UNIVERSIDAD NACIONAL DEL COMAHUE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE ELECTROTECNIA

DISEÑO E IMPLEMENTACIÓN DE LABORATORIO BAJO ESTÁNDAR IEC 61850

Proyecto Integrador Profesional presentado por:

ERICK EMANUEL CORTEZ

Ante la Facultad de Ingeniería de la Universidad Nacional del Comahue para acceder al título de:

INGENIERO ELÉCTRICO

Dirección

Director: ING. CASABONA GUSTAVO

Neuquén. Año 2026

Resumen

El estándar IEC 61850 se ha consolidado como una de las principales tecnologías utilizadas en los sistemas modernos de automatización de subestaciones eléctricas, permitiendo la interoperabilidad entre dispositivos electrónicos inteligentes (IED) mediante modelos de datos normalizados y servicios de comunicación avanzados. Sin embargo, su estudio y experimentación práctica presentan dificultades en entornos académicos debido al alto costo de los equipos industriales y la complejidad de las plataformas comerciales disponibles.

En este trabajo se presenta el diseño e implementación de un laboratorio educativo basado en el estándar IEC 61850, orientado al estudio y comprensión de los mecanismos de comunicación utilizados en sistemas de automatización de subestaciones. Para ello se desarrollaron dos dispositivos electrónicos inteligentes mediante programación en lenguaje C utilizando la biblioteca libiec61850. El primero de ellos simula el funcionamiento de un relé de protección ejecutándose en un entorno Windows, mientras que el segundo se implementa sobre una plataforma Raspberry Pi con sistema operativo Linux, representando una unidad de adquisición de datos. Ambos dispositivos implementan servidores IEC 61850 capaces de exponer modelos de datos, intercambiar mensajes GOOSE y permitir la conexión de clientes mediante el servicio MMS. Adicionalmente, el sistema desarrollado se integra con un entorno de simulación y con un sistema de supervisión SCADA, permitiendo la recepción de reportes MMS, la supervisión remota de variables y la ejecución de comandos de operación. La interacción con un tablero físico de comando construido para el laboratorio permite además representar de forma visual el estado de los dispositivos mediante indicadores luminosos y señales locales conectados a los pines GPIO de la Raspberry Pi.

Los resultados obtenidos demuestran el correcto funcionamiento de los mecanismos de comunicación implementados y validan la capacidad del laboratorio para reproducir las principales funciones de un sistema de automatización de subestaciones basado en el estándar IEC 61850.

Palabras Clave: IEC 61850, IED, GOOSE, MMS, Raspberry Pi, libiec61850.

Abstract

The IEC 61850 standard has become one of the most widely adopted technologies in modern substation automation systems, enabling interoperability between Intelligent Electronic Devices (IEDs) through standardized data models and advanced communication services. However, practical experimentation with this standard in academic environments is often limited due to the high cost of industrial equipment and the complexity of commercial platforms.

This work presents the design and implementation of an educational laboratory based on the IEC 61850 standard, intended to support the study and understanding of communication mechanisms used in substation automation systems. Two intelligent electronic devices were developed using the libiec61850 library and programmed in the C language. The first device simulates a transformer protection relay running on a Windows platform, while the second device is implemented on a Raspberry Pi running a Linux operating system, representing a simplified data acquisition unit. Both devices implement IEC 61850 servers capable of exposing data models, exchanging GOOSE messages, and allowing client connections through the MMS service. In addition, the developed system is integrated with a simulation environment and a SCADA supervision system, enabling the reception of MMS reports, remote monitoring of variables, and execution of control commands. The interaction with a physical control panel specifically built for the laboratory allows visual representation of device states through indicator lights connected to the Raspberry Pi digital outputs.

The obtained results demonstrate the correct operation of the implemented communication mechanisms and validate the capability of the laboratory to reproduce the main functions of a substation automation system based on the IEC 61850 standard.

Keywords: IEC 61850, IED, GOOSE, MMS, Raspberry Pi, libiec61850.

Reconocimientos y dedicatorias

En primer lugar, deseo expresar mi agradecimiento a la Universidad Nacional del Comahue y a la Facultad de Ingeniería, instituciones que me brindaron la oportunidad de formarme profesionalmente. Destaco el valor de la educación pública como pilar fundamental que hizo posible no solo mi formación, sino también la de tantos otros compañeros.

Agradezco especialmente a mi director de tesis, Ing. Gustavo Casabona, por su guía, tiempo, paciencia y acompañamiento a lo largo de todo el proceso. Su orientación y perseverancia fue fundamental para el desarrollo de este proyecto.

Quiero también reconocer a mis compañeros de carrera, con quienes compartí innumerables horas de estudio, charlas, discusiones y desafíos. Cada instancia, desde los parciales hasta los finales, estuvo marcada no solo por la dedicación, sino también por el compañerismo, la buena predisposición y, por supuesto, los mates que hicieron más llevadero el proceso. Destaco y agradezco la disposición por parte de la Facultad de Ingenieria de un Aula de estudio, nuestra querida Aula A, donde nos encontramos compartiendo el espacio todos los días, incluso fines de semana o feriados.

En el plano personal, agradezco profundamente a mis padres por el sacrificio y esfuerzo realizado en casa para permitirme estudiar esta carrera. A mis hermanos, Alejandra y Cristian, por estar siempre presentes acompañándome en cada etapa. De igual manera, agradezco a mi grupo de amigos de la infancia, quienes siempre confiaron en mí y me brindaron su apoyo incondicional.

Finalmente, quiero dedicar este trabajo a la memoria de mi madre. Aunque hoy no se encuentre físicamente, su presencia ha sido una fuente constante de motivación para seguir adelante y superar cada obstáculo. Su confianza en mí y su forma de enfrentar la vida me acompañaron en cada momento. Este logro es en gran parte gracias a ella, y representa el cumplimiento de un camino que siempre soñó para mí.

El presente trabajo representa no solo un logro académico, sino también el reflejo del acompañamiento de quienes formaron parte de este camino, marcando así el cierre de una etapa.

Índice

Contenido de Figuras

Contenido de Tablas

1. Introducción

El sector eléctrico está compuesto por diversos elementos que permiten generar energía en ubicaciones estratégicas, para luego ser transportada y distribuida hacia los lugares de consumo. Este proceso puede dividirse en tres grandes sectores: Generación, Transmisión y Distribución.
El sector de generación es el responsable de inyectar energía al sistema eléctrico. Esto lo logra mediante la transformación de otras fuentes de energía en las centrales eléctricas.
El sector de transmisión vincula eléctricamente los centros de generación con los sistemas de distribución, brindándoles el servicio del transporte de la energía de forma eficiente y confiable. Está compuesto principalmente de una red eléctrica mallada, en cuyos nodos se ubican las estaciones transformadoras.
Dichas estaciones son fundamentales ya que cumplen la función de modificar los niveles de tensión para una adecuada transmisión y distribución de energía eléctrica. Además, permiten reconfigurar o aislar parte de la red con el fin de mantener la seguridad y confiabilidad del sistema eléctrico de potencia. Esto se logra mediante los sistemas de protección y control, que cumplen la función de detectar condiciones de falla para actuar en consecuencia y permiten la supervisión y operación del sistema. Para ello, es necesario obtener las señales correspondientes a estados de interruptores, seccionadores y señalizaciones de algunos componentes clave de los equipos que puedan representar una alerta en su funcionamiento. Así mismo, obtienen señales de mediciones de magnitudes eléctricas, como son tensión, corriente y potencia del sistema. A partir de toda esta información los equipos de protección y control toman decisiones, acorde a la función que cumplen en el sistema, y emiten señales de apertura o cierre a los equipos.
Con la aparición de nuevas tecnologías impulsadas por el desarrollo de los microprocesadores los equipos electromecánicos fueron progresivamente reemplazados por equipos digitales. Actualmente los relés digitales poseen una mayor capacidad de procesamiento lo que les permite integrar múltiples funciones en un solo equipo. Estos son conocidos como IEDs (Intelligent Electronic Devices) y forman parte integral de los sistemas de automatización de subestaciones (SAS). Los IED modernos no son simplemente relés de protección sino una unidad multifunción que incorpora capacidades de protección, medición, control y comunicación en un solo dispositivo.
Con la aparición de los equipos digitales y debido a su capacidad de procesamiento, distintos fabricantes comenzaron a incorporar capacidades de comunicación en sus dispositivos mediante protocolos propietarios. Esto produjo una gran mejora permitiendo la integración entre protecciones y SCADA (Supervisory Control and Data Acquisition), pero también

algunos inconvenientes. Algunos de estos inconvenientes puede ser la falta de interoperabilidad debido a que cada equipo solo era compatible con protocolos del mismo fabricante, o el incremento en la complejidad de integración debido a los equipos intermedios como conversores de protocolo y gateway. Esto representa un gran problema desde el punto de vista del mantenimiento ya que cualquier cambio o expansión del sistema implica grandes esfuerzos de ingeniería.

Debido a esta necesidad de modernizar y unificar las comunicaciones es que aparece y toma importancia el estándar IEC 61850. IEC 61850 Communication Networks and Systems in Substations (Redes y sistemas de comunicación en subestaciones), es un estándar internacional para sistemas de automatización y comunicaciones en subestaciones, desarrollado por la Comisión Electrotécnica Internacional cuyo objetivo principal es definir un lenguaje común para la comunicación entre dispositivos en subestaciones eléctricas, eliminando la dependencia de protocolos propietarios y simplificando la integración de sistemas.

Pero este estándar no es tan solo un protocolo de comunicación, sino un concepto más amplio donde se definen las bases para el diseño de una arquitectura de automatización que garantice la interoperabilidad y estandarizar el intercambio de información, permitiendo una gestión y mantenimiento que optimice el sistema eléctrico.

Uno de los pilares fundamentales de este enfoque es el modelo de datos utilizado, que define una estructura jerárquica y orientada a objetos. En lugar de intercambiar información a través de bits o registros sin contexto, como en protocolos tradicionales, IEC 61850 organiza la información en nodos lógicos, que representan funciones específicas de automatización como protección, medición o control.

Para describir y configurar estos modelos de datos y su comportamiento dentro del sistema, IEC 61850 recurre al uso de un lenguaje basado en XML (eXtensible Markup Language), que comparte estructura con el HTML utilizado en la web, aunque con fines distintos. Este lenguaje permite representar de forma legible y estructurada toda la información relacionada con la configuración y funcionamiento de un sistema de automatización. Se emplean distintos tipos de archivos bajo el estándar SCL (Substation Configuration Language) que unifica toda la configuración del sistema completo. Estos archivos permiten no solo la correcta parametrización inicial, sino también una gestión y mantenimiento más eficiente, ya que pueden ser leídos, interpretados y verificados por distintos sistemas y herramientas, independientemente del fabricante.

Además, IEC 61850 define un conjunto de servicios de comunicación estandarizados para garantizar la interoperabilidad funcional entre los dispositivos. Entre los más destacados se

encuentra GOOSE (Generic Object Oriented Substation Event), un servicio de transmisión directa de eventos entre dispositivos, utilizado típicamente para señales de protección y control debido a su baja latencia y fiabilidad. También se incluyen los Sampled Values (SV), utilizados para transmitir señales analógicas digitalizadas. Por último el protocolo MMS (Manufacturing Message Specification) que se utiliza para la supervisión, configuración y control remoto de los dispositivos. MMS actúa como un canal más amplio que permite leer y escribir datos en los modelos de los IEDs, así como ejecutar comandos o realizar diagnósticos.

La combinación de estos elementos permite a IEC 61850 garantizar la interoperabilidad entre dispositivos de distintos fabricantes, facilitar la integración y escalar los sistemas de automatización de subestaciones de forma eficiente y robusta. Esta arquitectura no solo moderniza la comunicación dentro de las subestaciones, sino que también prepara el terreno para una evolución hacia redes eléctricas más inteligentes, flexibles y resilientes.

Entre las principales barreras para la implementación del estándar se identifican la complejidad técnica, la dificultad de integración con sistemas heredados y la limitada disponibilidad de personal capacitado. Esta situación pone de manifiesto la necesidad de generar entornos de formación y experimentación que faciliten la comprensión y aplicación práctica del estándar.

En este contexto, el presente trabajo propone el diseño y desarrollo de un laboratorio de simulación accesible y de bajo costo que permita replicar el funcionamiento de una subestación transformadora simplificada bajo el modelo de automatización basado en IEC 61850. El laboratorio contempla el uso de IEDs virtuales que intercambian información mediante los protocolos GOOSE y MMS, empleando herramientas y librerías como libIEC61850 sobre plataformas Linux (Raspberry Pi) y Windows. Asimismo, mediante el uso de señales de entrada y salida digitales (GPIOs), se representarán condiciones de operación reales sobre un tablero físico, permitiendo simular escenarios típicos de una subestación. Adicionalmente, se incorpora una interfaz SCADA conectada a un gateway de comunicaciones, lo que permitirá la supervisión y el comando del sistema en tiempo real a través de una red LAN simple.

Este trabajo busca no solo demostrar la viabilidad técnica de la aplicación del estándar IEC 61850 en entornos reducidos, sino también constituir un punto de partida para futuras experiencias de implementación, escalamiento y formación profesional, contribuyendo al proceso de modernización de los sistemas de automatización de subestaciones eléctricas.

En los capítulos siguientes se detallarán los objetivos específicos del proyecto, los fundamentos técnicos involucrados, el diseño del entorno de pruebas y el análisis de resultados obtenidos.

2. Objetivos

2.1. Objetivos generales

Investigar la implementación del estándar IEC 61850 en sistemas de automatización de subestaciones digitales, a través del diseño y desarrollo de un laboratorio de simulación accesible y de bajo costo, que permita representar una subestación transformadora simplificada, con el fin de facilitar su estudio, aplicación y adaptación a entornos reales.

2.2. Objetivos específicos

- Diseñar la arquitectura de comunicación del sistema simulado, definiendo los protocolos utilizados y las señales intercambiadas entre los IEDs.
- Simular IEDs virtuales con características funcionales específicas, utilizando librerías open-source como libIEC61850 y otras herramientas disponibles.
- Implementar un IED en una Raspberry Pi capaz de recibir señales digitales a través de entradas GPIO, actuando como unidad de adquisición de datos (merging unit).
- Implementar un IED para el nivel de bahía capaz de establecer comunicación con el nivel inferior y superior mediante los protocolos del estándar IEC 61850.
- Diseñar e implementar un tablero de control físico con pulsadores e indicadores que permitan representar comandos y estados de operación, conectados a la Raspberry Pi.
- Configurar un gateway de comunicación que recoja la información de los IEDs virtuales y permita su visualización mediante una plataforma SCADA.
- Configurar una interfaz SCADA que permita el control y monitoreo de las variables del sistema simulado en tiempo real.
- Validar la funcionalidad de la arquitectura implementada mediante pruebas de comunicación, interacción entre dispositivos y visualización de eventos en el entorno simulado.

2.3. Indicadores de logro

Los siguientes indicadores permitirán contrastar los resultados del proyecto con los objetivos planteados:

- Se establece una red de comunicación funcional basada en el estándar IEC 61850, con intercambio de señales mediante protocolos GOOSE y MMS.

- Los IEDs virtuales emiten y reciben señales correctamente, con modelos de datos coherentes y configuraciones validadas.
- Se logra establecer comunicación y control efectivo de los IEDs virtuales mediante un cliente IEC 61850 validando la correcta implementación de los servicios de comunicación definidos en la norma.
- La Raspberry Pi es capaz de actuar como punto de adquisición de señales físicas y de comunicar eventos al resto del sistema.
- El tablero físico responde correctamente a las acciones de protección y control simuladas.
- La información del sistema se visualiza de manera estable en la interfaz SCADA, y los comandos remotos son recibidos y ejecutados en tiempo real.
- Toda la configuración del entorno es replicable en otra instalación, utilizando hardware y software accesibles, sin necesidad de licencias costosas ni dispositivos especializados.
- Se documenta el proceso completo con sus configuraciones, dificultades encontradas y alternativas de mejora, permitiendo su reutilización o escalamiento en el futuro.

3. Marco teórico

3.1. Aspectos básicos de comunicación

Un sistema de comunicación es un conjunto de elementos interconectados que permiten la transmisión de información entre dos o varios dispositivos. En el caso más simplificado estos pueden definirse como un emisor y un receptor. Dicho intercambio se produce a través de un proceso ordenado en el cual un emisor genera un mensaje, lo transforma en una señal adecuada para su transmisión, y un receptor la interpreta para reconstruir los datos originales. Estos sistemas constituyen la base de la interconexión de dispositivos en contextos tan variados como las telecomunicaciones, las redes informáticas, la automatización industrial o los sistemas de control de energía.

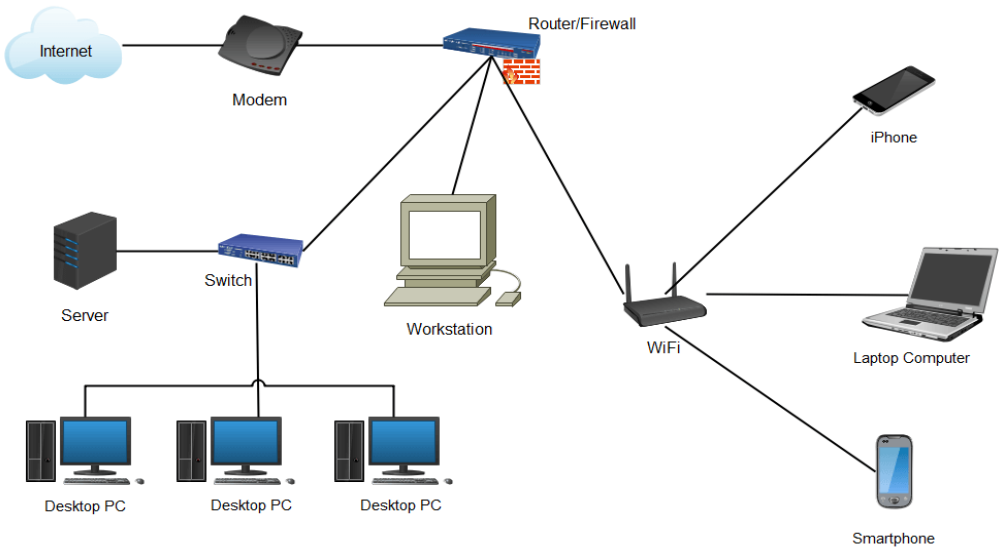


Figura 1: Sistema de comunicación entre varios dispositivos.

Los componentes básicos de un sistema de comunicación incluyen:

- Emisor: Convierte el mensaje en una señal adecuada para la transmisión.
- Receptor: Capta y decodifica la señal para recuperar el mensaje original.
- Canal: Medio físico por donde viaja la señal. Por ejemplo: cable, aire, fibra óptica.
- Mensaje: Información que se pretende llevar de un punto a otro.
- Código: Es el conjunto de símbolos o signos utilizado para la transmisión del mensaje, que debe ser conocido tanto por el emisor como por el receptor (protocolo, idioma).

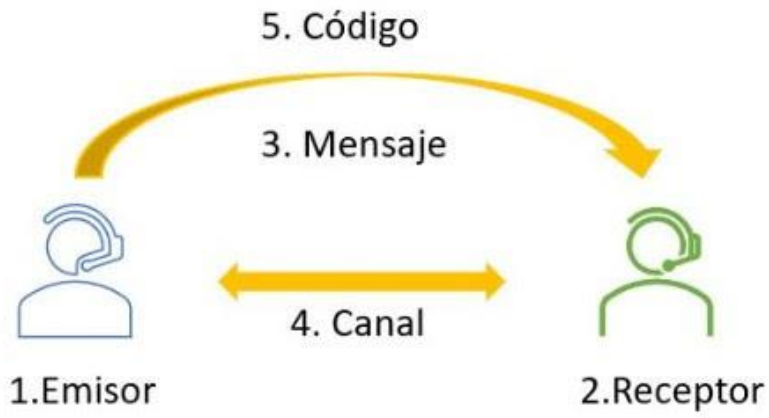


Figura 2: Componentes de un sistema de comunicación entre dos dispositivos.

En este proceso intervienen también fenómenos como la atenuación, que implica la pérdida de potencia de la señal durante su propagación, el ruido, que introduce distorsiones no deseadas, y la interferencia, que puede provenir de otras señales en el mismo medio.

Debido a estos factores para que la comunicación entre dispositivos sea posible no basta con la existencia de un medio físico entre el emisor y receptor, sino que es indispensable emplear protocolos de corrección de errores, técnicas de modulación y métodos de codificación que permitan garantizar la integridad y confiabilidad de la comunicación. [1]

Un protocolo de comunicación es un conjunto de normas y convenciones que definen cómo se conforman, transmiten y reciben los datos. Estos protocolos especifican desde la codificación de los bits hasta los procedimientos de sincronización, detección de errores y control de flujo. De forma análoga, en las redes de datos los protocolos garantizan que los mensajes se transmitan en orden, sin pérdidas y con un formato que todos los dispositivos comprendan.

3.1.1. Modelo de referencia OSI

La diversidad de tecnologías desarrolladas desde mediados del siglo XX condujo a la necesidad de crear modelos de referencia que establecieran una referencia común para el diseño e interoperabilidad de sistemas de comunicación. Uno de los más importantes es el modelo OSI (Open Systems Interconnection), diseñado por la ISO en 1983 y revisado en 1995, que estructura el proceso de comunicación en siete capas jerárquicas, cada una con funciones específicas y delimitadas.

Este modelo no es en sí un protocolo, sino un marco conceptual que facilita la comprensión, diseño y diagnóstico de redes. Su valor radica en que permite dividir un proceso complejo en tareas más pequeñas y manejables, distribuidas en distintos niveles. [2]

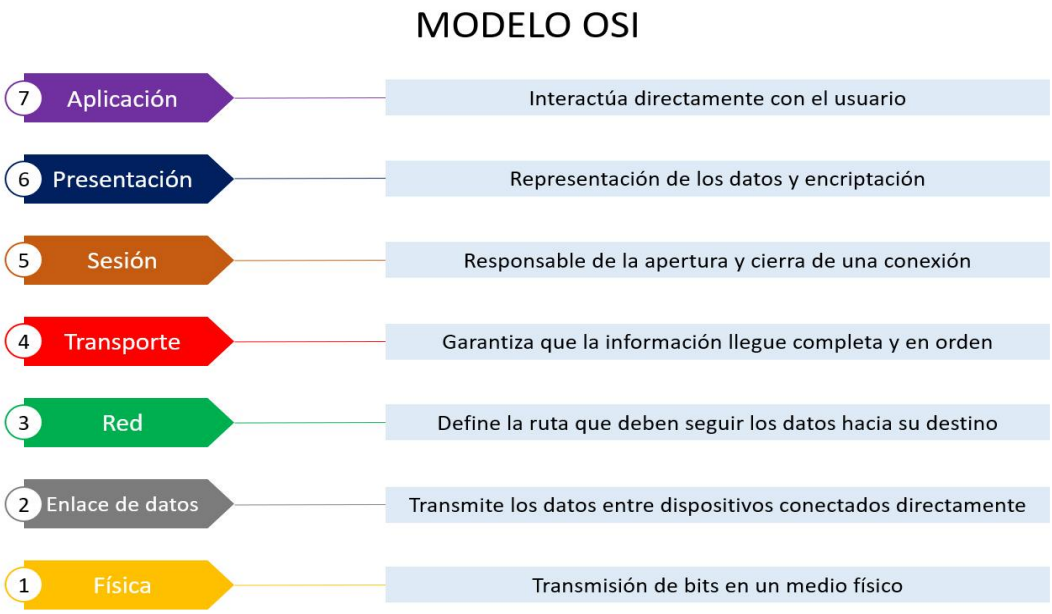


Figura 3: Capas del modelo OSI.

Aunque el modelo OSI es principalmente teórico, su estructura ha servido de base para modelos implementados como TCP/IP, ampliamente adoptado en redes modernas y sistemas de automatización.

Las capas se numeran de abajo hacia arriba, partiendo de la capa física hasta la capa de aplicación. El principio fundamental del modelo es que cada capa ofrece servicios a la capa superior y se apoya en los servicios de la capa inferior, estableciendo así una estructura escalonada y coherente.

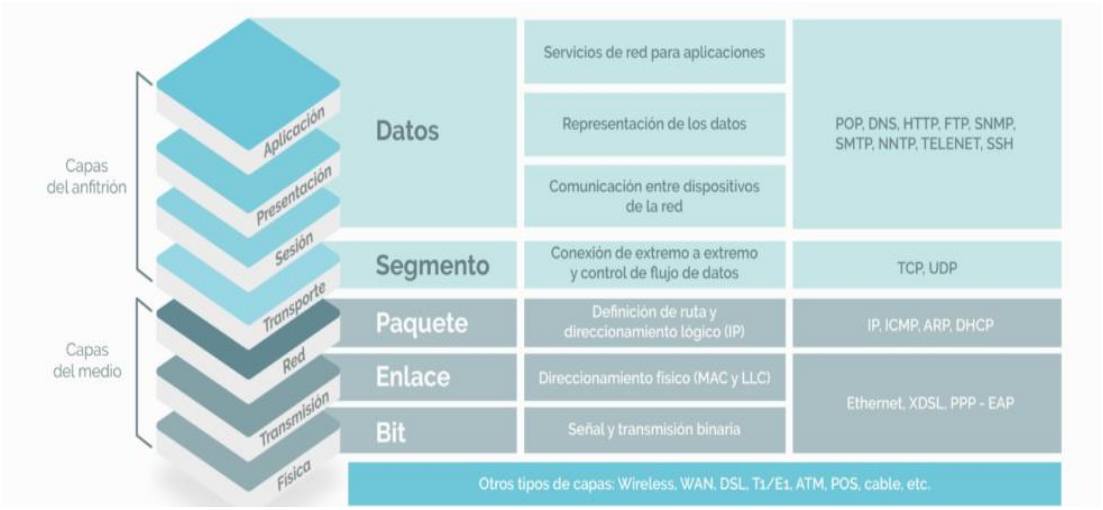


Figura 4: Modelo OSI. Datos y protocolos.

Un concepto esencial en el modelo OSI es el de encapsulamiento. Cada capa, al recibir los datos de la capa superior, añade su propia información de control (encabezados y, en algunos casos, tráileres). De esta forma, los datos viajan por la red en forma de unidades llamadas PDU

(Protocol Data Units). Bits en la capa física, tramas en la capa de enlace de datos, paquetes en la capa de red y segmentos en la capa de transporte.

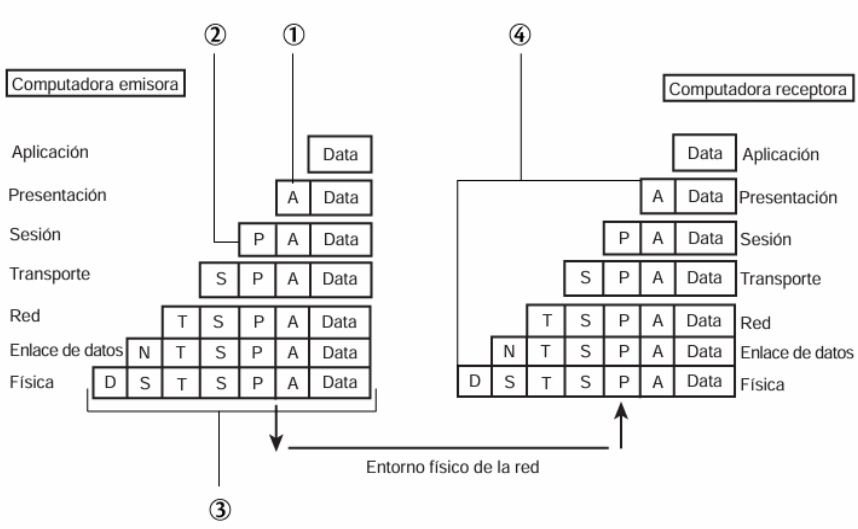


Figura 5: Modelo OSI. Encapsulamiento de la información.

1. Encabezado de la capa de aplicación
2. Encabezado de la capa de presentación
3. Paquete con todos los encabezados de las capas OSI
4. Los encabezados se van suprimiendo a medida que suben por las capas OSI del receptor.

3.1.2. Modelo TCP/IP

La evolución de las redes de comunicación y la necesidad de interconectar sistemas heterogéneos condujeron al desarrollo del modelo TCP/IP (Transmission Control Protocol / Internet Protocol). Este modelo, impulsado por el Departamento de Defensa de los Estados Unidos en la década de 1970, se consolidó como el estándar de facto para las comunicaciones en redes de datos, siendo la base del funcionamiento de Internet y de numerosas redes privadas industriales.

Figura 6: Comparación modelo OSI y TCP/IP

El modelo TCP/IP se concibió con un enfoque práctico, orientado a la implementación. Su arquitectura se organiza en cuatro capas principales, cada una de las cuales agrupa funciones equivalentes a varias capas del modelo OSI, con el objetivo de simplificar la comunicación y optimizar su desempeño.

Capa 1: Acceso a la red (Network Access Layer)

Esta capa corresponde a la capa física y de enlace de datos del modelo OSI. Se encarga de la transmisión de los datos a través del medio físico, incluyendo la codificación de señales, la definición de direcciones físicas (MAC) y el control de acceso al medio.

Define cómo se encapsulan los datagramas IP en las tramas que se transmiten por la red, así como los métodos utilizados para el direccionamiento local y la detección de errores.

Entre las tecnologías que operan en este nivel se incluyen Ethernet, Wi-Fi, Point-to-Point Protocol, y en entornos industriales, medios como RS-232, RS-485, fibra óptica, o enlaces de radiofrecuencia.

Capa 2: Internet (Internet Layer)

La capa de Internet se encarga del direccionamiento lógico y enrutamiento de los datos entre redes diferentes. Su función principal es garantizar que los paquetes lleguen desde el origen hasta el destino, incluso si ambas estaciones se encuentran en redes físicamente separadas.

El protocolo fundamental de esta capa es el IP (Internet Protocol), que define el formato de los paquetes y los mecanismos de direccionamiento mediante direcciones IP. Existen dos versiones ampliamente utilizadas: IPv4, con direcciones de 32 bits, e IPv6, con direcciones de 128 bits que amplían el espacio de direccionamiento y mejoran la seguridad y eficiencia.

Además de IP, otros protocolos asociados a esta capa son ICMP (Internet Control Message Protocol), ARP (Address Resolution Protocol) y RARP (Reverse ARP).

Capa 3: Transporte (Transport Layer)

La capa de transporte es la encargada de establecer la comunicación lógica extremo a extremo entre los dispositivos que participan en la red. Su función principal es garantizar que los datos generados por una aplicación en el equipo emisor lleguen correctamente al proceso correspondiente en el equipo receptor, manteniendo el orden, la integridad y la segmentación adecuada de la información. Los protocolos principales son los ya mencionados TCP y UDP.

Esta capa actúa como intermediaria entre las aplicaciones y la capa de red, proporcionando los mecanismos necesarios para la entrega confiable de datos o, en caso contrario, para transmisiones rápidas sin confirmación, según el tipo de servicio requerido.

Además, la capa de transporte implementa el concepto de puertos lógicos, que identifican los distintos servicios o aplicaciones que utilizan simultáneamente la red. Gracias a este esquema,

un mismo dispositivo puede mantener múltiples comunicaciones activas con diferentes procesos, sin interferencia entre ellas.

Capa 4: Aplicación (Application Layer)

La capa de aplicación agrupa las funcionalidades de las tres capas superiores del modelo OSI (sesión, presentación y aplicación). Es la responsable de proveer servicios de red directamente a las aplicaciones del usuario o del sistema.

En entornos industriales, esta capa alberga los protocolos de comunicación propios de la automatización, tales como:

- Modbus TCP, para el intercambio de datos entre dispositivos de campo y controladores.
- DNP3 sobre TCP/IP, utilizado en sistemas SCADA.
- MMS (Manufacturing Message Specification), base de la arquitectura de comunicación en IEC 61850, que permite el intercambio estructurado de información entre IEDs.

El proceso de comunicación en TCP/IP sigue el principio de encapsulamiento heredado del modelo OSI. Los datos generados por una aplicación se encapsulan sucesivamente en segmentos (capa de transporte), datagramas (capa de Internet) y tramas (capa de acceso a la red). Cada capa agrega su propio encabezado de control, necesario para la correcta entrega del mensaje.

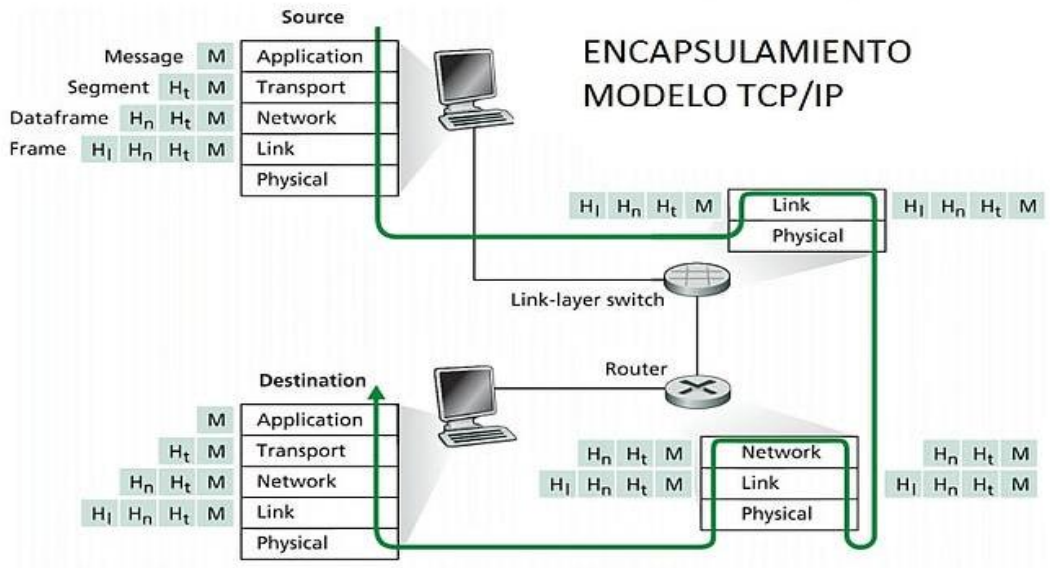


Figura 7: Encapsulamiento del modelo TCP/IP.

Aunque el modelo TCP/IP y el modelo OSI persiguen el mismo objetivo, presentan diferencias en su concepción y aplicación. En redes industriales, ambos modelos se complementan, ya que OSI proporciona el marco conceptual, y TCP/IP define la arquitectura utilizada en la mayoría de las infraestructuras de comunicación actuales.

3.1.3. Protocolos de comunicación industrial

La comunicación entre dispositivos de control, supervisión y protección constituye el núcleo funcional de todo sistema de automatización industrial. Para garantizar que los distintos equipos puedan intercambiar información de forma precisa, rápida y segura, se han desarrollado a lo largo de las décadas diversos protocolos de comunicación industrial, cada uno adaptado a necesidades específicas de aplicación, medio físico y nivel jerárquico dentro del sistema.

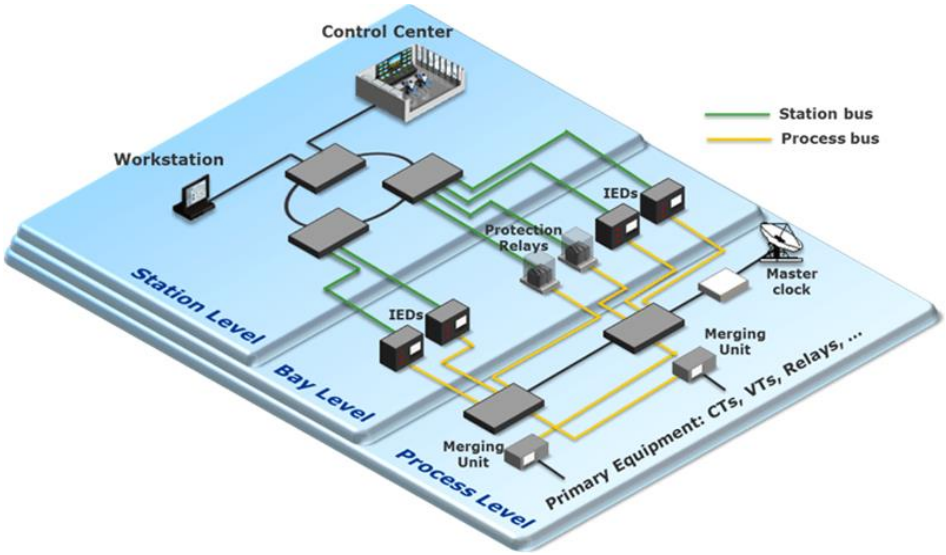


Figura 8: Arquitectura de un Sistema Automático en Estaciones Transformadoras.

Estos protocolos establecen las reglas, estructuras y procedimientos que definen cómo se transmiten los datos entre dispositivos de campo, controladores y sistemas SCADA, asegurando la interoperabilidad y la integridad de la información. En los sistemas eléctricos, su función resulta esencial para permitir la coordinación entre relés, RTUs, PLCs y centros de control. [3]
En términos generales, los protocolos industriales pueden agruparse en dos grandes familias:

1. Protocolos abiertos y normalizados, desarrollados por organismos internacionales (IEC, IEEE, ANSI), entre los que se destacan Modbus, DNP3, y la serie IEC 60870-5 (101 y 104). Estos protocolos sentaron las bases de la comunicación digital entre dispositivos en subestaciones, principalmente bajo arquitecturas maestro-esclavo, serie o TCP/IP.
2. Protocolos propietarios o licenciados por fabricantes, ampliamente utilizados en entornos de automatización industrial de procesos, tales como Profibus, Profinet, EtherNet/IP, DeviceNet o OPC. Aunque fueron diseñados para industrias de manufactura y control de procesos, muchos de ellos han sido adoptados en el ámbito energético, especialmente en aplicaciones de control local y supervisión de equipos secundarios.

Con la expansión de las redes Ethernet industriales y la necesidad de unificar arquitecturas bajo estándares interoperables, surgieron versiones adaptadas de protocolos clásicos como Modbus TCP y DNP3 LAN/WAN.

3.1.3.1. Protocolo Modbus

El protocolo Modbus es uno de los estándares de comunicación industrial más antiguos y extendidos a nivel mundial. Fue desarrollado por la empresa Modicon (actual Schneider Electric) en 1979, con el propósito de establecer un método sencillo y confiable para el intercambio de información entre PLC, dispositivos de campo y sistemas de supervisión.

Modbus define una estructura de comunicación maestro-esclavo, en la cual un dispositivo maestro inicia las solicitudes de información, y uno o varios dispositivos esclavos responden con los datos solicitados o ejecutan las acciones requeridas.

Originalmente, Modbus fue diseñado para trabajar sobre enlaces serie como RS-232 y RS-485, aunque con el tiempo se adaptó a redes Ethernet mediante la variante Modbus TCP, manteniendo la misma estructura de comandos y registros.

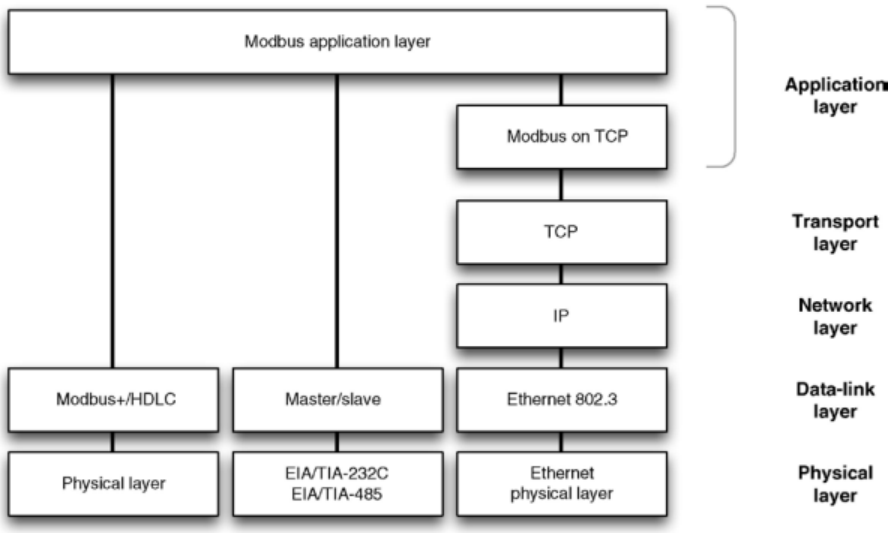


Figura 9: Arquitectura de un sistema automático bajo el protocolo Modbus.

Modbus RTU

Modbus RTU (*Remote Terminal Unit*) es la versión más utilizada del protocolo en entornos industriales. Opera sobre interfaces serie, típicamente RS-485, que permite la conexión multipunto de hasta 32 dispositivos en un mismo bus físico, o RS-232, en enlaces punto a punto de corto alcance.

En esta modalidad, los datos se transmiten en formato binario, lo que optimiza la eficiencia de la comunicación y permite mayores velocidades de transmisión (habitualmente entre 9600 y 115200 baudios).

Cada dispositivo esclavo posee una dirección única dentro de la red, y los mensajes siguen una estructura claramente definida, que incluye un campo de dirección, un código de función, los datos y un campo de verificación de errores. La detección de errores se realiza mediante un código CRC (Cyclic Redundancy Check), que garantiza la integridad del mensaje frente a interferencias o ruido eléctrico en el canal.

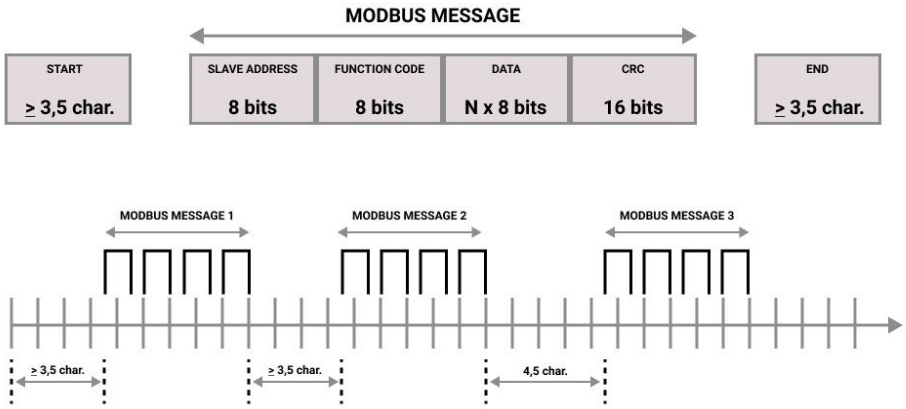


Figura 10: Arquitectura de un sistema automático bajo el protocolo Modbus.

Modbus ASCII

La variante Modbus ASCII comparte la misma estructura funcional que Modbus RTU, pero utiliza caracteres ASCII (American Standard Code for Information Interchange) para representar la información transmitida. Cada byte de datos se envía como dos caracteres ASCII, lo que facilita la legibilidad y depuración de los mensajes durante pruebas o mantenimiento.

Los mensajes Modbus ASCII comienzan con un carácter ":" (dos puntos) y finalizan con una secuencia de retorno de carro y salto de línea. La detección de errores se realiza mediante un checksum longitudinal (LRC).

Si bien esta versión resulta más lenta debido al mayor volumen de datos transmitidos, su facilidad para ser interpretada manualmente la hace útil en sistemas de baja velocidad o en comunicaciones donde la claridad de los datos sea prioritaria, como tareas de diagnóstico o configuración.

Modbus TCP

Con el avance de las redes Ethernet industriales y la necesidad de integrar dispositivos de campo en arquitecturas más complejas, surgió Modbus TCP, una evolución del protocolo original que conserva la estructura de mensajes pero reemplaza la capa física serie por una comunicación sobre redes Ethernet (IEEE 802.3).

En Modbus TCP, los mensajes se encapsulan dentro de tramas TCP/IP, eliminando la necesidad de campos de dirección y de verificación CRC, ya que la confiabilidad y la integridad de los datos son garantizadas por las capas inferiores del modelo TCP/IP.
Este enfoque permite aprovechar las ventajas de Ethernet, tales como:

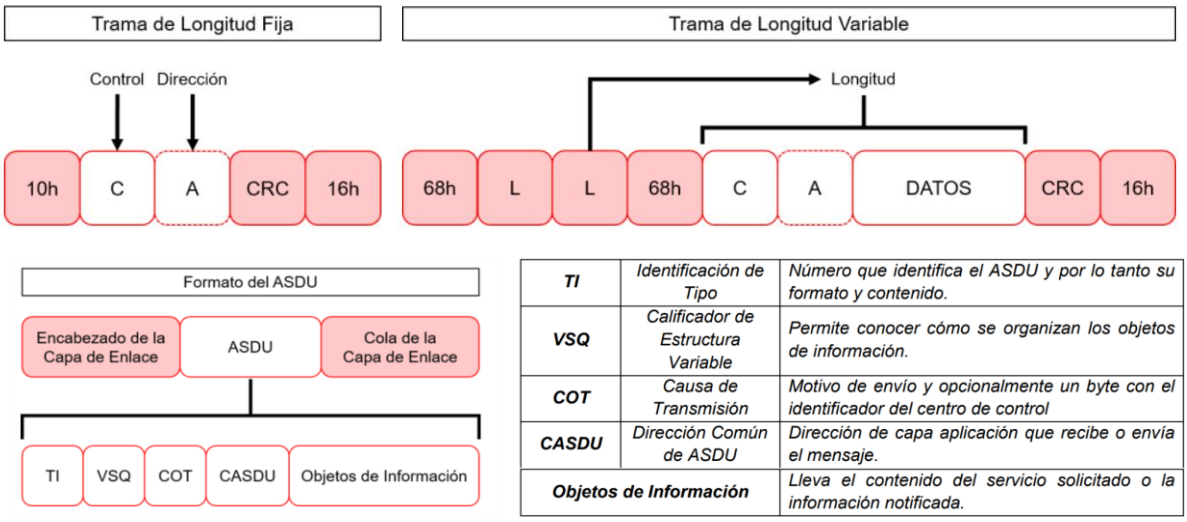
- Mayor velocidad de transmisión (10/100/1000 Mbps).
- Mayor distancia y cobertura de red.
- Capacidad de comunicación simultánea con múltiples dispositivos.
- Integración directa con redes SCADA y sistemas corporativos.

Cada transacción Modbus TCP utiliza el puerto 502 del protocolo TCP, designado internacionalmente por la IANA (Internet Assigned Numbers Authority) para este servicio.

