.1: setFunctionSend
2.8.2.1.1: send_recive
2.8.2.1.2: post
2.8.2.2: send_do_recive

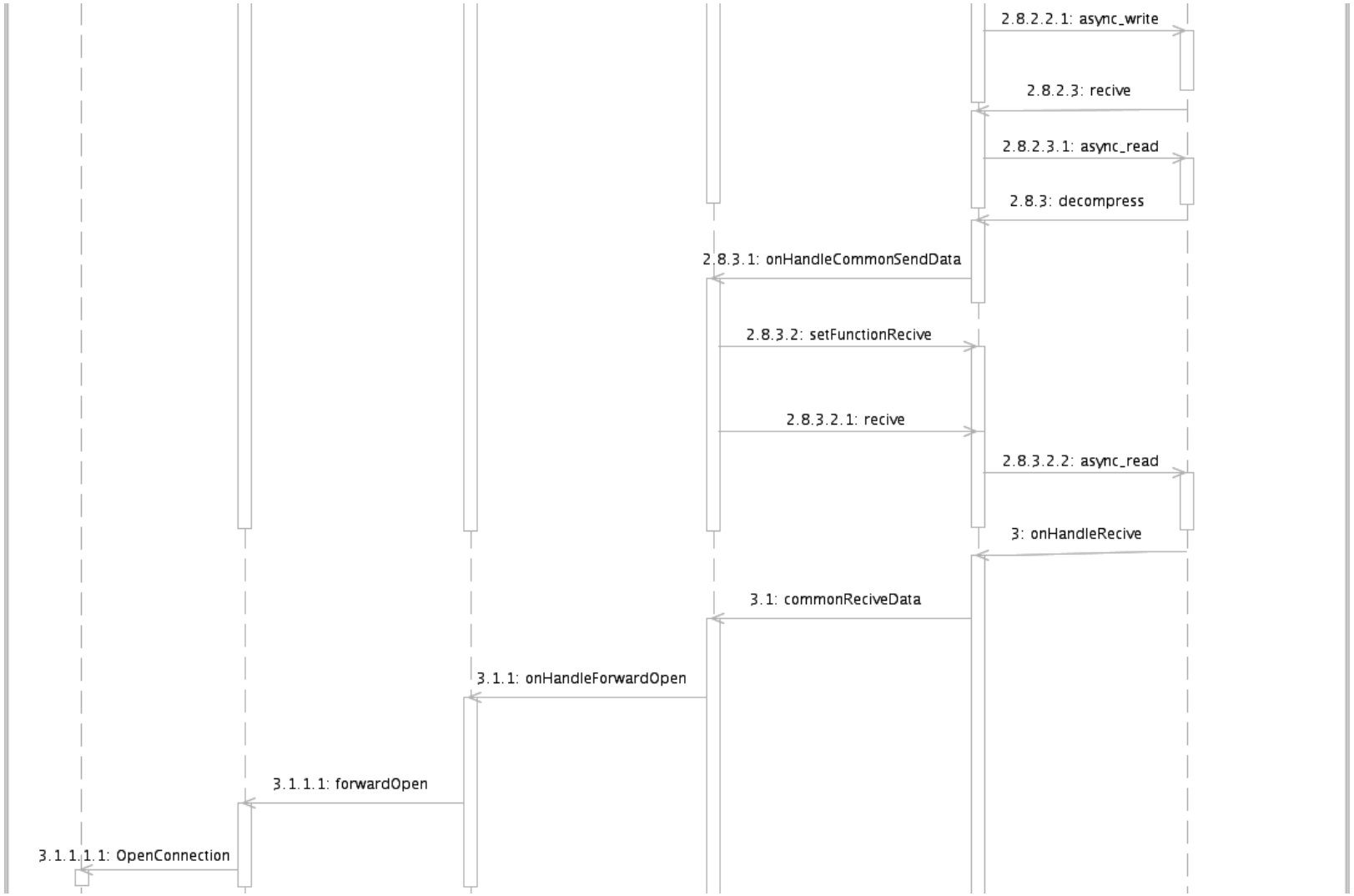
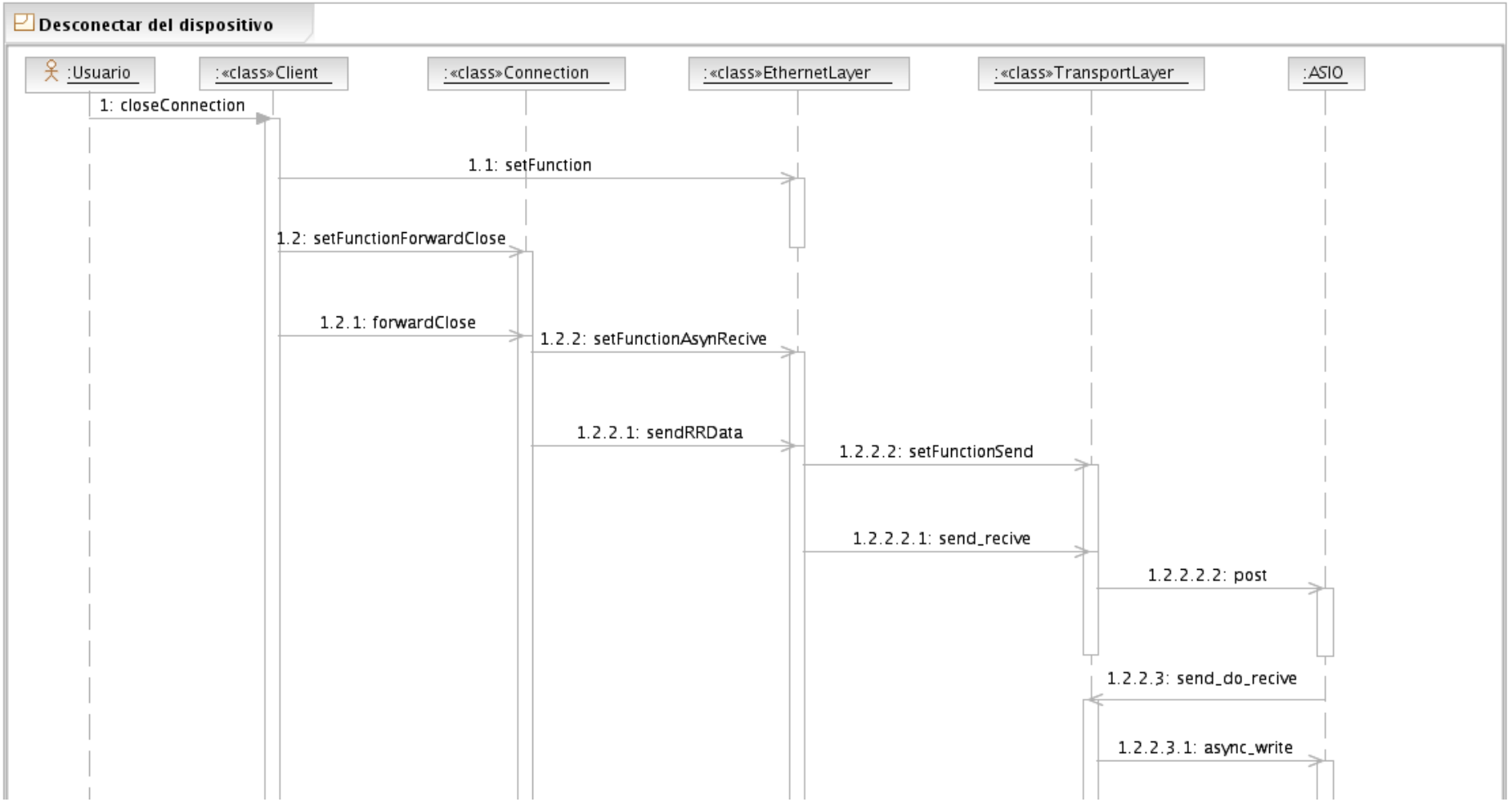
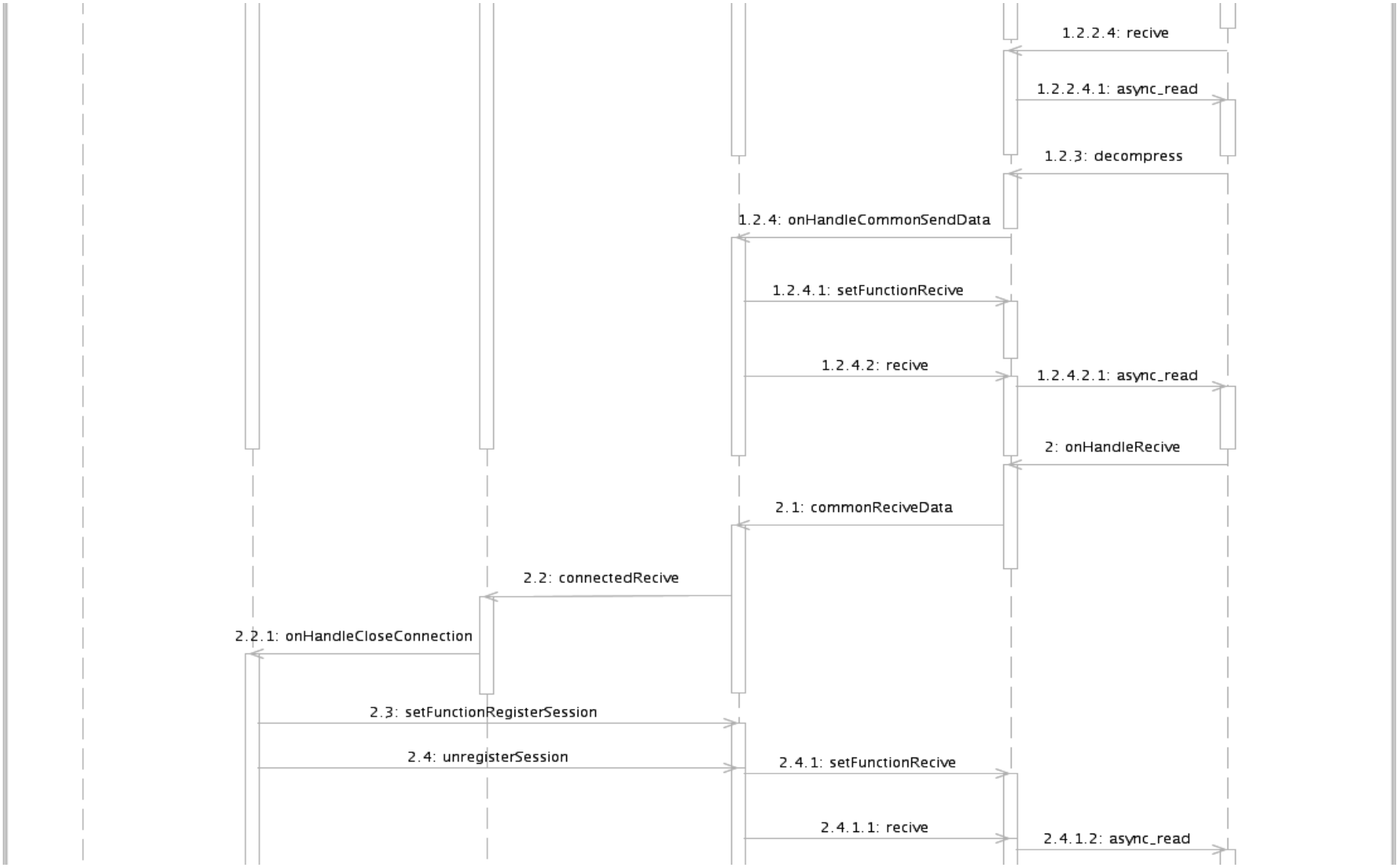


