UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y COMUNICACIONES

TEMA:

"INTEGRACIÓN DE LA CAPA DE COMUNICACIÓN DE LA NORMA IEC-61499 BASADA EN LOS PROTOCOLOS DE LA INDUSTRIA 4.0"

Trabajo de Titulación. Modalidad: Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Electrónica y Comunicaciones.

SUBLÍNEA DE INVESTIGACIÓN: Sistemas de Control

AUTOR: Xiomara Alejandra Cárdenas Medina

TUTOR: Ing. Geovanni Danilo Brito Moncayo

AMBATO – ECUADOR

SEPTIEMBRE 2020

APROBACIÓN DEL TUTOR

En calidad de tutor de Trabajo de Investigación sobre el tema: INTEGRACIÓN DE LA CAPA DE COMUNICACIÓN DE LA NORMA IEC-61499 BASADA EN LOS PROTOCOLOS DE LA INDUSTRIA 4.0, elaborado por la señorita Cárdenas Medina Xiomara Alejandra, estudiante de la Carrera de Ingeniería en Electrónica y Comunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 15 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y el numeral 7.4 del respectivo instructivo.

Ambato Septiembre,2020

EL TUTOR

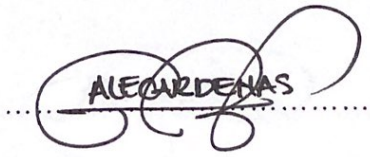
Firmado electrónicamente por:
GEOVANNI DANILO BRITO MONCAYO

Ing. Geovanni Brito

AUTORÍA

El presente Proyecto de Investigación titulado: INTEGRACIÓN DE LA CAPA DE COMUNICACIÓN DE LA NORMA IEC-61499 BASADA EN LOS PROTOCOLOS DE LA INDUSTRIA 4.0, es absolutamente original, autentico y personal. En tal virtud, los contenidos académicos e instrumentos legales que se manifiesten del mismo son de exclusiva responsabilidad del autor.

Ambato Septiembre, 2020



Xiomara Alejandra Cárdenas Medina

CC: 1804725057

AUTOR

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del Informe Final del Trabajo de Titulación presentado por la señorita Xiomara Alejandra Cárdenas Medina, estudiante de la Carrera de Ingeniería en Electrónica y Comunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado "INTEGRACIÓN DE LA CAPA DE COMUNICACIÓN DE LA NORMA IEC-61499 BASADA EN LOS PROTOCOLOS DE LA INDUSTRIA 4.0", nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 17 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y al numeral 7.6 del respectivo instructivo. Para cuya constancia suscribimos, conjuntamente con la señora Presidente del Tribunal.

Firmado electrónicamente por:
ELSA PILAR URRUTIA

……

Ing. Elsa Pilar Urrutia U Mg.

PRESIDENTA DEL TRIBUNAL

Firmado electrónicamente por:
FRANKLIN WILFRIDO SALAZAR LOGRONO

……

Ing. Franklin Salazar Mg.

DOCENTE CALIFICADOR

Firmado electrónicamente por:
MARCELO VLADIMIR GARCIA SANCHEZ

……

Dr. Marcelo García

DOCENTE CALIFICADOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que haga uso de este Trabajo de Titulación como un documento disponible para la lectura, consulta y procesos de investigación.

Cedo los derechos de mi Trabajo de Titulación en favor de la Universidad Técnica de Ambato, con fines de difusión pública. Además, autorizo su reproducción total o parcial dentro de las regulaciones de la institución.

Ambato Septiembre, 2020

Xiomara Alejandra Cárdenas Medina

CC: 1804725057

AUTOR

DEDICATORIA

A quien está en lo alto e inmenso universo, Dios que a pesar de que no lo vea, yo lo siento: en la brisa que roza mi rostro, en las mañanas deslumbrantes, en las noches estrelladas, cuando el sol brilla en mi mirada, al caer la lluvia sobre mí, ahí es cuando yo lo siento y sé que cada día que me brinda es una oportunidad de continuar el camino de la vida y seguir cumpliendo cada uno de mis propósitos.

Y porque desde que nací creyeron en mí, dedico cada uno de mis logros a las mujeres más fuertes y valientes de mi vida; a mi madre y a mi abuelita quienes a pesar de las adversidades no dudaron ni un segundo en darme su apoyo incondicional. Todo lo que soy ahora es gracias a su esfuerzo, ¡las amo!

Con nostalgia dedico este trabajo a mi mascota NINO, aquel perro que sin poder decirme nada me enseño tanto, su amor incondicional, su lealtad, su amistad y su compañía inspiraron cada uno de mis días e hicieron de mí una persona feliz, libre y agradecida de aquello que me aporta la vida. Fue, es y será siempre el mejor amigo que tuve desde muy pequeña. Estará presente como mi ángel de la guarda.