3.1.3.2. Protocolo IEC 60870-5

Los protocolos IEC 60870-5-101 y IEC 60870-5-104 forman parte de la familia de normas desarrolladas por la International Electrotechnical Commission (IEC) para el telecontrol y automatización de sistemas eléctricos de potencia. Estos protocolos definen los mecanismos de comunicación entre centros de control, estaciones remotas y dispositivos de campo, permitiendo la supervisión y operación de redes de transmisión y distribución eléctrica.
El protocolo IEC 60870-5-101, publicado en 1995, fue concebido para canales de comunicación serie (RS-232 o RS-485), mientras que IEC 60870-5-104, publicado en el año 2000, adapta la misma estructura de capas al entorno de redes TCP/IP sobre Ethernet, ofreciendo mayor velocidad, flexibilidad y compatibilidad con infraestructuras modernas. Ambos protocolos se basan en una arquitectura de comunicación en capas conocida como *Enhanced Performance Architecture* (EPA), una simplificación del modelo OSI que organiza la comunicación en tres capas funcionales. Capas de enlace de datos, pseudo-transporte y aplicación.
La capa de enlace es responsable de establecer una comunicación fiable entre dos nodos adyacentes. Se encarga de la detección de errores, el control de flujo y la sincronización de tramas. Los mensajes se transmiten bajo un esquema maestro-esclavo, donde el maestro inicia todas las solicitudes y el esclavo únicamente responde o notifica eventos.
En el caso de IEC 60870-5-101, la capa de pseudo-transporte opera sobre enlaces serie físicos, mientras que en IEC 60870-5-104 se encapsula dentro de las tramas TCP/IP para su transporte sobre Ethernet. Esta capa actúa como un nivel intermedio entre el enlace y la aplicación, y tiene como objetivo garantizar la entrega ordenada de las unidades de datos de aplicación (ASDU).
La capa de aplicación constituye el núcleo funcional del protocolo y define la estructura de los datos intercambiados y las funciones de telecontrol disponibles. Toda la información

transmitida se organiza en ASDUs. Estos se encuentran compuestos por un APCI (Application Protocol Control Information), que contiene los campos de control de secuencia y longitud y un ASDU propiamente dicho, donde se especifica el tipo de información, la causa de transmisión, el origen, la dirección del objeto de información y los valores transmitidos.



TI	*Identificación de Tipo*	*Número que identifica el ASDU y por lo tanto su formato y contenido.*
VSQ	*Calificador de Estructura Variable*	*Permite conocer cómo se organizan los objetos de información.*
COT	*Causa de Transmisión*	*Motivo de envío y opcionalmente un byte con el identificador del centro de control*
CASDU	*Dirección Común de ASDU*	*Dirección de capa aplicación que recibe o envía el mensaje.*
Objetos de Información		*Lleva el contenido del servicio solicitado o la información notificada.*

Figura 11: Encapsulamiento del protocolo IEC 60870-5

En la actualidad, los protocolos IEC 60870-5-101 y 104 continúan siendo ampliamente utilizados en sistemas SCADA de transmisión y distribución, donde se requiere comunicación confiable entre centros de control y estaciones remotas (RTUs).

3.1.3.3. Protocolo DNP3

El DNP3 (Distributed Network Protocol) es un protocolo de comunicación diseñado para la telemedición, control y supervisión en sistemas eléctricos. Desarrollado a comienzos de la década de 1990 por la empresa Westronic, Inc. (posteriormente Harris Controls), su objetivo fue superar las limitaciones de los protocolos serie tradicionales y ofrecer una solución más eficiente y estructurada para la comunicación entre dispositivos de campo, RTUs, IEDs y sistemas SCADA.

Desde su concepción, DNP3 fue pensado para operar en entornos eléctricos, priorizando la confiabilidad, la integridad de los datos y la transmisión eficiente en canales de baja calidad o alta latencia, condiciones frecuentes en redes de control extensas.

En 2010 fue adoptado como estándar internacional bajo la norma IEEE Std 1815, consolidando su uso en aplicaciones de transmisión, distribución y generación eléctrica.

DNP3 y los protocolos IEC 60870-5 (101/104) surgieron en épocas y contextos geográficos similares, con el objetivo de establecer un lenguaje común para el intercambio de información entre centros de control y estaciones remotas.

Figura 12: Distribución global en 2009 según Siemens.

DNP3 fue impulsado principalmente en Norte América, bajo la influencia de las utilidades eléctricas y fabricantes estadounidenses, buscando un protocolo abierto y robusto compatible con los sistemas SCADA existentes, mientras que IEC 60870-5 fue desarrollado en Europa por la International Electrotechnical Commission (IEC).

Ambos protocolos comparten una arquitectura jerárquica maestro-esclavo, el uso de tramas estructuradas con verificación de errores, y la capacidad de operar sobre enlaces serie o redes TCP/IP. Sin embargo, DNP3 introdujo mayores capacidades de registro temporal y reportes por excepción, mientras que IEC 60870-5 adoptó una organización más rígida en sus estructuras de datos.

Al igual que IEC 60870-5 el protocolo DNP3 se organiza siguiendo una estructura simplificada inspirada en el modelo OSI, que implementa únicamente tres capas funcionales (EPA).

La capa de enlace de datos es responsable de la transmisión confiable de las tramas, incorporando mecanismos de detección de errores mediante códigos CRC, numeración de tramas y control de flujo.

La capa de pseudo-transporte constituye una capa intermedia propia del protocolo DNP3, encargada de la segmentación, detección de pérdidas o desorden, y reensamblado de los datos provenientes de la capa de aplicación. Su función principal es adaptar los mensajes largos (fragmentos) a las limitaciones de tamaño de la capa de enlace.

La capa de aplicación es la más alta y representa la interfaz entre el protocolo y la lógica funcional del sistema SCADA o de automatización. En este nivel se generan, interpretan y procesan los mensajes de control, consulta y reporte de eventos, denominados fragmentos.

Cada fragmento de aplicación comienza con una cabecera que incluye:

- Un código de control de aplicación, donde se indica si el fragmento forma parte de un mensaje múltiple, si requiere confirmación o si fue enviado de forma no solicitada.
- Un código de función, que define la operación que debe realizarse (por ejemplo: lectura, escritura, confirmación, reinicio, control de salidas, etc.).

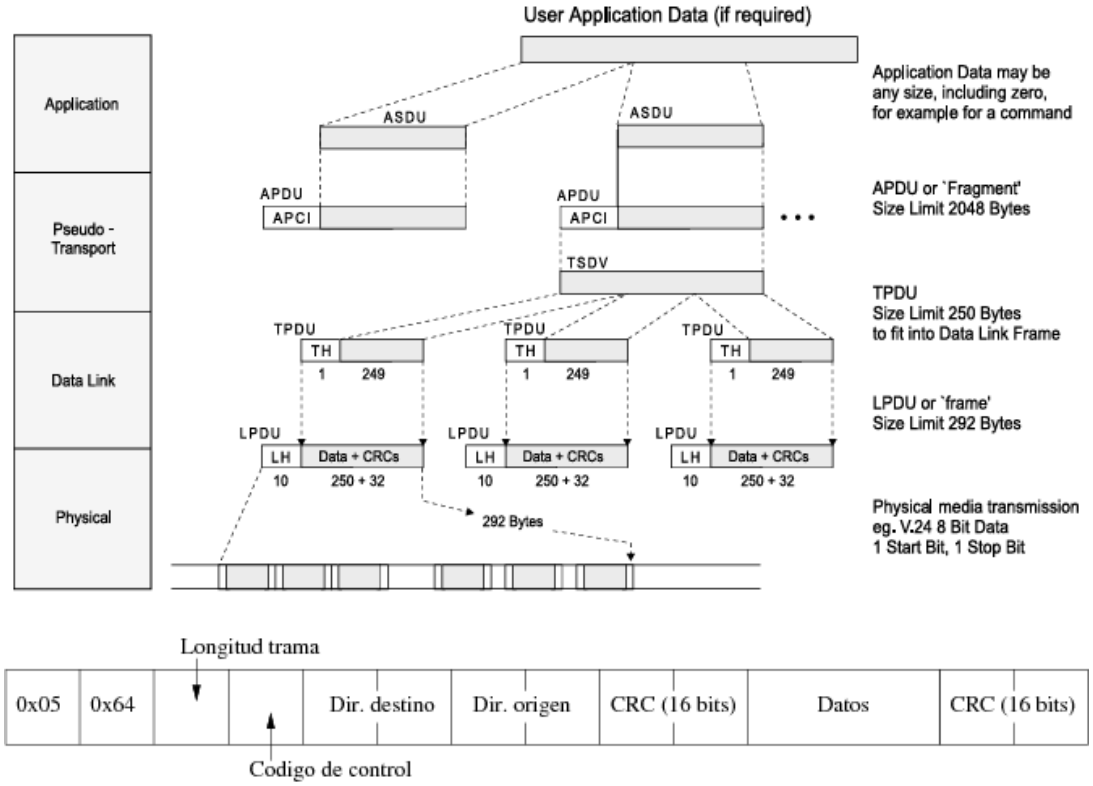


Figura 13: Encapsulamiento DNP3.

Una característica clave del DNP3 es su capacidad de reporte por excepción, mediante la cual solo se transmiten los datos que han cambiado desde la última actualización, optimizando el uso del canal de comunicación. Asimismo, todos los eventos y valores pueden asociarse a marcas de tiempo (timestamp) para garantizar la secuencia cronológica de los registros.

El protocolo mantiene una arquitectura maestro-esclavo, donde el maestro solicita lecturas o envía comandos, y los dispositivos remotos responden o notifican eventos según corresponda. A diferencia de los protocolos más antiguos, DNP3 permite la transmisión iniciada por el esclavo, característica esencial para reportes de eventos y alarmas. Es particularmente adecuado para comunicaciones a larga distancia, enlaces conmutados, radioenlaces o sistemas mixtos, donde la latencia y la pérdida de paquetes deben gestionarse con tolerancia. Debido a su diseño modular, DNP3 puede implementarse tanto en canales serie como en redes Ethernet, y es compatible con arquitecturas híbridas, donde conviven enlaces tradicionales y redes IP.

3.2. Automatización de estaciones transformadoras

3.2.1. Sistema eléctrico de potencia

El sistema eléctrico de potencia constituye una de las infraestructuras más complejas y críticas de la sociedad moderna, pensada para garantizar el suministro confiable, seguro y eficiente de energía eléctrica desde los centros de generación hasta los usuarios finales. Este se estructura en tres grandes etapas: generación, transmisión y distribución, las cuales interactúan de manera coordinada mediante una red interconectada que opera bajo estrictos criterios de estabilidad, calidad y seguridad.

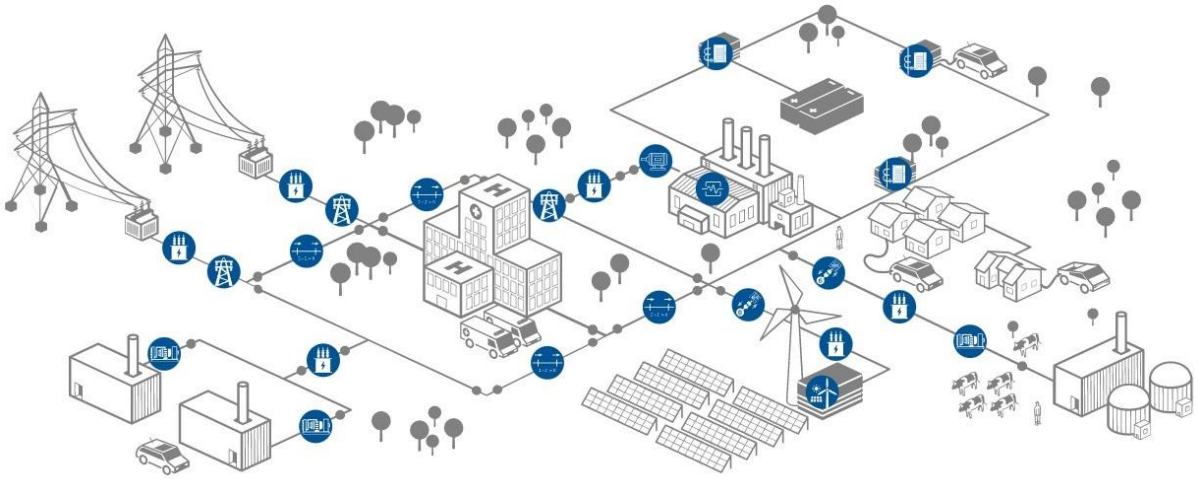


Figura 14: Sistema eléctrico de potencia.

En el subsistema de generación, la energía eléctrica se produce a partir de la conversión de diversas formas de energía primaria. En todos los casos, los niveles de tensión en la generación son relativamente bajos (10–30 kV), por lo que resulta necesario elevar la tensión para permitir el transporte eficiente de grandes potencias a largas distancias.

El subsistema de transmisión desempeña un rol esencial al permitir el transporte de grandes cantidades de energía a largas distancias y con mínimas pérdidas, lo que requiere operar a niveles de tensión elevados. El subsistema de transmisión opera habitualmente en niveles de Muy Alta Tensión (MAT), del orden de 220, 500 kV e incluso superiores en sistemas interconectados de gran escala, mientras que las Altas Tensiones (AT) abarcan rangos típicos de 66 a 132 kV. En el extremo opuesto, las redes de Media Tensión (MT), que se sitúan entre 10 y 33 kV, y las de Baja Tensión (BT), por debajo de 1 kV, se destinan a la distribución local y consumo final. La transición ordenada entre estos niveles de tensión es posible gracias a las estaciones transformadoras, que actúan como nodos estratégicos del sistema.

Una estación transformadora puede definirse como una instalación electromecánica en la que la energía eléctrica es sometida a procesos de transformación, control, protección y maniobra, con el objetivo de garantizar la transferencia de potencia entre diferentes niveles de tensión o entre distintos segmentos de la red. [4]

Desde un punto de vista funcional, las estaciones transformadoras pueden clasificarse en diferentes tipos de acuerdo con su ubicación dentro del sistema eléctrico y su propósito operativo:

- Elevadoras, que incrementan la tensión a la salida de la generación para permitir el transporte eficiente a través de la red.
- Reductoras de transmisión, que convierten los niveles de muy alta tensión hacia tensiones intermedias para alimentar redes regionales.
- De distribución, que reducen aún más la tensión hasta valores aptos para usuarios finales industriales o residenciales.
- De maniobra, que permiten interconectar, seccionar o reconfigurar partes de la red sin realizar transformación de tensión, aportando flexibilidad y confiabilidad operativa.

La importancia de cada tipo de estación no radica únicamente en su capacidad de transformación, sino también en su impacto en la confiabilidad, estabilidad y seguridad del sistema eléctrico. También depende de la ubicación geográfica y el nodo eléctrico que representa para la red, que se encuentra ampliamente asociado a la distribución de la generación y la demanda dentro del territorio nacional.

La configuración interna de una estación transformadora cobra especial relevancia en la continuidad y calidad del servicio. Las barras, que constituyen el punto común de conexión de líneas, transformadores y equipos de maniobra, pueden organizarse en distintos esquemas según el nivel de confiabilidad requerido, la flexibilidad deseada y los costos aceptables.

Por otra parte, también es necesario supervisar magnitudes eléctricas fundamentales como tensión, corriente, potencia activa, potencia reactiva, frecuencia y energía, las cuales resultan esenciales para garantizar la operación segura y mantener la calidad de servicio.

La creciente complejidad del sistema eléctrico de potencia, impulsada por la expansión de las redes interconectadas, la integración de generación distribuida y renovable, y la necesidad de elevar la confiabilidad del suministro, ha llevado a que las estaciones transformadoras evolucionen hacia esquemas de gestión más sofisticados basados en la automatización y el uso de protocolos estandarizados.

3.2.2. Evolución en la automatización de estaciones transformadoras

La automatización de estaciones transformadoras se entiende como la implementación de tecnologías de control, supervisión y protección que permiten operar la instalación de manera remota, coordinada y en tiempo real, reduciendo la necesidad de intervención manual y aumentando la capacidad de respuesta ante contingencias.

La automatización de estaciones eléctricas ha experimentado un proceso de transformación progresiva que refleja tanto los avances tecnológicos como la necesidad creciente de mayor confiabilidad, eficiencia y seguridad en la operación de los sistemas eléctricos. En sus primeras etapas, las estaciones eran instalaciones atendidas permanentemente por personal especializado, cuya función era operar equipos de maniobra de manera manual. Las órdenes se ejecutaban directamente desde tableros de control locales, donde se implementaban paneles con diagramas denominados "mímicos", que representaban de forma esquemática la disposición de barras, interruptores y líneas, permitiendo al operador seguir visualmente el estado de la estación. La supervisión se limitaba a indicadores analógicos como voltímetros, amperímetros, frecuencímetros y a señales luminosas que representaban el estado de los equipos primarios. Estas configuraciones eran limitadas, dependientes de la experiencia del personal y susceptibles a errores humanos.

Con el paso del tiempo, la creciente demanda y los avances tecnológicos impulsaron un incremento en la necesidad de supervisión remota. Esto dio lugar a la incorporación de la digitalización de magnitudes eléctricas, mediante el uso de transductores y equipos que convierten señales analógicas en señales binarias o impulsos que podían ser transmitidos hacia sistemas de control centralizados. Se introdujeron los primeros sistemas de telemando y telemedición, que permitieron llevar la información a centros de control lejanos y devolver órdenes de operación, disminuyendo la dependencia del personal en sitio.

Posteriormente, con la irrupción de la electrónica industrial y la aparición de protocolos de comunicación seriales, como Modbus RTU, se hizo posible el intercambio estructurado de datos entre dispositivos electrónicos inteligentes. Este avance dio paso a la automatización distribuida en las subestaciones, con la incorporación de relés digitales de protección, registradores de fallas y controladores lógicos programables. Sin embargo, en esta etapa inicial cada fabricante desarrollaba sus propios protocolos de comunicación propietarios, lo cual generaba dificultades en la interoperabilidad y obligaba a los usuarios a depender de soluciones cerradas. [5]

La evolución posterior se orientó hacia la estandarización y la integración, con el surgimiento de protocolos abiertos como DNP3 o IEC 60870-5-101/104, que facilitaron la comunicación

entre equipos de distintos fabricantes y su conexión con los centros de control. Estos protocolos permitieron que las subestaciones fueran operadas de forma más confiable y con mayor capacidad de supervisión, aunque todavía persistían limitaciones en cuanto a la velocidad de intercambio de información, la estructura de datos y la complejidad de la configuración.

La digitalización integral de las estaciones se consolidó con el desarrollo y publicación de la norma IEC 61850, que introdujo un modelo de datos orientado a objetos, estructurado y normalizado, capaz de describir de forma unificada las funciones de protección, control, medición y supervisión de cualquier equipo dentro de la estación. [6]

A diferencia de los protocolos anteriores, IEC 61850 no solo define la comunicación entre dispositivos, sino también su semántica, garantizando una interoperabilidad real y una integración transparente entre equipos de distintos fabricantes. Además, incorpora mecanismos de comunicación en tiempo real como GOOSE (Generic Object Oriented Substation Event) y Sampled Values (SV), basados en el intercambio directo entre dispositivos (peer-to-peer) sobre redes Ethernet, con tiempos de transmisión del orden de los milisegundos, imprescindibles para aplicaciones de protección y control de alta velocidad.

El concepto de automatización evolucionó así de un control remoto básico hacia un sistema inteligente y distribuido, capaz de procesar información, diagnosticar fallas y ejecutar acciones coordinadas de protección y control en tiempos mínimos. La automatización de estaciones bajo IEC 61850 constituye la base de las denominadas estaciones digitales, donde las señales analógicas tradicionales se reemplazan por flujos de datos transmitidos sobre redes Ethernet y fibra óptica, y las funciones de protección y control se implementan en plataformas modulares, flexibles y escalables. Esto no solo reduce el cableado y los costos de mantenimiento, sino que mejora la confiabilidad, eleva los niveles de seguridad y habilita la integración de tecnologías avanzadas vinculadas con la ciberseguridad, el mantenimiento predictivo y la gestión inteligente de la red eléctrica.

3.2.3. Equipos de una estación transformadora

Una estación transformadora constituye el vínculo esencial entre los distintos niveles de tensión del sistema eléctrico de potencia y cumple el propósito de adaptar las condiciones de la energía eléctrica para su transmisión, distribución o consumo final. Su concepción integra equipos principales y auxiliares que, en conjunto, garantizan la transformación de tensión, la maniobra segura de los circuitos, la protección de las instalaciones, la medición de magnitudes eléctricas y el control operativo.

Figura 15: Estación transformadora de Alta tensión.

El transformador de potencia es el equipo más relevante de la estación, responsable de modificar los niveles de tensión con el fin de optimizar el transporte y reducir pérdidas por efecto Joule. Está constituido por devanados de cobre inmersos en aceite mineral o sintético y dispone de sistemas de refrigeración clasificados como ONAN, ONAF u OFAF, según la combinación de circulación natural o forzada de aceite y aire. Pueden tener dos o tres devanados, o emplearse tres transformadores monofásicos en lugar de uno trifásico. Para mantener la regulación de tensión en condiciones de carga variable, incorporan cambiadores automáticos de tomas bajo carga (OLTC), capaces de operar de manera continua sin interrumpir el servicio. En caso de fallas internas, cuentan con dispositivos de protección específicos, tales como relés Buchholz, válvulas de alivio y sensores térmicos ubicados en el aceite y los devanados.

Figura 16: Transformador de potencia de Alta tensión.

La maniobra y el seccionamiento de los circuitos se realizan mediante interruptores de potencia y seccionadores. Los interruptores, generalmente basados en tecnología de vacío o gas SF_6, son capaces de interrumpir tanto corrientes nominales como de cortocircuito, garantizando la desconexión selectiva frente a fallas y permitiendo la operación bajo carga en condiciones normales. Existen diversos modelos dependiendo del nivel de tensión, la capacidad de ruptura y las características del fabricante. Los seccionadores, en cambio, aseguran una separación visible entre partes de la instalación, posibilitando tareas de mantenimiento en condiciones de seguridad. Su construcción contempla configuraciones tripolares y, en muchos casos, cuchillas de puesta a tierra integradas que permiten descargar los equipos aislados. Aunque no están diseñados para interrumpir corriente, algunos modelos para media o baja tensión admiten maniobras limitadas bajo carga en corrientes reducidas.

Para garantizar la supervisión y medición del sistema, las estaciones incorporan transformadores de medida, que comprenden transformadores de corriente (TC) y transformadores de tensión (TT). Estos equipos transforman de manera proporcional los valores de tensión y corriente a niveles adecuados para instrumentos de medición, relés y sistemas de protección, además de proporcionar aislamiento galvánico entre el sistema de potencia y los circuitos de control. Su precisión es fundamental para asegurar el correcto funcionamiento de los sistemas de protección diferencial, de distancia o de sobrecorriente, así como para la facturación de energía y el registro de variables operativas.

Figura 17: Elementos de maniobra de una ET.

Dada la exposición del equipamiento a fenómenos atmosféricos y maniobras de conmutación, las estaciones incorporan descargadores de sobretensión de óxido metálico (MOV), cuya

función es limitar las sobretensiones transitorias desviando la corriente hacia tierra, protegiendo así transformadores, interruptores y líneas asociadas. El correcto dimensionamiento y ubicación de estos dispositivos resulta esencial para preservar la vida útil del equipamiento principal y evitar daños ante descargas atmosféricas.

Todo el conjunto de la estación se encuentra respaldado por un sistema de puesta a tierra, diseñado para asegurar la disipación eficaz de corrientes de falla y descargas atmosféricas. La malla de tierra debe mantener potenciales de paso y contacto dentro de límites seguros para el personal, al tiempo que contribuye a la estabilidad eléctrica y al correcto funcionamiento de las protecciones.

En el ámbito de control, protección y supervisión, las estaciones modernas integran tableros de protección, control y comunicaciones, donde se concentran los relés digitales (IEDs), los sistemas de automatización y las interfaces SCADA. Los relés cumplen funciones de medición, disparo y registro de fallas, siendo los encargados de ejecutar las órdenes de apertura o cierre de interruptores ante condiciones anómalas. Estos dispositivos, junto con los IEDs de control, posibilitan la coordinación automática de protecciones y maniobras, el registro de eventos y la supervisión continua de variables eléctricas. Los sistemas de control local (HMI) y las unidades terminales remotas (RTU) permiten la operación directa en sitio o la comunicación con centros de control, integrando la estación dentro de un sistema de supervisión más amplio.

La comunicación de datos entre los distintos equipos y hacia los centros de control se realiza mediante redes de comunicación industrial, que constituyen la base de la automatización moderna. En este contexto, los switches Ethernet industriales, routers, convertidores de medio y gateway de protocolo garantizan la transmisión confiable y segura de información entre los IEDs, servidores locales y sistemas SCADA. Estos equipos permiten la integración de diferentes protocolos (IEC 61850, DNP3, Modbus TCP, IEC 60870-5-104, entre otros), la sincronización temporal de eventos mediante PTP/IEEE 1588, y la implementación de redes con topologías redundantes que aseguran la continuidad del servicio y la interoperabilidad entre dispositivos de distintos fabricantes.

Finalmente, la operación segura de una estación transformadora requiere de servicios auxiliares, entre los cuales destacan los sistemas de corriente continua alimentados por bancos de baterías y cargadores, imprescindibles para el accionamiento de interruptores y la continuidad del sistema de protección en caso de fallas en la red de potencia. A estos se suman los sistemas de refrigeración, ventilación, iluminación, climatización y comunicaciones internas, cuya función es mantener la operatividad y la seguridad de la instalación en todo momento.

3.2.4. Señales de protección y control

La automatización de una estación transformadora se fundamenta en la adquisición, procesamiento y transmisión de señales provenientes de los distintos equipos primarios. Estas señales permiten conocer el estado operativo de la instalación, supervisar magnitudes eléctricas y ejecutar maniobras desde sistemas de control o de protecciones. En este contexto, las señales se clasifican en digitales (binarias) y analógicas (cuantitativas), dependiendo de la naturaleza de la información representada.

Las señales digitales corresponden a estados discretos de equipos o condiciones específicas del sistema, como la posición de un interruptor (abierto/cerrado), la condición de un seccionador (en servicio/puesto a tierra), la activación de un relé (en servicio/fuera de servicio) o la presencia de una alarma en algún equipo. Estas señales se transmiten como valores lógicos binarios (0/1), garantizando una interpretación inequívoca de estado.

Las señales analógicas, en cambio, representan magnitudes eléctricas o físicas que varían de forma continua, tales como tensiones, corrientes, potencias, frecuencia, temperaturas, presiones o niveles de aceite. Son adquiridas en tiempo real por transformadores de medida y sensores, convertidas a valores digitales mediante conversores A/D, y procesadas por IEDs para fines de supervisión, registro y control.

En mayor medida, las señales son transmitidas desde la playa de maniobra hasta los equipos de protección y control, y pueden tener diferentes naturalezas, pero se clasifican fundamentalmente en:

- Señales digitales de entrada: indican el estado de un dispositivo o una condición de alarma.
- Señales digitales de salida: transmiten órdenes de operación o disparo hacia equipos de maniobra.
- Señales analógicas de entrada: corresponde a mediciones de magnitudes eléctricas y físicas.
- Señales analógicas de salida: representan señales de control o consignas hacia dispositivos reguladores.

Las señales adquiridas en una estación provienen de múltiples fuentes. En los transformadores de potencia, se supervisan variables asociadas a la temperatura de devanados y aceite, presión interna, posición del cambiador de tomas, alarmas de gas y disparo del relé Buchholz, todas ellas críticas para la protección térmica y dieléctrica del equipo.

En los interruptores de potencia, se registran señales digitales que indican su posición mecánica, permisos para operar, órdenes de apertura o cierre, y el estado de los mecanismos de disparo, junto con alarmas de presión del medio de extinción o fallas en el accionamiento.

Los seccionadores generan señales que informan su posición principal y la de las cuchillas de puesta a tierra, además de bloqueos mecánicos o enclavamientos que impiden maniobras indebidas. Debido a que no deben operar bajo carga, su correcta supervisión resulta esencial para la protección del equipo. Por otra parte, estos elementos cumplen con la apertura visible y aislamiento de zonas energizadas, apartados fundamentales para la seguridad del personal durante los mantenimientos.

En tanto, los transformadores de medida proporcionan señales analógicas en valores secundarios normalizados (1 A, 5 A, 100 V o 110 V), a partir de las cuales los equipos de protección y control determinan parámetros eléctricos fundamentales como tensiones, corrientes, ángulos de fase, potencias, energía o factor de potencia. Algunos modelos incorporan además salidas digitales que indican el estado interno del dispositivo o la validez de las mediciones.

La automatización no solo implica la adquisición de señales, sino también la capacidad de ejecutar maniobras a distancia. Desde la sala de control local, el operador puede enviar comandos remotos para abrir o cerrar interruptores, accionar seccionadores o modificar la posición del cambiador de tomas de un transformador. Estas órdenes, transmitidas como señales digitales de control, se ejecutan únicamente tras la verificación de enclavamientos lógicos y condiciones de seguridad que evitan maniobras peligrosas o contradictorias. Cuando los comandos se originan en el centro de control del sistema eléctrico, se habla de telecomando, mediante enlaces de comunicación basados en fibra óptica, microondas o redes IP seguras, integrados en sistemas SCADA/EMS. El telecomando permite coordinar el despacho de cargas, la regulación de tensión y la respuesta ante contingencias, sin necesidad de presencia física de operadores en la estación.

La ejecución de maniobras está protegida por interbloqueos eléctricos, lógicos y de software, que garantizan la secuencia correcta de operaciones. Entre ellos destacan las interdependencias entre interruptores y seccionadores, la imposibilidad de accionar equipos con fallas activas, o la verificación automática de sincronismo antes de realizar cierres de barra o de línea.

En las estaciones modernas, todas las señales y comandos se integran en un sistema de automatización basado en IEDs, que se comunican mediante protocolos. Las señales digitales y analógicas adquiridas en tiempo real se concentran en servidores locales, que a su vez interactúan con sistemas SCADA. De esta manera, la estación no solo se convierte en un punto de transformación eléctrica, sino también en un nodo de información digital, capaz de reportar su estado operativo, recibir instrucciones de manera inmediata y coordinar acciones con otros puntos de la red.

3.3. Estándar IEC 61850

El estándar IEC 61850 es una norma internacional desarrollada por la International Electrotechnical Commission (IEC) a través del comité técnico TC57, que define un marco integral para la comunicación, el modelado de datos y la ingeniería de los Sistemas de Automatización de Subestaciones (SAS), así como de otros entornos eléctricos relacionados con la generación, transmisión y distribución de energía.

El propósito fundamental de esta norma es establecer un lenguaje común de comunicación y descripción funcional que permita la interoperabilidad entre equipos electrónicos inteligentes (IEDs, Intelligent Electronic Devices) de distintos fabricantes, asegurando que puedan compartir información y ejecutar funciones conjuntas de control, protección y supervisión dentro del sistema eléctrico. [7]

Hasta ese momento, la comunicación entre equipos se realizaba mediante protocolos como DNP3, Modbus, o las series IEC 60870-5-101 y 104, los cuales presentaban limitaciones en cuanto a interoperabilidad, semántica de los datos y capacidad de integración entre fabricantes. Estas limitaciones se traducían en altos costos de ingeniería, dependencia de un único proveedor y dificultades para la ampliación o modernización de las instalaciones. Para enfrentar este problema, la IEC trabajó junto con el Institute of Electrical and Electronics Engineers (IEEE) y el Electric Power Research Institute (EPRI) en el marco del proyecto Utility Communication Architecture (UCA), cuyo objetivo fue definir un modelo de comunicación estandarizado, escalable y basado en tecnologías abiertas. De este esfuerzo surgió la primera edición del estándar IEC 61850, publicada en el año 2004, bajo el título "*Communication Networks and Systems for Power Utility Automation*".

Desde su publicación inicial, el estándar ha tenido una evolución constante. La primera edición sentó las bases conceptuales, definiendo el modelo de datos orientado a objetos, los servicios de comunicación abstractos (ACSI) y los principales protocolos asociados (MMS, GOOSE y Sampled Values). Posteriormente, la segunda edición (Ed.2, 2010-2013) introdujo mejoras significativas en la ingeniería de sistemas, la gestión de versiones de archivos SCL (Substation Configuration Language) y nuevas clases de datos, además de extender su alcance más allá de las subestaciones. Finalmente, la edición 2.1 (Ed.2.1, 2020-2022) consolidó revisiones técnicas para asegurar compatibilidad retroactiva, actualizó requisitos de ciberseguridad y amplió la aplicación hacia la automatización de redes de distribución, generación renovable y recursos energéticos distribuidos (DERs). La IEC se encuentra en desarrollo de la tercera edición (Ed.3),

cuyo objetivo es lograr una integración total con arquitecturas de comunicación basadas en IEC 61850 para subestaciones digitales y su conexión con Smart Grids y IoT industrial.

3.3.1. Estructura del estándar

El estándar IEC 61850 se compone de un conjunto estructurado de partes que abordan los diferentes aspectos técnicos y conceptuales del sistema. Cada parte define un dominio particular dentro de la automatización eléctrica, conformando en su conjunto una arquitectura estandarizada que abarca desde la definición de los modelos de datos hasta los mecanismos de comunicación.

La Figura 18 muestra la estructura general del estándar IEC 61850. Este conjunto de partes conforma una estructura coherente y jerárquica que cubre todos los niveles de un sistema automatizado, desde la definición conceptual y semántica de los datos hasta su implementación física en las redes de comunicación.

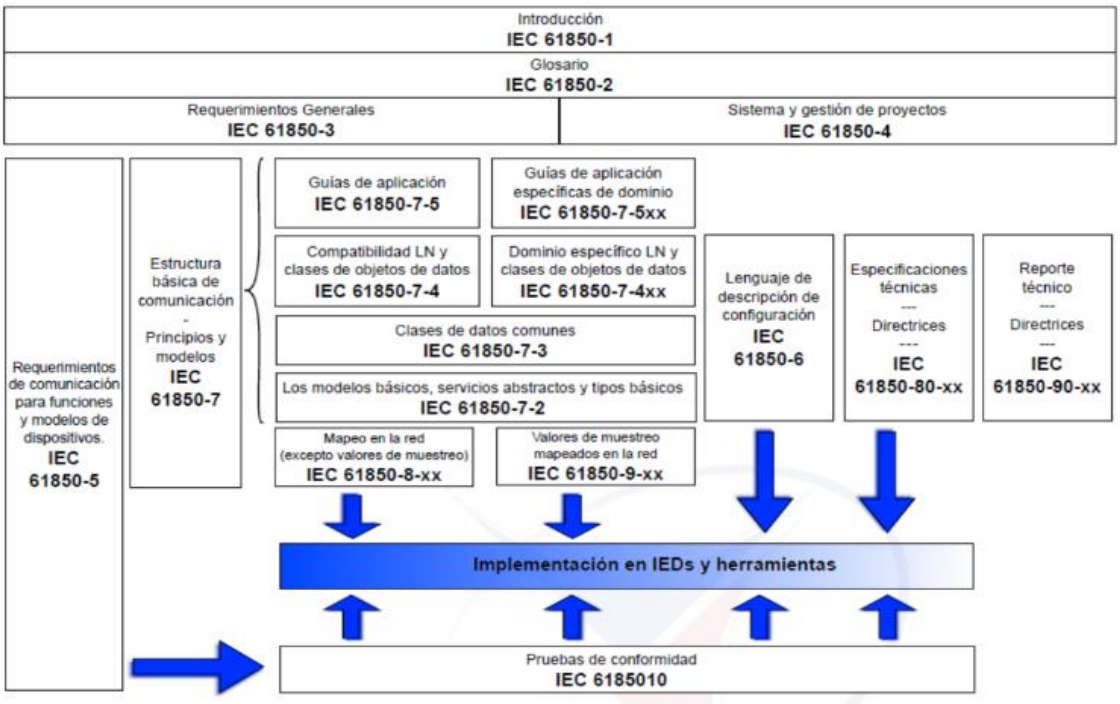


Figura 18: Estructura del estándar IEC 61850.

Las partes pueden describirse de la siguiente forma:

- IEC 61850-1 introduce los principios, alcances y objetivos generales del estándar, describiendo la arquitectura general de los sistemas de automatización de las empresas eléctricas (Power Utility Automation Systems) y la interacción entre sus componentes.
- IEC 61850-2 proporciona el glosario oficial de términos, abreviaturas y definiciones utilizadas en todas las partes del estándar.

- IEC 61850-3 establece los requisitos generales que deben cumplir los sistemas, incluyendo aspectos de confiabilidad, disponibilidad, seguridad, compatibilidad electromagnética y condiciones ambientales de operación.
- IEC 61850-4 define el proceso de ingeniería y gestión de proyectos, abordando el ciclo de vida completo del sistema, desde su diseño y parametrización hasta la puesta en servicio, mantenimiento y actualización de versiones.
- IEC 61850-5 especifica los requisitos funcionales y de comunicación entre dispositivos, detallando los tipos de datos, los tiempos de respuesta y las clases de desempeño requeridas para funciones de protección, control y supervisión.
- IEC 61850-6 introduce el Substation Configuration Language (SCL), un lenguaje basado en XML que permite describir la estructura física y lógica de la subestación, la configuración de IEDs, las redes de comunicación y las relaciones entre funciones. Este lenguaje se convirtió en una herramienta fundamental para el desarrollo de sistemas interoperables y reutilizables.
- Las partes IEC 61850-7-x conforman el núcleo lógico del estándar.
    - La parte 7-1 define los principios generales y la estructura de modelado.
    - La 7-2 describe los servicios abstractos de comunicación (ACSI) que establecen cómo los clientes y servidores intercambian información.
    - La 7-3 define las Common Data Classes (CDC), es decir, las clases de datos comunes que estructuran la información.
    - La 7-4 lista las clases de nodos lógicos (Logical Nodes) y sus funciones dentro del sistema.
    - La 7-410 y la 7-420 amplían el modelo para generación hidroeléctrica, eólica y recursos energéticos distribuidos.
- IEC 61850-8-1 especifica el mapeo de los servicios ACSI al protocolo MMS (Manufacturing Message Specification), utilizado principalmente para la comunicación cliente-servidor en redes Ethernet, así como la definición de mensajes GOOSE (Generic Object Oriented Substation Events) para el intercambio rápido de eventos en tiempo real.
- IEC 61850-9-2 define el mapeo para los valores muestreados (Sampled Values, SV) que transmiten medidas instantáneas de corriente y tensión entre transformadores de instrumentación y unidades de protección, permitiendo reemplazar el cableado analógico por comunicación digital sobre fibra óptica.

- IEC 61850-10 establece los procedimientos de pruebas de conformidad, determinando los criterios para verificar que un IED cumpla con los requisitos de interoperabilidad definidos por el estándar.

Debido a esto, IEC 61850 no se limita a ser un protocolo, sino un marco de ingeniería integral que define cómo diseñar, comunicar, probar y mantener un sistema eléctrico digital.

Si bien el estándar nació para las subestaciones de transmisión y distribución, su aplicación se ha extendido rápidamente hacia centrales generadoras, sistemas de energía renovable, microrredes, centros de control, redes de distribución inteligente (DER: Distributed Energy Resources) e incluso instalaciones industriales. Esta expansión responde al enfoque modular del modelo IEC 61850, que permite describir cualquier función de automatización utilizando nodos lógicos y estructuras de datos reutilizables, independientemente de la naturaleza del proceso físico.

El propósito esencial del estándar es facilitar la digitalización del sistema eléctrico de potencia, permitiendo el paso de un entorno basado en cableado y señales discretas, hacia uno basado en comunicación de datos sobre redes IP, en el cual las funciones de protección y control se distribuyen lógicamente entre los dispositivos. Esto reduce el cableado físico, simplifica la ingeniería, mejora la flexibilidad del sistema y posibilita la integración de nuevas tecnologías como sincronización temporal mediante PTP (Precision Time Protocol), procesamiento distribuido, inteligencia en el borde (edge computing) y analítica avanzada.

Entre las principales ventajas del estándar IEC 61850 frente a otros protocolos de automatización eléctrica se destaca, en primer lugar, la interoperabilidad entre dispositivos de distintos fabricantes, lograda mediante un modelo de datos estandarizado y orientado a objetos que garantiza una interpretación común de la información intercambiada. Esta característica reduce la dependencia de soluciones propietarias y facilita la integración de sistemas heterogéneos.

En términos de ingeniería y mantenimiento, el uso de mecanismos de auto descripción y del Substation Configuration Language (SCL) simplifica los procesos de configuración, puesta en servicio y ampliación de los sistemas, disminuyendo los tiempos de ingeniería y mejorando la gestión del ciclo de vida de las instalaciones.

Desde el punto de vista del desempeño, IEC 61850 incorpora servicios de comunicación diseñados para aplicaciones críticas del sistema eléctrico, como GOOSE y Sampled Values, que permiten la transmisión de eventos y mediciones con bajas latencias, habilitando esquemas avanzados de protección y control sobre redes Ethernet industriales.

Finalmente, su arquitectura abierta y escalable, basada en tecnologías de comunicación ampliamente adoptadas, facilita la evolución hacia subestaciones digitales y redes eléctricas inteligentes, permitiendo la integración de nuevas funciones y tecnologías sin modificaciones estructurales significativas.

3.3.2. Arquitectura

El estándar IEC 61850 propone una arquitectura de comunicación y control jerárquico que busca organizar el funcionamiento de una subestación eléctrica en distintos niveles funcionales, donde cada uno cumple tareas específicas y se comunica con los demás mediante redes normalizadas. Esta estructura permite integrar los equipos electrónicos inteligentes (IEDs), los sistemas de supervisión, los dispositivos de protección y los procesos eléctricos físicos dentro de un mismo entorno digital coherente.

La arquitectura de una subestación según IEC 61850 se basa en tres niveles principales: el nivel de estación, el nivel de bahía, y el nivel de proceso. Cada nivel agrupa funciones y dispositivos con roles definidos en el sistema de automatización, y su interacción se realiza a través de una infraestructura de comunicación basada en Ethernet industrial. [8]

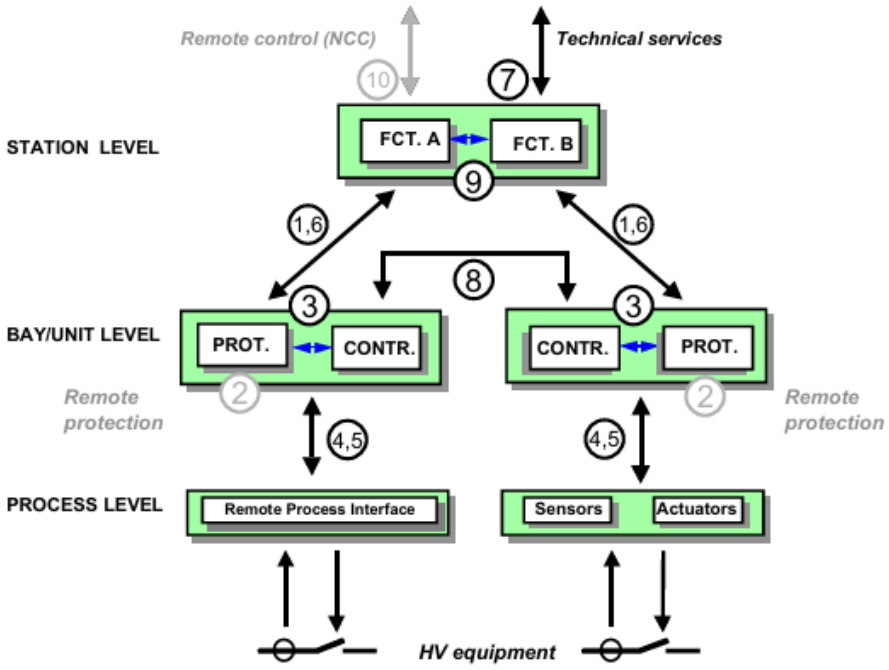


Figura 19: Niveles e interfaz lógica en SAS según IEC 61850-5.

1. **Nivel de estación**

En este nivel se concentran las funciones de supervisión, control general y coordinación de la subestación. Incluye los servidores SCADA locales, las interfaces hombre-máquina (HMI, Human Machine Interface), y los equipos encargados de la comunicación con el centro de

control. Su función principal es recibir información procesada desde los IEDs de bahía y proceso, visualizar el estado del sistema, registrar eventos y permitir el envío de comandos de operación. La comunicación con los niveles inferiores se realiza mediante el protocolo MMS (Manufacturing Message Specification), bajo el modelo cliente-servidor, donde el nivel de estación actúa generalmente como cliente que solicita o supervisa la información disponible en los IEDs.

2. **Nivel de bahía**

El nivel de bahía agrupa los dispositivos responsables de las funciones de protección, control y medida asociadas a una celda o circuito específico de la subestación. Aquí se ubican los IEDs de bahía, como relés de protección, controladores de interruptores, registradores de fallas y medidores inteligentes.

Estos dispositivos se comunican entre sí y con los niveles adyacentes utilizando tanto servicios MMS como GOOSE (Generic Object Oriented Substation Event), dependiendo de la criticidad y el tiempo de respuesta requerido.

El nivel de bahía constituye el núcleo funcional de la subestación, ya que integra la lógica de operación de cada circuito, coordina el disparo y enclavamiento de interruptores, y genera los reportes y alarmas que luego son gestionados en el nivel de estación.

3. **Nivel de proceso**

Este nivel representa la interfaz directa con el proceso físico, es decir, con los transformadores de corriente (TC), transformadores de tensión (TP), interruptores, seccionadores, y sensores. Tradicionalmente, las señales de estos equipos se transmitían a los relés mediante cableado analógico. Sin embargo, en la arquitectura IEC 61850 moderna, esta conexión se reemplaza por la digitalización de las señales mediante valores muestreados (Sampled Values, SV) y la utilización de módulos de entrada/salida digitales o Merging Units (MU). Las Merging Units capturan las señales analógicas de corriente y tensión, las convierten en datos digitales, y las transmiten sobre la red Ethernet del bus de proceso a los IEDs de bahía en tiempo real.

Este enfoque no solo reduce drásticamente el cableado, sino que también mejora la precisión y sincronización de las mediciones, permitiendo implementar subestaciones digitales completamente basadas en comunicación óptica.

La comunicación entre los distintos niveles de la arquitectura se realiza a través de dos buses principales: el bus de estación y el bus de proceso. El bus de estación conecta los dispositivos del nivel de bahía con los sistemas de supervisión del nivel de estación. Utiliza principalmente MMS sobre TCP/IP, adecuado para la transmisión de información no crítica en tiempo real, como reportes, registros o comandos operativos.

Por su parte, el bus de proceso conecta los dispositivos del nivel de bahía con el proceso físico, utilizando servicios como GOOSE y Sampled Values, diseñados para garantizar la entrega determinista de datos en tiempos del orden de microsegundos a milisegundos. Esta división permite asignar a cada tipo de comunicación la tecnología más adecuada, logrando simultáneamente velocidad, confiabilidad y flexibilidad.

Los mensajes GOOSE se utilizan para el intercambio rápido de eventos binarios entre dispositivos, como órdenes de disparo o enclavamientos lógicos. Gracias a su transmisión multicasting sobre Ethernet y su mecanismo de repetición periódica, garantizan alta disponibilidad y mínima latencia sin depender de servidores intermedios.