Figura 29. Diagrama de secuencia referente al CU “Conectar con el PLC”.


Desconectar del dispositivo
:Usuario
:«class»Client
:«class»Connection
:«class»EthernetLayer
:«class»TransportLayer
:ASIO
1: closeConnection
1.1: setFunction
1.2: setFunctionForwardClose
1.2.1: forwardClose
1.2.2: setFunctionAsynRecive
1.2.2.1: sendRRData
1.2.2.2: setFunctionSend
1.2.2.2.1: send_recive
1.2.2.2.2: post
1.2.2.3: send_do_recive
1.2.2.3.1: async_write


1.2.2.4: recive
1.2.2.4.1: async_read
1.2.3: decompress
1.2.4: onHandleCommonSendData
1.2.4.1: setFunctionRecive
1.2.4.2: recive
1.2.4.2.1: async_read
2: onHandleRecive
2.1: commonReciveData
2.2: connectedRecive
2.2.1: onHandleCloseConnection
2.3: setFunctionRegisterSession
2.4: unregisterSession
2.4.1: setFunctionRecive
2.4.1.1: recive
2.4.1.2: async_read

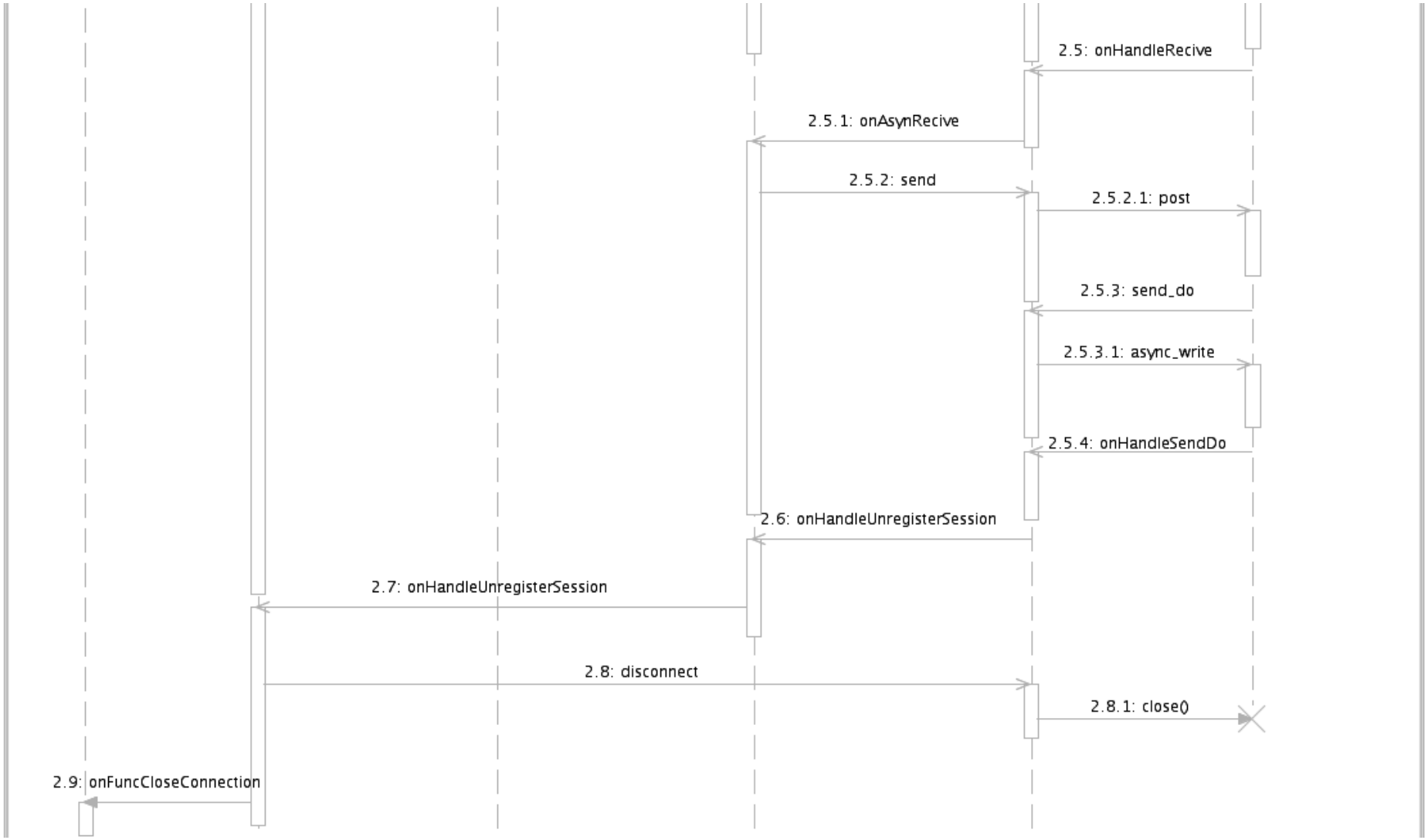
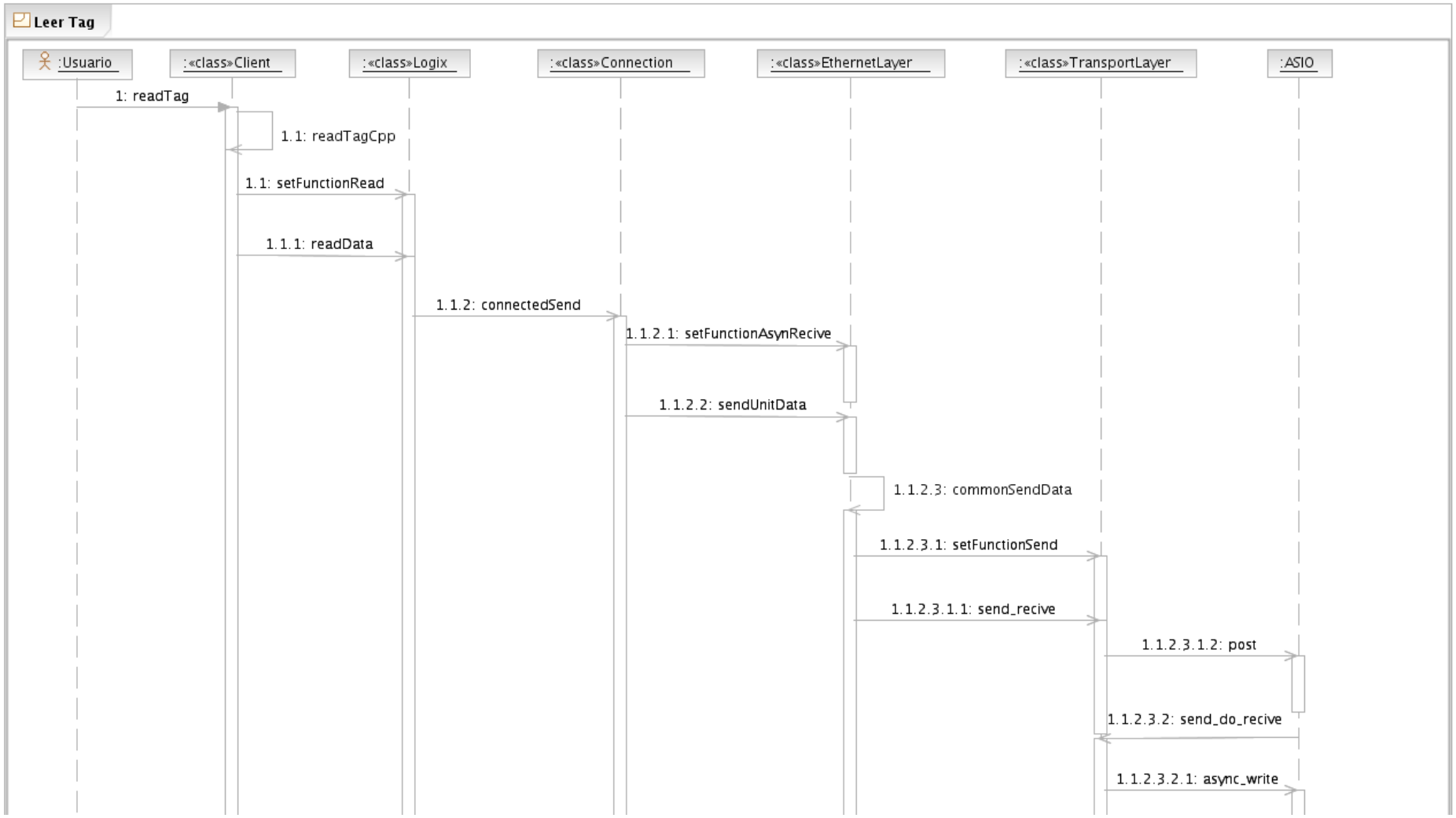