Alejandra Cárdenas Medina

AGRADECIMIENTO

Gracias a Dios por mi familia y por las bendiciones de todos los días, por darme la fuerza para no desistir, aunque muchas veces los obstáculos fueron muy grandes, por regalarle luz a mi camino y por este objetivo cumplido.

A mi madre Betty por forjarme como una mujer fuerte que sin importar a donde vaya jamás debo olvidar todos los valores que me enseñó. A mi abue Elsa por ser mi otra madre, porque en cada paso que di estabas y estas junto a mí, preocupada siempre por mi salud y con sus remedios caseros curarme hasta el alma. Gracias a mis madres por sus sabios consejos.

La compañía de mi hermana Camila hacen de mis días divertidos y felices además que sin importar nuestros desacuerdos también está pendiente y preocupada por mí. No podría olvidar el apoyo incondicional de mi tío Rolando Cárdenas, gracias.

Agradezco infinitamente a mi alma mater que brindo sus aulas y laboratorios para que docentes puedan compartirme sus conocimientos. Y de manera especial a mi tutor Ing. Geovanni Brito por su apoyo y orientarme en el desarrollo de este proyecto. Además, al Dr. Marcelo García por creer en mi potencial para lograr este reto. Gracias por sus enseñanzas y su amistad.

Aquellas personas que puedo llamar amigos quienes con paciencia fueron mis otros maestros sin importar los motivos me prestaron su tiempo para compartir y contribuir en mi aprendizaje.

A mis amigos "Los Gatos" quienes me abrieron las puertas de sus hogares y de sus corazones e hicieron de esta etapa una aventura, sin importar nuestras diferencias fueron un apoyo para mí.

¡GRACIAS TOTALES!

Alejandra Cárdenas Medina

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Los antiguos estándares de la industria no contemplan las necesidades de las grandes y pequeñas empresas, ya que requieren integrar redes de comunicaciones entre sistemas distribuidos. Con la Cuarta Revolución Industrial, se incorpora la Norma IEC-61499, la cual aporta una arquitectura de comunicación estándar para sistemas de control, esta arquitectura limita la integración de nuevas tecnologías y plataformas de la IIOT (Internet Industrial de las Cosas) para el monitoreo y ejecución de procesos físicos multidisciplinarios. Por lo cual, este proyecto propone desarrollar una arquitectura de comunicación reconfigurable, la cual será utilizada por los sistemas descentralizados para comunicarse entre ellos.

Entonces, se desarrolló una capa de comunicación con el 95 por ciento de eficiencia y sirve como guía para añadir diferentes tipos de protocolos de la Industria 4.0 así como, código reutilizable para otros desarrolladores. La arquitectura de comunicación incluye las librerías necesarias para acceder al control del PLC y las que encapsulan mensajes con protocolo MQTT, de esta forma, se pueda gestionar la recepción y él envió de datos hacia el Bróker de Mosquitto. Además, se realizó un algoritmo de control para el funcionamiento de la Estación Festo MPS-200-Clasificador, permitiendo comprobar la hipótesis de investigación, puesto que, el proceso permite que un cliente se conecte al Bróker con la finalidad suscribir y publicar mensajes para monitorizar el estado de sus sensores y actuadores. Para el diseño del sistema de control y de comunicación se utilizó los modelos de Bloques de Funciones del entorno de programación 4DIAC-IDE y el runtime FORTE de 4DIAC, adicionalmente se creó el módulo de comunicación en una Raspberry Pi en lenguaje de programación C++ compatible con el software de diseño y runtime.

Palabras Claves: SNAP7, MQTT, Mosquitto, Internet Industrial de las Cosas (IIOT).

ABSTRACT

Old industry standards do not address the needs of large and small businesses, as they require the integration of communications networks between distributed systems. With the Fourth Industrial Revolution, the IEC-61499 Standard is incorporated, which provides a standard communication architecture for control systems. This architecture limits the integration of new technologies and platforms of the IIOT (Industrial Internet of Things) for the monitoring and execution of multidisciplinary physical processes. Therefore, this project proposes to develop a reconfigurable communication architecture, which will be used by the decentralized systems to communicate with each other.