Los Sampled Values, en cambio, se emplean para el envío continuo de mediciones de corriente y tensión muestreadas a alta frecuencia, reemplazando los tradicionales circuitos de medida analógicos. Ambos servicios operan sobre una red compartida pero priorizada, lo que permite que la subestación actúe como un sistema distribuido de control en tiempo real.

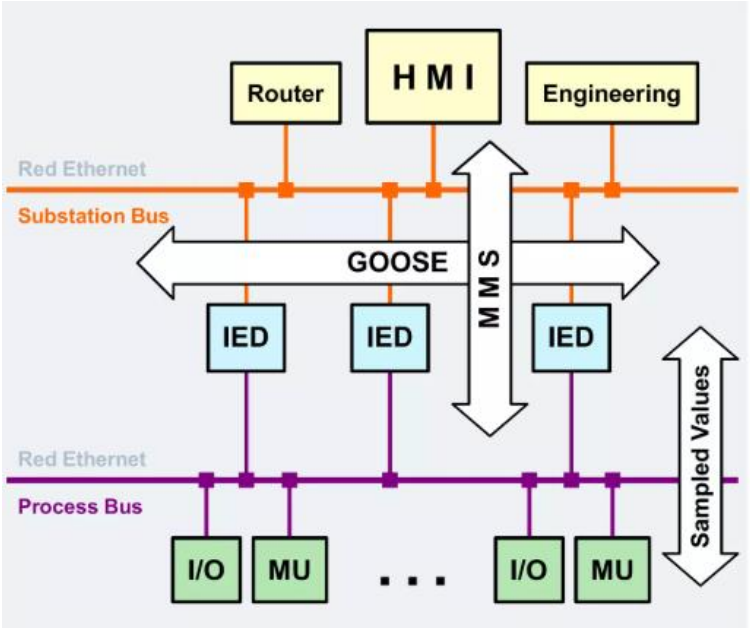


Figura 20: Interacción de protocolos definidos en IEC 61850 entre los niveles.

3.3.3. Lenguaje SCL y archivos

Uno de los pilares fundamentales del estándar IEC 61850 es la definición de un lenguaje formal para la descripción, configuración e intercambio de la información de ingeniería de los sistemas de automatización de subestaciones. Este lenguaje, denominado Substation Configuration Language (SCL), se encuentra especificado en la parte IEC 61850-6 y está basado en el estándar XML, lo que le otorga una estructura jerárquica, legible y extensible.

El objetivo principal del lenguaje SCL es permitir la interoperabilidad a nivel de ingeniería, posibilitando el intercambio de información de configuración entre herramientas de distintos

fabricantes sin ambigüedades. A diferencia de otros enfoques, en los cuales la ingeniería queda fuertemente ligada a software propietario, SCL establece una representación normalizada de la subestación, los dispositivos y las comunicaciones, independizándolos de la plataforma utilizada para su implementación.

Desde el punto de vista conceptual, el lenguaje SCL define distintos modelos de descripción, cada uno orientado a representar un aspecto particular del sistema de automatización. El modelo de subestación (Substation) describe la estructura eléctrica primaria, incluyendo el diagrama unifilar, los niveles de tensión, las bahías y los equipos de maniobra, así como los nodos lógicos asociados a cada función. El modelo de producto (Product) representa las capacidades funcionales de los IEDs, detallando los nodos lógicos disponibles, los tipos de datos soportados y los servicios de comunicación implementados. Por su parte, el modelo de comunicación (Communication) define la topología de red, los dispositivos de comunicación, las direcciones IP, VLANs y los parámetros necesarios para el intercambio de información. Finalmente, el modelo de flujo de datos (Data Flow) describe las relaciones de publicación y suscripción entre dispositivos, especificando cómo se intercambian los datos mediante servicios como GOOSE, Sampled Values o MMS.

Para materializar estos modelos, la norma IEC 61850 define distintos tipos de archivos SCL, cada uno con un propósito específico dentro del proceso de ingeniería.

El archivo ICD (IED Capability Description) describe las capacidades de un IED de forma genérica, independientemente de una subestación en particular. Incluye los nodos lógicos disponibles, los servicios soportados y las opciones de comunicación, y suele ser provisto por el fabricante como punto de partida para la ingeniería del sistema.

El archivo SSD (System Specification Description) representa la especificación funcional del sistema y describe la estructura de la subestación desde el punto de vista del proceso eléctrico, incluyendo el diagrama unifilar y los nodos lógicos requeridos, sin asociarlos aún a dispositivos físicos concretos. A partir de la combinación de los archivos ICD de los distintos IEDs y del archivo SSD, se genera el archivo SCD (Substation Configuration Description), que constituye la descripción completa del sistema de automatización. Este archivo contiene la topología de la subestación, los dispositivos instalados, la configuración de comunicaciones y las relaciones de intercambio de datos entre los IEDs.

Una vez definida la configuración global, el archivo SCD se utiliza para derivar los archivos CID (Configured IED Description), que contienen la configuración específica de cada IED y son finalmente descargados en los dispositivos durante la puesta en servicio. Adicionalmente, la norma contempla otros tipos de archivos, como el IID (Instantiated IED Description),

empleado en ciertos procesos de ingeniería incremental aunque su uso suele ser menos frecuente, y el SED (System Exchange Description), utilizado para el intercambio de información entre distintos sistemas.

La Figura 21 muestra un diagrama de flujo con los diferentes archivos SCL durante el proceso de configuración de un Sistema de Automatización de Subestaciones.

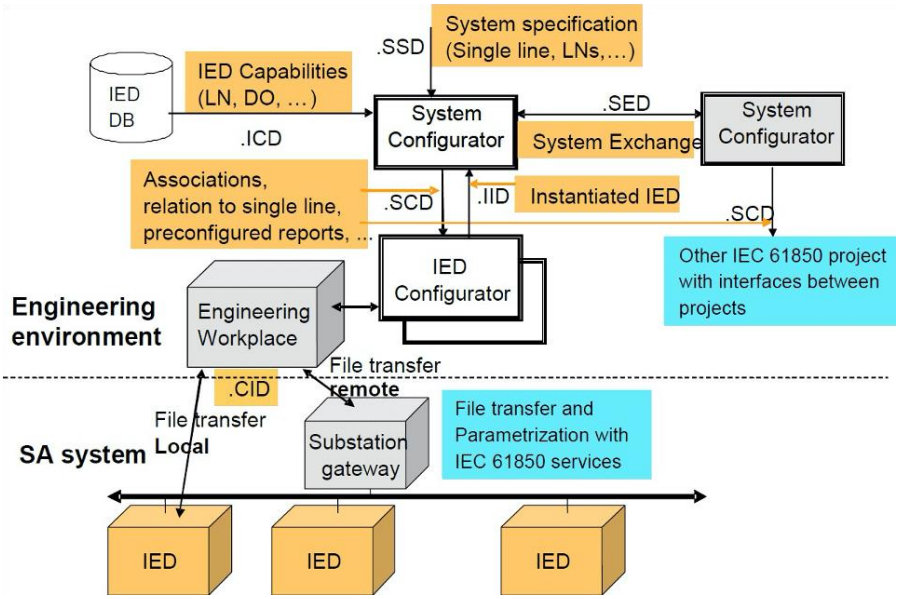


Figura 21: Archivos SCL durante el proceso de configuración de un SAS.

La utilización del lenguaje SCL permite estructurar el proceso de ingeniería de acuerdo con un flujo de trabajo estandarizado, en el cual la definición del sistema, la selección de dispositivos, la configuración de comunicaciones y la asignación de funciones se realizan de forma coherente y verificable. Este enfoque reduce errores de configuración, facilita la validación del sistema antes de su puesta en servicio y simplifica las tareas de mantenimiento y ampliación futuras. En este sentido, SCL no solo actúa como un formato de archivos, sino como un elemento clave para la estandarización de la ingeniería en subestaciones digitales basadas en IEC 61850.

3.3.4. Modelo de datos

Uno de los aspectos distintivos del estándar IEC 61850 es la adopción de un modelo de datos orientado a objetos para la representación de la información del sistema eléctrico. Esto implica que, en lugar de intercambiar señales o registros aislados, el estándar define estructuras de datos con significado semántico explícito, que representan funciones eléctricas, estados, mediciones y comportamientos asociados a los dispositivos de automatización. De esta manera, la información intercambiada en la red no solo contiene valores, sino también el contexto funcional necesario para su correcta interpretación. [9]

El enfoque orientado a objetos permite abstraer las funciones del sistema eléctrico de su implementación física, facilitando la interoperabilidad entre dispositivos de distintos fabricantes y la reutilización de modelos de datos en diferentes aplicaciones. Cada objeto definido por el estándar encapsula datos, atributos y reglas de acceso, lo que garantiza una representación coherente y normalizada de las funciones de protección, control y supervisión.

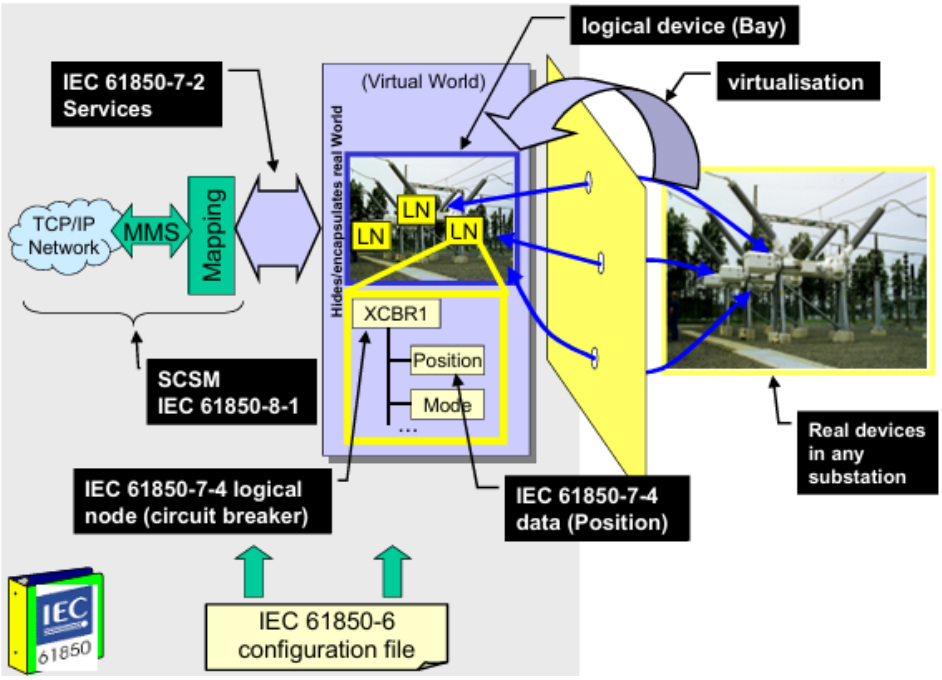


Figura 22: Concepto de modelado de datos según IEC 61850.

El modelo de datos IEC 61850 se estructura de forma jerárquica, siguiendo una organización lógica que permite descomponer el sistema en niveles funcionales claramente definidos.

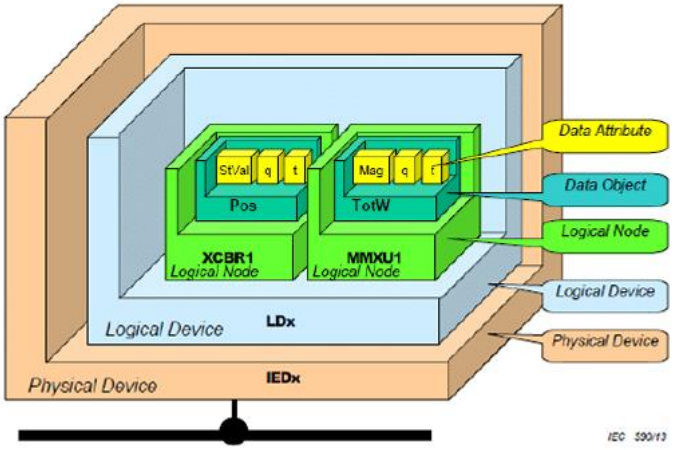


Figura 23: Jerarquía del modelo de datos IEC 61850.

En el nivel superior se encuentra el dispositivo lógico (Logical Device), que representa una agrupación funcional dentro de un IED. Cada dispositivo lógico contiene uno o más nodos lógicos (Logical Nodes), los cuales constituyen la unidad básica de modelado funcional del

estándar. A su vez, cada nodo lógico está compuesto por objetos de datos (Data Objects), que representan información específica asociada a una función, como estados, mediciones o comandos. Estos objetos de datos contienen atributos de datos (Data Attributes), que son los valores elementales intercambiados en la comunicación, tales como magnitudes eléctricas, banderas de estado, marcas temporales o indicadores de calidad.

Esta estructura jerárquica permite una representación ordenada y escalable del sistema, facilitando tanto la ingeniería como el acceso a la información por parte de los servicios de comunicación definidos por la norma.

El nodo lógico (Logical Node, LN) es el elemento central del modelo de datos IEC 61850. Cada nodo lógico representa una función específica del sistema eléctrico, como protección, control, medición, supervisión o señalización. Los nodos lógicos están normalizados por el estándar y se identifican mediante un nombre compuesto por cuatro caracteres, donde los primeros indican la clase funcional.

- I: Interfaces y registros
- A: Control Automático.
- C: Control.
- F: Bloques Funcionales.
- M: Medida.
- P: Protección.
- R: Relacionadas con protección.
- S: Supervisión y monitorización.
- T: Transformadores de instrumentación.
- X: Equipos de maniobra.
- Y: Transformadores de potencia.
- Z: Transformadores de potencia secundaria.
- L: Nodos lógicos del sistema.
- W: Energía Eólica (Ed2).
- G: Nodos lógicos genéricos.
- K: Nodos lógicos para equipamiento mecánico y no eléctrico.
- Q: Nodos lógicos para eventos de calidad de energía.

La norma IEC 61850 define un conjunto extenso de nodos lógicos que cubren funciones de protección, control, medición, supervisión y automatización, así como extensiones específicas para generación y recursos energéticos distribuidos. En estas tablas se presentan los nodos

lógicos más representativos y comúnmente utilizados en subestaciones digitales, mientras que la lista completa y detallada puede consultarse en la norma IEC 61850-7-4 y sus extensiones.

Nodo lógico	Descripción
LLN0	Nodo lógico común a todos los dispositivos lógicos (información general).
LPHD	Información física del dispositivo (IED).
LCCH	Control del canal de comunicación.
LPL	Información de placa y fabricante.

Tabla 1: Nodos lógicos del sistema.

Nodo lógico	Descripción
PTOC	Protección por sobrecorriente temporizada.
PIOC	Protección por sobrecorriente instantánea.
PTOV	Protección por sobretensión.
PDIS	Protección de distancia.
PDIF	Protección diferencial.
PTEF	Protección por falla a tierra.
PDPR	Protección direccional de potencia.
PTRC	Coordinación y emisión de órdenes de disparo de protección.
PSCH	Esquemas especiales de protección.

Tabla 2: Nodos lógicos de protección.

Nodo lógico	Descripción
CSWI	Control de conmutación y operación de dispositivos de maniobra.
CILO	Gestión de enclavamientos lógicos para operaciones seguras.
CALH	Generación de alarmas por superación de umbrales.
CPOW	Control de potencia activa y reactiva.

Tabla 3: Nodos lógicos de control.

Nodo lógico	Descripción
ATCC	Control automático del cambiador de tomas bajo carga.
AVCO	Control automático de tensión.
ARCO	Función de reconexión automática.
ANCR	Control automático de redes o esquemas de reconexión.
AZVT	Automatización basada en niveles de tensión.

Tabla 4: Nodos lógicos de automatización.

Nodo lógico	Descripción
MMXU	Medición trifásica de magnitudes eléctricas (V, I, P, Q, f).
MMTR	Medición de energía eléctrica (activa y reactiva).
MSQI	Medición de calidad de energía (armónicos, desequilibrios).
MHAI	Medición de armónicos individuales.

Tabla 5: Nodos lógicos de medición.

Nodo lógico	Descripción
XCBR	Control y supervisión de interruptores de potencia.
XSWI	Control y supervisión de seccionadores.

Tabla 6: Nodos lógicos de maniobra.

Nodo lógico	Descripción
GGIO	Entradas/Salidas genéricas y no contempladas en los demás nodos lógicos.

Tabla 7: Nodos lógicos genéricos.

En el modelo de datos IEC 61850, cada nodo lógico está compuesto por un conjunto predefinido de objetos de datos (*Data Objects*) que representan los distintos aspectos funcionales de la función modelada. Estos objetos de datos describen información específica asociada al comportamiento del nodo lógico, como estados operativos, valores medidos, órdenes de control, señales de alarma o condiciones de supervisión.

Cada objeto de datos está tipificado mediante una *Common Data Class* (CDC), la cual define su estructura interna y los atributos que lo componen. Las CDC establecen el conjunto de atributos de datos obligatorios y opcionales, así como su organización y tipos de dato, asegurando que objetos de datos con la misma función presenten una estructura común, independientemente del nodo lógico o del fabricante. De este modo, la CDC actúa como un vínculo formal entre la función representada por el nodo lógico y la información elemental intercambiada en la comunicación.

La Tabla 8 muestra los tipos de Common Data Class definidos en el estándar IEC 61850-7-1 anexo A, ordenados por tipo de información.

Tipo de información	CDC	Nombre
Estado	SPS	Single Point Status
	DPS	Double Point Status
	INS	Integer Status
	SIG	Status Indication Group
	ISI	Integer Status Information
	ACT	Protection Activation Information
	ACD	Directional Protection Activation Information
	SEC	Security Violation Counting
	BCR	Binary Counter Reading
Medición	MV	Measured Value
	CMV	Complex Measured Value
	SAV	Sampled Value
	WYE	Wye Measured Value
	DEL	Delta Measured Value
	SEQ	Sequence Measured Value
	HVWYE	Harmonic Value for WYE
	HDEL	Harmonic Value for DEL
Controlable – Estado	SPC	Controllable Single Point
	DPC	Controllable Double Point
	INC	Controllable Integer Status
	BSC	Binary Controlled Step Position
	ISC	Integer Controlled Step Position
Controlable – Analógico	APC	Controllable Analogue Point

Configuración – Estado	SPG	Single Point Setting
	ING	Integer Setting
Configuración – Analógica	ASG	Analogue Setting
	CURVE	Setting Curve
Descripción	DPL	Device Name Plate
	LPL	Logical Node Name Plate
	CSD	Curve Shape Description

Tabla 8: Common Data Class según tipo de información.

La Figura 24 muestra un ejemplo de una lista de clases de datos para un interruptor automático. El nombre de la clase es "XCBR". Las clases de datos que componen el interruptor automático se agrupan en tres categorías (información básica de LN, datos controlables e información de estado). Para ser más precisos, cada clase de datos también tiene una clase de datos común que define los detalles, es decir, los atributos de la clase de datos. La última columna especifica si esta clase de datos es obligatoria (M) u opcional (O).

Logical Node: Circuit breaker		Name: XCBR	
Data-Class	**DataName**	**Common Data Class (CDC)**	**M/O**
Basic Logical Node information			
Mode	Mod	INC - Controllable Integer Status	M
Behaviour	Beh	INS - Integer Status	M
Health	Health	INS - Integer Status	M
Name plate	NamPlt	LPL - Logical node name plate	M
Local operation (local means without sub-station automation communication, hard-wired direct control)	Loc	SPS - Single point status	
External equipment health	EEHealth	INS - Integer Status	
External equipment name plate	EEName	DPL - Device name plate	
Operation counter	OpCnt	INS - Integer Status	
Controllable Data			
Switch position	Pos	**DPC - Controllable Double Point**	M
Block opening	BlkOpn	SPC - Controllable Single Point	M
Block closing	BlkCls	SPC - Controllable Single Point	M
Charger motor enabled	ChMotEna	SPC - Controllable Single Point	O
Metered Values			
Sum of Switched Amperes, resetable	SumSwARs	BCR - Binary counter reading	O
Status information			
Circuit breaker operating capability	CBOpCap	INS - Integer Status	M
Point On Wave switching capability	POWCap	INS - Integer Status	O
Circuit breaker operating capability when fully charged	MaxOpCap	INS - Integer Status	O

Figura 24: Data Object y Common Data Class definidos para el NL XCBR.

Los atributos definidos por cada CDC constituyen el nivel más bajo del modelo de datos y contienen los valores concretos utilizados por el sistema, tales como estados binarios, magnitudes analógicas, indicadores de calidad o marcas temporales. Estos atributos se asocian a restricciones funcionales (*Functional Constraints*) que determinan su rol dentro del sistema, como valores de estado, medición, configuración o control. En la Tabla 9 se pueden observar los tipos de FC definidos en el estándar.

FC	Nombre	Tipo de información
ST	Status information	Estados instantáneos y condiciones operativas.
MX	Measurand values	Valores medidos instantáneos.
SP	Setpoint	Valores de consigna ajustables.
SV	Substitution values	Valores sustitutos utilizados ante fallas o pruebas.
CF	Configuration	Parámetros de configuración del dispositivo o función.
DC	Description	Información descriptiva del modelo o del dispositivo.
SG	Setting group	Parámetros pertenecientes a grupos de ajuste.
SE	Setting group editable	Parámetros de ajuste editables en línea.
EX	Extended definition	Información extendida o específica del fabricante.
CO	Control	Atributos asociados a órdenes de control.
RP	Report	Información utilizada para servicios de reporte.
BR	Buffered report	Información utilizada en reportes con buffer.
LG	Logging	Información destinada a registros de eventos.
GO	GOOSE	Información publicada mediante mensajes GOOSE.
GS	GOOSE status	Estado y supervisión de publicaciones GOOSE.
MS	MMS services	Información relacionada con servicios MMS.
US	Unbuffered report	Información para reportes sin buffer.

Tabla 9: Functional Constraints definidos por IEC 61850.

Esta clasificación permite que los servicios de comunicación accedan de manera selectiva y estructurada a la información, manteniendo la coherencia entre el modelo de datos y los mecanismos de intercambio definidos por la norma.

DPC class					
Attribute name	**Attribute type**	**FC**	**TrgOp**	**Value/value range**	**M/O/C**
DataName	Inherited from Data Class (see IEC 61850-7-2)				
DataAttribute					
control and status					
ctlVal	BOOLEAN	CO		off (FALSE) | on (TRUE)	AC_CO_M
operTim	TimeStamp	CO			AC_CO_O
origin	Originator	CO, ST			AC_CO_O
ctlNum	INT8U	CO, ST		0..255	AC_CO_O
stVal	CODED ENUM	ST	dchg	intermediate-state | off | on | bad-state	M
q	Quality	ST	qchg		M
t	TimeStamp	ST			M
stSeld	BOOLEAN	ST	dchg		AC_CO_O
substitution					
subEna	BOOLEAN	SV			PICS_SUBST
subVal	CODED ENUM	SV		intermediate-state | off | on | bad-state	PICS_SUBST
subQ	Quality	SV			PICS_SUBST
subID	VISIBLE STRING64	SV			PICS_SUBST
configuration, description and extension					
pulseConfig	PulseConfig	CF			AC_CO_O
ctlModel	CtlModels	CF			M
sboTimeout	INT32U	CF			AC_CO_O
sboClass	SboClasses	CF			AC_CO_O
d	VISIBLE STRING255	DC		Text	O
cdcNs	VISIBLE STRING255	EX			AC_DLNDA_M
cdcName	VISIBLE STRING255	EX			AC_DLNDA_M
dataNs	VISIBLE STRING255	EX			AC_DLN_M

Figura 25: DataAttribute definidos para un CDC DPC.

En la Figura 25 se observan los atributos definidos para el CDC DPC (Controllable Double Point). Se puede apreciar que cada atributo tiene asociado propiedades predefinidas como el

tipo (attribute type) y la restricción funcional (FC). Con esto se intenta aclarar que el tipo de variable en cada atributo es independiente de la restricción funcional.

La nomenclatura de cada atributo mostrado por un cliente IEC61850 será una composición del Dispositivo Físico, Dispositivo Lógico, Nodo Lógico y DataObject al que pertenece, finalizado con el nombre del atributo, por ejemplo:

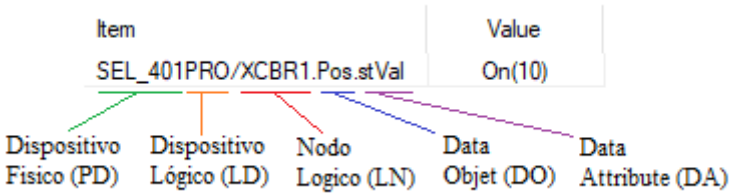


Figura 26: Conformación de la nomenclatura de un atributo.

3.3.5. Servicios de comunicación

El estándar IEC 61850 define un conjunto de servicios de comunicación destinados a permitir el intercambio de información entre los distintos dispositivos de un sistema de automatización de subestaciones, manteniendo la coherencia con el modelo de datos orientado a objetos previamente descripto. A diferencia de los protocolos tradicionales, IEC 61850 separa claramente la definición abstracta de los servicios de su implementación sobre protocolos de comunicación concretos, lo que constituye una de sus principales innovaciones.

3.3.5.1. Interfaz abstracta de servicios de comunicación (ACSI)

En el núcleo del estándar se encuentra la Abstract Communication Service Interface (ACSI), especificada en la norma IEC 61850-7-2.

ACSI define, de forma abstracta e independiente de cualquier tecnología de red, el comportamiento funcional del intercambio de información entre clientes y servidores para acceder al modelo de datos IEC 61850.

Estos se definen como servicios e incluyen

- Servicio de acceso: permite leer, escribir o acceder a atributos específicos.
- Servicio de control: Permite ejecutar acciones remotas. Incluye los mecanismos de seguridad Control Directo, Select Before Operate y Confirmaciones de ejecución.
- Servicios de reportes: Permite el envío automático de información ante eventos.
- Servicios de logging: permite el registro de eventos y acceder a los mismos.
- Servicios de gestión: del servidor, dataset, reportes o archivos.

La función principal de ACSI es proporcionar una interfaz lógica común entre el modelo de datos y los mecanismos de comunicación, garantizando que las funciones del sistema puedan

ser implementadas sobre diferentes tecnologías de transporte sin modificar su semántica. De este modo, ACSI actúa como un puente entre el modelo orientado a objetos y los protocolos reales utilizados en la red de subestación.

3.3.5.2. DataSet

Dentro del marco de los servicios definidos por la Interfaz Abstracta de Servicios de Comunicación (ACSI), los DataSets constituyen el mecanismo mediante el cual se selecciona y agrupa la información del modelo de datos que será intercambiada entre los dispositivos del sistema. Un DataSet es una colección lógica de objetos de datos o atributos de datos, pertenecientes a uno o varios nodos lógicos, que se consideran como una unidad coherente para su transmisión.

El propósito principal de los DataSets es desacoplar la definición de la información de los servicios de comunicación que la transportan. De este modo, un mismo DataSet puede ser utilizado por distintos servicios, como reportes MMS, mensajes GOOSE o transmisión de Sampled Values, sin necesidad de redefinir la información en cada caso. Esta separación mejora la modularidad del sistema y simplifica el proceso de ingeniería.

Los DataSets se definen y configuran durante la etapa de ingeniería del sistema, habitualmente mediante archivos SCL, y son referenciados por los distintos bloques de control asociados a los servicios de comunicación.

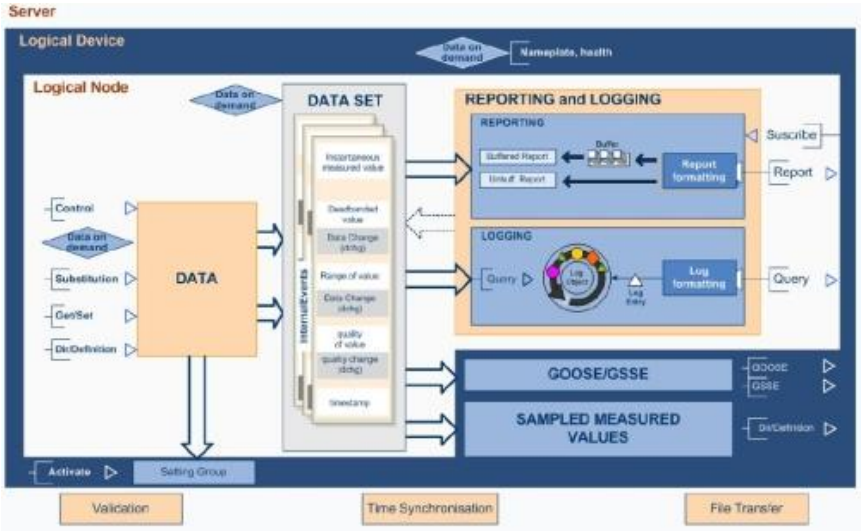


Figura 27: Relación entre el modelo de datos, DataSet y servicios de comunicación.

3.3.5.3. Manufacturing Message Specification

La implementación más extendida de los servicios ACSI para comunicaciones cliente-servidor se realiza mediante el protocolo Manufacturing Message Specification (MMS), definido originalmente en el estándar ISO 9506. En IEC 61850, MMS se utiliza para el intercambio de

información entre IEDs y sistemas de nivel de estación, como HMI, SCADA o estaciones de ingeniería.

MMS opera bajo un modelo cliente-servidor, donde los IEDs actúan como servidores que exponen su modelo de datos, y los sistemas de supervisión actúan como clientes que acceden a dicha información. A través de MMS se implementan servicios como la lectura de valores medidos, la consulta de estados, la escritura de parámetros de configuración y la ejecución de órdenes de control.

Un mecanismo central asociado a MMS es el uso de Report Control Blocks (RCB), los cuales permiten notificar automáticamente cambios en el modelo de datos sin necesidad de consultas periódicas. Los RCB definen qué información será reportada, bajo qué condiciones y hacia qué destinatarios. Existen dos tipos principales de RCB:

Unbuffered Report Control Blocks (URCB): generan reportes en tiempo real, sin almacenamiento intermedio. Si el cliente no está disponible en el momento del evento, la información se pierde.

Buffered Report Control Blocks (BRCB): almacenan los eventos en un buffer interno del IED, permitiendo su transmisión posterior cuando se restablece la comunicación con el cliente.

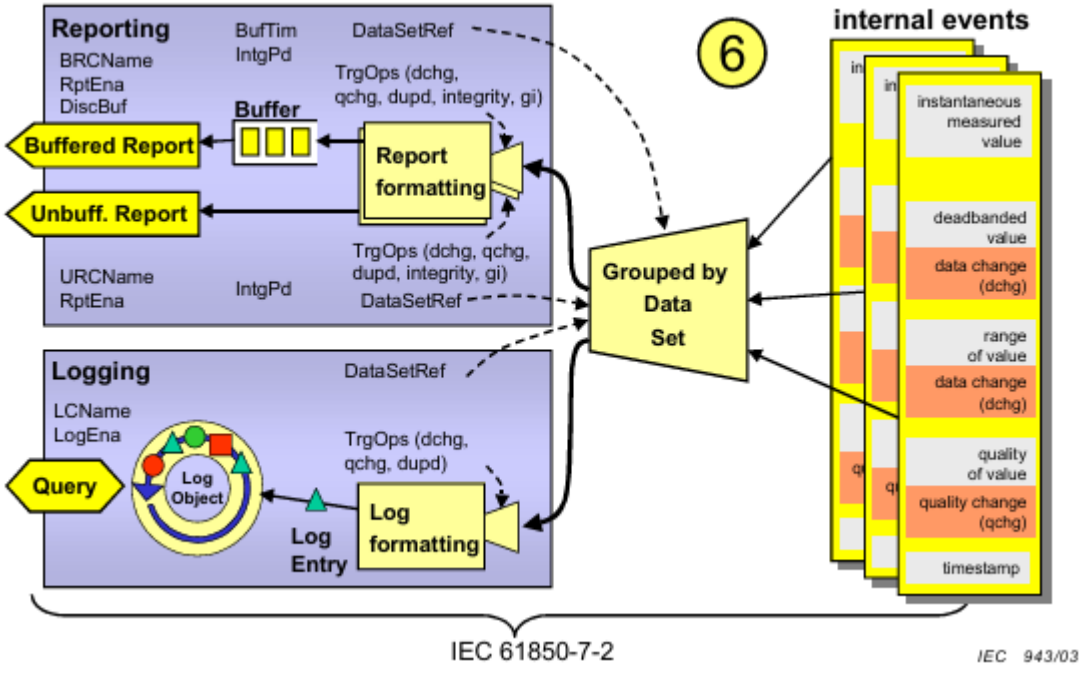


Figura 28: Modelo conceptual de Reportes y logging.

Los reportes se activan mediante triggers, que determinan cuándo se genera un reporte. Este mecanismo permite optimizar el tráfico de red y asegurar que solo la información relevante sea transmitida. En la Figura 29 se muestran las condiciones de disparo (TriggerConditions) definidos en el estándar IEC 61850-7-2.

TriggerConditions type			
Attribute name	Attribute type	TriggerOption (TrgOp) for use in DataAttributes	Value/value range/explanation
	PACKED LIST		
data-change	BOOLEAN	dchg	Trigger used in **DATA-Attributes** determined by common **DATA** classes of IEC 61850-7-3
quality-change	BOOLEAN	qchg	Trigger used in **DATA-Attributes** determined by common **DATA** classes of IEC 61850-7-3
data-update	BOOLEAN	dupd	Trigger used in **DATA-Attributes** determined by common **DATA** classes of IEC 61850-7-3
integrity	BOOLEAN	–	Trigger whose value (time) can be set by a service or by configuration; independent of an instance of **DATA**
general-interrogation	BOOLEAN	–	Trigger whose value (initiate general interrogation) can be set by a service or by configuration; independent of an instance of **DATA**

Figura 29: Tipos de TriggerConditions (Condición de disparo) según IEC 61850-7-2.

3.3.5.4. Generic Object Oriented Substation Event

Para aplicaciones que requieren tiempos de transmisión extremadamente bajos y alta confiabilidad, como la protección y el control rápido, IEC 61850 define el servicio GOOSE (Generic Object Oriented Substation Event). A diferencia de MMS, GOOSE utiliza un modelo de publicación-suscripción, donde los dispositivos publican eventos en la red y múltiples suscriptores pueden recibirlos simultáneamente. [10]

Los mensajes GOOSE se transmiten directamente sobre Ethernet, sin utilizar protocolos de capas superiores como TCP o UDP, lo que reduce significativamente la latencia. La información transmitida corresponde a DataSets previamente configurados y se envía de forma multicast a todos los dispositivos interesados.

La configuración de una publicación GOOSE se realiza mediante un GOOSE Control Block (GoCB), que define el DataSet asociado, los parámetros de retransmisión y las direcciones de comunicación. Ante la ocurrencia de un evento, el mensaje GOOSE se retransmite varias veces siguiendo una secuencia temporal decreciente, lo que aumenta la probabilidad de recepción incluso en presencia de pérdidas de paquetes.

GOOSE es ampliamente utilizado para la transmisión de órdenes de disparo, enclavamientos lógicos y señales de protección, reemplazando el cableado físico tradicional por comunicación digital de alta velocidad.

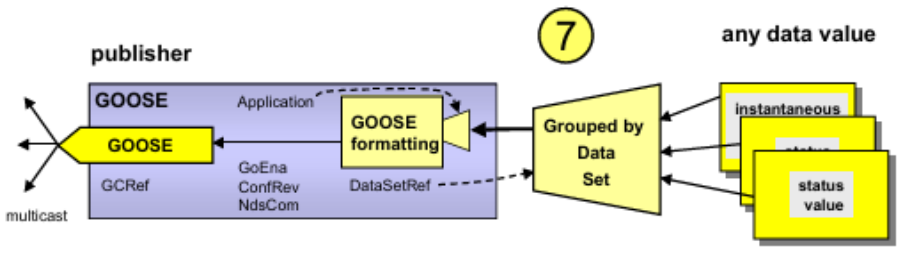


Figura 30: Modelo conceptual de publicación GOOSE.

3.3.5.5. Sampled Values

El servicio Sampled Values (SV) está destinado a la transmisión de valores muestreados de magnitudes eléctricas, como corrientes y tensiones, desde el nivel de proceso hacia los dispositivos de protección y control. Este servicio es fundamental en arquitecturas de subestaciones digitales, donde los transformadores de medida convencionales son reemplazados o complementados por dispositivos electrónicos de medición.

Al igual que GOOSE, Sampled Values utiliza comunicación Ethernet en modo multicast, permitiendo que múltiples dispositivos suscriptores reciban simultáneamente las muestras. Los datos transmitidos corresponden a valores muestreados a alta frecuencia, organizados en tramas temporales y asociados a marcas de tiempo precisas.

La utilización de SV permite reducir el cableado de cobre, mejorar la flexibilidad del sistema y facilitar la sincronización de mediciones en esquemas de protección y control avanzados.

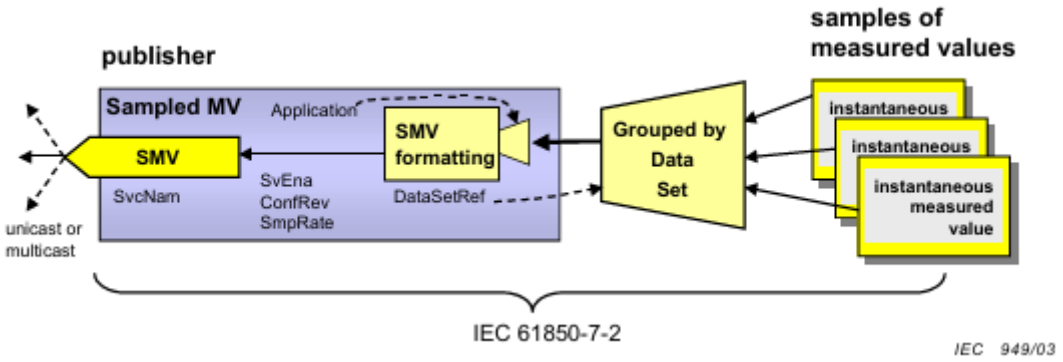


Figura 31: Modelo conceptual de publicación SV.

Desde el punto de vista del modelo OSI, los servicios de comunicación IEC 61850 presentan características diferenciadas según el tipo de servicio. MMS se apoya en capas superiores del modelo, utilizando TCP/IP y proporcionando mecanismos de comunicación confiables orientados a sesión, lo que lo hace adecuado para aplicaciones de supervisión y configuración. Por el contrario, GOOSE y Sampled Values operan directamente sobre la capa de enlace de datos, encapsulados en tramas Ethernet, lo que permite minimizar la latencia y garantizar tiempos de respuesta determinísticos. Esta coexistencia de distintos mecanismos de comunicación dentro del mismo estándar refleja el enfoque flexible de IEC 61850, que selecciona la tecnología de transporte más adecuada según los requisitos temporales y funcionales de cada aplicación.

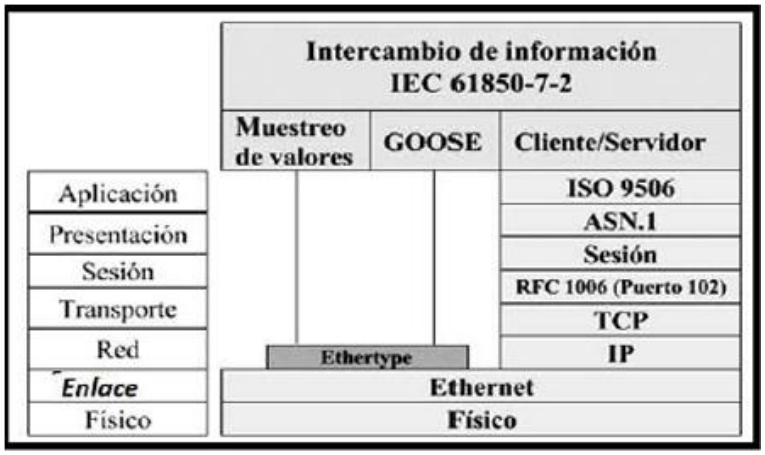


Figura 32: Mapeo de servicios IEC 61850 sobre el modelo OSI.

En conjunto, los servicios de comunicación definidos por IEC 61850 permiten implementar arquitecturas de automatización altamente integradas, combinando comunicaciones cliente/servidor, transmisión de eventos en tiempo real y envío de valores muestreados de alta velocidad. Esta diversidad de servicios, alineada con un modelo de datos estandarizado, constituye uno de los principales factores que han impulsado la adopción de IEC 61850 como base para la digitalización de las subestaciones eléctricas modernas.

4. Diseño e implementación

Como se mencionó en los apartados anteriores, los sistemas automatizados de protección y control de una estación transformadora cuentan con la intervención de diversos equipos y señales que interactúan con el fin de brindar un servicio de calidad y seguridad del sistema eléctrico.

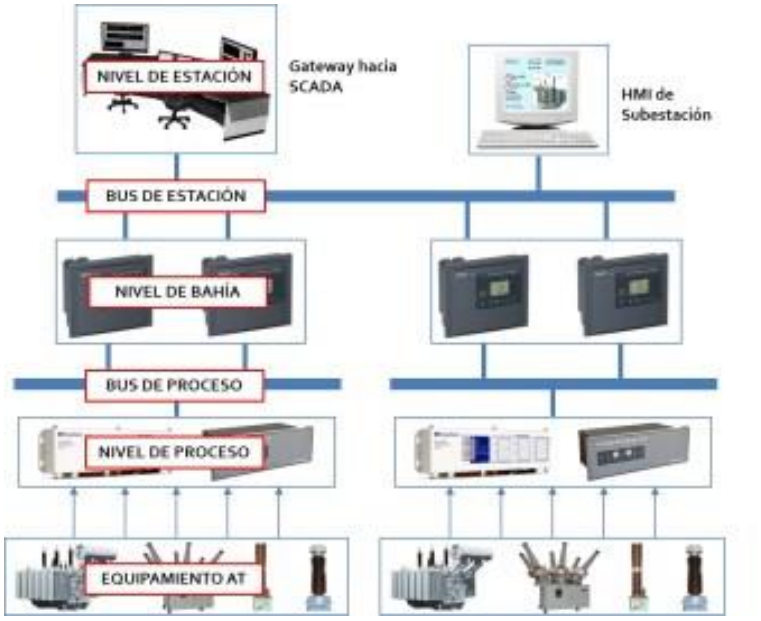


Figura 33: Arquitectura de sistema automático en estaciones transformadoras.

En el desarrollo de este proyecto se simularon equipos con el fin de recrear una pequeña estación transformadora, donde el intercambio de información se realiza bajo el estándar IEC 61850 e intentando representar todos los niveles de la estación. Para ello fue necesario modelar desde los equipos en playa, los equipos de recolección de información, los equipos de protección y control, y los sistemas de supervisión a nivel de estación.

4.1. Diseño del laboratorio

4.1.1. Escenario ideal

En esta sección se detalla la idea del proyecto y cómo se llevó a cabo el mismo a partir de herramientas y aplicaciones de uso libre o versiones de prueba. Para ello lo principal es definir el alcance del proyecto y todo lo que lo conforma.

Inicialmente la idea comenzó con la simulación y modelado de una estación transformadora con dos salidas de línea. El transformador de potencia se vincula a las barras mediante un esquema de interruptor simple, con sus respectivos seccionadores y transformadores de medición. Las dos salidas de línea de media tensión, que de ahora en más se denominarán

alimentadores, cuentan con el mismo esquema, es decir, interruptor simple, acompañado de sus seccionadores y un transformador de corriente.

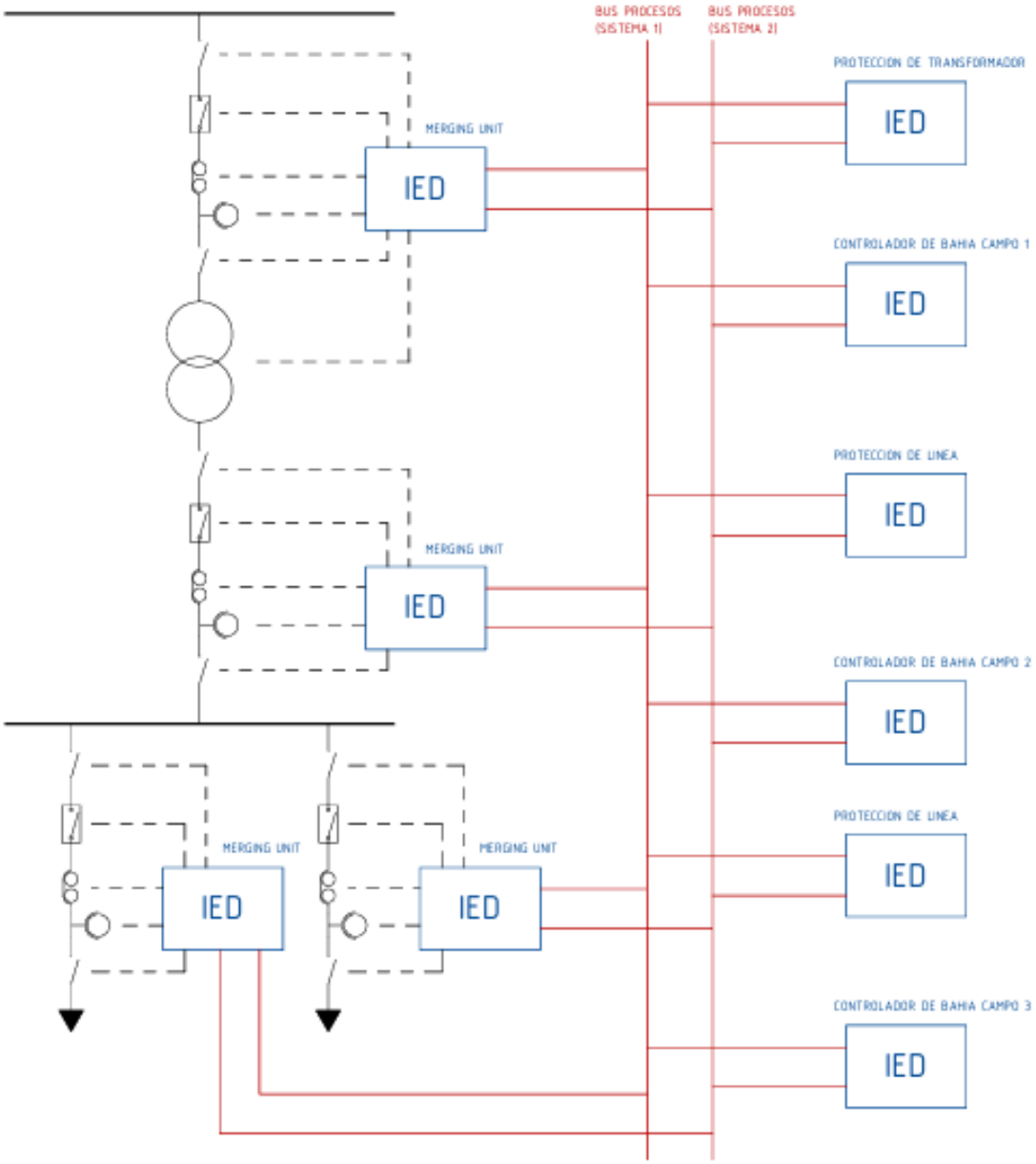


Figura 34: Esquema unifilar ideal de la ET.