Figura 30. Diagrama de secuencia referente al CU “Desconectar del dispositivo PLC”.


Leer Tag
:Usuario
:«class»Client
:«class»Logix
:«class»Connection
:«class»EthernetLayer
:«class»TransportLayer
:ASIO
1: readTag
1.1: readTagCpp
1.1: setFunctionRead
1.1.1: readData
1.1.2: connectedSend
1.1.2.1: setFunctionAsynRecive
1.1.2.2: sendUnitData
1.1.2.3: commonSendData
1.1.2.3.1: setFunctionSend
1.1.2.3.1.1: send_recive
1.1.2.3.1.2: post
1.1.2.3.2: send_do_recive
1.1.2.3.2.1: async_write


1.1.2.4: onHandleSendDo
1.1.2.5: onHandleCommonSendData
1.1.2.5.1: setFunctionRecive
1.1.2.5.1.1: recive
1.1.2.5.1.2: async_read
1.1.2.6: decompress
1.1.2.6.1: commonReciveDataHead
1.1.2.6.2: setFunctionRecive
1.1.2.6.2.1: recive
1.1.2.6.2.2: async_read
1.1.3: onHandleRecive
1.1.4: commonReciveData

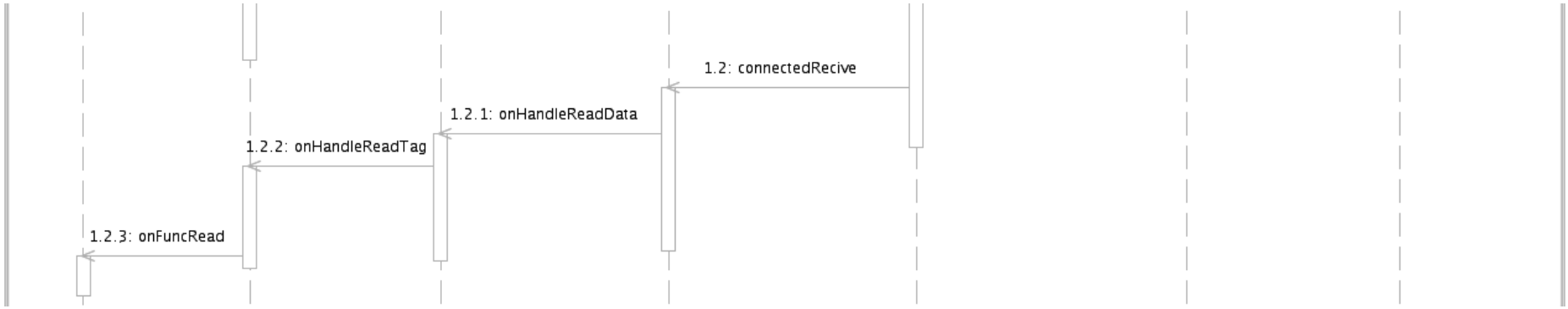


Figura 31. Diagrama de secuencia referente al CU “Leer Tag de un dispositivo PLC”.