Then, a communication layer was developed with 95 percent efficiency and serves as a guide to add different types of Industry 4.0 protocols as well as, reusable code for other developers. The communication architecture includes the necessary libraries to access the PLC control and those that encapsulate messages with MQTT protocol, in this way, it is possible to manage the reception and sending of data to the Mosquitto Broker. Also, a control algorithm was made for the operation of the Festo MPS-200-Classifier Station, allowing to check the research hypothesis, since, the process allows a client to connect to the Broker with the purpose of subscribing and publishing messages to monitor the state of its sensors and actuators. For the design of the control and communication system, the 4DIAC-IDE programming environment function block models and the 4DIAC FORTE runtime were used. In addition, the communication module was created on a Raspberry Pi in C++ programming language compatible with the design and runtime software.

Keywords: SNAP7, MQTT, Mosquito, Industrial Internet of Things (IIOT).

CAPÍTULO I

1 MARCO TEÓRICO

1.1 Antecedentes Investigativos

La Norma IEC-61499 ofrece un amplio ámbito para el desarrollo de aplicaciones para el control y automatización de sistemas distribuidos. Estas aplicaciones son herramientas de software que se pueden desarrollar y establecer con módulos reconfigurables, es decir que, cualquier dispositivo y sus componentes pueden ser configurados por múltiples fabricantes. Para lo cual, el desarrollo de este proyecto requiere de estudios realizados con anterioridad sobre temas afines a la propuesta, por lo tanto, para respaldar la investigación se tienen los siguientes trabajos:

Actualmente existen múltiples investigaciones sobre la nueva estandarización de la Norma IEC-61499 para el diseño e implementación de control de procesos industriales mediante Bloques de Funciones. Puesto que, la Norma IEC-61131 está mostrando muchas limitaciones, por tal motivo se crean varios enfoques en la ingeniería de software y sistemas con el objetivo de abordar los nuevos desafíos mediante la Norma IEC-61499 que ha sido desarrollada para abordar el campo de la automatización.

Así pues, algunas de estas investigaciones se enfocan en la implementación de sistemas distribuidos empleando los requisitos que tiene la Norma, debido a la versatilidad en el diseño, puesto que, combina software y hardware independientemente. El artículo [1] describe una aplicación mediante Bloques de Funciones para controlar el funcionamiento de sistemas distribuidos en dispositivos de bajo costo, la aplicación realiza el control de la estación MPS-500 y de esta forma se verifica los alcances para automatizar y modificar los parámetros de la planta. A su vez permite que el sistema sea reconfigurable y escalable. La Norma IEC-61499 permite que los sistemas distribuidos sean escalables debido a sus características de portabilidad e interoperabilidad, pero también cuenta con el inconveniente de

especificaciones exhaustivas para la ejecución de los Bloques de Funciones, por lo cual, [2] crea una semántica sincrónica llamado “código de Esterel” que permite acelerar la ejecución de los Bloques de Funciones con la ventaja de utilizar el lenguaje de programación C+. Es de gran ayuda que este tipo de investigaciones utilicen diferentes lenguajes de programación debido a que los próximos desarrolladores pueden mejorar cada vez más el desarrollo de aplicaciones basadas en la Norma IEC-61499.

También se tiene numerosos retos en cuanto al diseño de modelos de control que permitan manejar sistemas de producción industrial, que a su vez van aumentado su complejidad mientras pasa el tiempo. En este sentido, para sistemas de automatización distribuida con procesos físicos integrados son necesarias metodologías de diseño basadas en técnicas de modelado como las que soporta la Norma IEC-61499.

Por lo cual, [3] utiliza una nueva metodología para el desarrollo de aplicaciones de intercambio de información entre el nivel de planta y las capas de suscripción. A más de, integrar protocolos de redes industriales que permiten el acceso a los datos de proceso y también se emplea una comunicación M2M (Machine-to-Machine) que implementa servidores y clientes para la adquisición de información de la planta de forma sencilla y eficiente en diferentes dispositivos de bajo costo. Otro modelo de desarrollo para el diseño e implementación es una capa de inteligencia mecatrónica utilizando una combinación de sensor/actuador inalámbrico con redes de arquitectura orientada a servicios, como describe [4], donde los servicios son localizadas a nivel de dispositivo, así como en las Nubes locales y globales. Estas metodologías utilizadas en estas investigaciones orientan al desarrollo de nuevos modelos de aplicaciones para controlar diversos procesos de automatización.

Por otro lado, el uso de Bloques Funcionales permite la creación de programas más flexibles y reutilizables, ya que se entiende el Bloque de Función como una unidad de software en la que se encapsulan datos y código, además, cuentan con entradas y salidas. El diseño de Bloques de Funciones de [5] se utiliza para el manejo de entradas y salidas de datos en el control de selección, almacenamiento y clasificación de la maqueta Festo FMS-200. Los FB´s contienen algoritmos de control avanzado permitiendo obtener una arquitectura distribuida, lo cual permite utilizar diferentes dispositivos para controlar el proceso.