El diseño de la misma se pensó inicialmente bajo el estándar IEC 61850, por lo tanto las señales de playa se llevan a los equipos de protección y control haciendo uso de Merging Unit. Se ubicó una MU por cada interruptor y en cercanía del mismo para minimizar la cantidad de cable. Las señales asociadas al transformador se asociaron a la MU del nivel de Alta Tensión.

Con respecto a los equipos de protección y control el campo de transformación cuenta con un relé de transformador, con funciones de protección típicas como pueden ser sobrecorriente, diferencial, sobretensión, protecciones de tierra y propias del transformador. Por otra parte la

operación segura de estos equipos dependerá de una Unidad de bahía capaz de realizar la apertura y cierre de todos los elementos del campo, verificando las condiciones de seguridad necesarias como son los enclavamientos o sincrocheck, desde el gabinete de protección.
Para los alimentadores se consideró una protección de línea, con características típicas como pueden ser de sobrecorriente y distancia únicamente y un controlador de bahía para la operación.

4.1.2. Consideraciones y limitaciones

Debido al elevado nivel de complejidad que conlleva desarrollar un proyecto con estas características, debido a la cantidad de señales y elementos que posee, se procedió a limitar el alcance del mismo y restringir determinadas funciones intentando mantener la fidelidad con un proyecto real. Esto con el fin de diseñar un entorno de aprendizaje sencillo y accesible con las características básicas del estándar IEC 61850.
A continuación se describen las simplificaciones aplicadas para que este proyecto sea factible:

- Solo se cuenta con una Raspberry Pi por lo que todas las MU se simplificaron en una con todas las señales de los equipos de playa.
- Debido a la limitada cantidad de pines en la Raspberry Pi se decidió simplificar la cantidad de entradas/salidas digitales, por lo que los comandos y estados en los interruptores serán únicamente tripolares.
- No se consideraron seccionadores debido a la limitación de entradas y salidas digitales y para simplificar la lógica de los IED.
- Para este proyecto se decidió no implementar entradas analógicas, debido a que la Raspberry Pi no cuenta nativamente con entradas de este tipo, y por lo tanto tampoco se consideran equipos de medición en el esquema unifilar.
- Por la falta de mediciones se simplificaron las funciones de los IED de protección y se limitó a una actuación inmediata producto de una falla en el transformador.
- A causa de la cantidad limitada de puertos Ethernet de los equipos utilizados, solo se utilizó una red LAN unificando bus de estación y de procesos.
- Se omitió la sincronización horaria debido a la limitación de hardware, delegando la precisión al reloj de los sistemas operativos.
- Se implemento un enfoque Bottom-Up aunque de forma incompleta, ya que la generación de archivos SCD queda fuera del alcance de este proyecto.

Aplicando las condiciones mencionadas y simplificando los equipos a implementar el esquema unifilar resultante para el proyecto es el siguiente:

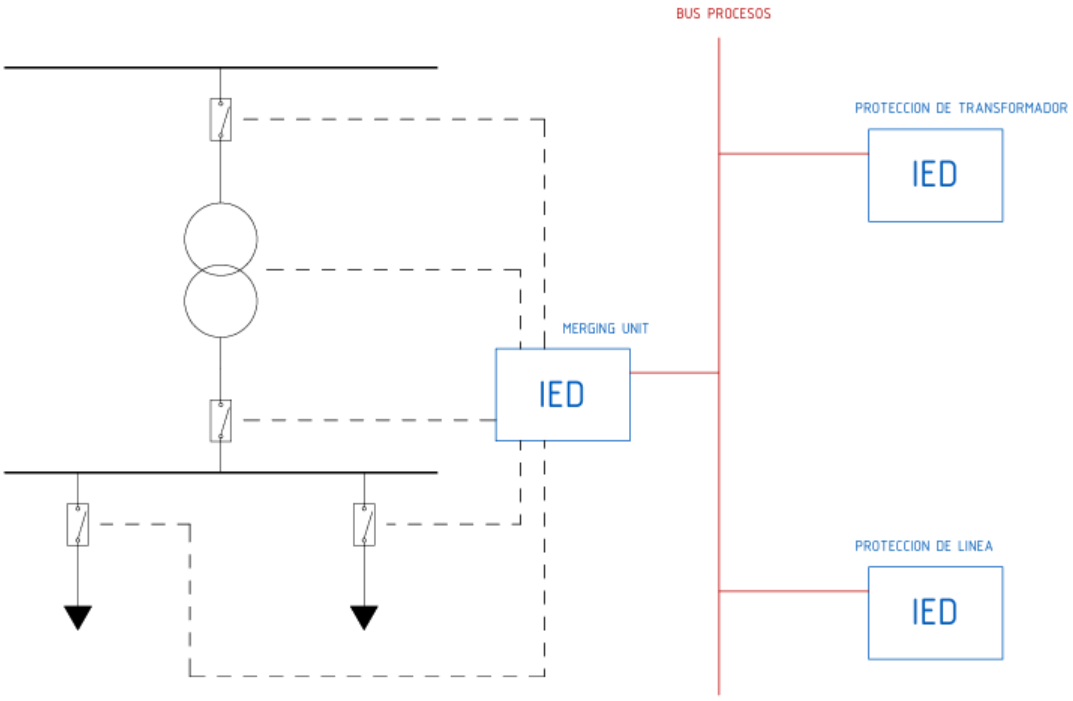


Figura 35: Esquema unifilar simplificado utilizado en el proyecto

Debido a que los IED se modelarán en función de las características necesarias y se programaron para ser ejecutados en un dispositivo electrónico, se simplificará el tablero de protección del campo de transformación por un único IED para la protección y control del transformador. Es decir, este será el encargado de controlar los dos interruptores asociados al transformador y reportar sus estados o alarmas a niveles lógicos superiores.

Como ya se mencionó, este proyecto no considera la aplicación del protocolo Sample Values ni de mediciones analógicas y por lo tanto no se puede avanzar con una lógica de protecciones muy compleja. Se considerarán dos señales digitales proveniente de las protecciones propias del transformador y representarán fallas graves que deben ser despejadas inmediatamente. Estas señales se simularán mediante el uso de dos pulsadores ubicados en tablero que representa la playa y al ser recibidas por la protección producirán los disparos y bloqueos de los interruptores asociados al transformador. El bloqueo puede ser retirado con otro pulsador del tablero y esto permitirá operar nuevamente los interruptores.

Los interruptores se representarán, de forma muy simple en el tablero mediante luces led que indican si se encuentran en estado abierto o cerrado. También se decidió instalar en este tablero pulsadores de apertura y cierre para cada interruptor y llaves selectoras para permitir la operación local, como si se operara a pie de equipo, o remota de los mismos.

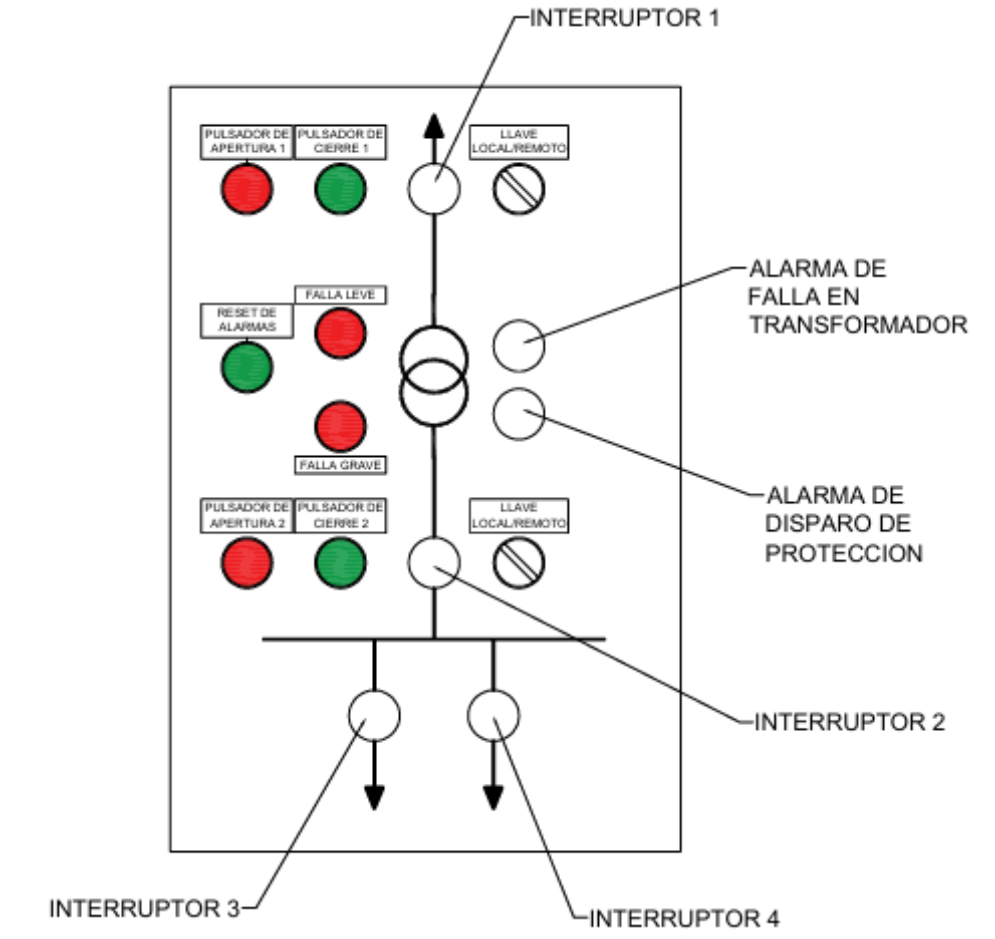


Figura 36: Esquema topográfico del tablero de comando.

Para la protección y control de los interruptores asociados a la barra se utiliza un único IED. En este caso el IED utilizado no será uno propiamente desarrollado, sino uno virtual desarrollado por **Axon Group**, por lo que se admitirán ciertas limitaciones. Si bien no existe problema en la comunicación entre dispositivos, este IED se encuentra limitado debido a que su configuración no puede ser modificada, evitando así la emisión o suscripción a nuevos mensajes GOOSE que permitan la integración total con este proyecto. Para evitar aumentar la complejidad innecesariamente los interruptores asociados a este IED solo serán representados por luces led, y no tendrán la posibilidad de ser operados desde el tablero.

4.1.3. Arquitectura de red

Si bien la norma específica buses de comunicación, tanto de procesos como de estación, la limitante de utilizar equipos que solo cuentan con un puerto de comunicación Ethernet se unificaron los buses en una única red LAN.

Por otra parte, la norma específica que debe asegurarse la comunicación con tiempo de convergencia cero y alta disponibilidad, lo cual no será posible ya que se utilizó un único switch para la red LAN impidiendo utilizar protocolos como PRP (Parallel Redundancy Protocol) o

HSR (High-availability Seamless Redundancy). En este proyecto no se realizó énfasis en estos conceptos buscando un enfoque simplificado con el objetivo de verificar la comunicación entre dispositivos.

El IED 1 del campo de transformación, el IED 2 implementado en Raspberry Pi y el IED 3 del campo de alimentadores intercambian información mediante los protocolos MMS y GOOSE, y para ello se conectaron a un switch con cables UTP y terminales RJ45.

A esta red LAN también se conectó otra PC para realizar la recolección de datos y la supervisión del sistema utilizando los software Axon Exchange y Axon Builder para operar como Gateway recolectando la información bajo el protocolo IEC 61850 y enviarlo al SCADA por protocolo DNP3. Este cambio de protocolo se realizó pensando en una estación maestra fuera de la red LAN aunque por cuestiones técnicas se descartó esta y solo se mantuvo un SCADA local. El esquema de red utilizado se muestra en la Figura 37.

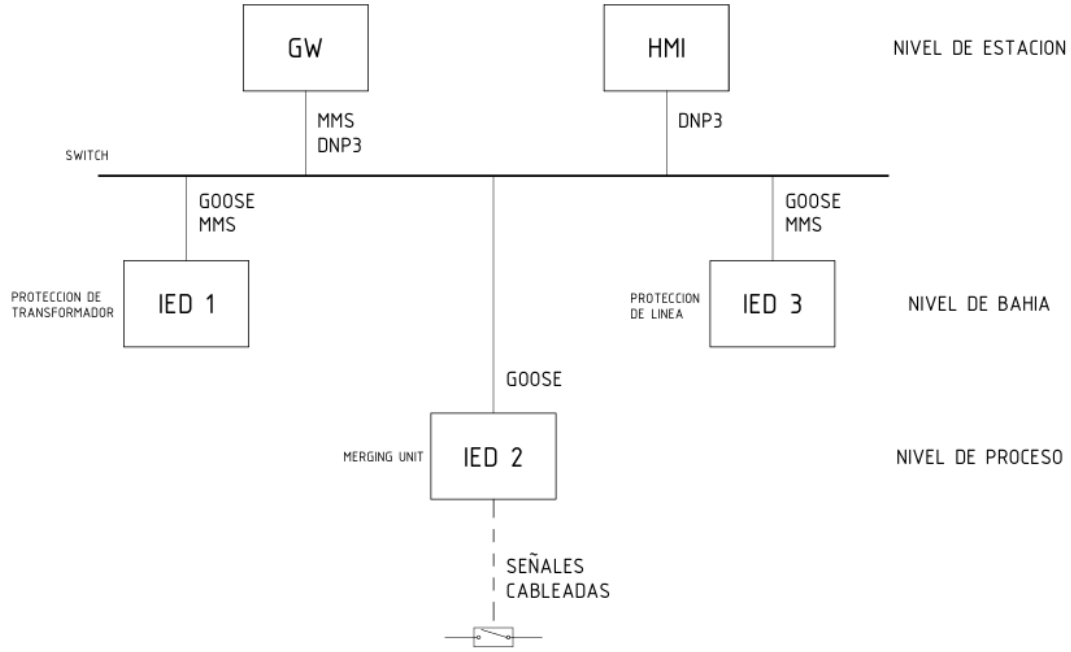


Figura 37: Arquitectura de red implementada en el proyecto.

A continuación en la Tabla 10 se especifican las direcciones utilizadas en cada equipo al momento de configurar la comunicación en la red.

Dispositivo / Rol	**Tipo de equipo**	**Dirección IP**	**Observaciones**
Merging Unit	Raspberry Pi	192.168.1.1	Publica GOOSE y MMS
Relé de transformador	PC	192.168.1.10	Publica GOOSE y MMS
IED virtual de protección	PC / Axon Bay	192.168.1.10	Publica GOOSE y MMS
Gateway IEC 61850 – DNP3	PC / Axon Exchange	192.168.1.20	Enlace con SCADA
SCADA local	PC / Axon Builder	192.168.1.20	Supervisión local

Tabla 10: Dispositivos y direcciones IP.

4.2. Diseño de tablero de comando

El tablero de comando es una representación simplificada de los equipos de playa de la estación transformadora del proyecto.

Como ya se mencionó previamente cuenta con cuatro interruptores, representados a partir de luces ojo de buey de color verde que se encontrarán apagadas cuando los interruptores se encuentren en el estado ABIERTO y se encenderán cuando el estado cambie a CERRADO. No se representan los estados típicos en los interruptores como son INDETERMINADO y EN FALLA. Las luces utilizadas tienen una tensión nominal de 220v.

Los interruptores asociados al transformador cuentan con la posibilidad de operar localmente, por lo que se instalaron pulsadores de color rojo y verde para los mandos de APERTURA y CIERRE respectivamente y una llave selectora para los comandos de LOCAL/REMOTO.

Figura 38: Elementos utilizados para la confección del tablero.

Por las cuestiones explicadas en las consideraciones del proyecto, los interruptores asociados a los alimentadores no cuentan con estos elementos.

Se instalaron dos pulsadores de color rojo para simular las señales de fallas internas en el transformador como si de un relé Buchholz se tratara, en donde el primer pulsador representa el nivel de alarma y la segunda el nivel de disparo. En los IED se programará la lógica de encender una luz de ALARMA DE FALLA ubicada en el tablero para el nivel de alarma, o encender la luz de ALARMA DE DISPARO Y BLOQUEO para el nivel de disparo, abrir los interruptores asociados al transformador y bloquear de forma permanente. El bloqueo permanente y la alarma pueden ser retirados con un pulsador verde ubicado en el tablero llamado RESET.

Los pulsadores y llaves tienen una tensión nominal de 220v, pero se encuentran conectados al circuito de señales de 3,3v que ingresan en la Raspberry pi ya que los enclavamientos se realizan de forma lógica y no eléctrica para este proyecto.

El tablero utilizado es de la marca Genrod de 450x300x150mm.

Figura 39: Gabinete metálico estanco Genrod 450x300x150 utilizado.

Dentro del tablero se ubicó la Raspberry Pi, junto con un módulo de expansión GPIO para facilitar el conexionado de las entradas y salidas digitales de la misma.

Figura 40: Módulo de expansión y Raspberry Pi 3B+ utilizados.

Se colocaron borneras sobre rieles DIN para distribuir los cables de forma segura y ordenada teniendo en cuenta que las señales GPIO se encuentran en 3.3Vcc y 5Vcc, mientras que los ojos de buey se accionan con 220Vac.

Para el control de las salidas digitales se utilizaron dos módulos de relé de 4 canales a 5v. Los mismos son alimentados externamente por una fuente de 5Vcc, aislando así la Raspberry Pi y controlando los pines asignados como salidas digitales dentro del programa diseñado.

Figura 41: Módulo de relé de 5v y 4 canales.

Dichas salidas trabajan con tensiones de 0 o 3.3v (0 o 1 binario), por lo que es necesario retirar el jumper que puentea los pines 1-2 del módulo, y conectar 3.3v en el pin 1(VCC) y 5v en el pin 2(JD-VCC). Los relés de este módulo se activan con una señal LOW y se apagan con una señal HIGH.

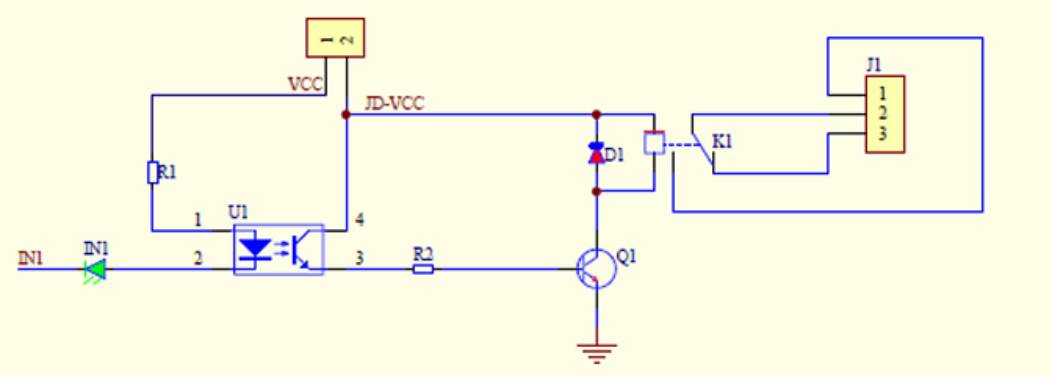


Figura 42: Circuito interno del módulo de relé.

Debido a que se contaba con 8 relés para las salidas digitales y solo 6 luces que necesitaban ser controladas, los 2 restantes relés se utilizaron para obtener el estado de los interruptores asociados al transformador e ingresarlos como señales de entrada de la Raspberry.

Para la protección del circuito de 220v se colocó una termomagnética de 16A, que luego pasa a dos tomacorrientes y a las borneras de distribución de 220Vac. Uno de los tomacorrientes se utiliza para conectar la fuente de la Raspberry Pi, mientras que el otro se utiliza para conectar la fuente de 5Vcc que alimenta los relés.

El primer diseño se realizó utilizando una protoboard con pulsadores, leds y resistencias como se puede ver en la Figura 43 para las pruebas de señales I/O con Raspberry pi.

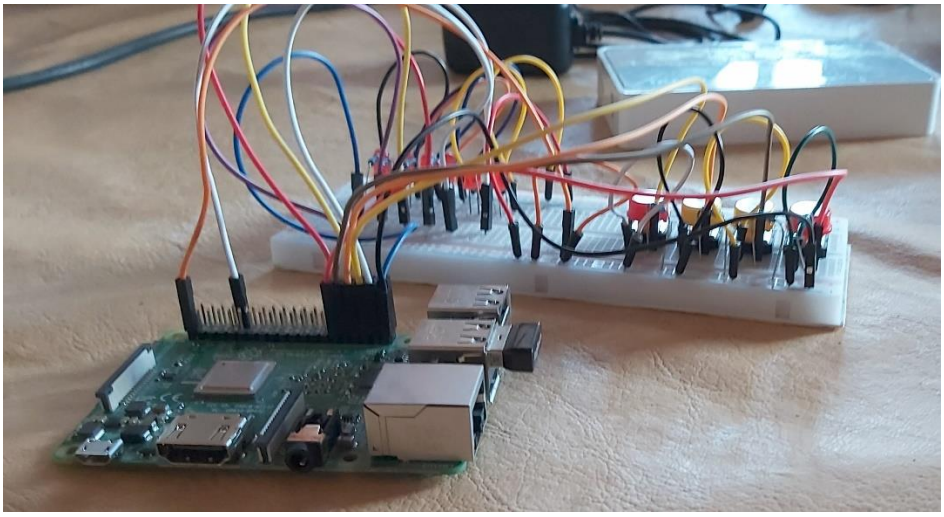


Figura 43: Primer prototipo del tablero de comando.

A continuación se muestran los esquemas de conexionado del circuito de entradas digitales en las Figura 44 y en la Figura 45 el esquema de las salidas digitales para el diseño del tablero.

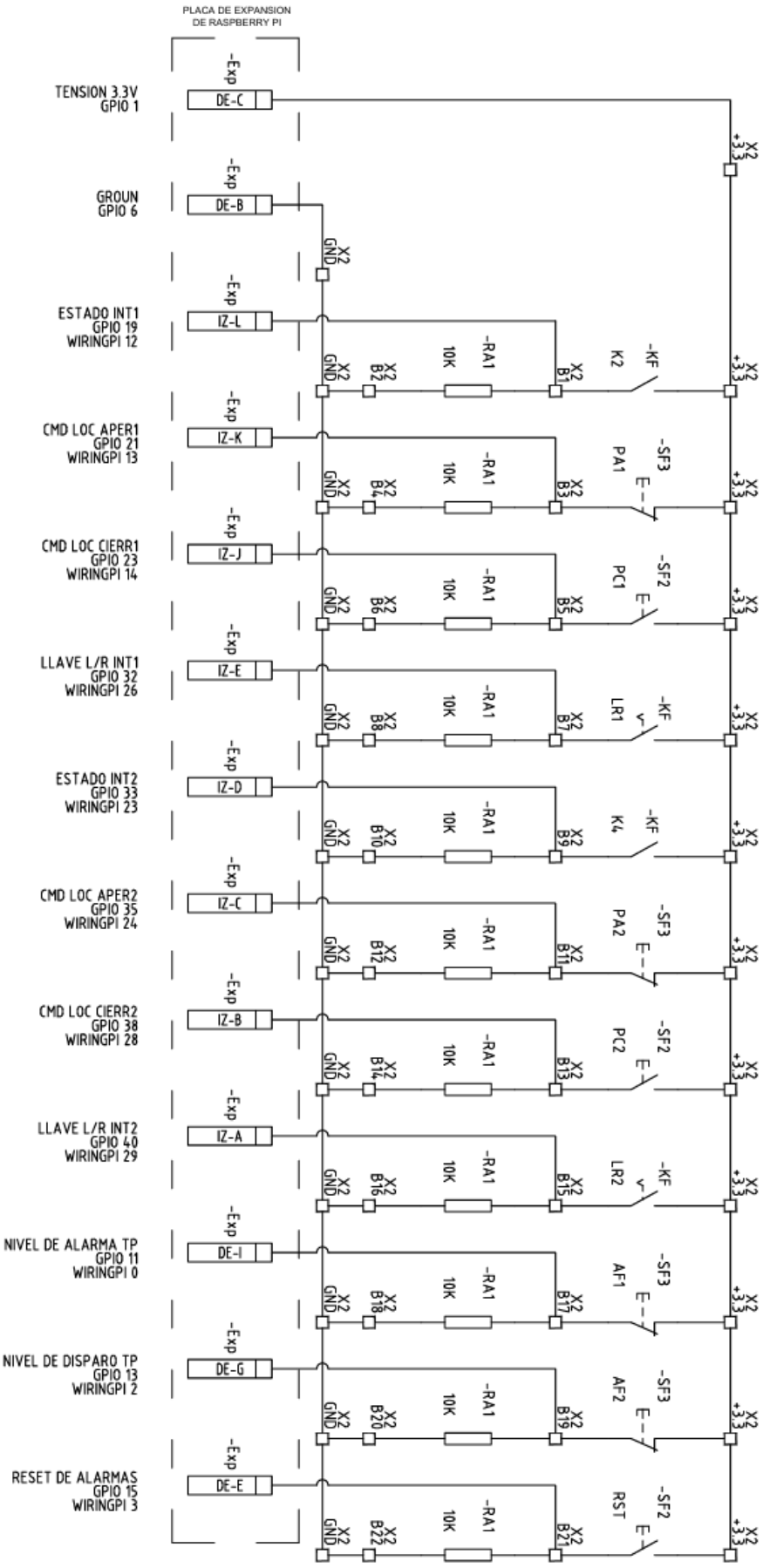


Figura 44: Esquema funcional de entradas digitales implementado en el tablero.

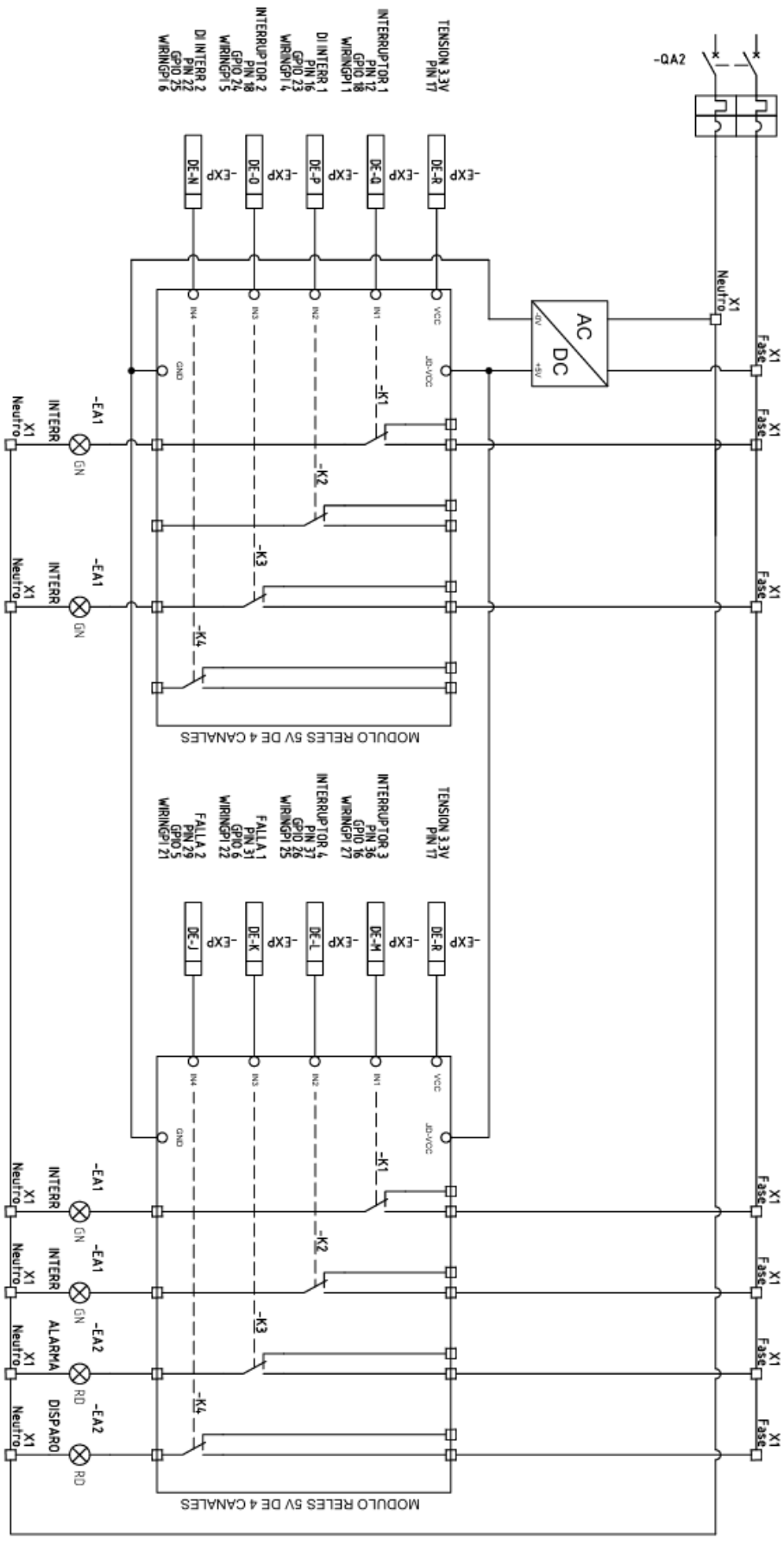


Figura 45: Esquema funcional de salidas digitales implementado en el tablero.

Previamente en la Figura 36 se mostró el esquema topográfico del frente del tablero indicando los distintos elementos que lo conforman. En la Figura 46 se muestra el esquema topográfico del interior del mismo, indicando la distribución de los elementos que lo conforman.

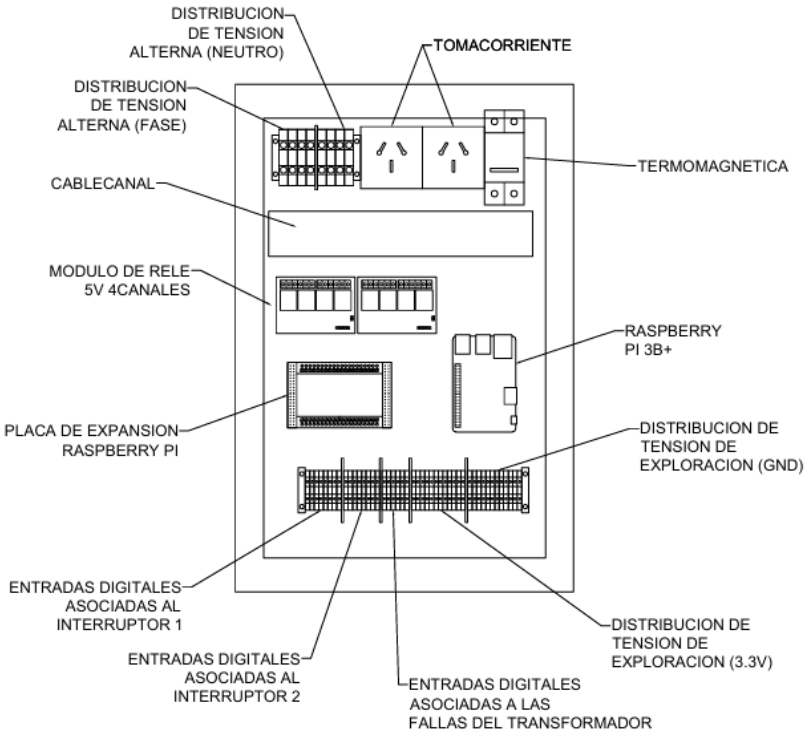


Figura 46: Esquema topográfico interior de tablero de comando.

A continuación se muestran algunas fotografías del tablero durante el proceso de armado.

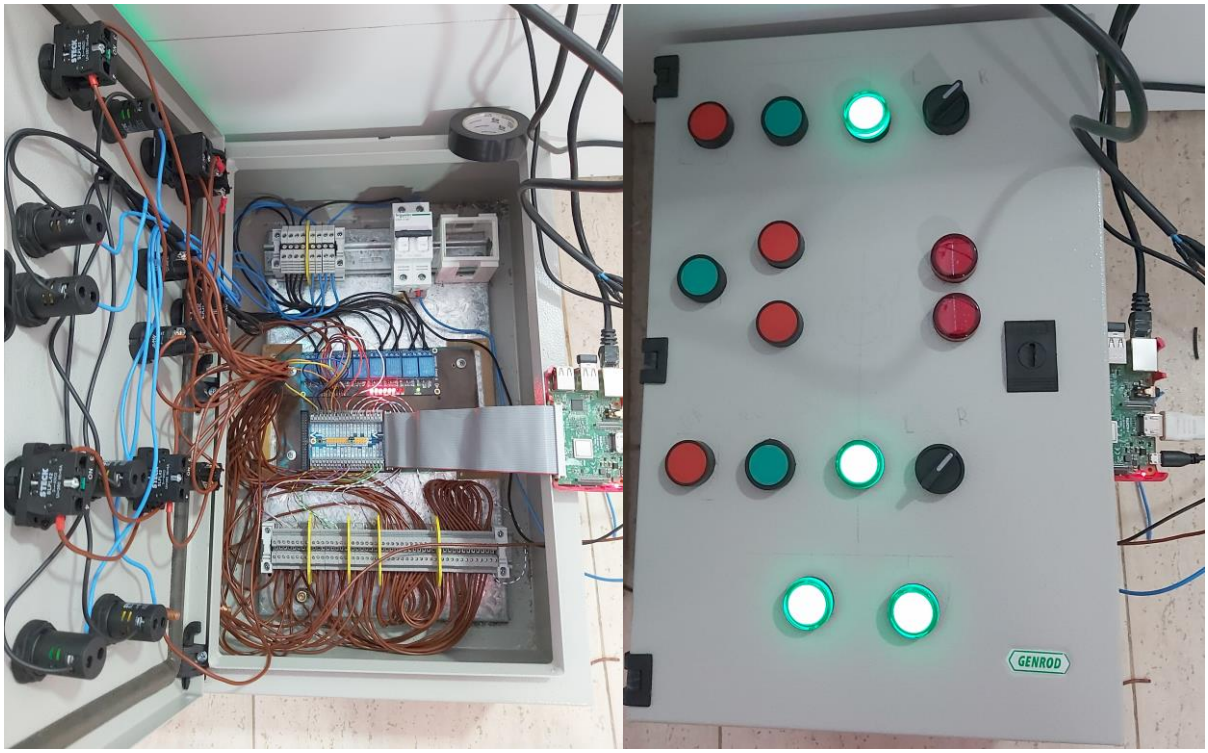


Figura 47: Topográfico frente de tablero de comando.

Por último se muestran en las siguientes figuras fotografías del tablero finalizado, conforme a los esquemas de conexión previamente presentados.

Figura 48: Tablero de comando finalizado.

4.3. Simulación de dispositivos

4.3.1. AXON AT61 - Simulator Bay

El AXON AT61 es una herramienta de software desarrollada por AXON Group, orientada a la simulación, prueba y validación de comunicaciones bajo el estándar IEC 61850, tanto en modo cliente como servidor. El conjunto de herramientas incluye un simulador interactivo de bahía (Simulator Bay), que permite representar de forma gráfica y funcional una bahía típica de subestación integrada bajo IEC 61850. [11]

La aplicación facilita la puesta en servicio y el diagnóstico de sistemas de automatización de subestaciones, permitiendo la exploración, supervisión y simulación del modelo de datos IEC 61850, así como la ejecución de comandos, reportes y mensajería GOOSE en un entorno controlado.

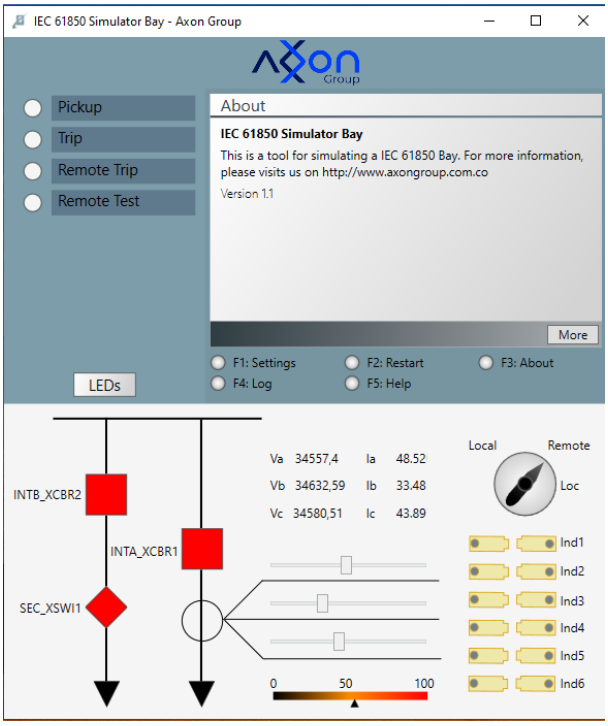


Figura 49: Software Simulator Bay de AXON.

El Simulator Bay implementa un modelo funcional de bahía, compuesto por interruptores de potencia, seccionadores, mediciones y puntos de mando, cuya interacción se refleja directamente en el modelo de datos IEC 61850 expuesto por el servidor simulado. La operación de los elementos puede realizarse de manera gráfica, permitiendo observar en tiempo real los cambios de estado y su propagación en los nodos lógicos correspondientes.

El modelo de datos se organiza conforme a la estructura jerárquica definida en IEC 61850-7, basada en Logical Devices (LD), Logical Nodes (LN), Data Objects (DO) y Data Attributes (DA).

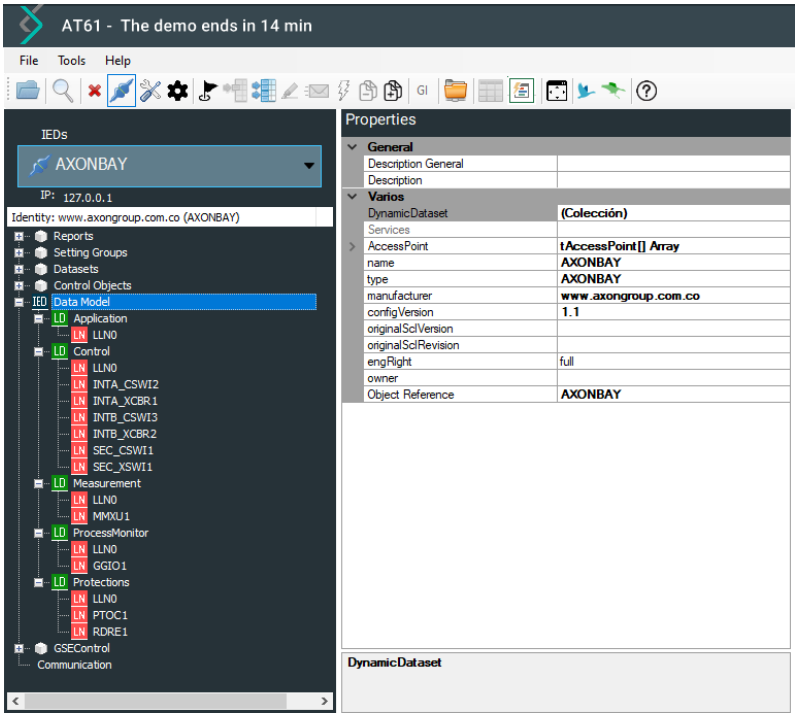


Figura 50: Modelo de datos obtenido con AXON AT61 client.

El AXON AT61 soporta los principales servicios cliente-servidor IEC 61850, incluyendo operaciones de lectura y escritura de atributos, configuración y habilitación de Report Control Blocks (Buffered y Unbuffered), creación de DataSets dinámicos, y ejecución de comandos de control bajo los modelos Direct Operate y Select Before Operate, con o sin seguridad mejorada. Los datos recibidos mediante lecturas, reportes o sondeos periódicos pueden ser supervisados a través del módulo Data Monitor, que incorpora registro de eventos y marcas temporales.

Para el desarrollo de este proyecto se utilizó Simulator Bay ejecutándose en una PC con la IP 192.168.1.20 y puerto 102, para el comando de los interruptores de los dos alimentadores haciendo uso de los dos interruptores que simula la aplicación.

Debido a que fue imposible modificar el modelo de datos que utiliza Simulator Bay se logró comunicación con la Raspberry mediante el protocolo MMS utilizando los reportes ya existentes y estableciendo una comunicación cliente-servidor. Esto permite replicar las señales existentes en el software Simulator Bay en la Raspberry Pi y así controlar los interruptores en el tablero de comando.

4.3.2. Biblioteca libiec61850

El proyecto **libiec61850** proporciona una biblioteca desarrollada en lenguaje C, destinada a la implementación de clientes y servidores compatibles con el estándar IEC 61850. La biblioteca es de código abierto, distribuida bajo licencia GPLv3, y se encuentra diseñada principalmente para la edición 2 del estándar IEC 61850, aunque mantiene compatibilidad con implementaciones ampliamente utilizadas en sistemas reales.

La biblioteca soporta los servicios definidos por la ACSI (Abstract Communication Service Interface), permitiendo el desarrollo de aplicaciones cliente-servidor que utilizan los protocolos MMS, GOOSE y Sampled Values para el intercambio de información entre dispositivos dentro de una misma red. Gracias a esta arquitectura, es posible implementar IEDs virtuales tanto en sistemas embebidos como en computadoras personales, bajo sistemas operativos Linux, Windows y macOS. [12]

libiec61850 se encuentra implementada conforme al estándar C99, lo que le otorga un alto grado de portabilidad. En este contexto, la portabilidad hace referencia a la capacidad de compilar y ejecutar el mismo código fuente en distintas plataformas de hardware y sistemas operativos, con mínimas o nulas modificaciones. Esto se logra gracias a la utilización de construcciones estándar del lenguaje C y a la abstracción de funcionalidades dependientes del sistema operativo, lo que maximiza la reutilización del código y facilita su despliegue en entornos heterogéneos.

La biblioteca permite generar modelos de datos IEC 61850 en código C a partir de archivos ICD o CID, previamente configurados con los nodos lógicos, DataSets y bloques de control correspondientes. Este proceso se realiza mediante una herramienta auxiliar desarrollada en Java, incluida dentro del paquete de la librería, que traduce la descripción SCL del IED a un modelo estático en C. De esta forma, es posible asignar a la aplicación desarrollada las características de un IED real, o bien utilizar modelos simplificados con una cantidad mínima de nodos lógicos, lo que brinda una elevada flexibilidad en el diseño de servidores y clientes. Adicionalmente, la librería permite la construcción dinámica del modelo de datos en tiempo de ejecución, ofreciendo una alternativa para aplicaciones que requieren mayor adaptabilidad.

Por otra parte, libiec61850 incorpora abstracciones de hardware y sistema operativo, tales como manejo de sockets de red, threads y temporización, encapsuladas dentro de la propia biblioteca. Estas abstracciones permiten desacoplar la lógica de la aplicación del sistema subyacente, simplificando el desarrollo y contribuyendo nuevamente a la portabilidad del software entre distintas plataformas.

Para utilizar la librería, el primer paso consiste en su descarga. La misma puede obtenerse desde el sitio oficial del proyecto https://libiec61850.com/ o desde su repositorio público en GitHub. En caso de contar con herramientas de control de versiones como Git, la descarga puede realizarse directamente mediante línea de comandos utilizando la instrucción:

```
git clone https://github.com/mz-automation/libiec61850.git
```

Una vez finalizada la descarga, es necesario proceder con la instalación de la librería. El procedimiento varía según el sistema operativo utilizado, siendo generalmente más sencillo en sistemas basados en **Linux** que en **Windows**, donde se requiere la instalación de herramientas y bibliotecas adicionales. En los apartados siguientes se describen los procedimientos de instalación y ejecución de ejemplos para ambos entornos.

4.3.2.1. Instalación en Linux

La instalación de la biblioteca libiec61850 se realiza de forma nativa sobre sistemas operativos Linux, aprovechando el entorno de compilación estándar basado en herramientas GNU. Este sistema operativo es el entorno de referencia del proyecto, por lo que el proceso de instalación resulta directo.

Una vez descargado el código fuente de la biblioteca desde su repositorio oficial, el proceso de construcción se realiza mediante CMake, herramienta que permite generar automáticamente los archivos de compilación en función del sistema anfitrión y de las opciones seleccionadas. Este mecanismo facilita la adaptación de la biblioteca a distintos entornos Linux y arquitecturas de hardware. En caso de no contar con las herramientas necesarias se debe ejecutar el comando:

```
sudo apt-get –y update
sudo apt-get install build-essential
```

Se procede a abrir una ventana de comandos desde la barra de tareas y se accede a la carpeta mediante líneas de comando utilizando el comando ‘cd’. Por ejemplo, si la librería fue descargada directamente sobre el escritorio se podrá acceder a la carpeta mediante el comando:

```
cd Desktop/libiec61850-1.5.1/
```

Una vez dentro de la raíz principal de la librería debemos ejecutar el archivo Makefile, donde se describen dentro de sus instrucciones una llamada install. Para ello se puede utilizar el comando:

```
sudo make install
```

A modo resumen rápido del lenguaje, la palabra clave *sudo* concede permisos de superusuario/administrador para realizar determinada acción o ejecución de comando. Por otra parte, el comando *make* es una herramienta de automatización que se utiliza para compilar y

construir programas a partir de un código fuente. Al ejecutar este comando se busca el archivo llamado *Makefile* en el directorio actual. Estos archivos pueden tener varias instrucciones y para acceder a cada una se debe utilizar la palabra clave con la cual se definió dentro del código por el creador de la librería como se puede ver en la imagen.