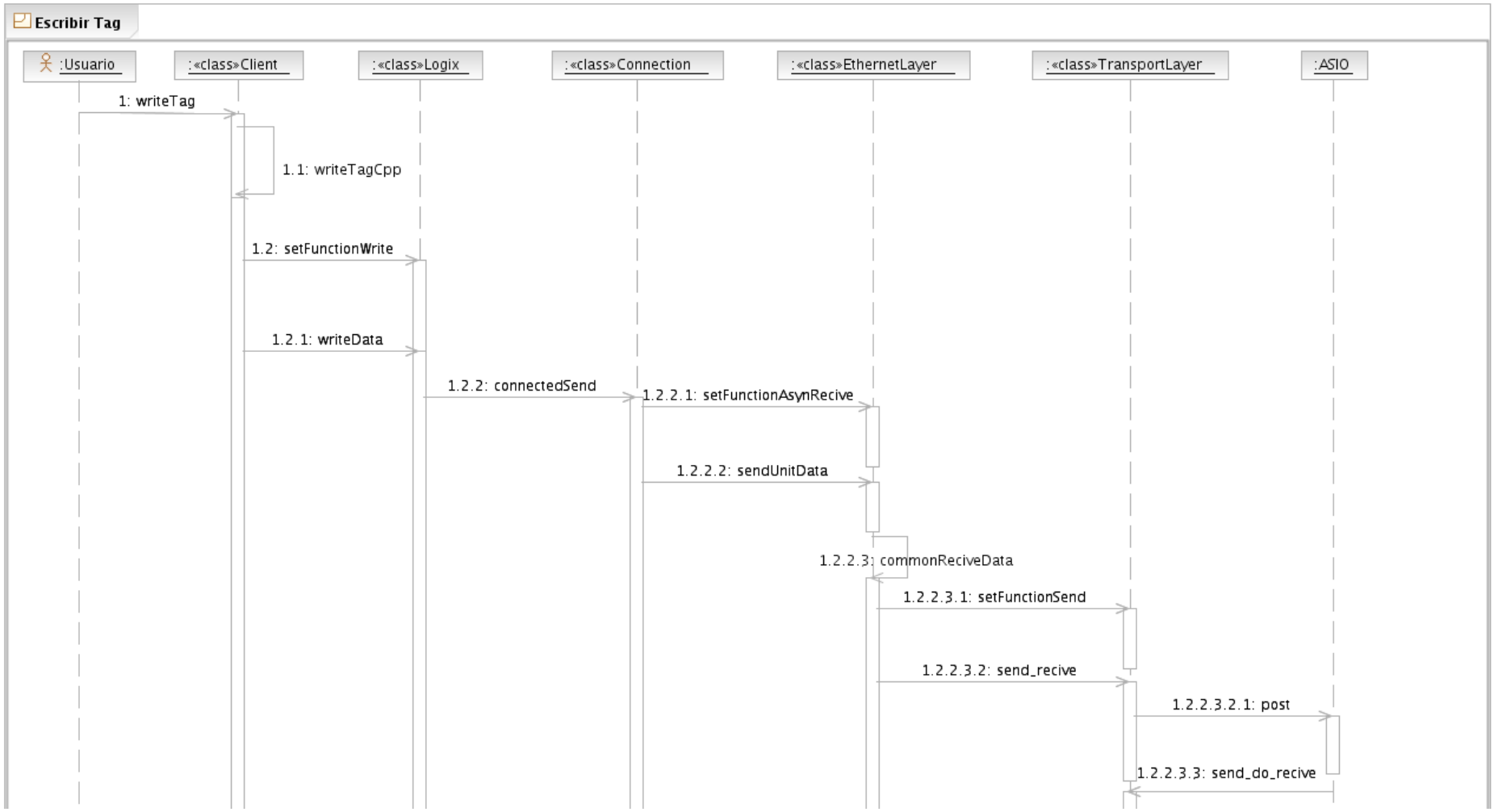

Escribir Tag
:Usuario
:«class»Client
:«class»Logix
:«class»Connection
:«class»EthernetLayer
:«class»TransportLayer
:ASIO
1: writeTag
1.1: writeTagCpp
1.2: setFunctionWrite
1.2.1: writeData
1.2.2: connectedSend
1.2.2.1: setFunctionAsynRecive
1.2.2.2: sendUnitData
1.2.2.3: commonReciveData
1.2.2.3.1: setFunctionSend
1.2.2.3.2: send_recive
1.2.2.3.2.1: post
1.2.2.3.3: send_do_recive

Figura 32. Diagrama de secuencia referente al CU ”Escribir Tag en un dispositivo PLC”.