De igual forma la Norma IEC-61499 permite utilizar varias herramientas para el desarrollo aplicaciones de control pero, el entorno 4DIAC y su plataforma FORTE son ideales para el diseño de un sistema distribuido basado en la programación mediante FBs con el objetivo de automatizar y manipular cualquier tipo de proceso industrial, [6] realiza un algoritmo mediante eventos para el control del robot Kuka YouBot, permitiendo controlar y manipular de forma independiente la plataforma y el brazo que conforma el robot. Adicionalmente, mediante algoritmos creados en los Bloques de Funciones se puede realizar lecturas de datos proporcionados por sensores analógicos, [7] implementa un sistema de control industrial para el proceso de nivel de la estación MPS-PA, logrando alcanzar los valores analógicos deseados para la prueba de funcionamiento y de esta forma verificando así la factibilidad de utilizar Bloques de Funciones basados en la Norma IEC-61499.

En compendio, las investigaciones descritas anteriormente mencionan diferentes formas de control para los sistemas distribuidos mediante la Norma IEC-61499, delimitando la utilidad del estándar en la parte de comunicación, puesto que, estos proyectos de investigación utilizan la misma arquitectura de conexión. Por lo cual, se propone crear un nuevo módulo de comunicación, así de esta forma abrir una gama de posibilidades para que cualquier desarrollador implemente varios tipos de conexión con los sistemas de control. Para validar esta propuesta se implementa de forma adicional una aplicación que permite evidenciar el control de la Estación Festo MPS-200-Clasificador.

1.2 Contextualización del problema

Numerosas normas se han desarrollado a lo largo de los años para estandarizar las actividades industriales, económicas o científicas, con el fin de obtener un óptimo ordenamiento de especificaciones técnicas, definiciones o características de productos, materiales, servicios y procesos. La Comisión Electrónica Internacional (IEC), es una organización que abarca los campos: eléctrico, electrónico y tecnologías relacionadas; 83 países alrededor del mundo son parte de esta organización para establecer

condiciones de interoperabilidad a sistemas complejos, contribuir con el ambiente, implementar la calidad de un producto y los servicios mediante sus normas, entre otras [8].

Por lo cual, el Estándar IEC-61131 durante varios años ha servido como guía para los sistemas de automatización industrial ya que fue el primer paso en la normalización de los autómatas programables con respecto a lenguajes de programación y aplicaciones de control. Pero hoy en día el estándar IEC-61131 ya no se ajusta a las nuevas necesidades de manejar sistemas distribuidos de medición y control, debido a que estos procesos van evolucionando con el tiempo y las nuevas tecnologías. Además, los sistemas distribuidos necesitan tener una conectividad online entre sí y los procesos que se están ejecutando en las plantas industriales [9].

Actualmente se experimenta la Cuarta Revolución Industrial con respecto a los sistemas de automatización y control, que integran redes de comunicaciones para el monitoreo y ejecución de procesos físicos multidisciplinarios que deben ser accesibles, de respuesta rápida y eficiente a los cambios de entorno de trabajo. Para lo cual, la Norma IEC-61499 contempla la mayor parte de las necesidades de esta gestión aportando con una arquitectura y requisitos de herramientas de software para sistemas de control distribuidos [9].

Por lo tanto, la Norma IEC-61499 aporta con una guía para la utilización de Bloques de Funciones que permiten modularidad, inteligencia distribuida y control flexible en procesos industriales. Por lo tanto, los Bloques de Funciones se integra a la programación de los sistemas de automatización, ya que la puesta en marcha de los mismos se realiza mediante eventos, lo que significa una especificación del orden correcto de ejecución de estos bloques. Además, los requisitos de herramientas de software deben soportar la ejecución múlti - tareas [10].

El proyecto se centra en desarrollar y validar una capa de conexión utilizando el modelo de arquitectura de comunicación de la Norma IEC-61499. Así pues, la Norma brinda modelos de Bloques de Funciones los cuales permiten encapsular y abstraer al usuario el acceso al hardware o a los recursos de interfaz de programación, de esta forma se facilita la recolección y procesamiento de datos entre sensores y actuadores. Se pretende entregar una aplicación que gestiona la información de entrada y salida de

datos para el control de la Estación Festo MPS-200-Clasificador a través de la nueva capa de comunicación.

Uno de los beneficios primordiales que provee el proyecto, es reconfigurar la comunicación que tienen los sistemas industriales distribuidos, esto quiere decir que en la nueva arquitectura de comunicación se podrán utilizar variedad de protocolos de Industria 4.0. De esta forma el personal de ingeniería de las empresas industriales podrá aportar con ideas para mejorar o actualizar las configuraciones de los sistemas distribuidos. Ya que tanto las empresas públicas y privadas requieren implementar sistemas industriales distribuidos bajo nuevas estandarizaciones que solicitan los avances tecnológicos.