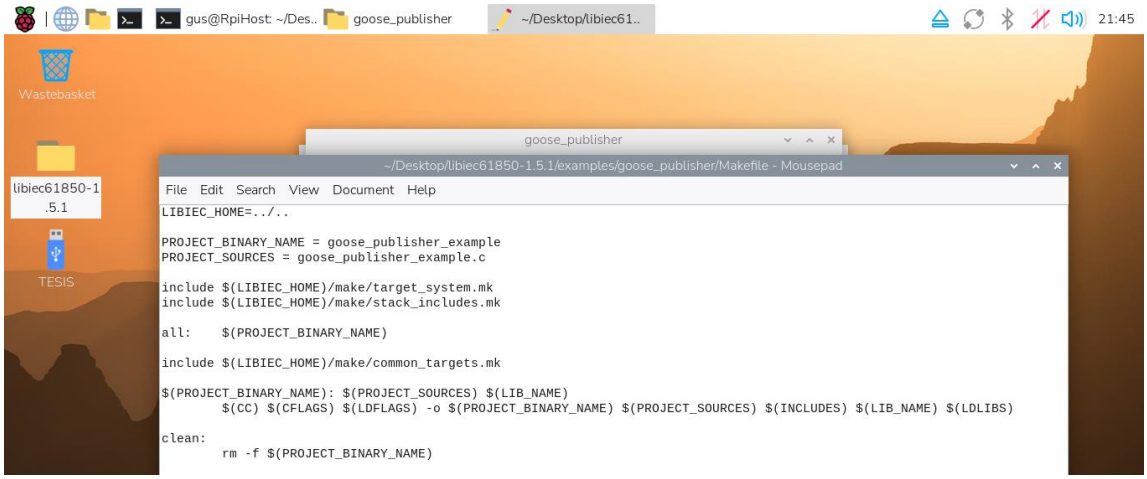


Figura 51: Archivo Makefile de un ejemplo.

De esta forma el archivo Makefile puede tener varias instrucciones para realizar distintos tipos de compilaciones según sea necesario. También cuenta con una instrucción que debe estar por defecto en todos los archivos Make denominada *all* y es el objetivo principal o por defecto del proyecto, ya que si no se ingresa una instrucción se realiza lo especificado en esta sección de código.

Luego de ejecutar el comando se instalan las librerías necesarias en el equipo y se podrá ejecutar cualquiera de los ejemplos de la carpeta *examples* sin problema.

Para ejecutar un servidor presente en la carpeta *examples* se debe acceder mediante líneas de comando desde la consola como se mostró anteriormente. Ubicado en la carpeta raíz de la librería se puede utilizar el comando *cd* para navegar por las diferentes carpetas. Por ejemplo para ingresar en la subcarpeta de *server_example_simple* dentro de la carpeta *example* se puede utilizar el comando:

```
cd examples/server_example_simple
```

Desde este punto se puede ejecutar el servidor ya compilado en un archivo ejecutable que debe estar en la carpeta. Si no se encuentra presente el archivo ejecutable se debe compilar haciendo uso del comando *make* con la siguiente instrucción:

```
sudo make
```

Luego, una vez terminada la compilación del archivo se ejecuta el mismo con el comando:

```
sudo ./server_example_simple
```

Este proceso es válido para ejecutar cualquier ejemplo de esta librería.

Con un software que se conecte como cliente se puede acceder al modelo de datos del servidor a partir de la dirección IP y el puerto configurado en el programa que por defecto es 102.

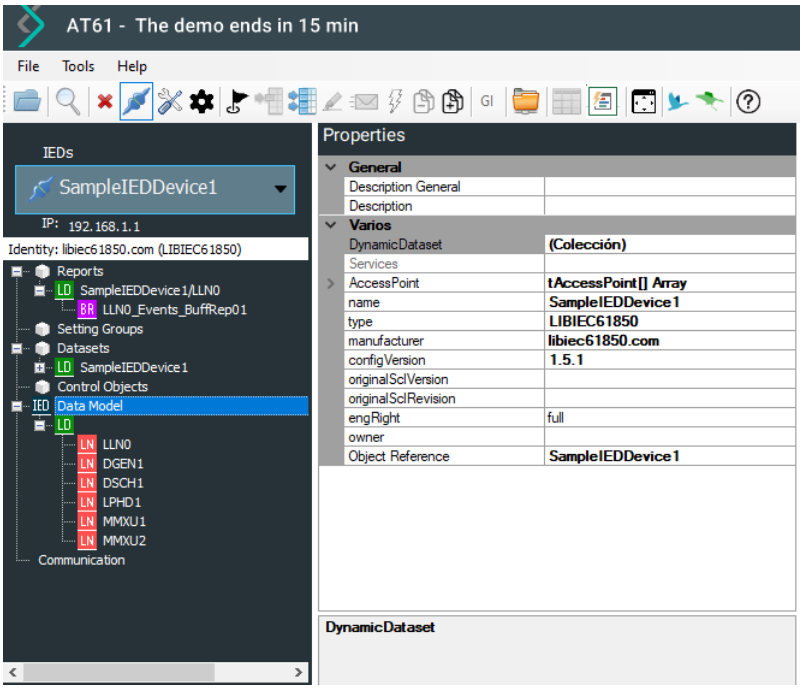


Figura 52: Cliente 61850 conectado al servidor en funcionamiento.

4.3.2.2. Instalación en Windows

A diferencia de los sistemas Linux, la instalación de libiec61850 en sistemas operativos Windows requiere una configuración previa más detallada del entorno de desarrollo, debido a la ausencia nativa de herramientas de compilación compatibles con el ecosistema GNU.

En este proyecto se utilizó Windows 10, por lo tanto esta explicación puede verse afectada para otras versiones. Para la instalación se recomienda seguir la siguiente secuencia de pasos:

1. **Obtención de una consola Bash**

Una consola Bash (Bourne-Again Shell) es una interfaz de línea de comandos que permite interactuar con el sistema operativo escribiendo comandos en texto. Es un tipo específico de "shell" o intérprete de comandos que es el nativo en sistemas tipo Unix/Linux y su sintaxis se basa en texto que permite ejecutar tareas complejas mediante comandos individuales. Para obtener una consola bash en Windows se recomienda utilizar alguna de las siguientes opciones:

Instalar MSYS2: es una plataforma de software para Windows que proporciona un entorno similar a Unix, con herramientas de desarrollo y un gestor de paquetes (*pacman*), permitiendo a los usuarios compilar y ejecutar fácilmente aplicaciones nativas de Windows, ofreciendo acceso a compiladores como GCC.

Instalar Visual Studio Code + Git bash: Visual Studio Code (VSCode) es un editor de código fuente ligero, gratuito y potente, ideal para desarrollo web y scripting, que se

extiende con extensiones para muchos lenguajes y funciones. Git Bash, por otro lado, es una aplicación para Windows que proporciona un entorno de línea de comandos (Bash) y las herramientas de Git, permitiendo a los usuarios ejecutar comandos Git como si estuvieran en Linux o macOS

2. Preparación del entorno de desarrollo

Es necesario instalar un entorno de desarrollo compatible con el estándar C99. Se recomienda instalar el toolchain MinGW-w64 es una implementación de los compiladores GCC para la plataforma Win32 que permite migrar la capacidad de este compilador en entornos Windows.

Para ello se puede utilizar el gestor de paquetes *pacman* de **MSYS2** utilizando el siguiente comando en la consola bash:

```
pacman -Syu mingw-w64-i686-toolchain
```

Otra opción es descargarlo directamente de la Web como un archivo ejecutable e instalarlo en el equipo.

Una vez descargado e instalado se configura las variables de entorno del sistema, añadiendo en variables del sistema, en Path, la ruta correspondiente al directorio **mingw32/bin**, permitiendo así la ejecución de las herramientas de compilación desde la línea de comandos.

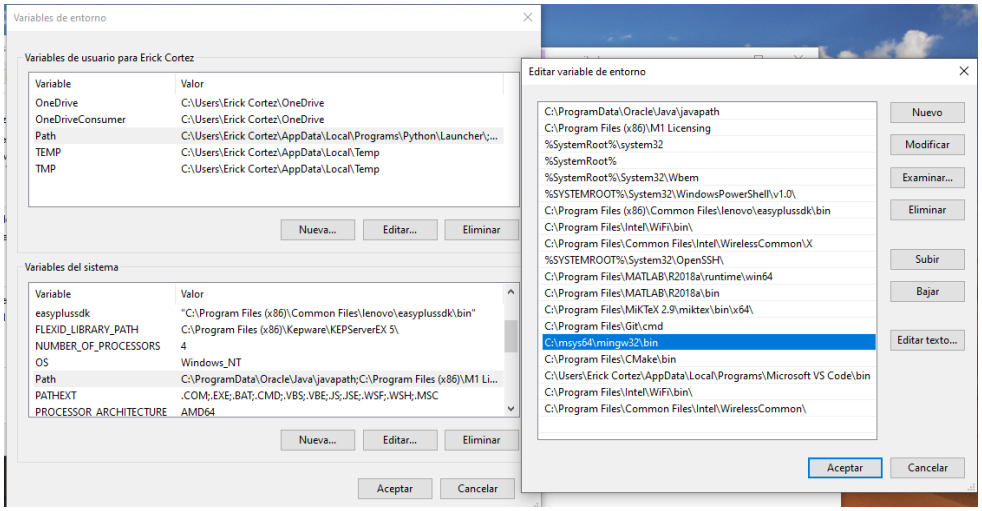


Figura 53: Editor de variables de entorno.

Una vez instalado el compilador, se puede verificar que el sistema lo detecta correctamente a partir del comando:

```
gcc --version
```

3. Implementar Cmake como Sistema de Generación de Proyectos

La librería libiec61850 utiliza Cmake para gestionar la compilación multiplataforma. Por lo tanto es necesario descargar e instalar los binarios desde la página oficial https://cmake.org/download/. Durante la instalación se debe seleccionar la opción “Add Cmake

to the system PATH for all users". Se puede verificar su correcta instalación escribiendo en la consola el comando:

```
cmake --version
```

CMake será el encargado de leer los archivos CMakeLists.txt del directorio donde se ejecute el comando cmake de ahora en más.

El comando Make y CMake no son iguales. El primero compila a partir de archivos Makefile escritos manualmente, mientras que el segundo se direcciona los archivos CMakeLists.txt y genera automáticamente los archivos Makefile para diferentes plataformas. Para el uso del comando make se duplico de la carpeta mingw32\bin el archivo mingw32-make y se renombro make. Esto permite utilizar comandos similares en ambos entornos de desarrollo como:

```
make model
```

4. Soporte para GOOSE y Sample Values

Los protocolos GOOSE y Sampled Values (SV) operan directamente sobre la capa de enlace de datos, omitiendo la pila TCP/IP. Windows requiere un controlador de captura de paquetes para esta función. Para habilitar el soporte de mensajería GOOSE en sistemas Windows, fue necesario instalar la biblioteca WinPcap, utilizada para la captura y envío de tramas Ethernet a bajo nivel. Las instrucciones correspondientes se encuentran documentadas en la carpeta third_party del proyecto libiec61850, así como en la documentación oficial de la biblioteca.

Se indica que debe descargarse los encabezados y librerías de WinPcap de la página https://www.winpcap.org/install/bin/WpdPack_4_1_2.zip y estos deben ser copiados en el directorio third_party\winpcap. Luego es necesario ejecutar cmake desde el directorio principal para que la configuración se complete.

También es necesario instalar WinPcap o Npcap en el equipo, aunque este último se instala automáticamente con aplicaciones como Wireshark.

5. Obtención del código fuente

El código fuente de la biblioteca fue obtenido desde el repositorio oficial del proyecto, utilizando un sistema de control de versiones Git mediante el comando:

```
git clone https://github.com/mz-automation/libiec61850.git
```

Este enfoque permitió asegurar la trazabilidad de la versión empleada durante el desarrollo de la tesis y facilitó la replicación del proceso en distintos entornos.

Alternativamente, la biblioteca puede descargarse directamente desde el repositorio público en formato comprimido.

6. Instalación de la librería

Una vez realizados todos los pasos anteriores, se cumplen las condiciones para iniciar el proceso de instalación de la librería en Windows. Se ingresa al directorio principal de la librería con la consola bash y se crea una nueva carpeta llamada build con el comando:

```
mkdir build
```

Se accede a la carpeta build y se realiza el llamado del archivo CMakeLists.txt del directorio principal con los comandos:

```
cd build
cmake -G “MinGW Makefiles” ..
```

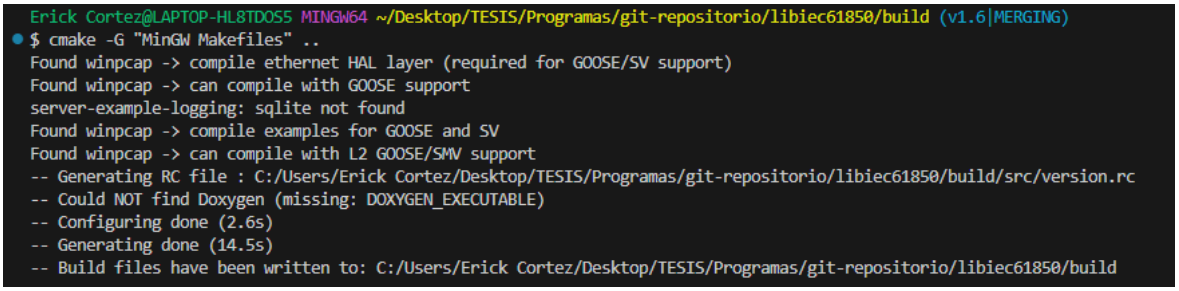


Figura 54: Consola en Visual Studio Code con ejecución de comando.

Esto genera todos los archivos y carpetas necesarios dentro de la carpeta build. También genera archivos Makefile adaptados para Windows que permiten instalar y compilar los proyectos de la carpeta examples.

```
make install
```

Nota: Se debe modificar el archivo socket_win32 ubicado en la carpeta hal/socket/win32. Aquí después de la instalación se crea este archivo que tiene por defecto para la variable DEBUG_SOCKET en WIN32 el valor de 1. Esto genera una gran cantidad de mensajes durante la ejecución del servidor, y si bien funciona correctamente puede ser molesto, por lo que se recomienda modificar su valor a 0 para utilizar la librería en Windows.

La ejecución de un ejemplo se realiza de forma similar a lo visto previamente, pero direccionado a uno de los archivos ubicado en la carpeta build/examples. Por ejemplo para ejecutar el server_example_simple se deben aplicar los siguientes comandos desde la raíz.

```
cd build/examples/server_example_simple
```

En caso de no encontrarse el archivo ejecutable se compila el mismo utilizando el comando:

```
make
./server_example_simple.exe
```

La ejecución del servidor también puede realizarse buscando con el explorador de Windows el archivo ejecutable .exe e iniciándolo directamente.

4.3.2.3. Funciones de la librería

La librería libiec61850 cuenta con una extensa cantidad de funciones que facilitan la implementación de los servidores y la interacción con el modelo de datos.
A continuación se muestran algunas de las funciones utilizadas en este proyecto con una breve explicación de sus parámetros de entrada, salida y utilización:

`void IedServer_start(IedServer self, int tcpPort)`

Esta función iniciar la gestión de conexiones de cliente. Esta funcion no devuelve ningún dato (*void*) y utiliza como parámetros de entrada a *self*, una instancia de IedServer en la que se operará y *tcpPort*, el puerto TCP en el que escucha el servidor.

`void GooseSubscriber_setListener(GooseSubscriber self, GooseListener listener, void *parameter)`

Establece una función de devolución de llamada que se invocará al recibir un mensaje GOOSE.

self: instancia de GooseSubscriber sobre la que se operará.

lister: función de devolución de llamada proporcionada por el usuario.

parameter: un parámetro proporcionado por el usuario que se pasará a la función de devolución de llamada.

`void IedServer_setControlHandler(IedServer self, DataObject *node, ControlHandler handler, void *parameter)`

Esta función asigna un controlador proporcionado por el usuario para un objeto de datos de control. El objeto de datos debe ser una instancia CDC (Clase de Datos Común) controlable, como por ejemplo, SPC, DPC o APC. El controlador es una función *callback* que el servidor IEC llamará cuando un cliente realice una operación de control sobre el objeto de datos.

self: la instancia de IedServer sobre la que se realizará la operación.

node: el controlador del objeto de datos controlable.

handler: una función de devolución de llamada de tipo ControlHandler.

parameter: un parámetro proporcionado por el usuario que se pasa al controlador de control.

A continuación se hará mención de otras funciones muy útiles para el desarrollo de este proyecto o uno nuevo, únicamente dando una breve descripción ya que los detalles se pueden consultar directamente en el repositorio de la librería o con ayuda de un editor de código.

void IedServer_setGooseInterfaceId(IedServer self,const char *interfaceId)

Esta función permite configurar el ID de la interfaz GOOSE. Si no se utiliza o se establece en NULL, se utiliza el ID de interfaz predeterminado de stack_config.h.

GooseReceiver GooseReceiver_create(void)

Crear una nueva instancia de receptor. Una instancia de GooseReceiver se utiliza para gestionar todos los mensajes GOOSE recibidos en una interfaz de red específica.

void GooseReceiver_addSubscriber(GooseReceiver self, GooseSubscriber subscriber)

Añadir un suscriptor a esta instancia del receptor. NOTA: No invoque esta función mientras el receptor esté en ejecución (después de llamar a GooseReceiver_start).

uint32_t IedServer_getBitStringAttributeValue(IedServer self, const DataAttribute *dataAttribute)

Obtener el valor de un atributo de datos de tipo MMS_BIT_STRING. Si el atributo de datos es de un tipo diferente, el valor devuelto no está definido. NOTA: El entero devuelto se determina calculando según la siguiente fórmula: bit0^0 + bit1^1 + ... + bitn^n. Las especificaciones IEC 61850 no especifican cómo se puede interpretar una cadena de bits como entero. Por lo tanto, esta función debe utilizarse con precaución.

void IedServer_updateBitStringAttributeValue(IedServer self, DataAttribute *dataAttribute, uint32_t value)

Actualizar el valor de un atributo de datos de cadena de bits IEC 61850. Actualizar el valor de un atributo de datos de cadena de bits sin procesarlo con instancias de MmsValue. Esta función también comprobará si se cumple una condición de activación cuando se habilita un informe o un bloque de control GOOSE.

bool IedServer_getBooleanAttributeValue(IedServer self, const DataAttribute *dataAttribute)

Obtener el valor de un atributo de datos de tipo MMS_BOOLEAN. Si el atributo de datos es de un tipo diferente, el valor devuelto no está definido.

void IedServer_updateBooleanAttributeValue(IedServer self, DataAttribute *dataAttribute, bool value)

Actualizar el valor de un atributo de datos booleano IEC 61850. Actualizar el valor de un atributo de datos booleano sin procesarlo con instancias de MmsValue. Esta función también comprobará si se cumple una condición de activación cuando se habilita un informe o un bloque de control GOOSE.

void IedServer_updateUTCTimeAttributeValue(IedServer self, DataAttribute *dataAttribute, uint64_t value)

Actualiza el valor de un atributo de datos de hora UTC sin procesar instancias de MmsValue. Esta función también comprueba si se cumple una condición de activación cuando se habilita un informe o un bloque de control GOOSE.

bool MmsValue_getBoolean(const MmsValue *value)

Obtiene el valor booleano de un objeto MmsValue.

void Thread_sleep(int millies)

Suspender la ejecución del hilo durante la cantidad de milisegundos especificada.

Existen muchas funciones más que se utilizan tanto en los códigos diseñados así como en los ejemplos que provee la librería. Se considera que estas son las más destacables ya que permiten la creación del servidor, identificar en que adaptador de red estará trabajando el servidor, gestionar las suscripciones e implementar lógica dentro del modelo de datos. Aun así es necesario complementar estas funciones con el lenguaje básico de C y el manejo de tipo de datos al momento de diseñar un nuevo servidor con las características deseadas.

4.3.2.4. Convertir archivos ICD o CID

La librería cuenta con una aplicación diseñada en Java que permite convertir archivos ICD o CID del lenguaje SCL a archivos en código C. Estos archivos tienen todas las características y configuraciones del archivo en lenguaje SCL y define un modelo estático con todos los Nodos Lógicos, DataSet y configuraciones GOOSE.

Para utilizar esta aplicación es necesario instalar Java si no se encuentra instalado por defecto. Para ello puede utilizarse el siguiente comando.

```
sudo apt-get update
sudo apt-install default jdk
```

Una vez instalado se puede utilizar la herramienta Model Generator de dos formas muy distintas. La primera es colocar el archivo ICD o CID en la carpeta tools/model_generator de la librería y mediante comandos se ejecuta la siguiente instrucción:

```
sudo java -jar genmodel.jar [nombre ICD] [ap – nombre de punto de acceso] [nombre archivo de salida(static_model)] [prefijo del modelo]
```

Una vez realizado esto se generarán los archivos static_model.c y static_model.h dentro de la carpeta. Otra alternativa es utilizar los archivos Makefile de los ejemplos, que son los mismos que se recomienda utilizar para diseñar los proyectos propios. Colocando el archivo ICD o CID en la carpeta donde se está realizando el proyecto nuevo, y modificando en el archivo Makefile en PROJECT_ICD_FILE con el nombre del nuevo archivo para que coincida.

Una vez realizado esto se accede por línea de comando a la carpeta del proyecto y se ejecuta el comando:

```
sudo make model
```

De esta forma se generan los archivos static_model.c y static_model.h dentro de la misma carpeta del proyecto.

Ya sea por el primer o segundo método, los archivos static_model.c y static_model.h deben estar en la carpeta del proyecto al momento de compilar. Para utilizarlo dentro de los códigos es necesario incluir el archivo de encabezado con la instrucción:

```
#include “static_model.h”
```

4.3.3. Codificación de programas

4.3.3.1. En Linux con Raspberry Pi

Para el diseño de un nuevo servidor, con características personalizadas se puede partir de la copia de un ejemplo existente, evitando así crear los archivos de compilación desde cero. Se recomienda que este tenga los archivos de Makefile, static_model y el archivo en lenguaje C. Una vez copiado el ejemplo se renombra la carpeta con el nombre del nuevo proyecto, al igual que el archivo C donde luego se escribirá el código.

Figura 55: Archivos de ejemplo de la librería libiec61850

Recordar que debido a estas modificaciones también se debe actualizar el código en el archivo Makefile, a fin de redireccionar la compilación hacia el nuevo proyecto donde escribiremos el código del servidor. También es necesario agregar elementos de compilación en caso de ser necesario, como en el caso de utilizar la librería wiringpi para que el servidor interactúe con los pines GPIO de la Raspberry Pi.

A partir de este punto se puede modificar el código existente añadiendo o quitando líneas con el fin de darle al servidor las características deseadas utilizando las funciones de la librería o añadiendo lógica sobre el funcionamiento del mismo.

4.3.3.2. Biblioteca WiringPi

Se pueden diseñar códigos, dedicados a la Raspberry Pi, que utilicen los pines de la misma para interactuar con el servidor como entradas o salidas digitales.

BCM	WiringPi	Name	Physical		Name	WiringPi	BCM
		3.3v	1	2	5v		
2	8	SDA.1	3	4	5V		
3	9	SCL.1	5	6	0v		
4	7	1-Wire	7	8	TxD	15	14
		0v	9	10	RxD	16	15
17	0	GPIO. 0	11	12	GPIO. 1	1	18
27	2	GPIO. 2	13	14	0v		
22	3	GPIO. 3	15	16	GPIO. 4	4	23
		3.3v	17	18	GPIO. 5	5	24
10	12	MOSI	19	20	0v		
9	13	MISO	21	22	GPIO. 6	6	25
11	14	SCLK	23	24	CE0	10	8
		0v	25	26	CE1	11	7
0	30	SDA.0	27	28	SCL.0	31	1
5	21	GPIO.21	29	30	0v		
6	22	GPIO.22	31	32	GPIO.26	26	12
13	23	GPIO.23	33	34	0v		
19	24	GPIO.24	35	36	GPIO.27	27	16
26	25	GPIO.25	37	38	GPIO.28	28	20
		0v	39	40	GPIO.29	29	21
BCM	WiringPi	Name	Physical		Name	WiringPi	BCM

Figura 56: Numeración de pines GPIO y funcionalidad en librería WiringPi.

Para ello es necesario utilizar una librería específica en el código que permita asignar las variables definidas en el servidor con las entradas/salidas digitales asignadas a cada pin de la placa. En el desarrollo de este proyecto se utiliza la librería WiringPi que es una biblioteca de acceso GPIO de alto rendimiento escrita en código C para el control rápido y eficiente de los pines GPIO. Esta librería viene instalada por defecto y le asigna un número a cada uno de los pines como se ve en la Figura 56, permitiendo utilizar comandos específicos para cada pin por separado.

Para comprobar que la misma se encuentra instalada se puede consultar la versión de la misma utilizando la línea de comando:

```
gpio -v
```

En caso de no encontrarse instalada se puede utilizar desde el repositorio git el comando:

```
git clone https://github.com/WiringPi/WiringPi.git
cd WiringPi
./build
```

Esta librería tiene comandos específicos que pueden utilizarse dentro del código del servidor. Algunas de estas funciones son:

`wiringPiSetup(void)`

Esto inicializa la librería y asume la configuración por defecto mostrada en la imagen anterior.

`pinMode(int pin , int mode)`

Configura el modo de funcionamiento del pin, ya sea INPUT, OUTPUT, PWM_OUTPUT (únicamente para el pin 1) o GPIO_CLOCK (solo en el pin 18).

`digitalWrite(int pin, int value)`

Escribe el valor HIGH (1) o LOW (0) al pin indicado. El mismo debe ser declarado en modo OUTPUT previamente.

`pwmWrite(int pin, int value)`

Escribe el valor al registro PWM del pin indicado. Solo se puede aplicar al pin 1 de la Raspberry Pi y previamente debe ser declarado con el modo PWM_OUTPUT.

`digitalRead(int pin)`

Devuelve el valor leído en el pin indicado, con los valores HIGH (1) o LOW (0).

Como ya se mencionó, para compilar correctamente el proyecto se debe modificar el archivo Makefile de compilación para agregar la librería como un recurso. Esto se logra modificando el archivo y agregando las modificaciones que se muestran en la figura:

```
1  EXTRA_LIBS=-lwiringPi -lm
2  LIBIEC_HOME=../..
3
4  PROJECT_BINARY_NAME = Server
5  PROJECT_SOURCES = Server_example.c
6  PROJECT_SOURCES += static_model.c
7
8  PROJECT_ICD_FILE = raspberry2.CID
9
10 include $(LIBIEC_HOME)/make/target_system.mk
11 include $(LIBIEC_HOME)/make/stack_includes.mk
12
13 all:    $(PROJECT_BINARY_NAME)
14
15 include $(LIBIEC_HOME)/make/common_targets.mk
16
17 model:  $(PROJECT_ICD_FILE)
18     java -jar $(LIBIEC_HOME)/tools/model_generator/genmodel.jar $(PROJECT_ICD_FILE)
19
20 $(PROJECT_BINARY_NAME): $(PROJECT_SOURCES) $(LIB_NAME)
21     $(CC) $(CFLAGS) $(LDFLAGS) -o $(PROJECT_BINARY_NAME) $(PROJECT_SOURCES) $(INCLUDES) $(LIB_NAME) $(LDLIBS) $(EXTRA_LIBS)
22
23 clean:
24     rm -f $(PROJECT_BINARY_NAME)
25
```

Figura 57: Archivo Makefile con modificaciones para WiringPi.

Se puede observar que se agregó la línea 1 y se modificó la línea 21.

4.3.3.3. En Windows con PC

Para crear un nuevo proyecto en Windows se puede copiar un ejemplo de la carpeta *examples* de la raíz librería (no desde la carpeta *build*). Se modifica el nombre de la carpeta del proyecto nuevo y del código fuente en C, de igual forma que se explicó para Linux. En esta carpeta se deben agregar los archivos static_model, aunque para facilitar el proceso se recomienda ingresar el archivo ICD o CID.

examples > IED1 > Makefile

```
LIBIEC_HOME=../..

PROJECT_BINARY_NAME = IED1
PROJECT_SOURCES = IED_1.c
PROJECT_SOURCES += static_model.c

PROJECT_ICD_FILE = IED1.cid

include $(LIBIEC_HOME)/make/target_system.mk
include $(LIBIEC_HOME)/make/stack_includes.mk

all:    $(PROJECT_BINARY_NAME)

include $(LIBIEC_HOME)/make/common_targets.mk

model:  $(PROJECT_ICD_FILE)
    java -jar $(LIBIEC_HOME)/tools/model_generator/genmodel.jar $(PROJECT_ICD_FILE)

$(PROJECT_BINARY_NAME): $(PROJECT_SOURCES) $(LIB_NAME)
    $(CC) $(CFLAGS) $(LDFLAGS) -o $(PROJECT_BINARY_NAME) $(PROJECT_SOURCES) $(INCLUDES) $(LIB_NAME) $(LDLIBS)

clean:
    rm -f $(PROJECT_BINARY_NAME)

```

Figura 58: Archivo Makefile de un ejemplo.

Modificar el archivo CMakeLists.txt de la carpeta del proyecto nuevo, para asociar el nombre del código fuente a la compilación del programa.

examples > IED1 > CMakeLists.txt

```
include_directories(
    .
)

set(IED1_SRCS
    IED_1.c
    static_model.c
)

IF(MSVC)
set_source_files_properties(${IED1_SRCS}
                            PROPERTIES LANGUAGE CXX)
ENDIF(MSVC)

add_executable(IED1
  ${IED1_SRCS}
)

target_link_libraries(IED1
    iec61850
)
```

Figura 59: Archivo CMakeLists.txt de un ejemplo.

Modificar el archivo CMakeLists.txt de la carpeta examples para añadir la carpeta creada agregando una línea al código:

```
add_subdirectory("Nombre de la carpeta del proyecto")
```

Esto generará una carpeta nueva con el nombre del proyecto nuevo en la carpeta build con todos los archivos necesarios para realizar la compilación del programa. Se puede agregar varios si se desea hacer diferentes proyectos.

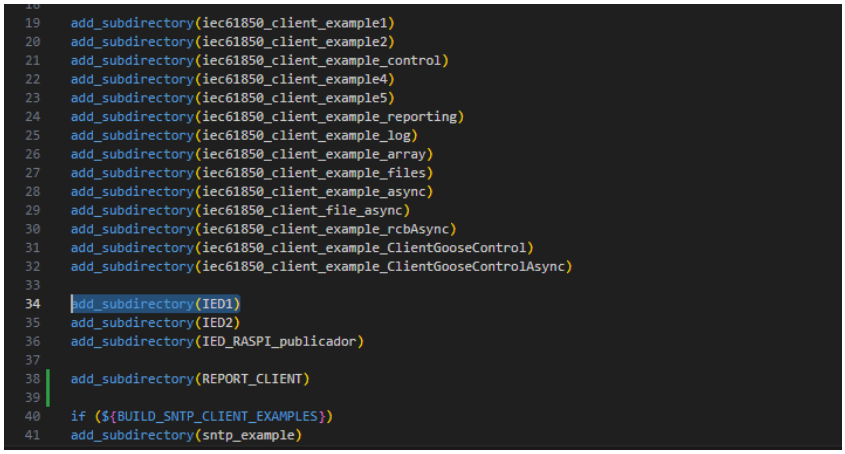


Figura 60: Archivo CMakeLists.txt de la carpeta examples.

Para que se ejecute esta modificación es necesario dirigirse a la carpeta build y realizar el llamado del archivo CMakeLists con la instrucción make. De esta forma se generarán nuevamente todas las carpetas con los ejemplos y sus archivos compilados, y donde aparecerá la nueva carpeta con el nuevo proyecto ya compilado. Las instrucciones son las siguientes:

```
cd ..
cd build
make
```

A partir de aquí se pueden realizar todas las modificaciones necesarias en el código fuente de la carpeta *example/"nuevo proyecto"*. La modificación de los archivos static_model debe hacerse en esta carpeta. Se recomienda direccionar la consola a esta carpeta y con el archivo ICD o CID en la misma ejecutar el comando

```
make model
```

Una vez finalizadas las modificaciones la compilación se realiza desde la dirección *build/examples/"nuevo proyecto"* con la instrucción

```
make
```

En esta carpeta aparecerá un archivo ejecutable al cual se puede acceder por consola o desde el navegador de carpetas.

4.3.4. Modelado de la Merging Unit

Como se mencionó previamente se diseñaron dos aplicaciones para modelar los IED utilizando la biblioteca libiec61850. El primero que se ejecutará en la raspberry pi basada en el sistema

Linux y que modela una Merging Unit para vincular las señales emitidas por los IED virtuales con el tablero eléctrico que representa los elementos de la estación.

4.3.4.1. Señales del dispositivo

A continuación se muestra un esquema donde se detallan las señales digitales de entrada y las señales de salida obtenida a partir de los pines GPIO desde el tablero de comando, y las señales intercambiadas por comunicación mediante los protocolos IEC 61850.

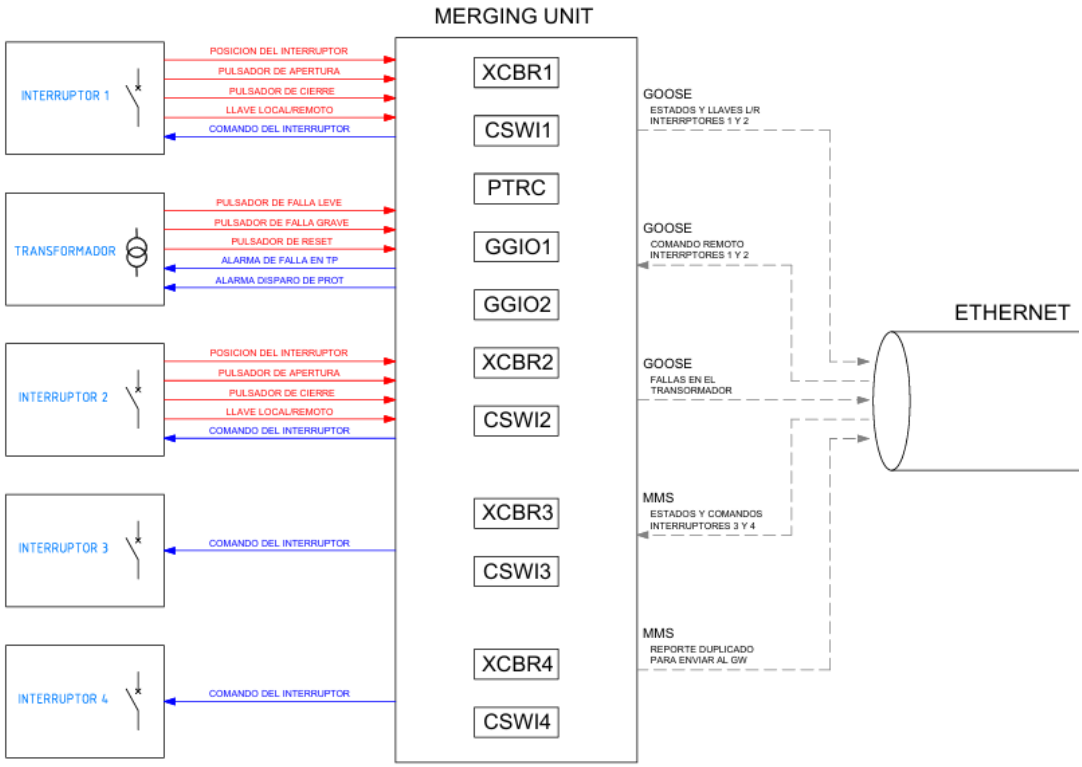


Figura 61: Esquema de señales que interactúan con la MU.

A partir de estas señales y el comportamiento que debe tener el dispositivo para comportarse como una Merging Unit se estimó la cantidad mínima de Nodos Lógicos que debería tener el IED y se determinaron los siguientes objetos:

- LLN0: Necesario para definir los DataSet, Report Control Block y GoCB.
- XCBR: Cuatro instancias para establecer el estado de cada interruptor.
- CSWI: Cuatro instancias para emitir los comandos a cada interruptor.
- PTRC: Necesario para el disparo producido desde la protección.
- GGIO: Necesario para obtener las señales provenientes de las protecciones internas del transformador.

Conociendo las señales que debe emitir y recibir el IED, y los Nodos Lógicos que conforman el modelo de datos se pueden definir los DataSet necesarios. Se emiten tres mensajes GOOSE,

el primero con el estado de los interruptores 1 y 2, así como la posición de sus llaves Local/Remoto, el segundo con las señales de falla del transformador y un tercero para indicar que se emite un comando hacia cualquiera de los cuatro interruptores.

Debido a la limitación de relé implementado con el software Simulator Bay se utilizó la Raspberry Pi para replicar los estados de los interruptores y poder emitirlos al tablero. Para ello es necesario establecer la comunicación Cliente-Servidor y así recibir un Reporte MMS configurado en el software por el fabricante. El problema en esto recae al momento de conectar el Gateway a la red, ya que este no puede obtener la información correspondiente al mismo reporte debido a que dicho canal ya se encuentra utilizado por la comunicación entre el software y la raspberry. Para solucionarlo se crea un Reporte duplicando la información obtenida para enviar al Gateway con los estados del interruptor 3, 4 y la posición de la llave Local/Remoto de la protección configurada como se puede ver en la Figura 62.

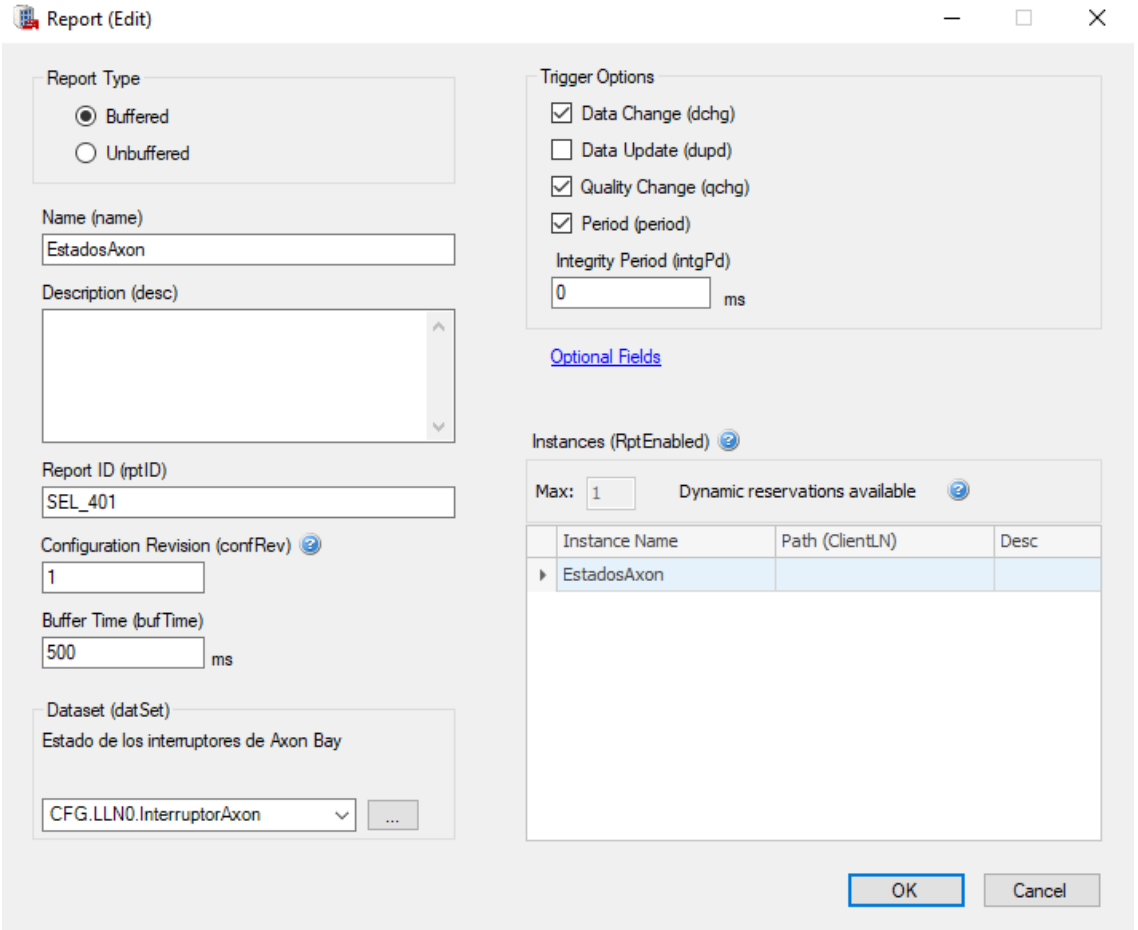


Figura 62: Configuración de reportes MMS en AcSELerator Architect.

Los comandos y disparos desde la protección del transformador (IED 1) hacia los interruptores se transmitirán mediante protocolo GOOSE, por lo tanto esto representará una suscripción para la MU, que se configura mediante la librería y código en C.

4.3.4.2. Configuración de archivo ICD

Para el modelado de los IED se utilizaron archivos ICD de dispositivos reales obtenidos y configurados con el software **acSELerator Architect**. Para la Merging Unit se utilizó un archivo ICD de un SEL 401, correspondiente con una MU comercial que actualmente se encuentra en el mercado.

Figura 63: Relé comercial SEL-401

Estos archivos poseen más nodos lógicos y una configuración por defecto distinta a la que se necesita para este proyecto, por lo que fue necesario adaptarlos para este caso.

El software **acSELerator Architect** permite modificar el archivo ICD para generar los DataSet necesarios, los RCB (Report Control Block) para los reportes MMS y los GoCB (Goose Control Block) para emitir mensajes Goose, pero no permite eliminar Nodos Lógicos.

Se eliminaron los reportes y DataSet que tenía el archivo por defecto y se crearon nuevos DataSet y GoCB acorde a las necesidades ya mencionadas.

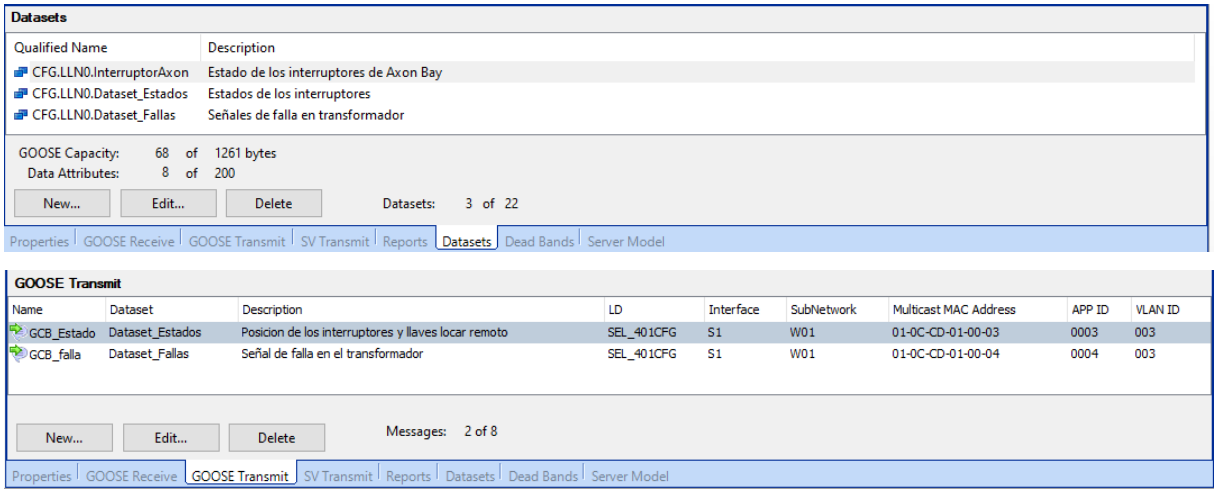


Figura 64: Configuración de nuevos DataSets y GoCB.

La lógica interna así como la recepción de mensajes MMS y GOOSE se configuran en la programación del servidor y no hace falta realizar la configuración en el archivo ICD.

4.3.4.3. Programación y diseño de MU

La programación del servidor que se ejecuta en Raspberry Pi se diseñó en función de los servidores de ejemplo que contiene la librería libiec61850. Estos ejemplos funcionaron como base del código desarrollado y se obtuvieron funciones específicas de la librería a partir de la exploración y prueba de estos ejemplos, para luego implementarlos en conjunto con el servidor creado.

Debido a esto el código fue desarrollado por 'bloques' o funciones que realizan una tarea específica del servidor y que interactúa únicamente con la función principal del código (main). En la sección de Anexos se encuentra el código completo con el archivo makefile correspondiente para la compilación del servidor.

A continuación se dará una breve explicación de cada sección del código sin realizar un análisis profundo de la sintaxis del lenguaje, sino más bien explicando a grandes rasgos la lógica de pensamiento o idea que busca resolver cada parte.

Definición de variables globales.

Se definen las variables utilizadas por varias funciones.

Gestión de interrupción del servidor.

Se establece un valor que detiene el bucle de la función principal al oprimir la combinación Ctrl+C en la consola.

Controlador de suscripciones GOOSE.

Posee la lógica que debe seguir el servidor cada vez que recibe un mensaje GOOSE con las características indicadas en la función principal.

Controlador para las entradas y salidas GPIO.

Se definen dos funciones con la lógica necesaria para cada señal. Estas funciones son llamadas continuamente por el bucle de la función principal y permite interrogar el valor de cada entrada detectando cambios en las mismas o escribiendo el valor guardado en el modelo interno para cada salida.

Controlador de cliente MMS.

Define la lógica que debe seguir el servidor cuando recibe un Reporte MMS que cumple con las características definidas en la función principal.

Función principal.

Inicia el servidor definiendo la IP y puerto.

Crea una instancia para recibir los mensajes GOOSE y define las suscripciones. Indica cual es la función que se debe llamar cuando recibe un mensaje.

Configura los parámetros para establecer la comunicación MMS con el relé AXON. Indica la IP, puerto, nombre del RCB y lo habilita. Solo intenta establecer la conexión una vez, si falla no intenta reestablecerla.

Define cuales son los pines utilizados como entrada y cuales son salidas.

Inicia el bucle que mantiene funcionando el servidor. Detiene el bucle por 100ms en cada vuelta para reducir la utilización de recurso. Actualiza el valor de las variables internas asociadas al modelo de datos en función de las entradas digitales. Actualiza los pines de salida en función de las variables del sistema.

Si el bucle se detiene elimina los objetos creados para el funcionamiento del servidor y la recepción de mensajes, liberando así los recursos del sistema utilizados.

Para la ejecución del mismo es necesario seguir el mismo procedimiento que al ejecutar un ejemplo de la librería como ya se explicó con anterioridad.

4.3.5. Modelado del Relé de Transformador

El segundo IED diseñado modela un relé de protección para el campo de transformación, capaz de enviar señales de disparo a los interruptores ante una falla, emitir comandos de apertura/cierre y reportar a la maestra el estado del sistema. Este se ejecuta en una PC con sistema Windows 10 y con las librerías Npcap instalada para el soporte de mensajes GOOSE.

4.3.5.1. Señales del dispositivo

A continuación se muestra un esquema donde se detallan las señales que interactúan con el relé del transformador y los protocolos de comunicación mediante el cual se recibe o envía la información.