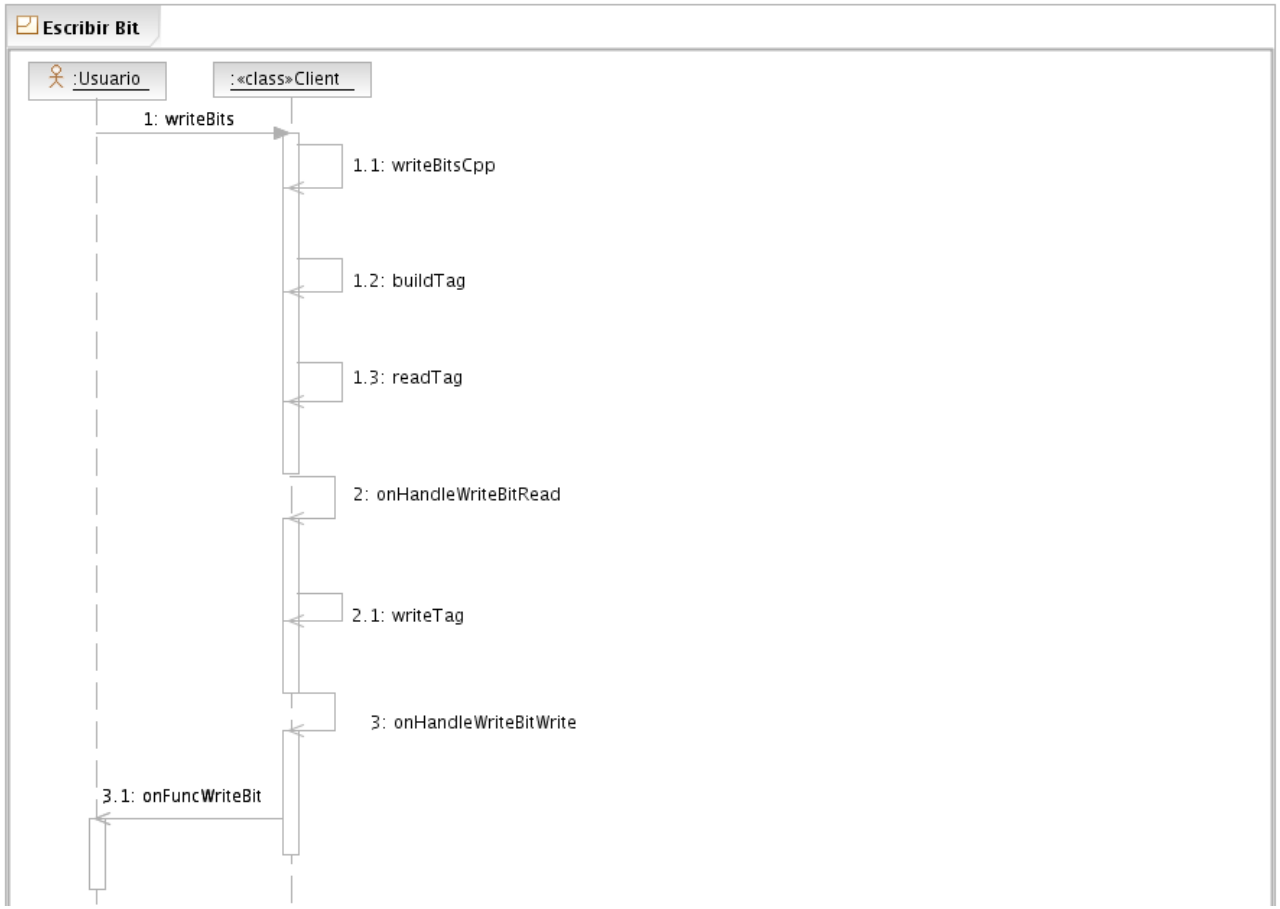


Figura 33. Diagrama de secuencia referente al CU ”Escribir Bit en un dispositivo PLC”.

Anexo 2. Descripción de las Clases del Diseño

Tabla 18. Descripción de la clase MessageRequest.

Nombre	MessageRequest	
Tipo de clase	Entidad	
Atributo		**Tipo**
requestSize		Word
service		Byte
pathSize		Byte
requestPath		Path
requestData		Stream
Para cada responsabilidad:		
Nombre:	getLength()	
Descripción:	Se encarga de devolver la longitud del mensaje.	
Nombre:	readFrom (Stream& input)	
Descripción:	Se encarga de llamar a los métodos setLength(dataSize), readFrom (input) y writeTo (requestData) de la clase VectorNode.	
Nombre:	writeTo (Stream& output)	
Descripción:	Se encarga de escribir para un Stream un requestData de tipo Stream.	

Tabla 19. Descripción de la clase Packet.

Nombre	Packet	
Tipo de clase	Entidad	
Atributo		**Tipo**
itemsCount		Word
addressItem		AbstractItem*
dataItem		AbstractItem*
tmpPackNumber		unsigned
Para cada responsabilidad:		
Nombre:	Packet()	
Descripción:	Constructor de la clase.	
Nombre:	~Packet()	
Descripción:	Destructor de la clase.	

Nombre:	packUnconnectedData (Stream& data)
Descripción:	Se encarga de empaquetar tramas a partir de data que se le pasa por parámetro y una cabecera que se crea.
Nombre:	unpackUnconnectedData (Stream& data)
Descripción:	Se encarga de desempaquetar tramas a partir de data que se le pasa por parámetro y una cabecera que se crea.
Nombre:	packConnectedData (DWord connectionID, Stream& data)
Descripción:	Se encarga de empaquetar tramas a partir de data que se le pasa por parámetro, un id de conexión y una cabecera que se crea.
Nombre:	unpackConnectedData (Stream& data)
Descripción:	Se encarga de desempaquetar tramas a partir de data que se le pasa por parámetro, y un una cabecera que se crea.
Nombre:	createItem (Word type)
Descripción:	Se encarga de crear un nuevo ítem.
Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud del packet.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer un Stream que se le entra por parámetro.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Tabla 20. Descripción de la clase Path.

Nombre	Path
Tipo de clase	Entidad
Atributo	**Tipo**
pathSize	unsigned
segments	QVector<AbstractSegment*>
Para cada responsabilidad:	
Nombre:	Path()
Descripción:	Constructor de la clase.
Nombre:	~Path()
Descripción:	Destructor de la clase.

Nombre:	clearSegments()
Descripción:	Se encarga de eliminar los segmentos, llamando el método clear().
Nombre:	addSegment (AbstractSegment* seg)
Descripción:	Se encarga de adicionar segmentos en una pila, llamando asi el método push_back (seg)
Nombre:	delSegment (unsigned index)
Descripción:	Se encarga de eliminar la posición del segmento.
Nombre:	setPathSize (unsigned byteSize)
Descripción:	Se encarga para actualizar el tamaño del path.
Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud del path.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer nuevos segmentos que van entrando por parámetros, de lo contrario leería un error detectando que el segmento que esta pasando es desconocido.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.
Nombre:	copyFrom (Path& from)
Descripción:	Se encarga de copiar los nuevos segmentos que ya han sido leídos.
Nombre:	createSegment (Byte type)
Descripción:	Se encarga de crear segmentos, devolviendo un nuevo segmento.

Tabla 21. Descripción de la clase Head.

Nombre	Head
Tipo de clase	Entidad
Atributo	**Tipo**
command	Word
length	Word
session	DWord
status	DWord
contextL	DWord
contextH	DWord
option	DWord
Para cada responsabilidad:	

Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud de la cabecera.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer un Stream que se le entra por parámetro.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Tabla 22. Descripción de la clase Data.

Nombre	Data
Tipo de clase	Entidad
Atributo	**Tipo**
subType	Byte
dataLength	Byte
data	VectorNode
Para cada responsabilidad:	
Nombre:	Data()
Descripción:	Constructor vacio de la clase.
Nombre:	Data (Byte subt, unsigned len, char * bytes)
Descripción:	Constructor de la clase.
Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud del segmento.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer un Stream que se le entra por parámetro.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Tabla 23. Descripción de la clase Logical.

Nombre	Logical
Tipo de clase	Entidad
Atributo	**Tipo**
subType	Byte
dataLength	Byte
data	VectorNode
Para cada responsabilidad:	

Nombre:	Data()
Descripción:	Constructor vacío de la clase.
Nombre:	Data (Byte subt, unsigned len, char * bytes)
Descripción:	Constructor de la clase.
Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud del segmento.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer un Stream que se le entra por parámetro.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Tabla 24. Descripción de la clase Network.