1.3 Fundamentación Teórica

1.3.1 Norma IEC-61499

La Norma IEC-61499 se desarrolló por la creciente demanda de nuevas tecnologías que crean arquitecturas más complejas en los sistemas industriales. Con la integración de estos estándares a lo largo del tiempo la industria ya tiene nuevas opciones para automatizar diversos procesos y de esta forma crear sistemas más flexibles, puesto que el software que se utilizará ya no estará limitado ni bloqueado por los proveedores, adicionalmente se reducirán en gran medida los costos en las implementaciones de los procesos industriales [1], [11].

Uno de los objetivos principales de esta Norma es desarrollar sistemas heterogéneos compuestos de dispositivos de control de cualquier fabricante, desde cierta perspectiva la Norma sirve para diseñar aplicaciones de control basadas en un concepto de Bloque Funcional donde la cuestión fundamental es la importancia del software ya que el código puede ser reutilizado en dos o más máquinas que realicen el mismo proceso. Por otro lado, facilita la recopilación y el procesamiento de información provenientes de sensores o de actuadores, y a su vez optimiza los métodos de mantenimiento de equipos e incorpora nuevas técnicas de control en la planta [12], [13].

Características de la Norma IEC-61499

Las herramientas de software de esta norma tienen como fin proveer un entorno abierto para la automatización y control de los sistemas distribuidos por lo cual, la Tabla 1 muestra un análisis de las principales características que identifican y respaldan a la Norma IEC-61499 con el objetivo de cumplir con cada una de las que se mencionan a continuación [14], [15]:

Tabla 1: *Descripción de las características de la Norma IEC-61499.* [14],[15].

Característica	Descripción
Portabilidad	Significa que puede soportar e interpretar componentes de software de otras herramientas del sistema.
Reconfigurabilidad	Se basa en la adaptación de componentes de software y hardware durante la operación del proceso.
Interoperabilidad	Es la funcionalidad de varios dispositivos en conjunto para que se ejecuten funciones propias de las aplicaciones distribuidas.
Configurabilidad	Puede ser configurado diversos dispositivos con sus componentes de software por medio de otras herramientas de varios fabricantes.
Seguridad	La comunicación entre controladores y con aplicaciones externas son importantes en un sistema de control y la IEC 61499 contempla el hecho, evitando riesgos en personas y equipamientos.
Fiabilidad	Es fiable relevantemente en el hardware, pero también en el software se tiene en cuenta al momento de la verificación y validación formal de los Bloques de Funciones.
Transparencia	Se toma en cuenta la transparencia de acceso lo que permite acceder a recursos locales y remotos de forma idéntica.
Distribución	Quiere decir que distribuye componentes de software para dispositivos de hardware de distintos fabricantes.

Rendimiento Y Escalabilidad	Quiere decir que se analizan prestaciones y escalabilidad con respecto a los protocolos de comunicación del Estándar. Y en tanto al rendimiento quiere decir que se debe cumplir con las restricciones del tiempo real o el tiempo de ejecución en cada procesos a realizarse.

Elaborado por: *La Investigadora*

La Figura 1 ilustra de forma esquemática como se desempeñan las características principales de la Norma IEC-61499 en el entorno de trabajo de los sistemas descentralizados:

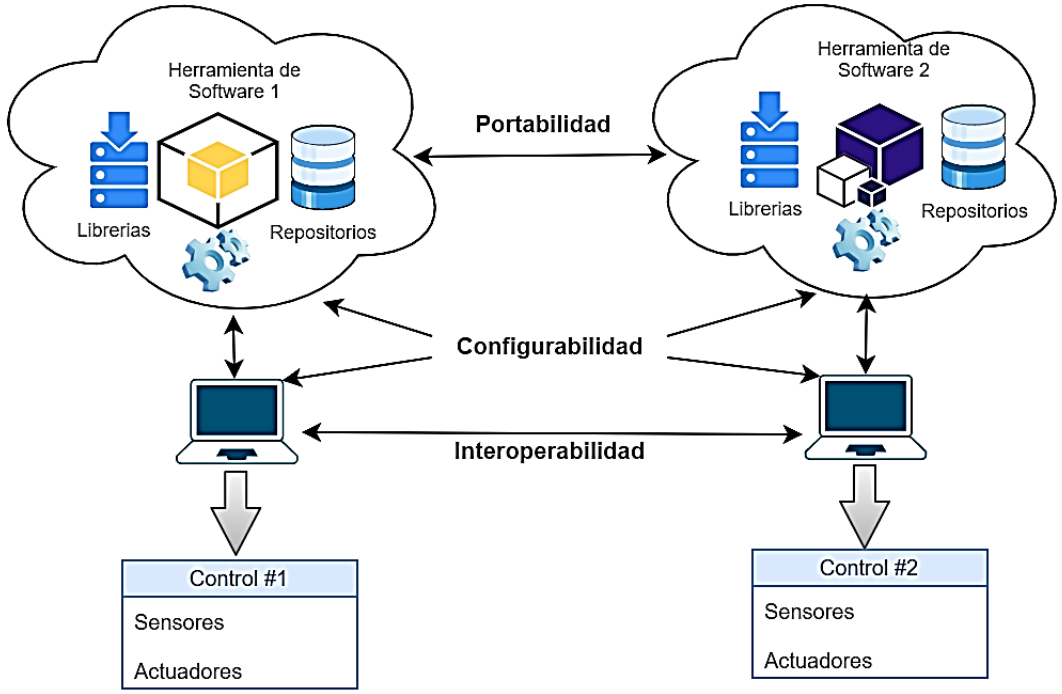


Figura 1: *Características de la Norma IEC-61499.*
Elaborado por: *La Investigadora.*

Estructura de la Norma IEC-61499:

En cuanto a la estructuración del estándar, este se encuentra distribuido en 4 partes, siendo cada una independiente de las otras y con objetivos diferentes. A continuación en la Tabla 2 se resume estas 4 partes [16]:

Tabla 2: *Postulados 1, 2, 3 y 4 de la Norma IEC-61499.* [16].

Postulados		**Descripción**
Parte 1	Arquitectura (IEC 61499-1)	Expone una arquitectura jerárquica y genérica para aplicaciones de control distribuido con modelos independientes, genéricos que permiten comprender la organización del sistema y sus componentes. Implementación de Bloques de funciones.
Parte 2	Requisitos de herramientas de software (IEC 61499-2)	Son los requisitos que deben cumplir las herramientas de software para desarrollar las arquitecturas del postulado de la Parte 1. Además, debe soportar la ejecución de tareas en los sistemas de control distribuido. Y garantiza que se cumplan sus características.
Parte 3	Manual de información (IEC 61499-3)	Tiene la información didáctica para la aceptación, entendimiento y aplicabilidad de la Norma en un amplio conjunto de dominios.
Parte 4	Reglas y Perfiles de Conformidad (IEC 61499-4)	Tiene como función definir perfiles de compilación y conformidad con la Norma, en estos perfiles se especifica características que se deben implementar en los postulados de la Parte 1 y 2.

Elaborado por: *La Investigadora.*

En el postulado de la Parte 1, menciona la Arquitectura la cual requiere que se detallen los modelos que existen en esta sección y de esta forma abordar mejor el tema de Bloque de Funciones. Entonces como primer punto se analiza cómo funciona internamente un FB; lo que hace es seguir una secuencia de acciones al momento de la ejecución y a continuación se enumera el orden del funcionamiento [11],[17]:

1. Primero llega un evento a la entrada del Bloque de Funciones.
2. Luego la entrada de datos se relaciona con el evento de entrada y se actualiza.
3. El evento pasa al control de ejecución.
4. La funcionalidad interna se va activar dependiendo del tipo de Bloque de Función y el control de ejecución.

5. Se proporciona nuevos datos de salida cuando la funcionalidad interna finaliza su ejecución.
6. Los datos de salida se relacionan con los eventos de salida y conjuntamente se actualizan.
7. Y finalmente se envía un evento de salida.

Para comprender el funcionamiento interno de los FB, la Figura 2 ilustra la secuencia de activación de un Bloque de Funciones con el orden que se describe anteriormente. Todos los Bloques deberán cumplir esta secuencia ya que de esta forma cumple su función correctamente [17]:

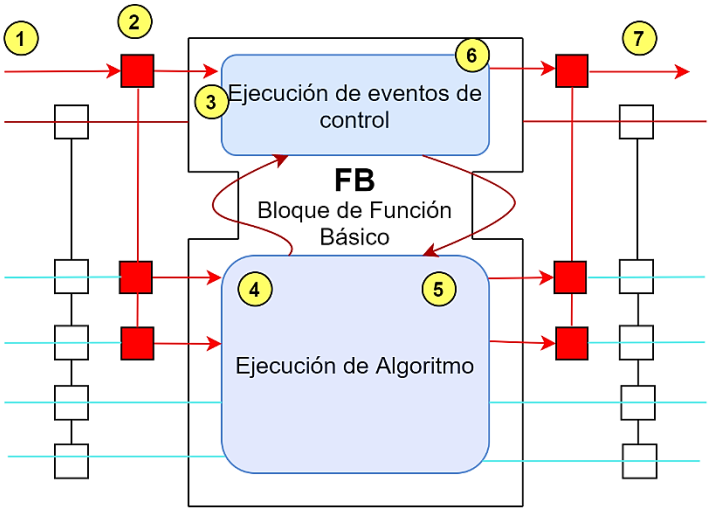


Figura 2: *Características de la Norma IEC-61499.* [17].