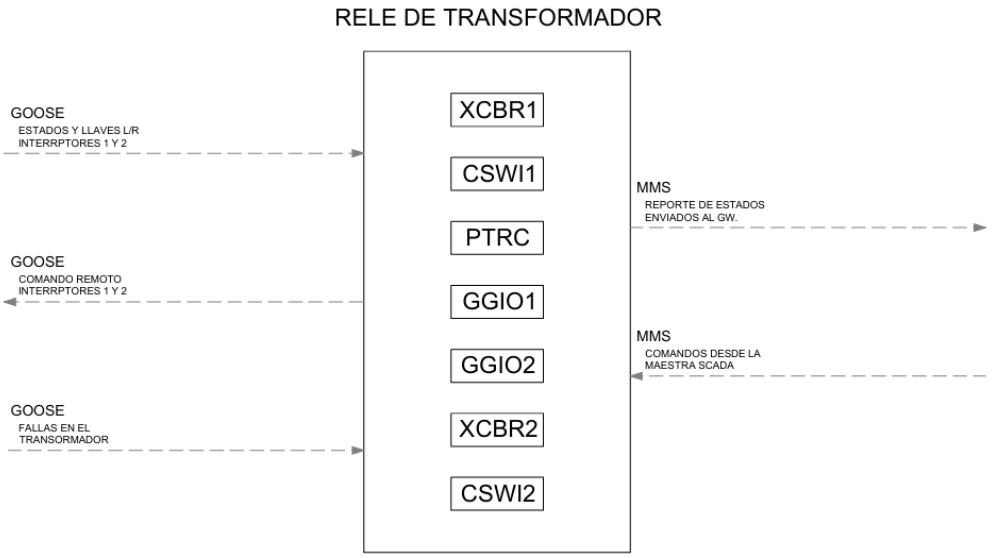


Figura 65: Esquema de señales del relé de transformador.

A partir de estas señales se determinó la cantidad mínima de Nodos Lógicos que debe tener el modelo de datos utilizado en el IED, los DataSet, Reportes y suscripciones necesarias.
Se determinó que como mínimo deben estar presentes los siguientes objetos:

- LLN0: Necesario para definir los DataSet, Report Control Block y GoCB.
- XCBR: Dos instancias para establecer el estado de cada interruptor.
- CSWI: Dos instancias para emitir los comandos a cada interruptor.
- PTRC: Necesario para el disparo producido desde la protección.
- GGIO: Necesario para obtener las señales provenientes de las protecciones internas del transformador.

Los comandos hacia los interruptores y el disparo por detección de falla se emitirán desde este IED como un mensaje GOOSE, por lo tanto se debe generar un DataSet con esta información y se asigna el mismo a un Goose Control Block. Este mismo DataSet se envía al Gateway por protocolo MMS.
Adicionalmente se generó otro DataSet para el reporte de las posiciones de cada interruptor, sus llaves local/remoto, la llave local/remoto de la protección y el estado de las alarmas debido a fallas en el transformador.

4.3.5.2. Configuración de archivo ICD

De forma similar al mencionado previamente se utilizó un archivo ICD obtenido y configurado en el software **acSELerator Architect**. En este IED se seleccionó el modelo de datos de un SEL-487E, que se trata de un Relé de protección de transformadores de la marca SEL que actualmente se encuentra en el mercado. El mismo cuenta con diversos nodos lógicos para la protección y supervisión de transformadores.

Figura 66: Relé comercial SEL-487E

El archivo obtenido cuenta con todos los Nodos Lógicos necesarios y aún más que no serán utilizados. También cuenta con DataSet y reportes por defecto que pueden ser modificados con el software **acSELerator Architec** para adecuarlo a las necesidades del proyecto.

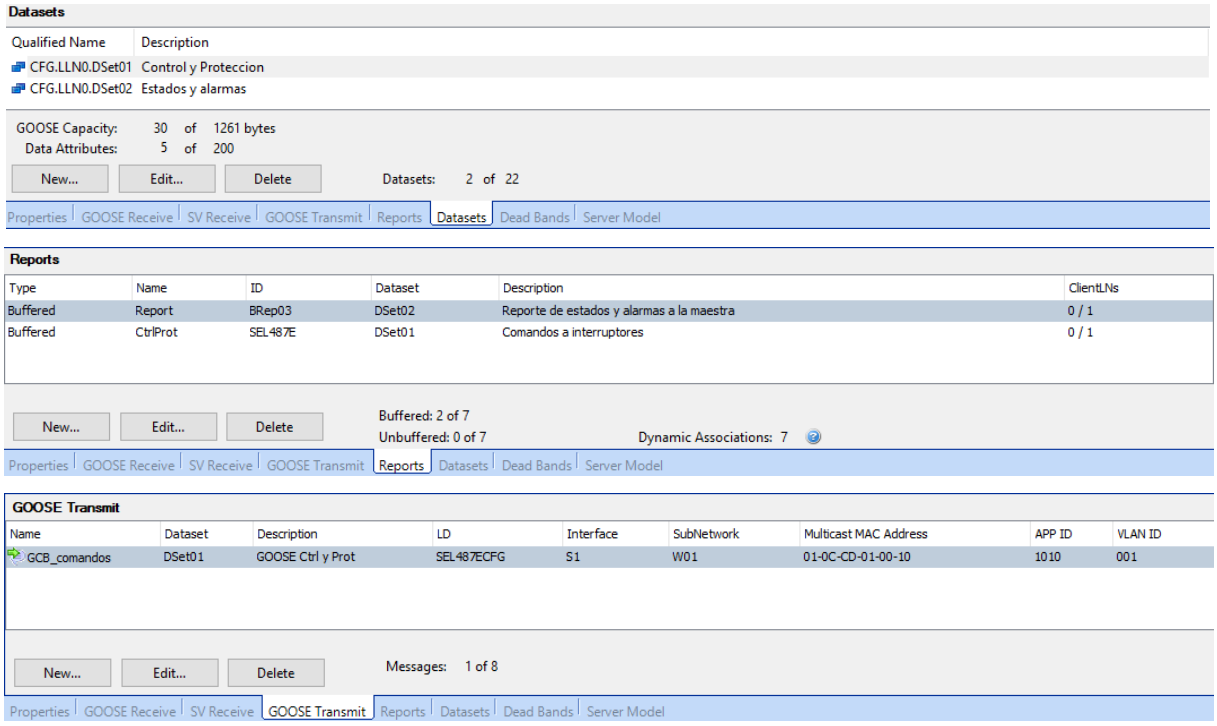


Figura 67: Configuración de DataSets, Reportes y GoRCB para Relé de Transformador.

Se puede observar en la siguiente figura el detalle de la configuración de Reportes MMS utilizado para enviar la información a la maestra GW.

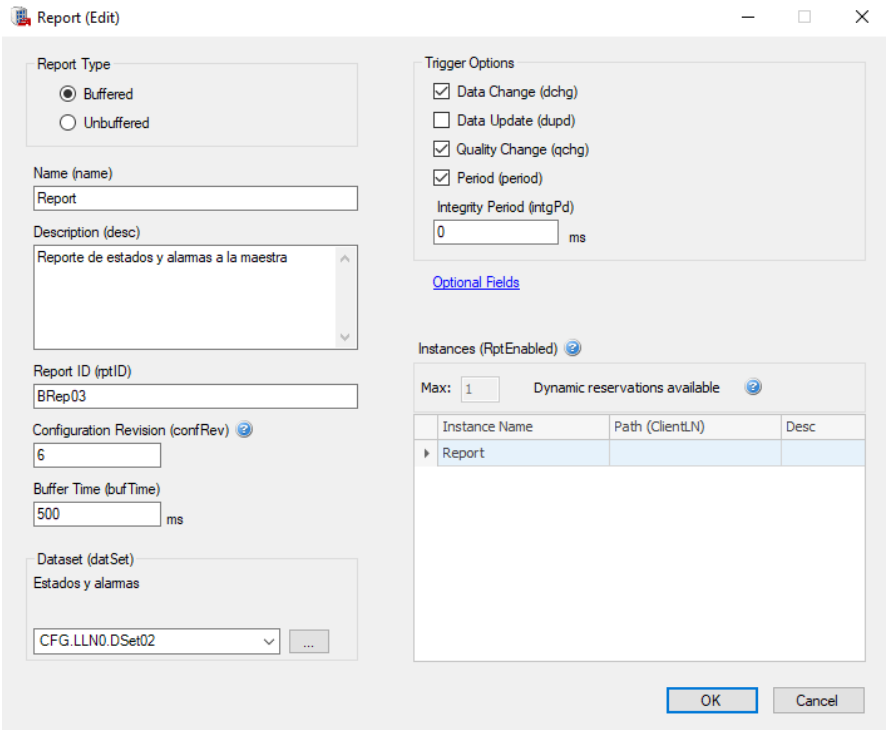


Figura 68: Configuración de Reportes en archivo ICD del relé de transformador.

Las suscripciones GOOSE y lógica interna del servidor se lograron mediante la programación del código fuente utilizando la librería libiec61850.

4.3.5.3. Programación y diseño de Relé

La programación del servidor que se ejecuta en Windows se realizó en base a los ejemplos de la librería libiec61850. El código fuente se realizó con un enfoque modular de manera tal que las propiedades del servidor se programaron en 'bloques'. Cada uno de estos resuelve independientemente una determinada características del servidor.

A diferencia del código diseñado para la Raspberry Pi este permite la interacción del usuario a partir de comandos escritos en la consola durante la ejecución. Se implementaron los siguientes comandos:

- help: Despliega el menú de comandos
- detener: Detiene el funcionamiento del servidor
- protlocal: Cambiar estado de la llave D/T de la protección
- 1open: Emite el comando de apertura del interruptor 1
- 1close: Emite el comando de cierre del interruptor 1
- 2open: Emite el comando de apertura del interruptor 2
- 2close: Emite el comando de cierre del interruptor 2
- monitoreo: Activar/Desactivar monitoreo por consola
- subgoose: Activar/Desactivar visualización de mensajes GOOSE
- actualizar: Muestra por pantalla el estado actual de las señales

Adicionalmente el servidor permite activar o desactivar la escritura por consola de los mensajes GOOSE subscritos recibido y mostrar las variables del sistema con la última estampa de tiempo en que fueron modificadas.

A continuación se dará una breve explicación de cada 'bloque' del código fuente, sin entrar en grandes detalles de la lógica y funciones implementadas.

Definición de variables globales:

Se definen variables que serán utilizadas en varias funciones o que tienen que mantener su valor aún cuando la función que la utiliza finaliza.

Manejo de interrupciones:

Define la combinación Ctrl+C como una forma de detener el bucle principal.

Funciones de estados para los atributos:

Define varias funciones utilizadas para asociar el valor de una variable con una cadena de caracteres. Estas funciones se utilizan para devolver el texto que se mostrará por pantalla. También se define la función que se encarga de realizar el print.

Actualización de estados en consola:

En caso que se encuentre activada la opción de mostrar los estados por pantalla esta función utiliza a las mencionadas previamente para escribir por pantalla los valores del modelo de datos en el instante que sea llamada.

Función de retardo para emisión de comandos:

Se crea un hilo que se ejecuta en paralelo con el servidor de la función principal, con el objetivo de asignarle el valor true a una variable, mantenerla durante 10 segundos y luego asignarle el valor false. Da soporte a la emisión de comandos desde el Relé.

Ejecución de comandos locales:

Es la función encargada de ejecutar los comandos desde el Relé verificando las condiciones de enclavamiento con la llave local/remoto de la protección.

También se encuentra la función encargada de modificar el estado de la llave local/remoto de la protección.

Interfaz de consola para operación manual:

Cuando esta función es llamada mantiene el servidor en Standby hasta que se ingrese un comando y se finalice el mismo con la tecla Enter. Si el comando se encuentra dentro de los admitidos realiza la acción correspondiente o llamado de las funciones de comandos locales.

Controlador para recepción MMS:

Define la lógica interna del IED ante un comando. En esta sección se define cuáles son los NL que pueden ser operados por telemando, el modo de operación y los enclavamientos. Informa por consola cuando un cliente se conecta e indica su dirección.

Controlador para suscripciones GOOSE:

Define la lógica interna del IED cuando recibe un nuevo mensaje GOOSE al que se encuentra suscripto. Tiene por condición principal verificar el parámetro stNum de cada mensaje GOOSE por separado, por lo tanto la lógica solo se ejecuta cuando ocurre una modificación en los atributos del mensaje. Permite asignar una lógica diferente a cada suscripción.

Función principal:

Al iniciar crea la instancia del servidor en el puerto 5102 (para evitar conflicto con otro servidor 61850 en la misma PC), y utilizando el adaptador de red indicado como parámetro de entrada.

Define los controladores que operan en hilos separados del servidor a la espera del llamado de un cliente para realizar un comando sobre un objeto controlable especificado.

Define una instancia para la recepción de mensajes GOOSE y establece los parámetros de las suscripciones, así como el controlador de cada una.

Inicia el bucle principal y ejecuta la función de interfaz de comandos en cada iteración hasta que el bucle se detenga. Si bien el servidor se encuentra continuamente en Standby a la espera de un comando para operar de forma local, los controladores se encuentran activos y esperando la recepción de nueva información para actualizar el modelo de datos interno. Este es un enfoque muy diferente al implementado en la Raspberry y con el cual se logró una estructura funcional que permite la interacción humano/maquina desde la consola, simulando la operación desde el frente del tablero de protecciones.

Cuando el bucle se detiene elimina las instancias de servidor y receptor GOOSE.

Para la ejecución del servidor se sigue el procedimiento ya explicado al igual que un ejemplo de la librería, con el detalle que debe especificarse un parámetro de entrada para indicar el adaptador de red que debe utilizar. El adaptador Ethernet suele estar direccionado con el valor 0, pero en algunos casos puede cambiar. Se puede determinar el ID que debe utilizarse con AT61 Server, viendo la pestaña Network Adapter al momento de crear un servidor. El ID coincide con el número de la izquierda para cada adaptador de red

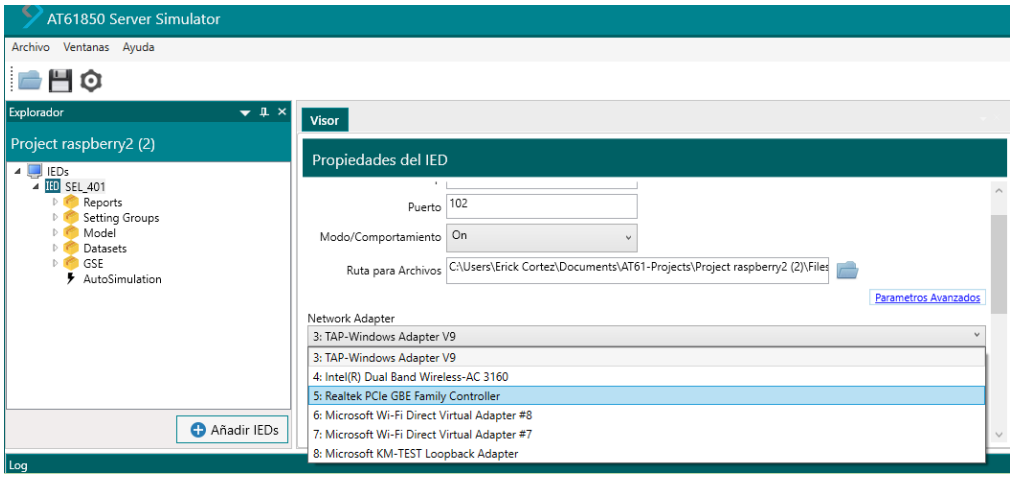


Figura 69: Adaptadores de red desde el software AT61 server.

Por ejemplo, en el caso mostrado si la ejecución del servidor se desea realizar sobre el adaptador Realtek Pcle GBE Family Controller se realiza el llamado por consola:

Figura 70: Servidor iniciado por consola por línea de comando.

4.4. Configuración de GW y SCADA

En este capítulo se presenta la configuración del sistema de supervisión y control implementado en el proyecto, basada en la utilización de **Axon Exchange** como gateway de comunicaciones y **Axon Builder** como plataforma SCADA. El objetivo principal es describir la integración entre los dispositivos de campo y el sistema de monitoreo centralizado.

Axon Exchange se emplea como elemento intermedio para la concentración y gestión de la información proveniente de los IEDs, permitiendo la comunicación entre distintos protocolos y el posterior envío de los datos al sistema SCADA. Por su parte, Axon Builder se utilizó para el diseño del entorno de supervisión, proporcionando las herramientas necesarias para la visualización y operación del sistema.

4.4.1. Configuración de Gateway

El gateway de comunicaciones del proyecto fue instalado y ejecutado en una PC conectada a la red LAN, la cual actúa como nodo central para la recopilación de la información proveniente de los distintos IEDs previamente descritos.

La comunicación entre el gateway y los IEDs se realiza utilizando el protocolo MMS, conforme al estándar IEC 61850, lo que posibilita el acceso a las variables y estados publicados por cada equipo. De esta manera, el gateway obtiene la información necesaria para su posterior procesamiento y redistribución hacia los niveles superiores del sistema.

Para lograrlo fue necesario instalar el software observando que se instalan tres aplicaciones.

Para la configuración del gateway se utilizó la herramienta **AE Config**, comenzando con la creación de un nuevo proyecto, al cual se solicita que se asigne una contraseña para restringir el acceso a la configuración. Una vez creado el proyecto, se procede a la definición de los dispositivos esclavos bajo el protocolo IEC 61850, accediendo a la pestaña correspondiente y agregando un nuevo equipo por cada IED a integrar. Para cada dispositivo se configura un nombre identificatorio, la dirección IP y el puerto de comunicación.

Cuando los parámetros de comunicación son correctos, es posible realizar el descubrimiento del dispositivo mediante la opción “Descubrir”, lo que permite obtener automáticamente el modelo de datos del IED, siempre y cuando se encuentre en funcionamiento. A partir de este modelo, se pueden explorar y seleccionar las distintas señales disponibles, organizadas en entradas digitales, salidas digitales, entradas analógicas y salidas analógicas. También se pueden activar o desactivar los reportes previamente configurados en cada IED. Este procedimiento se repite para cada uno de los IEDs hasta completar la totalidad de los esclavos IEC 61850 definidos en el proyecto.

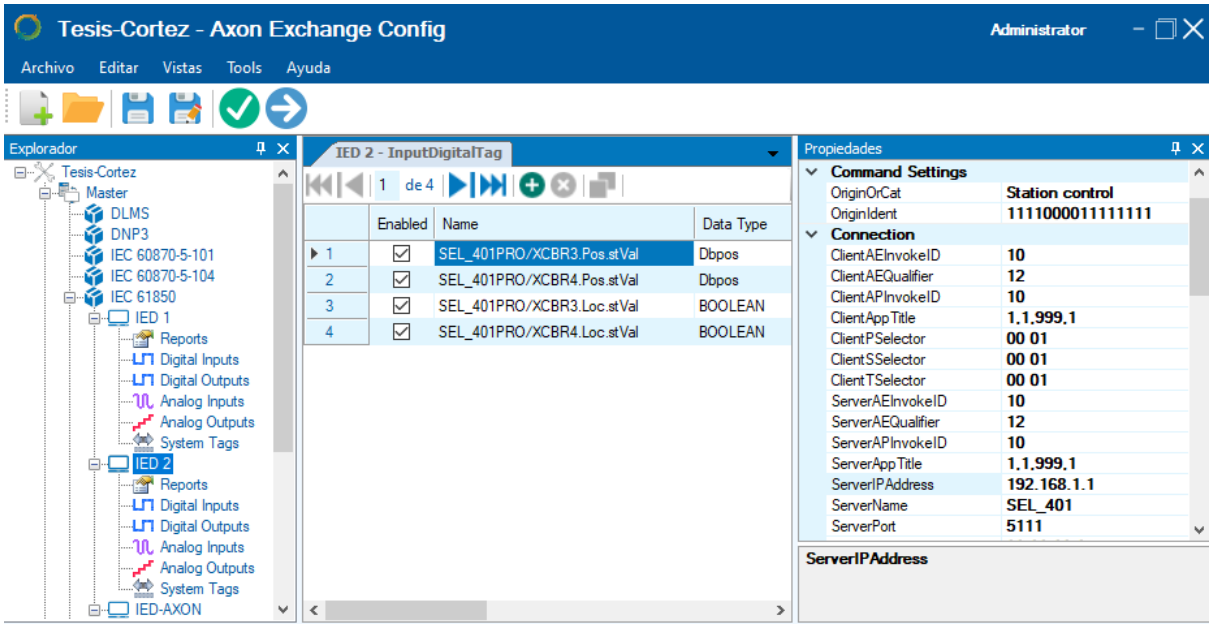


Figura 71: Entradas digitales descubiertas por Axon Exchange para el IED2.

Como se mencionó en capítulos previos para evitar conflictos con el Reporte que contiene los estados del IED Axon y que son utilizados por el IED2, se deshabilitan los mismos en la configuración de Axon Exchange como se puede ver en la Figura 72.

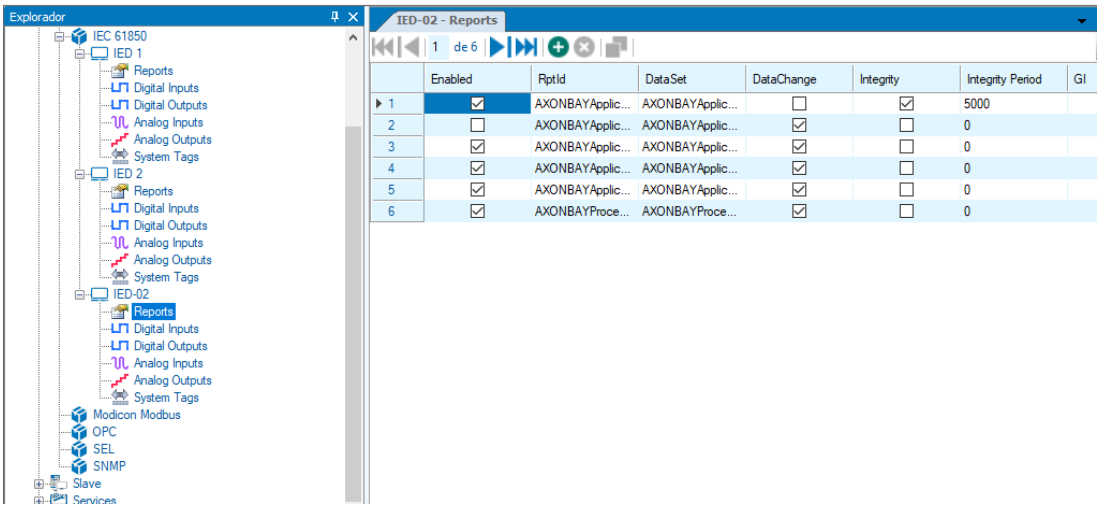


Figura 72: Configuración de Reportes obtenidos del IED Axon.

Se debe prestar especial atención al momento de configurar las Digital Outputs de cada IED ya que en la columna Data type se puede observar el tipo de comando que se enviara a cada Data Object controlable, además de los checkeos necesarios.

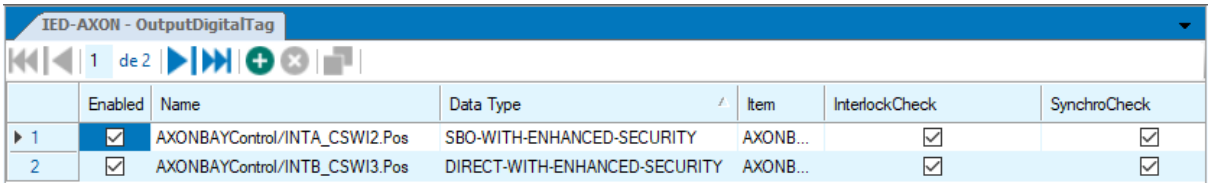


Figura 73: Señales de comando descubiertas para el IED1.

Una vez configurados los dispositivos esclavos, se procede a la configuración del dispositivo maestro DNP3, encargado de enviar la información hacia el sistema SCADA. En la pestaña correspondiente se agrega un nuevo dispositivo maestro, definiendo su nombre y la dirección IP de la PC donde se ejecuta el SCADA. El puerto de comunicación se mantiene en su valor predeterminado, 20000, correspondiente al protocolo DNP3. También la dirección configurando correctamente los valores Link address y Master link address.

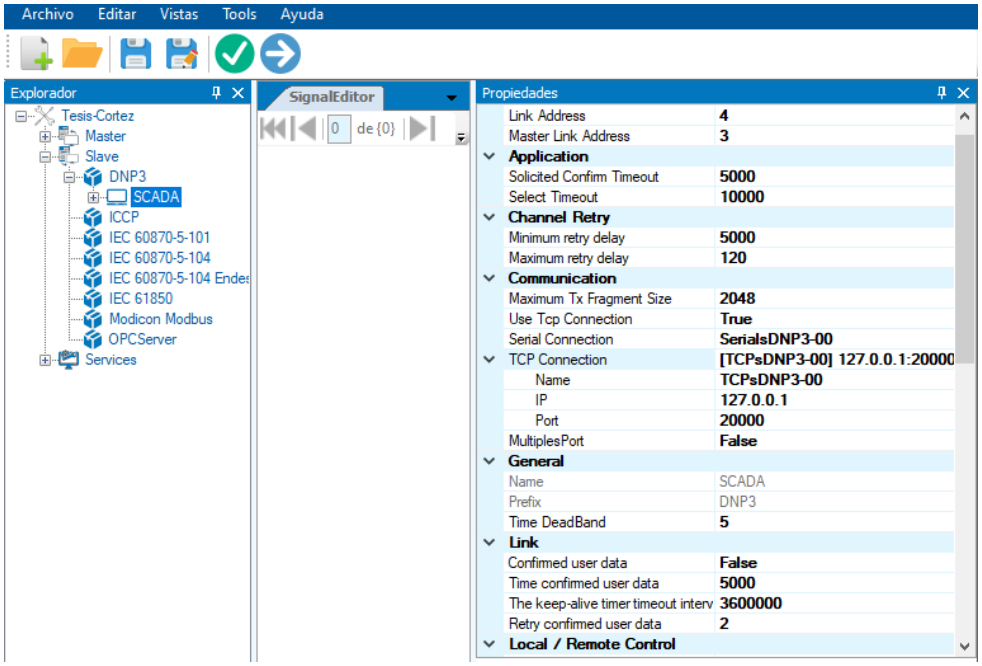


Figura 74: Configuración de parámetros de comunicación con la maestra SCADA.

Posteriormente, mediante la opción de **Mapeado**, se seleccionan las señales que serán enviadas a la maestra DNP3, estableciendo la correspondencia entre las variables obtenidas de los esclavos IEC 61850 y los puntos DNP3. En esta etapa resulta fundamental configurar el tipo de dato y el número de ítem de cada variable ya que esta debe configurarse de igual manera en el SCADA para la recepción de información.

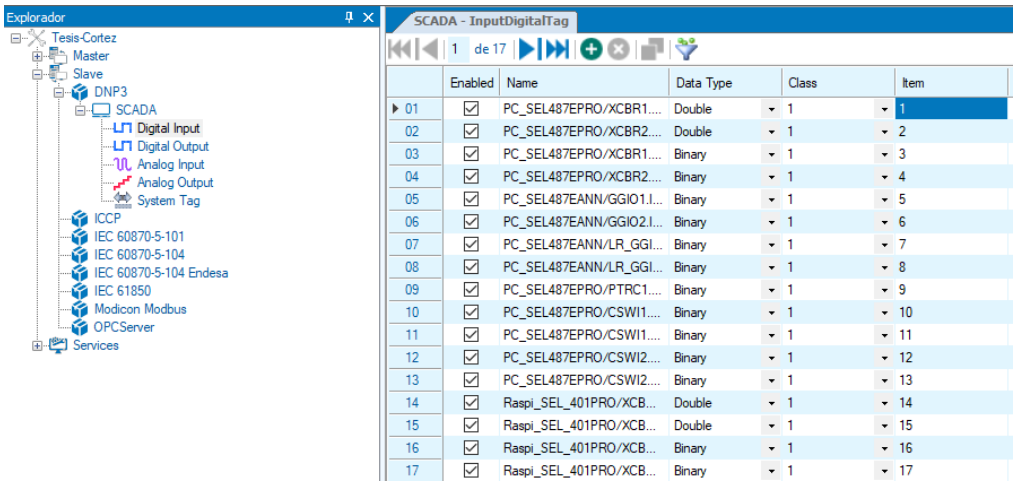


Figura 75: Entradas digitales enviadas a la maestra SCADA.

Para las Digital Outputs se debe indicar el número de ítem de cada señal, si se requiere selección previo a la emisión del comando y si el tipo de comando es persistente, pulso o mediante la modalidad trip/close.

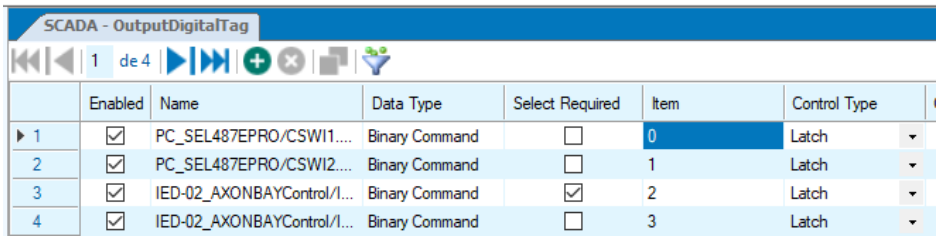


Figura 76: Salidas digitales configuradas en SCADA

Finalizada la configuración, el proyecto se guarda y se procede a la generación y ejecución del runtime, lo que da lugar a un archivo de configuración que contiene toda la lógica de comunicación definida. Este runtime puede ser visualizado mediante AE Runtime Viewer, ya sea desde la misma PC donde se realizó la configuración o desde otro equipo conectado a la misma red, indicando la dirección IP y el puerto de acceso. En el caso de ejecutarse localmente, se utilizó la dirección 127.0.0.1 (localhost) y el puerto 7654, asignado por defecto para la lectura del runtime.

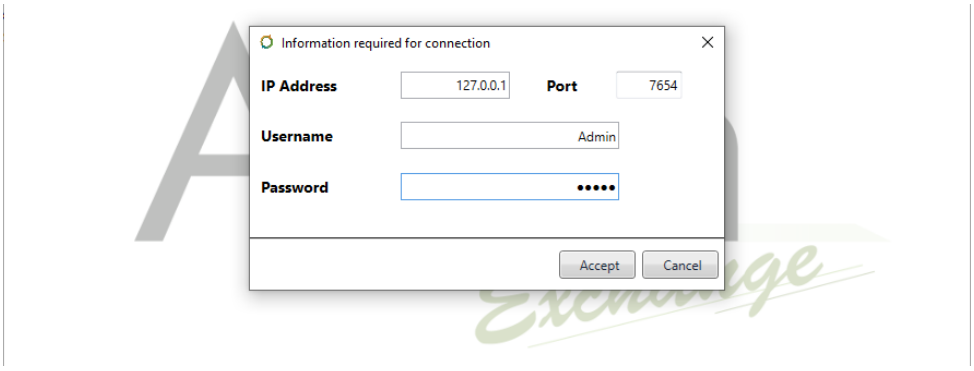


Figura 77: Pantalla de inicio de AE Viewer.

Desde **AE viewer** se ingresa con la contraseña del proyecto y la dirección IP. En caso de ejecutar la configuración con otro equipo se puede acceder a la misma mediante la IP de dicho equipo y el mismo puerto.

Una vez iniciado este se podrán visualizar las variables configuradas en cada dispositivo, así como su estado de conexión, el nombre de cada variable, status y calidad de la señal.

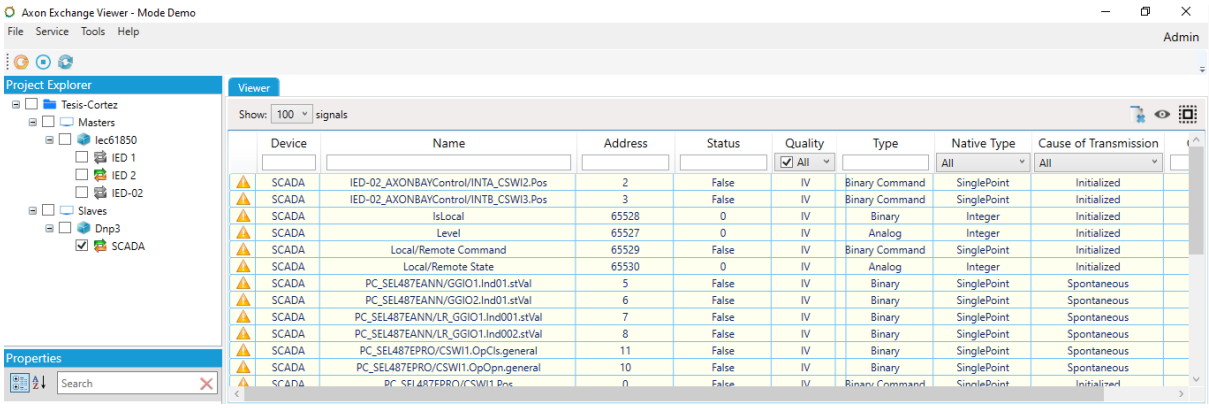


Figura 78: Señales configuradas en Axon Exchange.

Desde aquí es posible forzar señales para realizar pruebas contra el SCADA o emitir comandos hacia los IEDs.

4.4.2. Diseño de SCADA

El diseño del interfaz humano-máquina se realizó con el software AXON BUILDER. Debido a las limitaciones del software solo se modelaron las señales asociadas a los estados de los interruptores y sus comandos. La pantalla diseñada es la mostrada en la siguiente Figura 79:

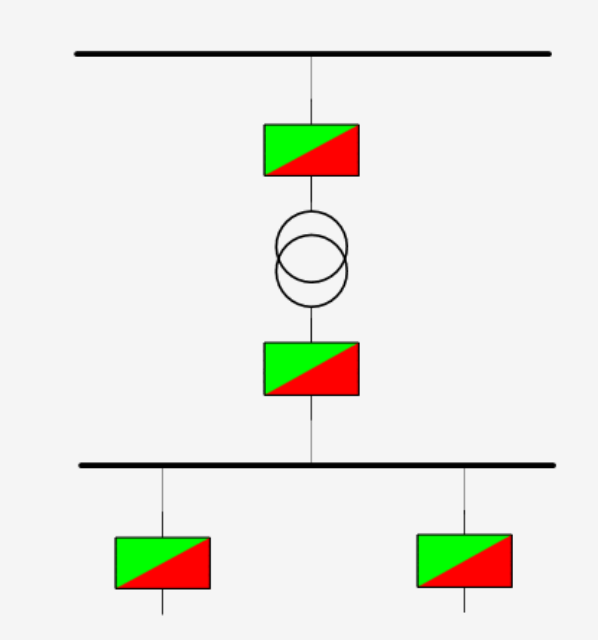


Figura 79: Pantalla diseñada para el HMI.

Debido a que el software se ejecuta en la misma PC que el gateway los parámetros de comunicación son los mostrados en la figura. Se establece como dirección de enlace local 3 y dirección remota en 4, que son valores preestablecidos que suele utilizar el protocolo DNP3 y previamente fueron configurados en el gateway también. La dirección IP utilizada es la localhost 127.0.0.1 y el puerto 20000 utilizado por defecto para el protocolo DNP.

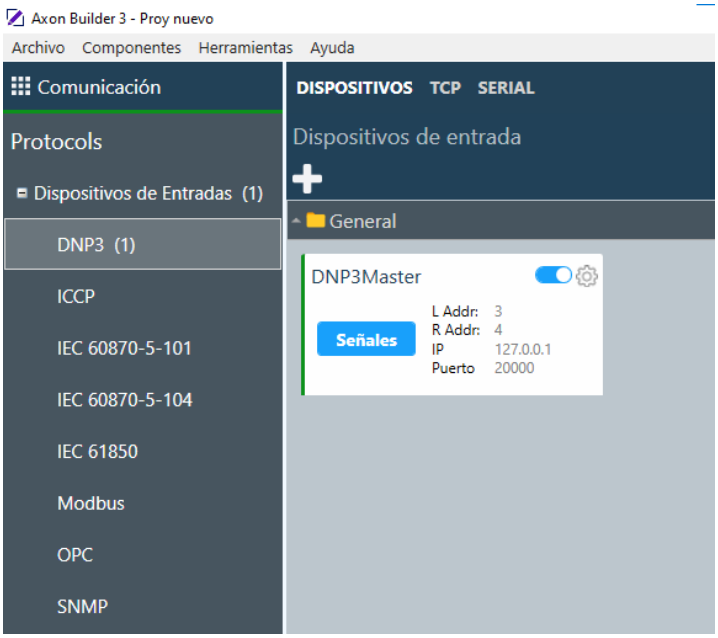


Figura 80: Configuración de comunicación con el Gateway.

Con los parámetros de comunicación ya configurados se agregaron las señales de entradas y salidas digitales manualmente, ya que este software no posee una opción para descubrir las variables desde el dispositivo configurado. Se debe prestar atención al tipo de señal configurado y la dirección, ya que estas deben coincidir con las señales enviadas.

	Habilitar	Topología	Nombre	Alarma	Tipo	Dirección	Inicial	Invertido	Enumeración	Log	Comentario
1	☑	Gateway	POS INT 1	NONE	Double	1	1	☐	NONE	☐	
2	☑	Gateway	POS INT 2	NONE	Double	2	1	☐	NONE	☐	
3	☑	Gateway	DT PROT 1	NONE	Binary	3	1	☐	NONE	☐	
4	☑	Gateway	LR INT 1	NONE	Binary	7	1	☐	NONE	☐	
5	☑	Gateway	LR INT 2	NONE	Binary	8	1	☐	NONE	☐	
6	☑	Gateway	POS INT 3	NONE	Double	14	1	☐	NONE	☐	
7	☑	Gateway	POS INT 4	NONE	Double	15	1	☐	NONE	☐	
8	☑	Gateway	DT PROT 2	NONE	Binary	16	1	☐	NONE	☐	

Figura 81: Entradas digitales del SCADA configuradas en Axon Builder.

Dentro del mismo dispositivo se configuran los comandos digitales también teniendo en cuenta la dirección y tipo de comando así como tipo de control.

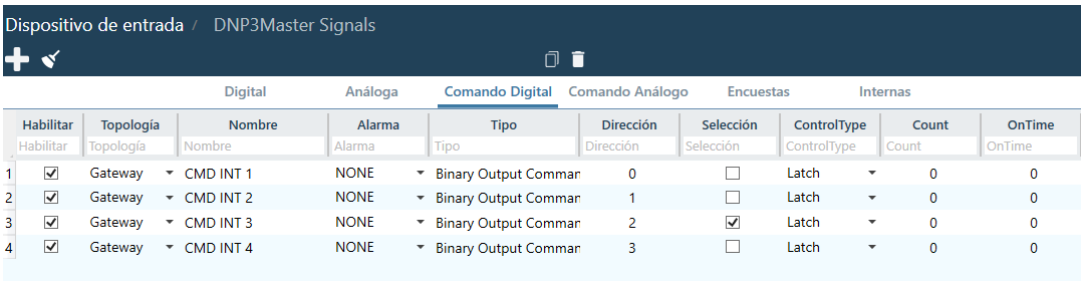


Figura 82: Salidas digitales del SCADA configuradas en Axon Builder.

El diseño de la pantalla se realiza una vez finalizada la configuración de todas las señales. Para ello se selecciona el elemento al que se le asignara una variable y en la sección diseño se asocia el estado con una función que utiliza la señal correspondiente.

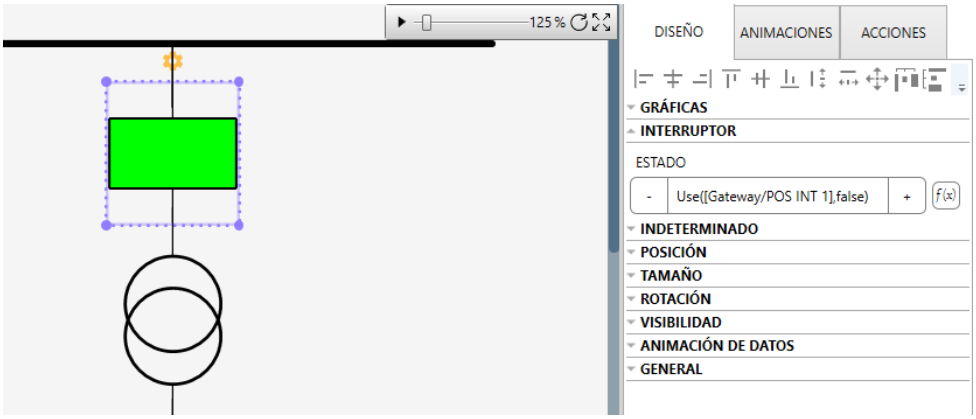


Figura 83: Asociación entre figura y variable en Axon Builder.

En la sección de Acciones se configura el envió de comandos para cada interruptor, asociando las señales de comando y posición del interruptor como se puede ver en la Figura 1Figura 84 para el interruptor 1. Se desactivo la opción de solicitar contraseña en la pestaña de Seguridad.

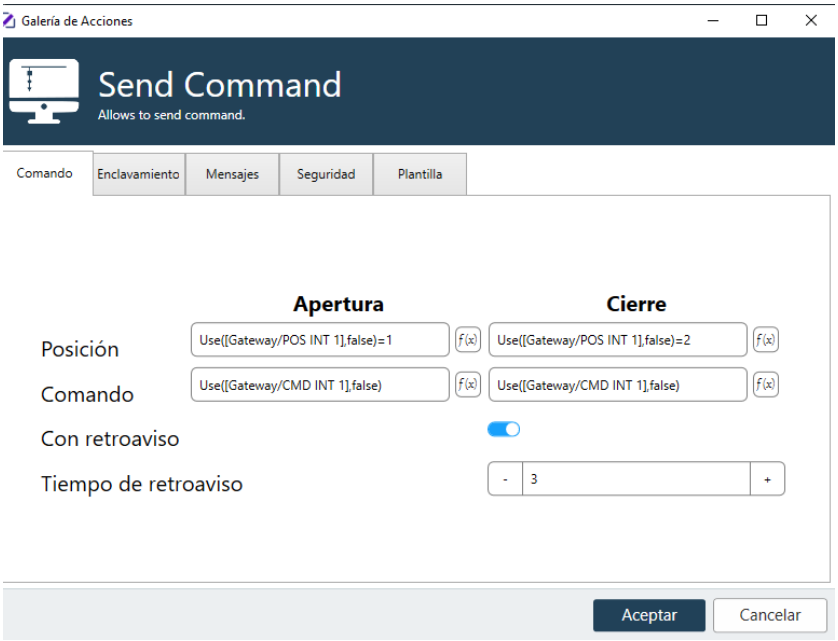


Figura 84: Configuración de acciones de comando.

5. Resultados y análisis

En este capítulo se presentan los resultados obtenidos a partir de la implementación del laboratorio de automatización basado en el estándar IEC 61850 desarrollado en los capítulos anteriores. El objetivo principal de esta sección es verificar el correcto funcionamiento del sistema implementado, evaluar su desempeño desde el punto de vista operativo y analizar el grado de cumplimiento de los objetivos propuestos al inicio del trabajo.
Para ello, se realizan distintas pruebas sobre los dispositivos diseñados, los mecanismos de comunicación implementados y la integración con sistemas externos, tales como el tablero de comando, el gateway de comunicaciones y el sistema SCADA.

5.1. Verificación del funcionamiento

Con el objetivo de validar el correcto desempeño del laboratorio, se realizaron ensayos funcionales y de comunicación sobre los dos IED diseñados mediante programación en lenguaje C utilizando la biblioteca libiec61850. Las pruebas realizadas se orientaron a verificar:

- Correcta inicialización de los servidores IEC 61850.
- Funcionamiento del servicio MMS.
- Intercambio bidireccional de mensajes GOOSE.
- Integración completa con el sistema SCADA y actuación física sobre el tablero de comando.

Se destaca que al momento de realizar pruebas entre varios equipos conectados a la red se realiza la configuración y verificación de las direcciones IP de cada dispositivo así como un ping entre los mismos, desactivando el firewall en caso que sea necesario.

5.1.1. Verificación funcional de los servidores implementados

El IED1 fue implementado como servidor IEC 61850 ejecutado en un entorno Windows, representando un relé de protección de transformador con capacidad de comunicación MMS y publicación/suscripción de mensajes GOOSE. Por otra parte, el IED2 fue implementado sobre Raspberry Pi utilizando sistema operativo Raspbian. Este dispositivo representa una Merging Unit simplificada orientada al manejo de estados binarios y control de interruptores.

5.1.1.1. Inicialización del servidor

Se verificó el correcto arranque del servidor mediante su ejecución desde la consola del sistema operativo. Durante la inicialización se comprobó:

1. Carga correcta del modelo de datos: Esta verificación es fundamental, ya que permite confirmar que los Logical Devices, Logical Nodes, DataSets y Control Blocks configurados se encuentran efectivamente integrados al servidor y disponibles para su utilización.
2. Apertura del puerto TCP correspondiente al servicio MMS: La apertura exitosa de dicho puerto indica que la pila de comunicación implementada por la biblioteca libiec61850 ha sido correctamente inicializada y que el servidor se encuentra en estado de escucha para aceptar solicitudes de conexión externas.
3. Disponibilidad del servidor para aceptar conexiones remotas desde un cliente MMS: Esta prueba permite validar el establecimiento correcto de sesiones cliente-servidor, la gestión de solicitudes de lectura y escritura, y la interacción completa con el modelo de datos implementado.

La Figura 85 muestra la ejecución del archivo compilado correspondiente al servidor IED1 desde la consola del sistema operativo. En la imagen se observa el llamado al ejecutable, la confirmación por pantalla del inicio correcto del servidor, así como la interfaz de red utilizada y el puerto asociado al servicio MMS.

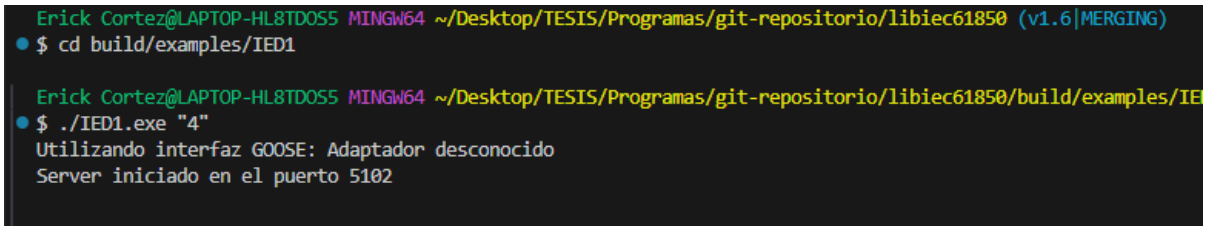


Figura 85: Inicio de servidor IED1 desde consola de Visual Studio Code.

La inicialización del servidor sin mensajes de error asociados a la creación del modelo, asignación de adaptador de red y puerto permite verificar estos puntos. Asimismo, establecer la conexión desde el cliente MMS AT61 Client permitió asegurar que el servidor se encuentra efectivamente operativo.

5.1.1.2. Verificación del modelo de datos y servicio MMS

Una vez confirmada la correcta inicialización del servidor, se procedió a verificar la exposición del modelo de datos y la operación del servicio MMS mediante la conexión desde un cliente externo compatible con IEC 61850.