Nombre	Network
Tipo de clase	Entidad
Atributo	**Tipo**
productionInhibitTime	Byte
Para cada responsabilidad:	
Nombre:	Network ()
Descripción:	Constructor vacio de la clase.
Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud del segmento.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer un Stream que se le entra por parámetro.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Tabla 25. Descripción de la clase Port.

Nombre	Port
Tipo de clase	Entidad
Atributo	**Tipo**
portIdentifier	Word

linkAddress	VectorNode
Para cada responsabilidad:	
Nombre:	Port ()
Descripción:	Constructor de la clase.
Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud del puerto.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer un Stream que se le entra por parámetro.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Tabla 26. Descripción de la clase Symbolic.

Nombre	Symbolic
Tipo de clase	Entidad
Atributo	**Tipo**
symbol	VectorNode
Para cada responsabilidad:	
Nombre:	Symbolic ()
Descripción:	Constructor vacío de la clase.
Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud del símbolo.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer un Stream que se le entra por parámetro.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Tabla 27. Descripción de la clase ConnectedAddress.

Nombre	ConnectedAddress
Tipo de clase	Entidad
Atributo	**Tipo**
connectionID	DWord

Para cada responsabilidad:	
Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud del id de la conexión.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer un id de conexión de tipo Stream, que se le pasa por parámetro.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Tabla 28. Descripción de la clase ConnectedData.

Nombre	ConnectedData	
Tipo de clase	Entidad	
Atributo		**Tipo**
packetNumber		Word
transportPacket		VectorNode
Para cada responsabilidad:		
Nombre:	getLength()	
Descripción:	Se encarga de devolver la longitud de un número de packet.	
Nombre:	readFrom (Stream& input)	
Descripción:	Se encarga de leer número de packet de tipo Stream, que se le pasa por parámetro.	
Nombre:	writeTo (Stream& output)	
Descripción:	Se encarga de escribir para un Stream.	

Tabla 29. Descripción de la clase NullAddress.

Nombre	NullAddress
Tipo de clase	Entidad
Para cada responsabilidad:	
Nombre:	getLength()
Descripción:	Se encarga de devolver la longitud de una dirección nula.
Nombre:	readFrom (Stream& input)
Descripción:	Se encarga de leer la longitud de una dirección nula de tipo Stream, que se le pasa por parámetro.

Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Tabla 30. Descripción de la clase SequencedAddress.

Nombre	SequencedAddress	
Tipo de clase	Entidad	
Atributos		**Tipo**
connectionID		DWord
packetNumber		DWord
Para cada responsabilidad:		
Nombre:	getLength()	
Descripción:	Se encarga de devolver la longitud de id de una dirección junto con el número del packet..	
Nombre:	readFrom (Stream& input)	
Descripción:	Se encarga de leer un número de packet de tipo Stream, que se le pasa por parámetro.	
Nombre:	writeTo (Stream& output)	
Descripción:	Se encarga de escribir para un Stream.	

Tabla 31. Descripción de la clase UnconnectedData.

Nombre	UnconnectedData	
Tipo de clase	Entidad	
Atributos		**Tipo**
transportPacket		VectorNode
Para cada responsabilidad:		
Nombre:	getLength()	
Descripción:	Se encarga de devolver la longitud del packet..	
Nombre:	readFrom (Stream& input)	
Descripción:	Se encarga de leer el packet de tipo Stream, que se le pasa por parámetro.	
Nombre:	writeTo (Stream& output)	
Descripción:	Se encarga de escribir para un Stream.	

Tabla 32. Descripción de la clase RegisterSession.

Nombre	RegisterSession	
Tipo de clase	Entidad	
Atributos		**Tipo**
protocolVersion		Word
optionFlags		Word
Para cada responsabilidad:		
Nombre:	RegisterSession()	
Descripción:	Constructor de la clase.	
Nombre:	getLength()	
Descripción:	Se encarga de devolver la longitud del campo protocolVersion sumado con la longitud del campo optionFlags .	
Nombre:	readFrom (Stream& input)	
Descripción:	Se encarga de leer el la versión de protocolo y las opciones de banderas de tipo Stream, que se le pasa por parámetro.	
Nombre:	writeTo (Stream& output)	
Descripción:	Se encarga de escribir para un Stream.	

Tabla 33. Descripción de la clase SendRRData.

Nombre	SendRRData	
Tipo de clase	Entidad	
Atributos		**Tipo**
interfaceHandle		DWord
operationTimeout		Word
packedData		VectorNode
Para cada responsabilidad:		
Nombre:	SendRRData ()	
Descripción:	Constructor de la clase.	
Nombre:	getLength()	
Descripción:	Se encarga de devolver la longitud del packedData.	
Nombre:	readFrom (Stream& input)	
Descripción:	Se encarga de leer el packedData de tipo Stream, que se le pasa por	

	parámetro.
Nombre:	writeTo (Stream& output)
Descripción:	Se encarga de escribir para un Stream.

Glosario

SCADA: Acrónimo de Supervisory Control and Data Acquisition. Sistema que se encarga de capturar y manipular la información referente a un proceso automatizado, esta información es utilizada para la realización análisis de indicadores y en la retroalimentación sobre algún operador.

Ethernet: Popular tecnología de red de área local creada en el Palo Alto *Research Center de Xerox Corpo*ration. Ethernet es un sistema de entrega con el mejor esfuerzo que utiliza tecnología CSMA/CD*. Xerox Corporation*, *Digital Equipment Corporation*, e *Intel Corporation* desarrollaron y publicaron el estándar para Ethernet de 10 Mbps. Originalmente, Ethernet utilizaba un cable coaxial. En versiones posteriores empezó a utilizar un cable coaxial delgado o un cable de par trenzado (10Base-T).

ISO (*International Organization for Standardizntion*): Organización internacional que discute, propone y especifica estándares para los protocolos de red. ISO es mejor conocido por su modelo de referencia de siete capas que describe la organización conceptual de los protocolos. Aun cuando se propuso como un conjunto de protocolos para la interconexión de sistemas abiertos, los protocolos OSI no han sido ampliamente aceptados a nivel comercial.