Las pruebas realizadas permitieron validar tres aspectos fundamentales del funcionamiento del servidor.

1. Navegación del modelo de datos: Se comprobó la correcta exposición jerárquica del modelo IEC 61850 implementado, verificando la estructura de *Logical Devices*, *Logical Nodes*, Data Objects y Data Attributes definidos durante la etapa de diseño.
2. Lectura de atributos de nodos lógicos: Se verificó la capacidad del cliente para realizar operaciones de lectura sobre atributos pertenecientes a distintos nodos lógicos del IED. Esta prueba permite validar el correcto funcionamiento del servicio MMS en operaciones de tipo *GetDataValues*, así como la coherencia entre el estado interno del servidor y los valores transmitidos al cliente.
3. Escritura remota de variables de control: Se verificó la operación de escritura remota sobre variables de control asociadas al modelo implementado. Esta prueba permite validar la capacidad del servidor para procesar solicitudes *SetDataValues* y ejecutar las acciones correspondientes dentro de su lógica interna.

La Figura 86 muestra la consola del IED1 durante la recepción de un comando de apertura emitido por un cliente MMS y la actualización de las variables internas del sistema mostrada por pantalla mediante el listado de estados.

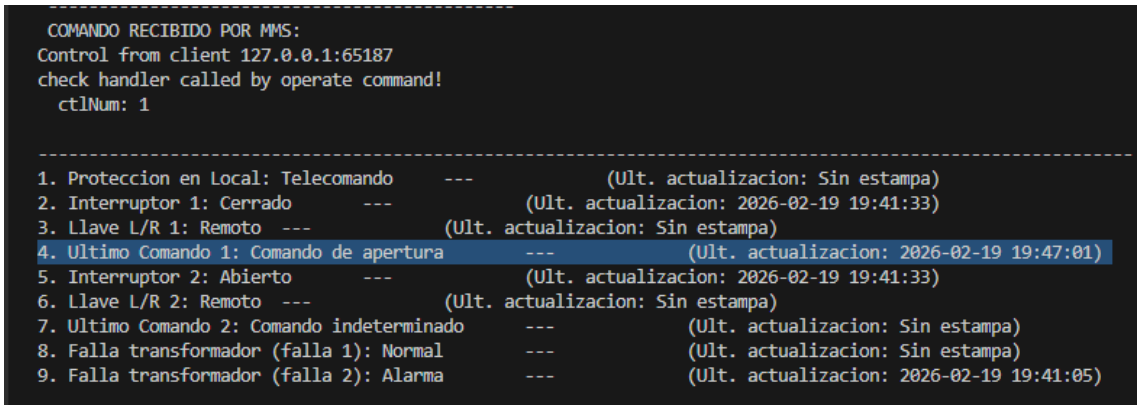


Figura 86: Recepción de comando por MMS en consola.

Las pruebas de lectura y escritura confirmaron la correcta exposición del modelo IEC 61850 implementado y la adecuada respuesta del servidor ante solicitudes MMS.

Prueba realizada	**Descripción de la prueba**	**IED1**	**IED2**
Navegación completa	Verifica conexión mediante el software AT61 Client obteniendo todos los NL del modelo de datos.	SI	SI
Lectura de atributos	Desde AT61 Client se selecciona un Data Attribute tipo SPS o DPS y se consulta su estado en el servidor.	SI	SI
Escritura remota	Desde AT61 Client se envía un comando hacia un Data Attribute de tipo controlable verificando si el cambio es aceptado por el servidor.	SI	NO

Tabla 11: Pruebas de verificación del modelo de datos.

El IED2 no verifica la prueba de escritura remota ya que se diseñó con el fin de recibir los comandos de disparos mediante el protocolo GOOSE, y por lo tanto la falta de respuesta ante un comando MMS es lo esperado.

5.1.1.3. Verificación de recepción de reportes MMS

Con el objetivo de validar el funcionamiento como cliente MMS implementado en el IED2 para recibir los estados de los interruptores simulados en el software Axon Bay se realizaron pruebas con los dos dispositivos conectados en red.

Una vez establecida la conexión, se verificó la recepción automática de reportes ante cambios de estado dentro de Axon Bay. Los eventos fueron generados mediante modificaciones en el estado de los interruptores simulados, observándose la actualización inmediata de los valores mostrados por consola y el cambio de estado medido en los pines GPIO correspondientes.

Esta prueba permitió confirmar el correcto funcionamiento del mecanismo de notificación basado en reportes MMS, validando la capacidad del servidor implementado para transmitir eventos de cambio de estado hacia clientes externos conforme a los servicios definidos en el estándar IEC 61850.

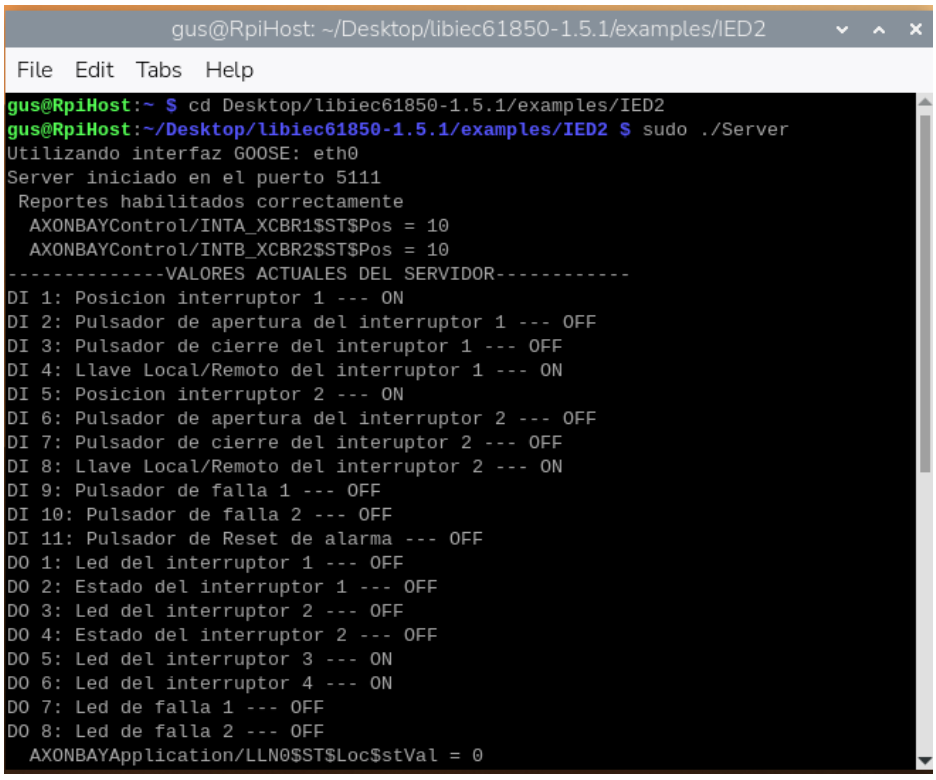


Figura 87: Recepción de estados del Axon Bay en consola de Raspberry Pi.

5.1.2. Verificación del intercambio GOOSE entre dispositivos

Una vez verificado el correcto funcionamiento individual de los servidores implementados, se procedió a evaluar el intercambio de mensajes GOOSE entre los mismos.

El objetivo de esta prueba fue validar el funcionamiento del mecanismo de comunicación orientado a eventos definido en el estándar IEC 61850, verificando tanto la publicación como la suscripción de mensajes GOOSE entre ambos dispositivos configurados respectivamente como publicador y suscriptor según el escenario de operación del laboratorio.

5.1.2.1. Publicación de mensajes GOOSE

En primer lugar se verificó la correcta generación y transmisión de mensajes GOOSE ante cambios de estado en las variables asociadas a los DataSets configurados en cada servidor. Dichos cambios fueron generados mediante modificaciones en variables internas del programa por comandos desde la interfaz de consola y mediante operaciones realizadas desde el cliente MMS para el IED1. Para el IED2 los cambios de las variables internas se realizaron forzando eléctricamente las entradas digitales mediante los pines GPIO de la Raspberry Pi. Ante la ocurrencia de estos eventos, los servidores generaron automáticamente la publicación de los mensajes GOOSE correspondientes, los cuales fueron transmitidos a través de la red Ethernet del laboratorio y capturados con el software Wireshark como puede verse en la Figura 88.

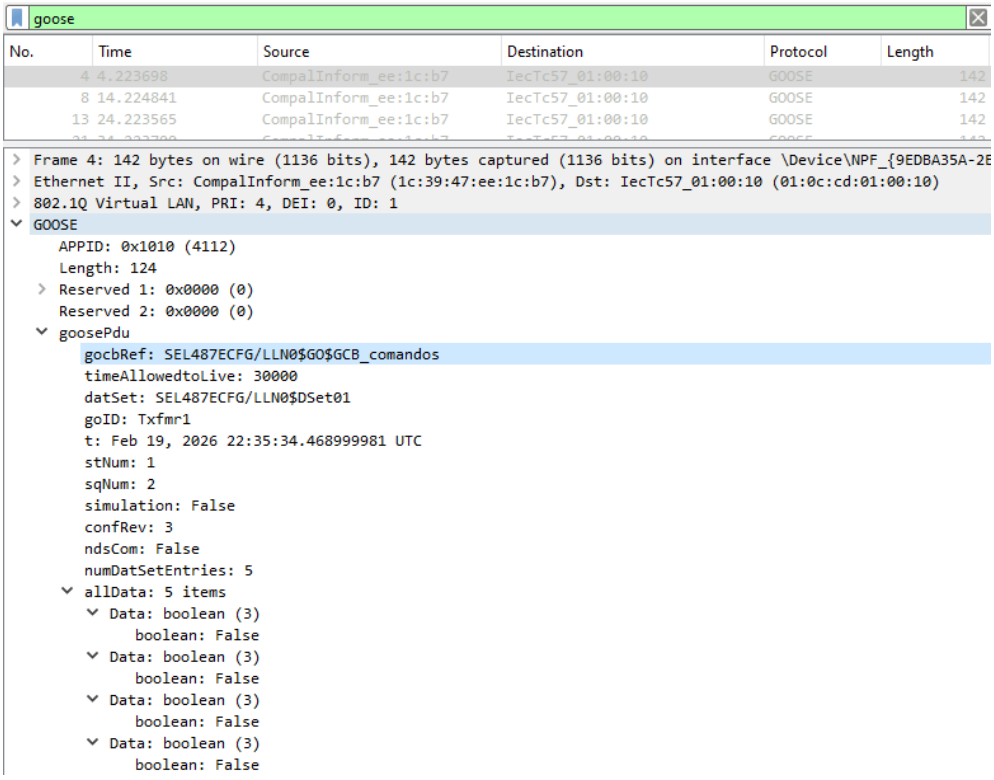


Figura 88: Verificación de emisión GOOSE mediante captura de paquetes en Wireshark.

5.1.2.2. Recepción y procesamiento de mensajes GOOSE

Posteriormente se verificó la correcta recepción de los mensajes GOOSE en el dispositivo suscriptor correspondiente. Al recibir una trama GOOSE válida, el servidor actualizó las variables internas asociadas al DataSet recibido, permitiendo que la lógica implementada en cada dispositivo reaccionara ante los eventos generados por el otro IED.

Esta prueba permitió confirmar la correcta implementación del mecanismo de suscripción y procesamiento de mensajes GOOSE en ambos dispositivos, validando la interoperabilidad entre los servidores desarrollados mediante la biblioteca libiec61850.

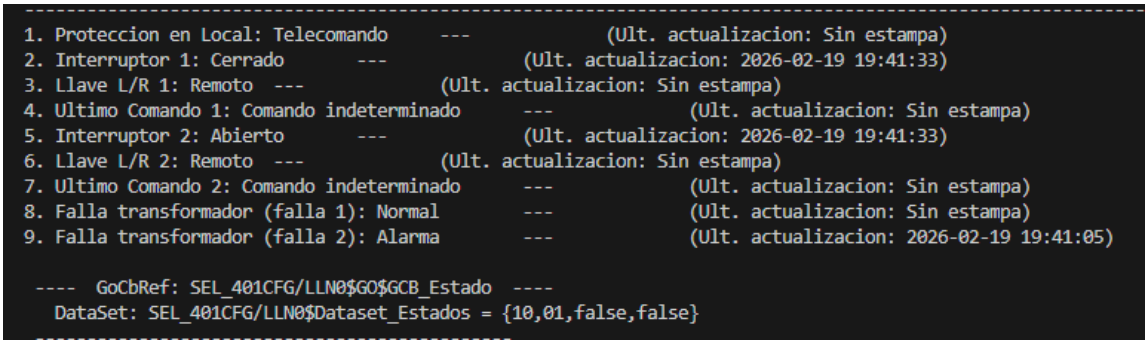


Figura 89: Verificación de recepción GOOSE por consola del IED1.

En la Figura 89 se puede observar la recepción del mensaje GOOSE en la consola del IED1, con el GoCB SEL_401CFG/LLN0GOGCB_Estado emitido desde el IED2 con la posición de los interruptores 1 y 2 así como la posición de sus respectivas llaves L/R.

La prueba confirmó la coherencia funcional entre el modelo lógico IEC 61850 y la actuación física implementada.

5.1.3. Vinculación del tablero de comando con el sistema de control

Con el objetivo de validar la interacción entre los IED implementados y los elementos físicos del laboratorio, se evaluó la integración del tablero de comando con el IED2 conectado y en funcionamiento.

5.1.3.1. Verificación de señales de entrada

En primer lugar se verificó la correcta adquisición de señales provenientes del tablero de comando hacia el IED2. Para ello se realizaron cambios de estado en los dispositivos de entrada conectados a los pines GPIO de la Raspberry Pi, observándose la actualización correspondiente de las variables internas del servidor IEC 61850.

Estas variaciones fueron posteriormente transmitidas a través de mensajes GOOSE hacia el IED1, permitiendo validar la correcta propagación de los eventos dentro del sistema de comunicación.

5.1.3.2. Verificación de comandos de operación

Posteriormente se evaluó la ejecución de comandos, tanto locales como provenientes de otros dispositivos del sistema, incluyendo el IED1 y el sistema SCADA. Ante la recepción de dichas órdenes, el IED2 procesó los comandos recibidos y actuó sobre las salidas digitales de la Raspberry Pi, produciendo el encendido o apagado de los indicadores luminosos presentes en el tablero. Se comprobó la lógica de enclavamiento en cada caso verificando el comportamiento esperado para las órdenes de carácter local y remoto.

Este comportamiento permitió confirmar la correcta implementación de la lógica de control dentro del servidor, así como la adecuada vinculación entre los eventos de comunicación y la actuación física sobre el sistema.

La Figura 90 muestra el tablero de comando durante las pruebas de funcionamiento del sistema. En la imagen puede observarse la activación de los indicadores luminosos asociados a los estados de operación del sistema, evidenciando la correcta interacción entre los dispositivos IEC 61850 implementados y los elementos físicos del laboratorio.

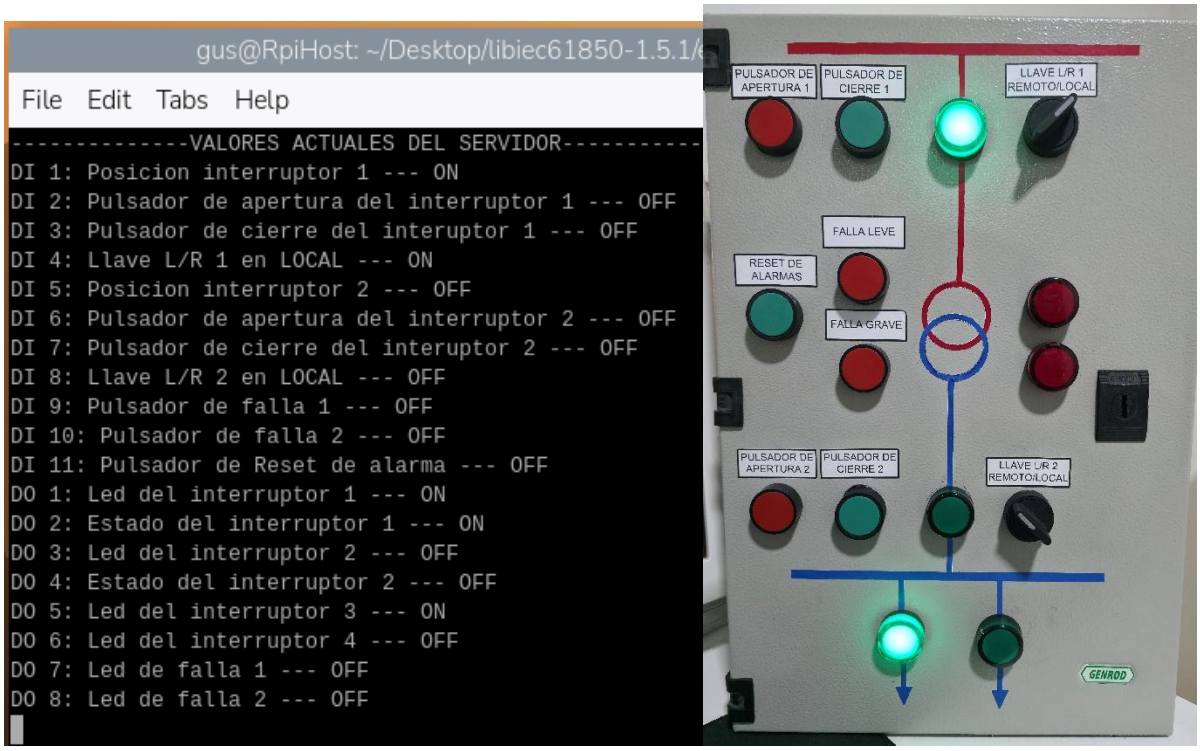


Figura 90: Verificación de comandos y estados entre IED2 y tablero de comando.

5.1.4. Integración con sistema SCADA y actuación física

Una vez verificado el funcionamiento individual de los dispositivos implementados y la correcta comunicación entre ellos mediante mensajes GOOSE, se procedió a evaluar la integración del sistema con el Gateway y el sistema SCADA utilizados en el laboratorio.

5.1.4.1. Verificación de la supervisión de variables

En primer lugar se verificó la correcta visualización de las variables de estado provenientes de los dispositivos implementados dentro de la interfaz del sistema SCADA. Para ello se estableció la conexión entre el Gateway y los servidores IEC 61850, permitiendo la adquisición de los valores asociados a los nodos lógicos definidos en el modelo de datos.

Una vez establecida la comunicación, se comprobó que los cambios de estado generados en los dispositivos eran correctamente reflejados en la interfaz del sistema SCADA, permitiendo la supervisión remota del estado del sistema.

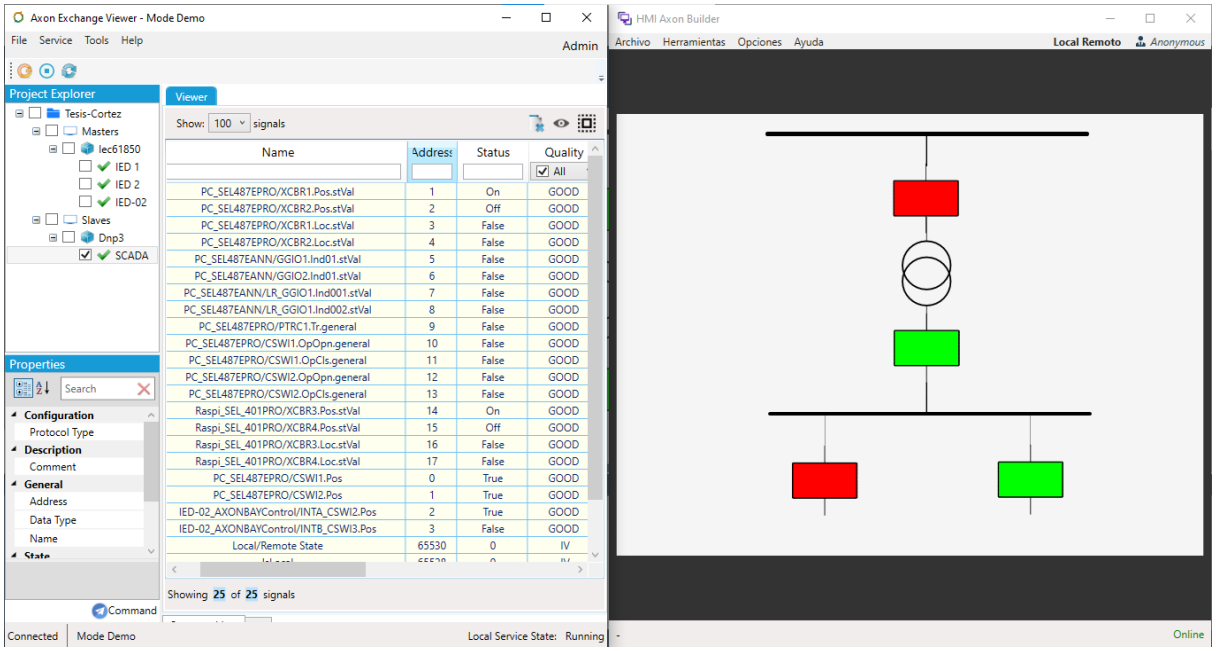


Figura 91: GW y SCADA con los dispositivos online.

5.1.4.2. Verificación de comandos de operación desde SCADA

Posteriormente se evaluó la capacidad del sistema SCADA para enviar comandos de operación hacia los dispositivos implementados. Para ello se ejecutaron órdenes de control desde la interfaz del sistema de supervisión, las cuales fueron transmitidas a través del Gateway hacia el servidor correspondiente mediante el servicio MMS.

Al recibir estos comandos, los dispositivos procesaron las solicitudes y ejecutaron las acciones asociadas dentro de su lógica de control, produciendo los cambios de estado correspondientes. Este procedimiento permitió verificar la correcta implementación del flujo de control dentro de la arquitectura del laboratorio, desde el sistema de supervisión hasta los dispositivos de campo.

En la Figura 92 se puede observar la implementación del sistema completo. A la izquierda el dispositivo donde se ejecuta el Gateway y SCADA, en el medio el IED 1 y el IED AXON, y a la derecha la pantalla de Raspberry Pi ejecutando el IED con el tablero de comando.

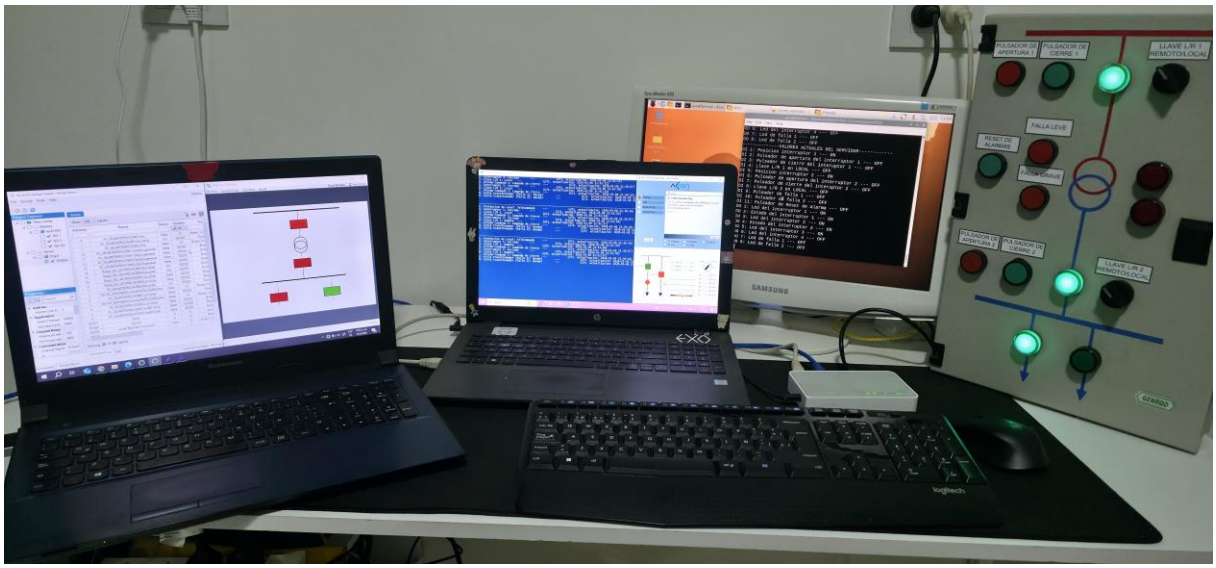


Figura 92: Implementación del sistema completo.

5.2. Evaluación de desempeño

Además de las pruebas funcionales presentadas en la sección anterior, se realizó una evaluación general del comportamiento del sistema implementado durante su operación en el laboratorio. Esta evaluación tuvo como objetivo analizar la estabilidad de la comunicación entre los dispositivos desarrollados, así como identificar ciertas limitaciones observadas durante el desarrollo del proyecto y la ejecución de las pruebas experimentales.

En términos generales, el sistema presentó un funcionamiento estable durante las pruebas realizadas. Los servidores IEC 61850 implementados en ambos dispositivos permitieron el intercambio de mensajes GOOSE y el acceso al modelo de datos mediante servicios MMS, mientras que la integración con el gateway de comunicaciones y el sistema SCADA permitió supervisar el estado de los interruptores representados en el laboratorio.

No obstante, durante el proceso de desarrollo e integración del sistema se identificaron algunas limitaciones y aspectos relevantes asociadas al comportamiento de la comunicación entre dispositivos, a las herramientas utilizadas para la configuración del modelo IEC 61850 y a determinados aspectos del diseño del hardware del laboratorio.

5.2.1. Comportamiento ante desconexión y reconexión de dispositivos

Durante las pruebas de funcionamiento se evaluó el comportamiento del sistema frente a la desconexión temporal de alguno de los dispositivos presentes en la red. En el caso de la comunicación entre los IED desarrollados y el gateway de comunicaciones, se observó que el sistema es capaz de restablecer automáticamente la comunicación una vez que el dispositivo

vuelve a conectarse a la red, sin requerir reinicio del sistema o intervención adicional del operador.

Sin embargo, se observó un comportamiento diferente en la comunicación entre el IED2 y el entorno de simulación implementado mediante el software Axon Bay. En esta configuración, el IED2 establece la conexión con el servidor MMS únicamente durante el proceso de inicialización del servidor. En caso de que la conexión se interrumpa posteriormente, el dispositivo no vuelve a consultar automáticamente el estado de la conexión ni intenta restablecerla.

Como consecuencia, si el servidor remoto se desconecta o se reinicia durante la operación del sistema, el IED2 no recupera automáticamente la recepción de los reportes MMS, siendo necesario reiniciar el servidor implementado en el dispositivo para restablecer la comunicación. Este comportamiento se debe a la lógica de implementación adoptada en el cliente MMS desarrollado para el dispositivo y no de una limitación inherente del estándar.

5.2.2. Limitaciones en la conexión MMS con el entorno de simulación

Durante la integración del IED2 con el entorno de simulación implementado en Axon Bay se observó una limitación relacionada con la gestión de clientes asociados a los reportes MMS. En particular, se verificó que el reporte configurado en el servidor no admite más de un cliente suscripto simultáneamente.

Como consecuencia, si el gateway de comunicaciones se conecta primero al reporte MMS configurado en Axon Bay, el IED2 no puede establecer posteriormente su propia conexión para recibir los reportes correspondientes al estado de los interruptores simulados. En estas condiciones, el dispositivo no logra suscribirse al reporte y, por lo tanto, no recibe las actualizaciones del estado de los interruptores.

Para evitar esta situación, durante las pruebas realizadas se estableció como procedimiento de inicialización iniciar primero el servidor implementado en el IED2 y posteriormente el gateway de comunicaciones. De esta manera se asegura que el dispositivo pueda establecer la suscripción al reporte antes de que otros clientes accedan al mismo recurso.

5.2.3. Conversión de archivos de configuración SCL

Durante el desarrollo del sistema se observó que la utilización de archivos de configuración SCL de tipo CID provenientes de herramientas comerciales, que contienen una gran cantidad de nodos lógicos y estructuras de datos asociadas, puede generar tiempos relativamente elevados durante el proceso de conversión hacia los archivos static_model.c y static_model.h.

Este proceso se realiza mediante las herramientas proporcionadas por la biblioteca libiec61850, las cuales generan una representación estática del modelo de datos que posteriormente es utilizada durante la compilación del servidor IEC 61850. Cuando el archivo de configuración contiene un número elevado de Logical Nodes, el tiempo requerido para generar los archivos correspondientes puede aumentar significativamente.

No obstante, esta situación no afecta el funcionamiento del sistema durante la ejecución, ya que la conversión del modelo se realiza únicamente durante la etapa de desarrollo. Una vez generados los archivos correspondientes al modelo estático, el proceso de compilación y ejecución del servidor no se ve significativamente afectado por el tamaño del archivo de configuración original.

5.2.4. Consideraciones en la alimentación de los módulos de relé

Durante la implementación del tablero de comando se observó que la alimentación directa de los módulos de relé de 5 V desde los pines GPIO de la Raspberry Pi puede generar condiciones de consumo elevadas durante el arranque del sistema.

Los módulos utilizados se activan mediante señales de nivel lógico bajo (LOW). Como consecuencia, durante la inicialización del sistema operativo los pines GPIO pueden permanecer momentáneamente en estado bajo, provocando la activación simultánea de los relés. Esta condición genera picos de consumo que pueden afectar la estabilidad de la alimentación del sistema.

En las primeras pruebas realizadas se observaron fallos asociados a la tarjeta microSD que contiene el sistema operativo de la Raspberry Pi, lo que obligó a reinstalar el sistema en nuevas memorias. Para evitar este inconveniente se optó por utilizar una fuente de alimentación externa de 5 V para los módulos de relé, manteniendo la Raspberry Pi únicamente como dispositivo de control de las señales de activación.

5.2.5. Comportamiento de las entradas digitales

Durante las pruebas realizadas con las entradas digitales del sistema se observó que el uso de resistencias de polarización con valores muy elevados, del orden de megaohmios, puede generar efectos indeseados debido a la captación de interferencias electromagnéticas presentes en el entorno.

En estas condiciones, los conductores asociados a las entradas pueden comportarse como antenas, induciendo pequeñas tensiones que pueden provocar activaciones erróneas de las entradas digitales o incluso dañar los circuitos asociados. Para evitar este efecto se recomienda utilizar resistencias de polarización con valores menores que permitan establecer una referencia

de potencial más estable y reduzcan la susceptibilidad del sistema frente a interferencias externas.

5.3. Análisis respecto a los objetivos propuestos

El objetivo general del proyecto consistió en investigar la implementación del estándar IEC 61850 en sistemas de automatización de subestaciones mediante el diseño e implementación de un laboratorio de simulación accesible y de bajo costo. Los resultados presentados en este capítulo demuestran que fue posible desarrollar una plataforma experimental capaz de reproducir los principales mecanismos de comunicación definidos por este estándar, permitiendo representar de forma simplificada el funcionamiento de una subestación digital en un entorno de laboratorio.

En relación con los objetivos específicos, se logró diseñar una arquitectura de comunicación basada en el estándar IEC 61850 que permite el intercambio de información entre dispositivos mediante servicios MMS y mensajes GOOSE. Asimismo, se implementaron dispositivos IED utilizando la biblioteca *libiec61850*, capaces de exponer modelos de datos compatibles con el estándar y establecer comunicación con otros dispositivos del sistema.

Por otra parte, la implementación de un IED sobre una Raspberry Pi permitió integrar el sistema de comunicación con señales físicas mediante el uso de pines GPIO, permitiendo la interacción con un tablero de control que representa estados e indicadores del sistema simulado.

La integración del laboratorio con un gateway de comunicaciones y una plataforma SCADA permitió además supervisar el estado de los dispositivos y visualizar la información del sistema en tiempo real, reproduciendo la estructura típica de los sistemas de automatización de subestaciones.

Las pruebas realizadas permitieron validar el correcto funcionamiento de los distintos componentes de la arquitectura implementada, confirmando el intercambio de mensajes entre dispositivos, la actualización del modelo de datos y la correcta representación de los estados del sistema tanto en el entorno SCADA como en el tablero físico.

En conjunto, los resultados obtenidos permiten afirmar que los objetivos planteados para el desarrollo del laboratorio fueron satisfactoriamente alcanzados, demostrando la viabilidad de implementar plataformas educativas basadas en el estándar IEC 61850 utilizando herramientas de software open-source y hardware de bajo costo.

6. Conclusiones

6.1. Conclusiones generales

El presente trabajo permitió desarrollar un laboratorio educativo basado en el estándar IEC 61850 orientado al estudio práctico de los sistemas de automatización de subestaciones. Mediante el uso de la biblioteca *libiec61850* y programación en lenguaje C se implementaron dos dispositivos electrónicos inteligentes capaces de intercambiar información mediante los servicios MMS y GOOSE, reproduciendo las principales funciones de comunicación definidas por el estándar.

La integración del sistema con un entorno de simulación, un sistema SCADA y un tablero físico de comando permitió validar experimentalmente el funcionamiento de la plataforma desarrollada, demostrando su capacidad para representar de manera simplificada el comportamiento de una red de automatización de subestaciones.

En este sentido, el laboratorio implementado constituye una herramienta útil para la enseñanza y experimentación con tecnologías IEC 61850, facilitando la comprensión de los modelos de datos y mecanismos de comunicación utilizados en sistemas eléctricos de potencia.

6.2. Propuesta para trabajos futuros

Una posible línea de trabajo consiste en el estudio e implementación del mecanismo de comunicación Sampled Values, utilizado softwares publicadores como *AT61 SVPublisher* presente en el paquete de AXON Group. Asimismo, incorporar mecanismos de sincronización horaria mediante protocolos como NTP o PTP, permitiendo mejorar la precisión temporal de los eventos registrados. A partir de estas señales también sería posible implementar funciones básicas de protección, como algoritmos de sobrecorriente programados dentro de los IED desarrollados. No obstante, debe considerarse que funciones de protección más complejas basadas en el procesamiento intensivo de señales y altas tasas de muestreo podrían superar las capacidades de procesamiento de plataformas de bajo costo como la Raspberry Pi.

Otra posible extensión del laboratorio consiste en la integración con IED comerciales, ya sea mediante el protocolo IEC 61850. Esto permitiría validar la interoperabilidad del sistema desarrollado y aproximar el laboratorio a escenarios más representativos de instalaciones reales.

Del mismo modo, podría profundizarse el proceso de ingeniería IEC 61850 mediante la utilización de archivos SSD y SCD, implementando un flujo de diseño Top-Down completo desde la especificación del sistema hasta la configuración de los IED.

7. Bibliografía

[1] B. A. Forouzan, Transmisión de datos y redes de comunicaciones, 2002.

[2] A. S. Tanenbaum, Redes de computadoras, 2003.

[3] G. Clarke y D. Reynders, Modern SCADA Protocols, 2004.

[4] K. P. Brand, V. Lohmann y W. Wimmer, Substation Automation Handbook, 2003.

[5] S. Kumar, A. Abu-Siada, N. Das y S. Islam, Review of the Legacy and Future of IEC 61850 Protocols Encompassing Substation Automation System, 2023.

[6] G. Suarez, Criterios de automatización de subestaciones con la norma IEC 61850, 2012.

[7] P. Bishop y N. K. C. Nair, IEC 61850: Principles and Applications to Electric Power Systems, 2022.

[8] A. Apostolov, IEC 61850: Digitizing the Electric Power Grid, 2022.

[9] H. Falk, IEC 61850 Demystified, 2018.

[10] Y. Yuan y Y. Yang, IEC 61850-Based Smart Substations: Principles, Testing, Operation and Maintenance, 2019.

[11] Axon Group. Documentación: https://axongroup.com.co/productos/.

[12] Libiec61850. Documentación: https://libiec61850.com/documentation/.

[13] J. A. L. Alvarado y M. E. V. Ceron, Diseño e implementación de un laboratorio virtual basado en una metodología para el estudio de pruebas de configuración de protocolos en equipos virtuales bajo el estándar IEC 61850, 2021.

[14] Raspberry Pi. Official Documentation: https://www.raspberrypi.com/documentation/.

8. Anexos

En esta sección se presentan los códigos diseñados en lenguaje C para compilar los archivos ejecutables en los lenguajes Linux y Windows correspondientemente. Durante el desarrollo del proyecto se especificó como utilizar los archivos Makefile de la librería y la obtención de los archivos static_model.c y static_model.h.

8.1. Código para Merging Unit

```
/*
================================================================================
    PROGRAMA SERVIDOR IEC 61850 CON SUSCRIPCIÓN A MENSAJES GOOSE Y CONTROL GPIO.
    Autor: ERICK EMANUEL CORTEZ
    Descripción general:
    - Este programa crea un servidor IEC 61850 con un modelo de datos de una merging unit
SEL 401.
    - Se suscribe a un mensaje GOOSE con comandos para dos interruptores que son emitidos
por otro dispositivo.
    - Controla los pines de la raspberry pi, definiendo las entradas y salidas.
    - Realiza un bucle cada 100 ms para verificar el estado de las entradas y actualizar
las salidas.
    - Lee las entradas digitales (DI) para actualizar el modelo de datos cargado.
    - Escribe las salidas digitales en funcion de los datos del modelo en cada periodo.
    - Tiene programada la logica de control de los equipos para verificar enclavamientos
con la llave L/R.

    Componentes principales:
    - gooseListener: función que procesa los mensajes GOOSE entrantes a la vez que se
ejecuta el codigo principal.
    - ActualizarServidor: funcion que actualizar los atributos del servidor con los datos
obtenidos por las entradas digitales. Cada señal de entrada tiene su logica en esta funcion
    - ObtenerSalida: funcion que actualiza las salidas digitales en base a los datos del
modelo iec61850
    - report_callback: funcion que se ejecuta en otro hilo cada vez que llega un reporte
MMS desde el servidor con los parametros seteados en el main y que actualiza los estados de
XCBR3 y XCBR4
    - main: inicializa el servidor, configura las suscripciones GOOSE y mantiene el
programa activo el bucle para interrogar y actualizar las entradas y salidas digitales.
================================================================================
*/

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <stdint.h>
// Bibliotecas específicas de libiec61850
#include "iec61850_common.h"
#include "goose_receiver.h"
#include "goose_subscriber.h"
#include "iec61850_server.h"
#include "hal_thread.h"
```

```
#include "mms_value.h"
#include "static_model.h"
#include "iec61850_client.h"
#include "mms_value.h"
// Bibliotecas para el manejo de pines
#include <wiringPi.h>
//================= VARIABLES GLOBALES =================//
static int running = 0;                         // Bandera de ejecución
static IedServer iedServer = NULL;              // Instancia del servidor IEC 61850
static unsigned int subgoose_stNumAnt;          // Almacena el número StNum anterior del
mensaje GOOSE
bool mostrar_SubGoose = true;                   // Muestra por consola los datos recibidos
bool permitir_comando = true;
bool cambio_detectado = false;

#define POS_ABIERTO 1
#define POS_CERRADO 2

//================= GESTIÓN DE INTERRUPCIÓN CTRL+C =================//
void sigint_handler(int signalId) {
    running = 0;  // Cuando se presiona Ctrl+C, se termina el bucle principal
}

//================= FUNCIÓN DE ESCUCHA GOOSE =================//
//    gooseListener: Esta función se llama automáticamente cuando se recibe un nuevo
mensaje GOOSE
//    - Verifica si hubo cambio en el número de estado (StNum).
//    - Compara los valores actuales con los anteriores.
//    - Si hay un cambio modifica el atributo correspondiente en el modelo.
//    - Esta preparada para un DataSet con 5 atributos booleanos.
void gooseListener(GooseSubscriber subscriber, void* parameter) {
    if (subgoose_stNumAnt != GooseSubscriber_getStNum(subscriber)) {
        // Obtener los valores del DataSet recibido
        MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
        if (values == NULL) {
            printf("Error: valores NULL en mensaje GOOSE\n");
            return;
        }
        // Mostrar por consola la información del mensaje GOOSE recibido
        char buffer[1024];
        MmsValue_printToBuffer(values, buffer, 1024);
        if (mostrar_SubGoose) {
            printf("----------------------------------------\n");
            printf(" GoCbRef: %s \n", GooseSubscriber_getGoCbRef(subscriber));
            printf(" DataSet: %s = %s\n", GooseSubscriber_getDataSet(subscriber), buffer);
        }
        uint64_t timestamp = GooseSubscriber_getTimestamp(subscriber);
        bool disparo =MmsValue_getBoolean(MmsValue_getElement(values,0));
        bool orden_SW1_Opn=MmsValue_getBoolean(MmsValue_getElement(values,1));
        bool orden_SW1_Cls=MmsValue_getBoolean(MmsValue_getElement(values,2));
        bool orden_SW2_Opn=MmsValue_getBoolean(MmsValue_getElement(values,3));
        bool orden_SW2_Cls=MmsValue_getBoolean(MmsValue_getElement(values,4));
        // Si el valor que llega por Goose es distinto al guardado en el servidor lo
actualiza.
        if (!MmsValue_equals(MmsValue_getElement(values,0),
IedServer_getAttributeValue(iedServer, IEDMODEL_PRO_PTRC1_Tr_general))){
            IedServer_updateBooleanAttributeValue(iedServer,
IEDMODEL_PRO_PTRC1_Tr_general,disparo);
```

```
            IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_PTRC1_Tr_t,timesta
mp);
            if (disparo){
                printf("Disparo por proteccion detectado\n");
                IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Pos_t,
timestamp);
                IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI1_Pos_stVal, POS_ABIERTO);
                IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Pos_t,
timestamp);
                IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI2_Pos_stVal, POS_ABIERTO);
                permitir_comando = false;
            } else {
                permitir_comando = true;
            }
        }
        // CONTROL SOBRE EL INTERRUPTOR 1.  COMANDO DE APERTURA
        if (!MmsValue_equals(MmsValue_getElement(values,1),
IedServer_getAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_OpOpn_general))) {
            if (IedServer_getBooleanAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Loc_stVal)
== false){   // Verificacion de la posicion de llave L/R del interruptor 1
                if
(orden_SW1_Opn==true){
                    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_PRO_CSWI1_Pos_t, timestamp);
                    IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI1_Pos_stVal, POS_ABIERTO);
                }
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpOpn_ge
neral,orden_SW1_Opn);
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpOpn_t,
timestamp);
            } else {
                if (orden_SW1_Opn==true){
                    printf("Comando de APERTURA REMOTO rechazado. LLave Local/Remoto del
interruptor 1 en LOCAL.\n");
                }
            }
        }
        //  CONTROL SOBRE EL INTERRUPTOR 1.   COMANDO DE CIERRE
        if (!MmsValue_equals(MmsValue_getElement(values,2),
IedServer_getAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_OpCls_general))){
            if (IedServer_getBooleanAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Loc_stVal)
== false && permitir_comando==true){   // Verificacion de la posicion de llave L/R del
interruptor 1
                if
(orden_SW1_Cls==true){
                    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_PRO_CSWI1_Pos_t, timestamp);
                    IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI1_Pos_stVal, POS_CERRADO);
                }
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpCls_ge
neral,orden_SW1_Cls);
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpCls_t,
timestamp);
            } else {
```

```
                if (orden_SW1_Cls==true){
                    printf("Comando de CIERRE rechazado. LLave Local/Remoto del interruptor
1 en LOCAL.\n");
                }
            }
        }
        // CONTROL SOBRE EL INTERRUPTOR 2.  COMANDO DE APERTURA
        if (!MmsValue_equals(MmsValue_getElement(values,3),
IedServer_getAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_OpOpn_general))) {
            if (IedServer_getBooleanAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Loc_stVal)
== false){   // Verificacion de la posicion de llave L/R del interruptor 2
                if
(orden_SW2_Opn==true){
                    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_PRO_CSWI2_Pos_t, timestamp);
                    IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI2_Pos_stVal, POS_ABIERTO);
                }
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpOpn_ge
neral,orden_SW2_Opn);
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpOpn_t,
timestamp);
            } else {
                if (orden_SW2_Opn==true){
                    printf("Comando de APERTURA rechazado. LLave Local/Remoto del
interruptor 2 en LOCAL.\n");
                }
            }
        }
        // CONTROL SOBRE EL INTERRUPTOR 2. COMANDO DE CIERRE
        if (!MmsValue_equals(MmsValue_getElement(values,4),
IedServer_getAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_OpCls_general))){
            if (IedServer_getBooleanAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Loc_stVal)
== false && permitir_comando==true){   // Verificacion de la posicion de llave L/R del
interruptor 2
                if
(orden_SW2_Cls==true){
                    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_PRO_CSWI2_Pos_t, timestamp);
                    IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI2_Pos_stVal, POS_CERRADO);
                }
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpCls_ge
neral,orden_SW2_Cls);
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpCls_t,
timestamp);
            } else {
                if (orden_SW2_Cls==true){
                    printf("Comando de CIERRE rechazado. LLave Local/Remoto del interruptor
2 en LOCAL.\n");
                }
            }
        }
        // Guardar el número de estado actual para futura comparación
        subgoose_stNumAnt = GooseSubscriber_getStNum(subscriber);
    }
}
```

```
//============== FUNCIONES PARA MANEJAR LAS ENTRADAS Y SALIDAS DE LA RASPBERRY ============
void ActualizarServidor(int valor,int posicion){
    uint64_t timestamp = Hal_getTimeInMs();
    if (posicion == 0){     // INTERRUPTOR 1
        uint32_t estado_actual = IedServer_getBitStringAttributeValue(iedServer,
IEDMODEL_PRO_XCBR1_Pos_stVal);
        if (valor == 0 && estado_actual != POS_ABIERTO) { // Interruptor abierto
            IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_XCBR1_Pos_stVal, POS_ABIERTO);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Pos_t,
timestamp);
            cambio_detectado = true;
        } else if (valor == 1 && estado_actual != POS_CERRADO) { // Interruptor cerrado
            IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_XCBR1_Pos_stVal, POS_CERRADO);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Pos_t,
timestamp);
            cambio_detectado = true;
        }
    }
    if (posicion == 1){     // PULSADOR DE APERTURA INTERRUPTOR 1
        if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpOpn_general)
!= valor && permitir_comando==true) {
            if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR1_Loc_stVal)
== true){
                cambio_detectado = true;
                if (valor==1){
                    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_PRO_CSWI1_Pos_t, timestamp);
                    IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI1_Pos_stVal, POS_ABIERTO);
                }
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpOpn_ge
neral,valor);
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpOpn_t,
timestamp);
            } else {
                if (valor==1){
                    printf("Comando de APERTURA LOCAL rechazado. LLave Local/Remoto del
interruptor 1 en REMOTO.\n");
                }
            }
        }
    }
    if (posicion == 2){     // PULSADOR DE CIERRE INTERRUPTOR 1
        if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpCls_general)
!= valor && permitir_comando==true) {
            if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR1_Loc_stVal)
== true){
                cambio_detectado = true;
                if (valor==1){
                    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_PRO_CSWI1_Pos_t, timestamp);
                    IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI1_Pos_stVal, POS_CERRADO);
                }
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpCls_ge
neral,valor);
```

```
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_OpCls_t,
timestamp);
            } else {
                if (valor==1){
                    printf("Comando de CIERRE LOCAL rechazado. LLave Local/Remoto del
interruptor 1 en REMOTO.\n");
                }
            }
        }
    }
    if (posicion == 3){     // LLAVE LOCAL/REMOTO INTERRUPTOR 1
            if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR1_Loc_stVal)
== false && valor == 1){
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR1_Loc_stVa
l,true);
                IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Loc_t,
timestamp);
                cambio_detectado = true;
            } else if
(IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR1_Loc_stVal) == true &&
valor == 0){
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR1_Loc_stVa
l,false);
                IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Loc_t,
timestamp);
                cambio_detectado = true;
            }
    }
    if (posicion == 4){     // INTERRUPTOR 2
        uint32_t estado_actual = IedServer_getBitStringAttributeValue(iedServer,
IEDMODEL_PRO_XCBR2_Pos_stVal);
        if (valor == 0 && estado_actual != POS_ABIERTO) { // Interruptor abierto
            IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_XCBR2_Pos_stVal, POS_ABIERTO);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Pos_t,
timestamp);
            cambio_detectado = true;
        } else if (valor == 1 && estado_actual != POS_CERRADO) { // Interruptor cerrado
            IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_XCBR2_Pos_stVal, POS_CERRADO);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Pos_t,
timestamp);
            cambio_detectado = true;
        }
    }
    if (posicion == 5){     // PULSADOR DE APERTURA INTERRUPTOR 2
        if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpOpn_general)
!= valor && permitir_comando==true) {
            if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR2_Loc_stVal)
== true){
                cambio_detectado = true;
                if (valor==1){
                    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_PRO_CSWI2_Pos_t, timestamp);
                    IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI2_Pos_stVal, POS_ABIERTO);
                }
```

```
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpOpn_ge
neral,valor);
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpOpn_t,
timestamp);
            } else {
                if (valor==1){
                    printf("Comando de APERTURA LOCAL rechazado. LLave Local/Remoto del
interruptor 2 en REMOTO.\n");
                }
            }
        }
    }
    if (posicion == 6){     // PULSADOR DE CIERRE INTERRUPTOR 2
        if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpCls_general)
!= valor && permitir_comando==true) {
            if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR2_Loc_stVal)
== true){
                cambio_detectado = true;
                if (valor==1){
                    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_PRO_CSWI2_Pos_t, timestamp);
                    IedServer_updateBitStringAttributeValue(iedServer,
IEDMODEL_PRO_CSWI2_Pos_stVal, POS_CERRADO);
                }
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpCls_ge
neral,valor);
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_OpCls_t,
timestamp);
            } else {
                if (valor==1){
                    printf("Comando de CIERRE LOCAL rechazado. LLave Local/Remoto del
interruptor 2 en REMOTO.\n");
                }
            }
        }
    }
    if (posicion == 7){     // LLAVE LOCAL/REMOTO INTERRUPTOR 2
        if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR2_Loc_stVal) ==
false && valor == 1){
            IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR2_Loc_stVal,tr
ue);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Loc_t,
timestamp);
            cambio_detectado = true;
        } else if
(IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR2_Loc_stVal) == true &&
valor == 0){
            IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR2_Loc_stVal,fa
lse);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Loc_t,
timestamp);
            cambio_detectado = true;
        }
    }
    if (posicion == 8){     // FALLA TIPO 1
        if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_ANN_GGIO1_Ind01_stVal) ==
false && valor == 1) {
```

```
            IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_ANN_GGIO1_Ind01_stVal,
true);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_GGIO1_Ind01_t,
timestamp);
            cambio_detectado = true;
        }
    }
    if (posicion == 9){     // FALLA TIPO 2
        if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_ANN_GGIO2_Ind01_stVal) ==
false && valor == 1) {
            IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_ANN_GGIO2_Ind01_stVal,
true);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_GGIO2_Ind01_t,
timestamp);
            cambio_detectado = true;
        }
    }
    if (posicion == 10){    // RESET DE LEDs
        if (valor == 1) {
            IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_ANN_GGIO1_Ind01_stVal,
false);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_GGIO1_Ind01_t,
timestamp);
            IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_ANN_GGIO2_Ind01_stVal,
false);
            IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_GGIO2_Ind01_t,
timestamp);
            cambio_detectado = true;
        }
    }
}

// ------ ASIGNAR VALOR A LAS SALIDAS EN FUNCION DEL VALOR DENTRO DEL MODELO --------
int ObtenerSalida(int posicion){
    switch (posicion)
    {
    case 0:     // INTERRUPTOR 1
        uint32_t interruptor1 =
IedServer_getBitStringAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_Pos_stVal);
        if (interruptor1 == POS_CERRADO)
            return 1;
        else
            return 0;
    case 1:
        uint32_t est_int1 =
IedServer_getBitStringAttributeValue(iedServer,IEDMODEL_PRO_CSWI1_Pos_stVal);
        if (est_int1 == POS_CERRADO)
            return 1;
        else
            return 0;
    case 2:     // INTERRUPTOR 2
        uint32_t interruptor2 =
IedServer_getBitStringAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_Pos_stVal);
        if (interruptor2 == POS_CERRADO)
            return 1;
        else
            return 0;
    case 3:
```

```
        uint32_t est_int2 =
IedServer_getBitStringAttributeValue(iedServer,IEDMODEL_PRO_CSWI2_Pos_stVal);
        if (est_int2 == POS_CERRADO)
            return 1;
        else
            return 0;
    case 4:     // INTERRUPTOR 3
        uint32_t interruptor3 =
IedServer_getBitStringAttributeValue(iedServer,IEDMODEL_PRO_CSWI3_Pos_stVal);
        if (interruptor3 == POS_CERRADO)
            return 1;
        else
            return 0;
    case 5:     // INTERRUPTOR 4
        uint32_t interruptor4 =
IedServer_getBitStringAttributeValue(iedServer,IEDMODEL_PRO_CSWI4_Pos_stVal);
        if (interruptor4 == POS_CERRADO)
            return 1;
        else
            return 0;
    case 6: return
IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_ANN_GGIO1_Ind01_stVal);
    case 7: return
IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_ANN_GGIO2_Ind01_stVal);
    default:
        break;
    }
}

//=========== CALLBACK PARA LOS ESTADOS DE INTERRUPTORES AXON ======================//
// Callback que se ejecuta automáticamente al recibir un nuevo reporte MMS del IED
void report_callback(void* parameter, ClientReport report)
{
    // Obtener los valores del DataSet recibido
    MmsValue* DataSetRecibido = ClientReport_getDataSetValues(report);
    int numElements = MmsValue_getArraySize(DataSetRecibido);
    uint64_t timestamp = Hal_getTimeInMs();

    // Recorrer cada elemento del DataSet recibido
    for (int i = 0; i < numElements; i++) {
        char valBuffer[500];
        const char* ref = ClientReport_getDataReference(report, i);
        MmsValue* valores = MmsValue_getElement(DataSetRecibido, i);
        // Validación de punteros
        if (valores == NULL || ref == NULL) {
            continue;
        }
        // Obtener el primer campo (stVal) dentro de la estructura Pos
        MmsValue* estado_actual = MmsValue_getElement(valores, 0);
        if (i== 0 ){
            bool estado = MmsValue_getBoolean(valores);
            printf("  %s = %d\n", ref, estado);
            if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR3_Loc_stVal)
!= estado){
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR3_Loc_stVa
l, estado);
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI3_Loc_stVa
l, estado);
```

```
            }
            if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR4_Loc_stVal)
!= estado){
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR4_Loc_stVa
l, estado);
                IedServer_updateBooleanAttributeValue(iedServer,IEDMODEL_PRO_CSWI4_Loc_stVa
l, estado);
            }
        }
        // Verificar y mostrar cambios en el interruptor 3
        if (i == 2){
            if
(!MmsValue_equals(IedServer_getAttributeValue(iedServer,IEDMODEL_PRO_XCBR3_Pos_stVal),
estado_actual)){
                IedServer_updateAttributeValue(iedServer,IEDMODEL_PRO_XCBR3_Pos_stVal,
estado_actual);
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_XCBR3_Pos_t,
timestamp);
                IedServer_updateAttributeValue(iedServer,IEDMODEL_PRO_CSWI3_Pos_stVal,
estado_actual);
                MmsValue_printToBuffer(estado_actual, valBuffer, 500);
                printf("  %s = %s\n", ref, valBuffer);
            }
        }
        // Verificar y mostrar cambios en el interruptor 4
        if (i == 3){
            if
(!MmsValue_equals(IedServer_getAttributeValue(iedServer,IEDMODEL_PRO_XCBR4_Pos_stVal),
estado_actual)){
                IedServer_updateAttributeValue(iedServer,IEDMODEL_PRO_XCBR4_Pos_stVal,
estado_actual);
                IedServer_updateUTCTimeAttributeValue(iedServer,IEDMODEL_PRO_XCBR4_Pos_t,
timestamp);
                IedServer_updateAttributeValue(iedServer,IEDMODEL_PRO_CSWI4_Pos_stVal,
estado_actual);
                MmsValue_printToBuffer(estado_actual, valBuffer, 500);
                printf("  %s = %s\n", ref, valBuffer);
            }
        }
    } //cambio_detectado=true;
}


//====================== FUNCIÓN PRINCIPAL ====================================//
int main(int argc, char** argv) {
    int tcpPort = 5111;  // 102 es el puerto TCP estándar para IEC 61850

    // Crear configuración del servidor IEC 61850 basado en el modelo estático
    IedServerConfig config = IedServerConfig_create();
    iedServer = IedServer_createWithConfig(&iedModel, NULL, config);
    IedServerConfig_destroy(config);

    // Determinar la interfaz de red a utilizar
    char* ethernetIfcID;
    if (argc > 1)
        ethernetIfcID = argv[1];  // Si se pasa como argumento
    else
        ethernetIfcID = "eth0";   // Por defecto, usar eth0
```

```
    printf("Utilizando interfaz GOOSE: %s\n", ethernetIfcID);
    IedServer_setGooseInterfaceId(iedServer, ethernetIfcID);

    // Iniciar el servidor IEC 61850
    IedServer_start(iedServer, tcpPort);
    printf("Server iniciado en el puerto %d\n", tcpPort);

    // Verificar que el servidor se haya iniciado correctamente
    if (!IedServer_isRunning(iedServer)) {
        printf("Inicializacion de servidor fallida! Saliendo.\n");
        IedServer_destroy(iedServer);
        exit(-1);
    }

    // Habilitar publicación de mensajes GOOSE propios
    IedServer_enableGoosePublishing(iedServer);

    // Crear e iniciar el receptor GOOSE
    GooseReceiver receiver = GooseReceiver_create();
    GooseReceiver_setInterfaceId(receiver, ethernetIfcID);

    // Crear el suscriptor al mensaje GOOSE (control de interruptores)
    GooseSubscriber sub_ctr_SW = GooseSubscriber_create("SEL487ECFG/LLN0$GO$GCB_comandos",
NULL);
    GooseSubscriber_setListener(sub_ctr_SW, gooseListener, "SW1");
    GooseReceiver_addSubscriber(receiver, sub_ctr_SW);

    // Iniciar el receptor
    GooseReceiver_start(receiver);
    if (!GooseReceiver_isRunning(receiver)) {
        printf("Ocurrio un error en la creacion de la instancia receiver\n");
        GooseReceiver_destroy(receiver);
        IedServer_destroy(iedServer);
        exit(-1);
    }

    // Configuración de conexión al IED AXON GROUP
    const char* ip_axon = "192.168.1.10";
    int port_axon = 102;
    IedClientError error_axon;
    IedConnection ied_axon = IedConnection_create();
    IedConnection_connect(ied_axon, &error_axon, ip_axon, port_axon);
    if (error_axon == IED_ERROR_OK) {
        // Referencia al Report Control Block
        const char* rcb_ref = "AXONBAYApplication/LLN0.BR.rcbControls";
        // Obtener los valores actuales del RCB desde el servidor
        ClientReportControlBlock rcb = IedConnection_getRCBValues(ied_axon, &error_axon,
rcb_ref, NULL);
        if (error_axon == IED_ERROR_OK) {
            // Configurar el RCB localmente antes de enviarlo al IED
            ClientReportControlBlock_setResv(rcb, true); // Reservar el RCB
            ClientReportControlBlock_setTrgOps(rcb, TRG_OPT_DATA_CHANGED); // Solo cambios
de datos
            ClientReportControlBlock_setDataSetReference(rcb,
"AXONBAYApplication/LLN0$BR$rcbControls"); // Nombre del DataSet
            ClientReportControlBlock_setRptEna(rcb, true); // Habilitar reportes
```

```
            ClientReportControlBlock_setGI(rcb, true); // Deshabilitar interrogación
general en esta etapa
            IedConnection_installReportHandler(ied_axon,rcb_ref,
                ClientReportControlBlock_getRptId(rcb),report_callback,NULL); // Registrar
el callback para reportes entrantes
            // Enviar al IED la configuración de habilitación de reportes
            IedConnection_setRCBValues(ied_axon, &error_axon, rcb, RCB_ELEMENT_RPT_ENA,
true);
            if (error_axon == IED_ERROR_OK) {
                printf(" Reportes habilitados correctamente\n");
                // Enviar una solicitud de interrogación general (GI) para obtener valores
actuales
                ClientReportControlBlock_setGI(rcb, true);
                IedConnection_setRCBValues(ied_axon, &error_axon, rcb, RCB_ELEMENT_GI,
true);}
            else {
                printf(" Error al habilitar los reportes: %d\n", error_axon);
            }
        }else {
            printf("getRCBValues service error!. Error %d\n",error_axon);}

    }else {
        printf(" Error al conectar con el IED AXON GROUP\n");
        IedConnection_destroy(ied_axon);
        //return -1;
    }

// Configuracion de pines GGIO de la raspberry
    wiringPiSetup();
    pinMode(12,INPUT);    // Posicion interruptor 1
    pinMode(13,INPUT);    // Pulsador de apertura del interruptor 1
    pinMode(14,INPUT);    // Pulsador de cierre del interruptor 1
    pinMode(26,INPUT);    // Llave Local/Remoto del interruptor 1
    pinMode(23,INPUT);    // Posicion interruptor 2
    pinMode(24,INPUT);    // Pulsador de apertura del interruptor 2
    pinMode(28,INPUT);    // Pulsador de cierre del interruptor 2
    pinMode(29,INPUT);    // Llave Local/Remoto 2
    pinMode(0,INPUT);     // Pulsador de falla 1
    pinMode(2,INPUT);     // Pulsador de falla 2
    pinMode(3,INPUT);     // Pulsador de Reset de alarma

    int valores_entrada[]={0,0,0,0,0,0,0,0,0,0,0};
    int PinesEntradasDigitales[]={12,13,14,26,23,24,28,29,0,2,3};
    char *CaracteresEntrada[]={"Posicion interruptor 1","Pulsador de apertura del
interruptor 1","Pulsador de cierre del interruptor 1","Llave L/R 1 en LOCAL",
        "Posicion interruptor 2","Pulsador de apertura del interruptor 2","Pulsador de
cierre del interruptor 2","Llave L/R 2 en LOCAL",
        "Pulsador de falla 1","Pulsador de falla 2","Pulsador de Reset de alarma"};

    pinMode(1,OUTPUT);    // Led del interruptor 1
    pinMode(4,OUTPUT);    // Estado del interruptor 1, se utiliza como DI
    pinMode(5,OUTPUT);    // Led del interruptor 2
    pinMode(6,OUTPUT);    // Estado del interruptor 2, se utiliza como DI
    pinMode(27,OUTPUT);   // Led del interruptor 3
    pinMode(25,OUTPUT);   // Led del interruptor 4
    pinMode(22,OUTPUT);   // Led de falla 1
    pinMode(21,OUTPUT);   // Led de falla 2
```

```
    int valores_salida[]={0,0,0,0,0,0,0,0};
    int PinesSalidasDigitales[]={1,4,5,6,27,25,22,21};
    char *CaracteresSalida[]={"Led del interruptor 1","Estado del interruptor 1","Led del
interruptor 2","Estado del interruptor 2",
        "Led del interruptor 3","Led del interruptor 4","Led de falla 1","Led de falla 2"};

    // Activar la interrupción por Ctrl+C
    running = 1;
    signal(SIGINT, sigint_handler);
    // Bucle principal: mantiene el programa activo
    while (running){
        Thread_sleep(100);  // el servidor descansa durante 100 ms para ahorrar recursos
        // ACTUALIZACION DEL SERVIDOR CON LAS SEÑALES DE ENTRADA
        for (int i = 0; i < sizeof(PinesEntradasDigitales)/sizeof(int); i++) {
            int entrada = digitalRead(PinesEntradasDigitales[i]);
            if (i==1 || i==5 || i==8 || i==9){
                entrada=!entrada;
            }
            valores_entrada[i]=entrada;
            ActualizarServidor(entrada, i);
        }
        // ACTUALIZACION DE LAS SEÑALES DE SALIDA CON LOS VALORES DEL SERVIDOR
        for (int j=0; j<sizeof(PinesSalidasDigitales)/sizeof(int) ; j++){
            int salida = ObtenerSalida(j);
            valores_salida[j]=salida;
            digitalWrite(PinesSalidasDigitales[j],!salida);
        }
        if (cambio_detectado){
            printf("--------------VALORES ACTUALES DEL SERVIDOR------------\n");
            for (int i = 0; i < sizeof(PinesEntradasDigitales)/sizeof(int); i++) {
                printf("DI %d: %s ---
%s\n",i+1,CaracteresEntrada[i],valores_entrada[i]?"ON":"OFF");
            }
            for (int j=0; j<sizeof(PinesSalidasDigitales)/sizeof(int) ; j++){
                printf("DO %d: %s --- %s
\n",j+1,CaracteresSalida[j],valores_salida[j]?"ON":"OFF");
            }
            cambio_detectado=false;
        }
    }
    // Finalización y limpieza de recursos
    GooseReceiver_stop(receiver);
    GooseReceiver_destroy(receiver);
    IedServer_stop(iedServer);
    IedServer_destroy(iedServer);
    if (ied_axon != NULL ) {
        IedConnection_destroy(ied_axon); }
    printf("El servidor se detuvo con exito.\n");
}
```

8.2. Código para Relé del transformador

```
/*
================================================================================
    PROGRAMA SERVIDOR IEC 61850 PARA PROTECCION Y CONTROL DE CAMPO DE TRANSFORMADOR
    Autor: ERICK EMANUEL CORTEZ

  Este servidor permite seleccionar la tarjeta de red utilizada seleccionando un valor
  numerico para el parametro de entrada.
  Permite interactuar con el mediante comandos escritos en la consola durante su ejecucion
  Crea un servidor IEC61850 en el puerto 5102 por defecto para poder ejecutar dos
servidores en la misma PC
  Posee suscripciones GOOSE indicadas en el codigo y publicaciones configuradas en el
archivo CID
  Puede establecer comunicacion cliente-servidor bajo el protocolo MMS recibir comandos a
los NL CSWI
================================================================================
*/
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdint.h>
#include <time.h>
#include "goose_receiver.h"
#include "goose_subscriber.h"
#include "iec61850_server.h"
#include "hal_thread.h"
#include "mms_value.h"
#include "goose_publisher.h"
#include "iec61850_common.h"
#include "static_model.h"

// ======================== VARIABLES GLOBALES ================================== //

static int running = false;                          // Estado general del bucle principal
static IedServer iedServer = NULL;                   // Instancia del servidor IEC 61850
char comando[100];                                   // Buffer de entrada de comandos desde
consola
int monitor[2] = {1, 0};                             // Control de monitoreo: [0] estados,
[1] mensajes GOOSE
#define POS_ABIERTO 1
#define POS_CERRADO 2

// ======================== MANEJO DE INTERRUPCIONES ============================== //

void sigint_handler(int signalId) {
    running = false;
}

// ======================== FUNCIONES DE ESTADO DE ATRIBUTOS ======================== //
const char* estadoInterruptor(IedServer servidor, const DataAttribute* atributo) {
    uint32_t valor = IedServer_getBitStringAttributeValue(servidor, atributo);
    switch (valor) {
        case 1: return "Cerrado";
        case 2: return "Abierto";
```

```
        case 0: return "Indeterminado";
        case 3: return "En falla";
        default: return "Valor invalido";
    }
}
const char* estadoComando(IedServer servidor, const DataAttribute* atributo) {
    uint32_t valor = IedServer_getBitStringAttributeValue(servidor, atributo);
    switch (valor) {
        case 1: return "Comando de apertura";
        case 2: return "Comando de cierre";
        case 0: return "Comando indeterminado";
        case 3: return "Comando no valido";
        default: return "Valor de comando invalido";
    }
}
const char* estadoLlaveDT(IedServer servidor, const DataAttribute* atributo) {
    return IedServer_getBooleanAttributeValue(servidor, atributo) ? "Local" :
"Telecomando";
}

const char* estadoLlaveLR(IedServer servidor, const DataAttribute* atributo) {
    return IedServer_getBooleanAttributeValue(servidor, atributo) ? "Local" : "Remoto";
}

const char* estadoFallaTransformador(IedServer servidor, const DataAttribute* atributo) {
    return IedServer_getBooleanAttributeValue(servidor, atributo) ? "Alarma" : "Normal";
}

// Imprime estado con estampa de tiempo legible
void imprimirEstadoConTimestamp(const char* descripcion, const char* estado, uint64_t
timestamp) {
    if (timestamp > 0) {
        time_t ts_seconds = timestamp / 1000;
        struct tm* tiempo = localtime(&ts_seconds);
        char buffer[64];
        strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", tiempo);
        printf("%s: %s\t---\t \t(Ult. actualizacion: %s)\n", descripcion, estado, buffer);
    } else {
        printf("%s: %s\t---\t \t(Ult. actualizacion: Sin estampa)\n", descripcion, estado);
    }
}

// ========================= ACTUALIZACION DE ESTADO EN CONSOLA ========================= //
void actualizacion_estado() {
    if (monitor[0] != 0) {
        printf("\n---------------------------------------------------------------------------
-----------------------------------\n");
        imprimirEstadoConTimestamp("1. Proteccion en Local", estadoLlaveDT(iedServer,
IEDMODEL_PRO_LLN0_Loc_stVal),
            IedServer_getUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_LLN0_Loc_t));
        imprimirEstadoConTimestamp("2. Interruptor 1", estadoInterruptor(iedServer,
IEDMODEL_PRO_XCBR1_Pos_stVal),
            IedServer_getUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Pos_t));
        imprimirEstadoConTimestamp("3. Llave L/R 1", estadoLlaveLR(iedServer,
IEDMODEL_PRO_XCBR1_Loc_stVal),
            IedServer_getUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Loc_t));
        imprimirEstadoConTimestamp("4. Ultimo Comando 1", estadoComando(iedServer,
IEDMODEL_PRO_CSWI1_Pos_stVal),
```

```
            IedServer_getUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Pos_t));
        imprimirEstadoConTimestamp("5. Interruptor 2", estadoInterruptor(iedServer,
IEDMODEL_PRO_XCBR2_Pos_stVal),
            IedServer_getUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Pos_t));
        imprimirEstadoConTimestamp("6. Llave L/R 2", estadoLlaveLR(iedServer,
IEDMODEL_PRO_XCBR2_Loc_stVal),
            IedServer_getUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Loc_t));
        imprimirEstadoConTimestamp("7. Ultimo Comando 2", estadoComando(iedServer,
IEDMODEL_PRO_CSWI2_Pos_stVal),
            IedServer_getUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Pos_t));
        imprimirEstadoConTimestamp("8. Transformador Alarma",
estadoFallaTransformador(iedServer, IEDMODEL_ANN_GGIO1_Ind01_stVal),
            IedServer_getUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_GGIO1_Ind01_t));
        imprimirEstadoConTimestamp("9. Transformador Falla",
estadoFallaTransformador(iedServer, IEDMODEL_ANN_GGIO2_Ind01_stVal),
            IedServer_getUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_GGIO2_Ind01_t));
        printf("\n");
    }
}

// ======================== FUNCION DE RETARDO PARA COMANDO ======================== //
typedef struct {
    IedServer iedServer;
    DataAttribute* atributo_general;
    DataAttribute* atributo_t;
} RetardoComandoArgs;

// Funcion que corre en un hilo independiente y desactiva el bit general tras 10s
static void* RetardoComandoThread(void* param) {
    RetardoComandoArgs* args = (RetardoComandoArgs*) param;
    uint64_t timestamp_inicio = Hal_getTimeInMs();
    IedServer_updateBooleanAttributeValue(args->iedServer, args->atributo_general, true);
    IedServer_updateUTCTimeAttributeValue(args->iedServer, args->atributo_t,
timestamp_inicio);
    Thread_sleep(10000);  // Espera de 10 segundos
    uint64_t timestamp_fin = Hal_getTimeInMs();
    IedServer_updateBooleanAttributeValue(args->iedServer, args->atributo_general, false);
    IedServer_updateUTCTimeAttributeValue(args->iedServer, args->atributo_t,
timestamp_fin);
    free(args); // Libera la memoria usada por los argumentos
    return NULL;
}

// Lanza un hilo que activa y luego desactiva el atributo general con estampas de tiempo
void RetardoComando(IedServer iedServer, DataAttribute* atributo_general, DataAttribute*
atributo_t) {
    RetardoComandoArgs* args = (RetardoComandoArgs*) malloc(sizeof(RetardoComandoArgs));
    args->iedServer = iedServer;
    args->atributo_general = atributo_general;
    args->atributo_t = atributo_t;

    Thread thread = Thread_create(RetardoComandoThread, args, false);
    Thread_start(thread);
}

// ====================== FUNCIONES DE COMANDO LOCAL Y LLAVES ======================== //
// Ejecuta un comando local desde la proteccion (apertura/cierre) si la llave D/T está en
"Distancia"
```

```
void comando_local(int paramet, char* orden) {
  uint64_t timestamp = Hal_getTimeInMs();

  if (paramet == 1) {
    printf("Accionando comando remoto para interruptor 1\n");
    if (strcmp(estadoLlaveDT(iedServer, IEDMODEL_PRO_CSWI1_Loc_stVal), "Local") == 0) {
      IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Pos_t,
timestamp);
      if (strcmp(orden, "cierre") == 0) {
        IedServer_updateBitStringAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Pos_stVal,
POS_CERRADO);
        RetardoComando(iedServer, IEDMODEL_PRO_CSWI1_OpCls_general,
IEDMODEL_PRO_CSWI1_OpCls_t);
      } else if (strcmp(orden, "apertura") == 0) {
        IedServer_updateBitStringAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Pos_stVal,
POS_ABIERTO);
        RetardoComando(iedServer, IEDMODEL_PRO_CSWI1_OpOpn_general,
IEDMODEL_PRO_CSWI1_OpOpn_t);
      }
    } else {
      printf("La operacion del interruptor 1 no esta permitida\n");
    }
  } else if (paramet == 2) {
    printf("Accionando comando remoto para interruptor 2\n");
    if (strcmp(estadoLlaveDT(iedServer, IEDMODEL_PRO_CSWI2_Loc_stVal), "Local") == 0) {
      IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Pos_t,
timestamp);
      if (strcmp(orden, "cierre") == 0) {
        IedServer_updateBitStringAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Pos_stVal,
POS_CERRADO);
        RetardoComando(iedServer, IEDMODEL_PRO_CSWI2_OpCls_general,
IEDMODEL_PRO_CSWI2_OpCls_t);
      }else if (strcmp(orden, "apertura") == 0) {
        IedServer_updateBitStringAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Pos_stVal,
POS_ABIERTO);
        RetardoComando(iedServer, IEDMODEL_PRO_CSWI2_OpOpn_general,
IEDMODEL_PRO_CSWI2_OpOpn_t);
      }
    }else {
      printf("La operacion del interruptor 2 no esta permitida\n");
    }
  }
  actualizacion_estado();
}

// Cambia el estado de la llave D/T de la proteccion
void llave_DT() {
    uint64_t timestamp = Hal_getTimeInMs();
    printf("Cambiando estado de la llave Distancia/Telecomando.\n");
    bool val = IedServer_getBooleanAttributeValue(iedServer, IEDMODEL_PRO_LLN0_Loc_stVal);
    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_LLN0_Loc_stVal, !val);
    IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_LLN0_Loc_t, timestamp);
    // Es el unico enclavamiento que debe verificar por lo tanto los bloqueos de operacion
se mueven en conjunto
    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Loc_stVal, !val);
    IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Loc_t, timestamp);
    IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Loc_stVal, !val);
    IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Loc_t, timestamp);
```

```
    actualizacion_estado();
}

// ================ INTERFAZ DE CONSOLA PARA OPERACION MANUAL ======================== //
// Procesa los comandos ingresados por teclado para accionar elementos o consultar estado
void comandos_consola() {
    if (fgets(comando, sizeof(comando), stdin) != NULL) {
        comando[strcspn(comando, "\n")] = '\0';  // Elimina el salto de línea
        printf("\n");
        if (strcmp(comando, "help") == 0) {
            printf("\n--------------------------------- Elige una opcion para continuar: --
------------------------------\n");
            printf("help -> Despliega el menu de comandos\n");
            printf("detener -> Detiene el funcionamiento del servidor\n");
            printf("protlocal -> Cambiar estado de la llave D/T de la proteccion\n");
            printf("1open -> Accionar apertura del interruptor 1\n");
            printf("1close -> Accionar cierre del interruptor 1\n");
            printf("2open -> Accionar apertura del interruptor 2\n");
            printf("2close -> Accionar cierre del interruptor 2\n");
            printf("monitoreo -> Activar/Desactivar monitoreo por consola [%s]\n",
monitor[0] ? "Activado" : "Desactivado");
            printf("subgoose -> Activar/Desactivar visualizacion de mensajes GOOSE [%s]\n",
monitor[1] ? "Activado" : "Desactivado");
            printf("actualizar -> Muestra por pantalla el estado actual de las señales\n");
            printf("-------------------------------------------------------------------------
----------------------\n\n");
        }
        else if (strcmp(comando, "detener") == 0) {
            printf("Deteniendo el servidor...\n");
            running = false;
        }
        else if (strcmp(comando, "1open") == 0) {
            comando_local(1, "apertura");
        }
        else if (strcmp(comando, "1close") == 0) {
            comando_local(1, "cierre");
        }
        else if (strcmp(comando, "2open") == 0) {
            comando_local(2, "apertura");
        }
        else if (strcmp(comando, "2close") == 0) {
            comando_local(2, "cierre");
        }
        else if (strcmp(comando, "protlocal") == 0) {
            llave_DT();
        }
        else if (strcmp(comando, "monitoreo") == 0) {
            monitor[0] = !monitor[0];
            printf("Mostrar estatus: '%s'\n", monitor[0] ? "Activado" : "Desactivado");
        }
        else if (strcmp(comando, "subgoose") == 0) {
            monitor[1] = !monitor[1];
            printf("Mostrar suscripciones GOOSE: '%s'\n", monitor[1] ? "ACTIVADO" :
"DESACTIVADO");
        }
        else if (strcmp(comando, "actualizar") == 0) {
            int aux = monitor[0];
            monitor[0] = 1;
```

```
            actualizacion_estado();
            monitor[0] = aux;
        }
        else {
            printf("Comando desconocido: '%s'\n", comando);
        }
    }
}


// ============== HANDLER PARA COMANDOS BINARIOS DESDE SCADA (MMS) ==================== //
// Esta función procesa las escrituras sobre atributos binarios desde un cliente MMS
static ControlHandlerResult controlHandlerForBinaryOutput(ControlAction action, void
*parameter, MmsValue *value, bool test)
{
  uint64_t timestamp = Hal_getTimeInMs();
  printf(" COMANDO RECIBIDO POR MMS:\n");
  ClientConnection clientCon = ControlAction_getClientConnection(action);
  if (clientCon)
    printf("Control from client %s\n", ClientConnection_getPeerAddress(clientCon));
  else
    printf("clientCon == NULL!\n");
  if (ControlAction_isSelect(action))
    printf("check handler called by select command!\n");
  else
    printf("check handler called by operate command!\n");
  printf("  ctlNum: %i\n", ControlAction_getCtlNum(action));
  if (parameter == IEDMODEL_PRO_CSWI1_Pos){       //Control del interruptor 1 por MMS
    if (strcmp(estadoLlaveDT(iedServer, IEDMODEL_PRO_CSWI1_Loc_stVal), "Telecomando") ==
0){
        IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Pos_t,
timestamp);
      if (MmsValue_getBoolean(value)==true) {                // Valor = 2 (bitstring "10")
para "cerrado"
        IedServer_updateBitStringAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Pos_stVal,
POS_CERRADO);
        RetardoComando(iedServer,
IEDMODEL_PRO_CSWI1_OpCls_general,IEDMODEL_PRO_CSWI1_OpCls_t);
      } else if (MmsValue_getBoolean(value)==false) {        // Valor = 1 (bitstring "01")
para "abierto"
        IedServer_updateBitStringAttributeValue(iedServer, IEDMODEL_PRO_CSWI1_Pos_stVal,
POS_ABIERTO);
        RetardoComando(iedServer,
IEDMODEL_PRO_CSWI1_OpOpn_general,IEDMODEL_PRO_CSWI1_OpOpn_t);
      }
    } else {
      printf("Proteccion en: %s\n", estadoLlaveDT(iedServer, IEDMODEL_PRO_LLN0_Loc_stVal));
      printf("Modifique el estado de la llave D/T para accionar desde la maestra.\n");
      return CONTROL_RESULT_FAILED;
    }
  }
  if (parameter == IEDMODEL_PRO_CSWI2_Pos){         //Control del interruptor 2 por MMS
    if (strcmp(estadoLlaveDT(iedServer, IEDMODEL_PRO_CSWI2_Loc_stVal), "Telecomando") ==
0){
      IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Pos_t,
timestamp);
      if (MmsValue_getBoolean(value)==true) {                // Valor = 2 (bitstring "10")
para "cerrado"
```

```
        IedServer_updateBitStringAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Pos_stVal,
POS_CERRADO);
        RetardoComando(iedServer,
IEDMODEL_PRO_CSWI2_OpCls_general,IEDMODEL_PRO_CSWI2_OpCls_t);
        uint64_t timestp = Hal_getTimeInMs();
      } else if (MmsValue_getBoolean(value)==false) {        // Valor = 1 (bitstring "01")
para "abierto"
        IedServer_updateBitStringAttributeValue(iedServer, IEDMODEL_PRO_CSWI2_Pos_stVal,
POS_ABIERTO);
        RetardoComando(iedServer,
IEDMODEL_PRO_CSWI2_OpOpn_general,IEDMODEL_PRO_CSWI2_OpOpn_t);
      }
    } else {
      printf("Proteccion en: %s\n", estadoLlaveDT(iedServer, IEDMODEL_PRO_LLN0_Loc_stVal));
      printf("Modifique el estado de la llave D/T para accionar desde la maestra");
      return CONTROL_RESULT_FAILED;
    }
  }
  actualizacion_estado();
  return CONTROL_RESULT_OK;
}
// ============ FIN DE COMANDOS BINARIOS DESDE SCADA (MMS) ============================== //

//////////////////  SUSCRIPCIONES GOOSE  ////////////////////////////////////////
//inicializacion de las suscripciones Goose que espera el servidor
static unsigned int gsub1_StNumAnterior ;
static unsigned int gsub2_StNumAnterior ;
static unsigned int gsub3_StNumAnterior ;
void gooseListener(GooseSubscriber subscriber, void* parameter) {
  char buffer[1024];
  uint64_t timestamp = Hal_getTimeInMs();
  if (parameter== "FALLAS") {
    if (gsub1_StNumAnterior != GooseSubscriber_getStNum(subscriber)) {
      MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
      if (values == NULL) {
        printf("Error: valores NULL en mensaje GOOSE\n");
        return;
      }
      //uint64_t timestamp = GooseSubscriber_getTimestamp(subscriber);
      MmsValue* falla_1=MmsValue_getElement(values,0);
      MmsValue* falla_2=MmsValue_getElement(values,1);
      // FALLA 1
      MmsValue* actual1 = IedServer_getAttributeValue(iedServer,
IEDMODEL_ANN_GGIO1_Ind01_stVal);
      if (!MmsValue_equals(actual1, falla_1)) {
          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_GGIO1_Ind01_t,
timestamp);
          IedServer_updateAttributeValue(iedServer, IEDMODEL_ANN_GGIO1_Ind01_stVal,
falla_1);
      }
      // FALLA 2
      MmsValue* actual2 = IedServer_getAttributeValue(iedServer,
IEDMODEL_ANN_GGIO2_Ind01_stVal);
      if (!MmsValue_equals(actual2, falla_2)) {
          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_ANN_GGIO2_Ind01_t,
timestamp);
          IedServer_updateAttributeValue(iedServer, IEDMODEL_ANN_GGIO2_Ind01_stVal,
falla_2);
```

```
      }
      if (IedServer_getBooleanAttributeValue(iedServer, IEDMODEL_ANN_GGIO2_Ind01_stVal)){
          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_PTRC1_Tr_t,
timestamp);
          IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_PTRC1_Tr_general,
true);
          printf("Se detecto una falla. Se enviara disparo a los interruptores 1 y 2.\n");
      } else {
          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_PTRC1_Tr_t,
timestamp);
          IedServer_updateBooleanAttributeValue(iedServer, IEDMODEL_PRO_PTRC1_Tr_general,
false);
      }
      gsub1_StNumAnterior = GooseSubscriber_getStNum(subscriber);
      actualizacion_estado();
    }
  }
  else if (parameter== "INTERRUPTORES") {
    if (gsub2_StNumAnterior != GooseSubscriber_getStNum(subscriber)) {
      MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
      if (values == NULL) {
        printf("Error: valores NULL en mensaje GOOSE\n");
        return;
      }
      //uint64_t timestamp = GooseSubscriber_getTimestamp(subscriber);
      MmsValue* interruptor_1=MmsValue_getElement(values,0);  //bit_string
      MmsValue* interruptor_2=MmsValue_getElement(values,1);  //bit_string
      MmsValue* llaveLR_1=MmsValue_getElement(values,2);      //boolean
      MmsValue* llaveLR_2=MmsValue_getElement(values,3);      //boolean
      // Interruptor 1
      MmsValue* actual1 = IedServer_getAttributeValue(iedServer,
IEDMODEL_PRO_XCBR1_Pos_stVal);
      if (!MmsValue_equals(actual1, interruptor_1)) {
          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Pos_t,
timestamp);
          IedServer_updateAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Pos_stVal,
interruptor_1);
      }
      // Interruptor 2
      MmsValue* actual2 = IedServer_getAttributeValue(iedServer,
IEDMODEL_PRO_XCBR2_Pos_stVal);
      if (!MmsValue_equals(actual2, interruptor_2)) {
          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Pos_t,
timestamp);
          IedServer_updateAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Pos_stVal,
interruptor_2);
      }
      // Llave LR 1
      MmsValue* actual3 = IedServer_getAttributeValue(iedServer,
IEDMODEL_PRO_XCBR1_Loc_stVal);
      if (!MmsValue_equals(actual3, llaveLR_1)) {
          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Loc_t,
timestamp);
          IedServer_updateAttributeValue(iedServer, IEDMODEL_PRO_XCBR1_Loc_stVal,
llaveLR_1);
      }
      // Llave LR 2
```

```
      MmsValue* actual4 = IedServer_getAttributeValue(iedServer,
IEDMODEL_PRO_XCBR2_Loc_stVal);
      if (!MmsValue_equals(actual4, llaveLR_2)) {
          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Loc_t,
timestamp);
          IedServer_updateAttributeValue(iedServer, IEDMODEL_PRO_XCBR2_Loc_stVal,
llaveLR_2);
      }
      gsub2_StNumAnterior = GooseSubscriber_getStNum(subscriber);
      actualizacion_estado();
    }
  }
  if (parameter== "COMANDOS") {
    if (gsub3_StNumAnterior != GooseSubscriber_getStNum(subscriber)) {
      MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
      if (values == NULL) {
        printf("Error: valores NULL en mensaje GOOSE\n");
        return;
      }
      MmsValue* Opn1=MmsValue_getElement(values,0);
      MmsValue* CLs1=MmsValue_getElement(values,1);
      MmsValue* Opn2=MmsValue_getElement(values,2);
      MmsValue* CLs2=MmsValue_getElement(values,3);
      if (IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR1_Loc_stVal)){
        if (MmsValue_getBoolean(Opn1)){
          printf("Comando de APERTURA para el interruptor 1 emitido desde el tablero\n");
        }if (MmsValue_getBoolean(CLs1)){
          printf("Comando de CIERRE para el interruptor 1 emitido desde el tablero\n");
        }
      }
      if
(IedServer_getBooleanAttributeValue(iedServer,IEDMODEL_PRO_XCBR1_Loc_stVal)){
        if (MmsValue_getBoolean(Opn2)){
          printf("Comando de APERTURA para el interruptor 2 emitido desde el tablero\n");
        }if (MmsValue_getBoolean(CLs2)){
          printf("Comando de CIERRE para el interruptor 2 emitido desde el tablero\n");
        }
      }
    gsub3_StNumAnterior = GooseSubscriber_getStNum(subscriber);
    }
  }
  if (monitor[1] != 0){
    MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);
    MmsValue_printToBuffer(values, buffer, 1024);
    printf(" ----  GoCbRef: %s  ----\n",GooseSubscriber_getGoCbRef(subscriber));
    printf("   DataSet: %s = %s\n", GooseSubscriber_getDataSet(subscriber),buffer);
    printf(" ---------------------------------------------\n");
  }
}

////////////////////// FIN DE SUSCRIPCIONES GOOSE  ////////////////////////////////////


//********************   FUNCION PRINCIPAL  ************************************//
int main(int argc, char** argv) {
  int tcpPort = 5102;  // Puerto a utilizar
//Creación del servidor a partir del archivo ICD usando el static_model
  IedServerConfig config = IedServerConfig_create();
```

```
  iedServer = IedServer_createWithConfig(&iedModel, NULL, config);
  IedServerConfig_destroy(config);

  //***   INICIO ENVIO DE DATOS GOOSE DE LOS GoCBs DEL MODELO IED   ***//
  // Se selecciona la interface para el envio de estos mjes GOOSE
  // El adaptador Ethernet suele estar direccionado con el valor 0, pero en algunos casos
puede cambiar
  // Se puede determinar el ID que debe utilizarse con AT61 Server, viendo la pestaña
Network Adapter
  // Al momento de crear un servidor. El ID coincide con el numero de la izquierda para
cada adaptador de red
  char* ethernetIfcID;
  if (argc > 1){
    ethernetIfcID = argv[1];}
  else{
    ethernetIfcID = "0";
  }
  // Define la interface para todos los GoCbs
  IedServer_setGooseInterfaceId(iedServer, ethernetIfcID);

//********* CONFIGURACION DE CONTROLADORES  *********//
  // Setea un controlador para los mjes MMS de un data object especifico
  IedServer_setControlHandler(iedServer, IEDMODEL_PRO_CSWI1_Pos,(ControlHandler)
controlHandlerForBinaryOutput,IEDMODEL_PRO_CSWI1_Pos);
  IedServer_setControlHandler(iedServer, IEDMODEL_PRO_CSWI2_Pos,(ControlHandler)
controlHandlerForBinaryOutput,IEDMODEL_PRO_CSWI2_Pos);

//*****   INICIO DEL SERVIDOR   ****//
  // El servidor MMS comenzara a recibir conexiones de clientes por el puerto 102
  IedServer_start(iedServer, tcpPort);
  printf("Server iniciado en el puerto %d\n",tcpPort);
  // Verifica la correcta ejecucion del servidor
  if (!IedServer_isRunning(iedServer)) {
    printf("Inicializacion de servidor fallida! Saliendo.\n");
    IedServer_destroy(iedServer);
    exit(-1);
  }
  // Habilita la publicacion de los GoCbs definidos en el archivo CID.
  IedServer_enableGoosePublishing(iedServer);

//******* INICIO CONFIGURACION PARAMETROS PARA SUSCRIPCION GOOSE  ******//
  // Crea una instancia que recibe todos los mjes GOOSE del servidor
  GooseReceiver receiver = GooseReceiver_create();
  // Selecciona el adaptador de red para recibir mensajes GOOSE
  GooseReceiver_setInterfaceId(receiver, ethernetIfcID);
  // Crea una nueva instancia (objeto) subscriber para recibir los mjes GOOSE relacionados
con el GoCbRef indicado y el AppId
  GooseSubscriber subscriber_falla = GooseSubscriber_create("SEL_401CFG/LLN0$GO$GCB_Falla",
NULL);
  GooseSubscriber subscriber_interruptor =
GooseSubscriber_create("SEL_401CFG/LLN0$GO$GCB_Estado", NULL);
  GooseSubscriber subscriber_comando =
GooseSubscriber_create("SEL_401CFG/LLN0$GO$GCB_Comando", NULL);
  // GooseSubscriber_setAppId(subscriber_falla, 1);  // Tiene que ser el mismo que se le
asigno en el servidor que envia el mje GOOSE
  /* Configuramos funcion que manejara los mensajes GOOSE. Cuando se
  recibe un mje en el objeto subscriber llama a la funcion gooseListener */
  GooseSubscriber_setListener(subscriber_falla, gooseListener, "FALLAS");
```

```
  GooseSubscriber_setListener(subscriber_interruptor, gooseListener, "INTERRUPTORES");
  GooseSubscriber_setListener(subscriber_comando, gooseListener, "COMANDOS");
  // Añade a la instancia receiver la instancia subscriber
  GooseReceiver_addSubscriber(receiver, subscriber_falla);
  GooseReceiver_addSubscriber(receiver, subscriber_interruptor);
  GooseReceiver_addSubscriber(receiver, subscriber_comando);
  // Inicia la recepcion de mensajes GOOSE en la instancia receiver
  GooseReceiver_start(receiver);
  if (!GooseReceiver_isRunning(receiver)) {
    printf("Ocurrio un error en la creacion de la instancia receiver\n");
    GooseReceiver_destroy(receiver);
    IedServer_destroy(iedServer);
    exit(-1);}

  running = true;
  signal(SIGINT, sigint_handler);

//*** INICIO DEL BUCLE ****//
  // Mientras la variable running se encuentre en 1
    printf("Introduzca un comando para interactuar con el servidor. \nPuede utilizar help
para desplegar el menu.\n");
  while (running) {
    comandos_consola();
  }
  // Cuando la variable running se pone en cero se detiene el bucle y se frena el
servidor
  GooseReceiver_stop(receiver);
  GooseReceiver_destroy(receiver);
  IedServer_stop(iedServer);
  IedServer_destroy(iedServer);
  printf("El servidor se detuvo con exito.\n");
}
//.....................................................................//
```