TCP (*Transmission Control Protocol*): Protocolo de nivel de transporte TCP/IP estándar que proporciona el servicio de flujo confiable full dúplex y del cual dependen muchas aplicaciones. El TCP/IP permite que el proceso en una máquina envíe un flujo de datos hacia el proceso de otra. El TCP está orientado a la conexión en el sentido de que, antes de transmitir datos, los participantes deben establecer la conexión. Todos los datos viajan en segmentos TCP, en donde cada viaje se 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.

ISO/OSI: Modelo de referencia para redes de computadoras, basado en capas, propuesto por la *International Standards Organization* en su estándar ISO 7498. El propósito de cada una de las capas (física, enlace de datos, red, transporte, sesión, presentación y aplicación) es proveer un conjunto bien definido de funciones que permitan la interoperabilidad e interconexión entre distintas computadoras.

Unix: Sistema operativo multiusuario y multitarea, desarrollado originalmente por Ken Thompson y Dennis Ritchie en los laboratorios Bell en 1969, para su uso en minicomputadoras. Ofrece múltiples ventajas y se considera potente, más portable e independiente de equipos concretos que otros sistemas operativos.

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

Sockets: Interfaz de comunicación que ofrece un mecanismo de comunicación general entre dos procesos cualquiera que pertenezcan a un mismo sistema, a sistemas diferentes o a sistemas en ordenadores diferentes.

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.

IP: Conjunto de reglas que regulan la transmisión de paquetes de datos a través de Internet. El IP es la dirección numérica de una computadora en Internet de forma que cada dirección electrónica se asigna a una computadora conectada a Internet y por lo tanto es única. La dirección IP esta compuesta de cuatro octetos como por ejemplo, 10.34.4.84.

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

C++: Lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos (POO). C++ está considerado por muchos como uno de los lenguajes más potentes, debido a que permite trabajar tanto a alto como a bajo nivel.

API: Del inglés Application Programming Interface. Interfaz de Programación de Aplicaciones. Un juego de rutinas usados por una aplicación para gestionar generalmente servicios de bajo nivel, realizados por el sistema operativo de la computadora. Uno de los principales propósitos de un API consiste en proporcionar un conjunto de funciones de uso general, de esta forma los programadores se benefician de las ventajas del API, ahorrándose el trabajo de programar todo de nuevo.

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

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

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

PDVSA: Petróleos de Venezuela Sociedad Anónima.

EIA: Alianza de las industrias electrónicas, en inglés, *Electronic Industries Alliance*.

TIA: Asociación de la Industria de las Telecomunicaciones, en inglés, *Telecommunications Industry Association*.

Hilos: Los hilos son las distintas partes en las que un programa se puede dividir para ejecutarse en varias tareas simultáneas o casi simultáneas.

***Gateway*:** Combinación de programa y hardware que comunica dos tipos diferentes de redes.

Callbacks: Comunicación de vuelta, unión o link de comunicación en el cual el servidor comunica al usuario, luego que éste informa su deseo de comunicarse.

CPU: Unidad de proceso central, el corazón del sistema ordenador; contiene la unidad aritmética y lógica, los circuitos de control y la memoria principal.

Switches: Un dispositivo de red capaz de realizar una serie de tareas de administración, incluyendo el redireccionamiento de los datos.

Release: Versión alfa de un software en desarrollo.

Hubs: Concentrador. Dispositivo que se utiliza típicamente en topología en estrella como punto central de una red, donde por ende confluyen todos los enlaces de los diferentes dispositivos de la red.

CVS (*Concurrent Versioning System*): Aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros y permite que distintos desarrolladores colaboren.

IBM: Compañía productora de computadoras reconocida internacionalmente.

Plugin: Un plugin (o plug-in -en inglés "enchufar", también conocido como addin, add-in, addon o add-on) es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica. Ésta aplicación adicional es ejecutada por la aplicación principal.

Multicast (multidifusión): Modo de difusión de información en vivo que permite que ésta pueda ser recibida por múltiples nodos de la red y por lo tanto por múltiples usuarios.

Broadcast (difusión amplia): Término utilizado originariamente en el mundo de la radio y de la televisión para indicar que sus emisiones las puede recibir cualquiera que sintonice una emisora. Hoy en Internet se emite también radio y televisión en modo broadcast, y la misma WWW es un medio de esta misma naturaleza.

Modbus: Protocolo de comunicaciones situado en el nivel 7 del Modelo OSI, basado en la arquitectura cliente-servidor, diseñado en 1979 por Modicon para su gama de controladores lógicos programables.

Profinet: Version de Profibus adaptada a Ethernet.

GCC: Conjunto de compiladores creados por el proyecto GNU. GCC es software libre y se distribuye bajo la licencia GPL.

CVS (*Concurrent Versioning System*): es una aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros y permite que distintos desarrolladores colaboren.

Tag: Nombre simbólico por el cual se accede a la información, es un identificador con no más de 40 caracteres de longitud. Los tags tienen un alcance. De acuerdo a su alcance se clasifican en tags del controlador (o sea globales) a los cuales se pueden acceder directamente y tags del programa (locales) a los cuales no se puede acceder desde un dispositivo externo. Cada tag 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.

Conection Path: Se compone de un stream de octetos que defina un objeto dentro del nodo.

IEEE 802.3: El comité de la IEEE definió un estándar el cual incluye el formato del paquete de datos para EtherNet, el cableado a usar y el máximo de distancia alcanzable para este tipo de redes. Describe una LAN usando una topologia de bus, con un método de acceso al medio llamado CSMA/CD y un cableado coaxial de banda base de 50 ohms capaz de manejar datos a una velocidad de 10 Mbs.

Message Router: El objeto dentro de un nodo que distribuye el mensaje explícito a los objetos que va dirigido el mensaje.

Router: Es un dispositivo con capacidad de procesamiento que conecta dos redes y cuya función principal es retransmitir datos desde una red a otra siguiendo la ruta adecuada para alcanzar al destino.