Como segundo punto se detalla a continuación los diferentes modelos de Bloques de Funciones con los que se puede trabajar:

Modelo de Bloque Funcional (FB)

Los FB ayudan al programador a descomponer el comportamiento interno de un sistema y la funcionalidad de estos bloques esta proporcionada mediante algoritmos escritos en lenguajes de alto nivel. Como se observa en la Figura 3, el FB está formado por la cabeza que se conecta al flujo de eventos y el cuerpo que se conecta al flujo de datos de información. En conjunto, estas dos partes ejecutan y producen eventos que son utilizados para activar datos de salida [18].

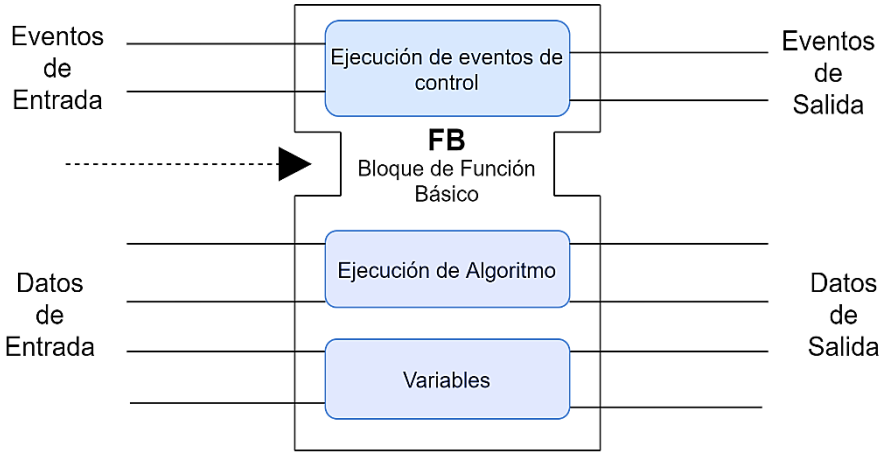


***Figura 3**: Modelo de Bloque Funcional.* [15].

La Norma IEC-61499 define tres tipos de Bloques de Función: FB Básico, FB Compuesto y FB Interfaz de Servicio.

Modelo de Recurso

Este modelo provee servicios a aplicaciones como la planificación y ejecución de los algoritmos en un proceso como se observa en la Figura 4. Sus funciones principales son [18]:

- Aceptar los eventos de entrada y salida de las interfaces de proceso y comunicación.
- Procesar los eventos de envío, además de remitir los eventos a las interfaces de proceso y comunicación.

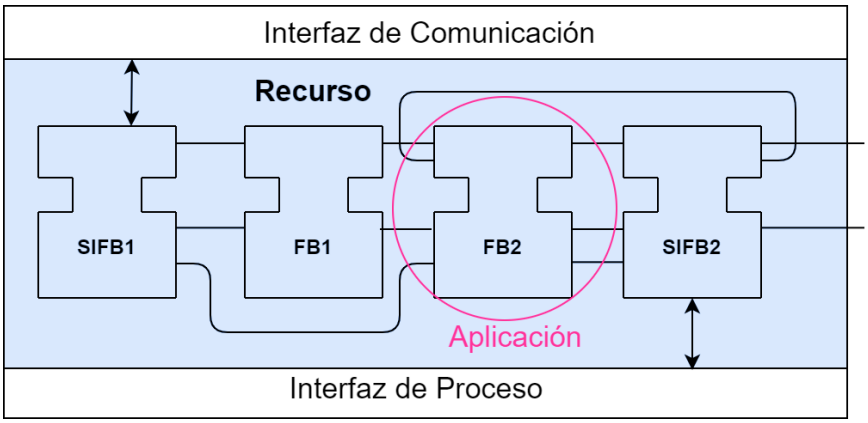


***Figura 4:** Modelo de Recurso.* [15].

Modelo de Dispositivo

Es considerado como un contenedor de recursos y una entidad física independiente, capaz de proporcionar un entorno de ejecución para las aplicaciones. Tiene dos interfaces como se visualiza en la Figura 5 [15], [19]:

- Interfaz de proceso
- Interfaz de comunicación

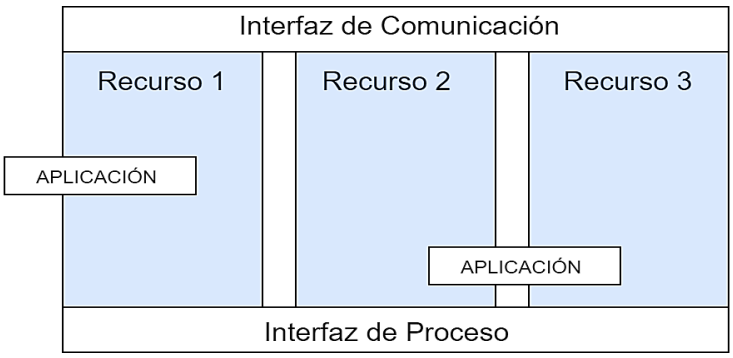


Figura 5: *Modelo de Dispositivo.* [15].

Modelo de Sistema

Es el conjunto de dispositivos interconectados y comunicados entre sí. Su comunicación es a través de segmentos como también de enlaces, capaz de formar una interacción de aplicaciones. En la Figura 6 se ilustra el Modelo Sistema que es una red de comunicación con varios dispositivos [19].

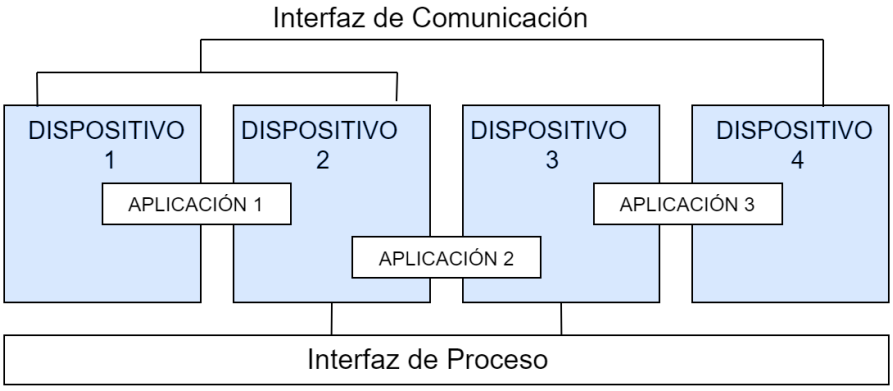


Figura 6: *Modelo de Sistema .* [15].

Modelo de Aplicación

Este modelo utiliza la comunicación entre aplicaciones para determinar una respuesta adecuada de eventos entre las interfaces de proceso y comunicación. Además, permite

que se modifiquen las variables, como también admite la generación de eventos e interacción de interfaces mediante la programación y ejecución de algoritmos internos en las interfaces de proceso y comunicación [19]. En la Figura 7 se puede observar que cada aplicación es una red de FBs.

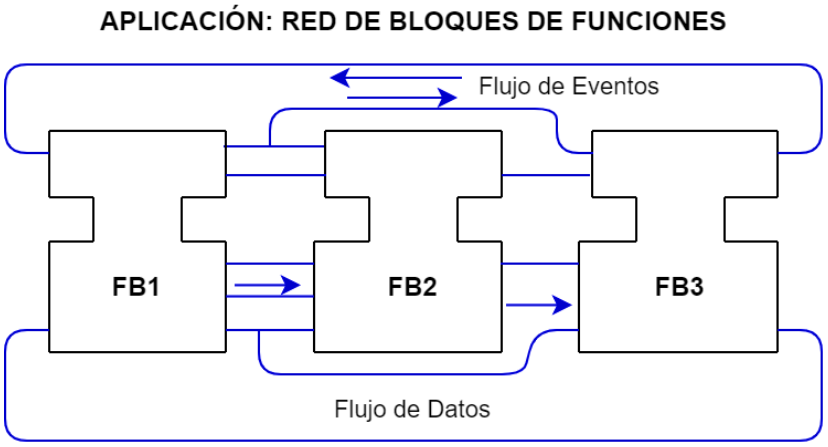


Figura 7: *Modelo Aplicación.* [18].

Modelo de Gestión

Aquí se provee herramientas para la gestión de recursos necesarios para los dispositivos. La interfaz de este modelo es un tipo FB de gestión. La Norma propone dos esquemas [19]:

- 1^{ro}: es la gestión de recursos compartidos.
- 2^{do}: es la gestión de servicios de distribución.

Modelo de Distribución

Este modelo es la fase final de la Norma ya que consiste en la distribución de la aplicación con otros dispositivos de control donde serán mapeados y ejecutados. Se puede observar en la Figura 8, que el modelo de distribución es un complemento del modelo de aplicación [19].

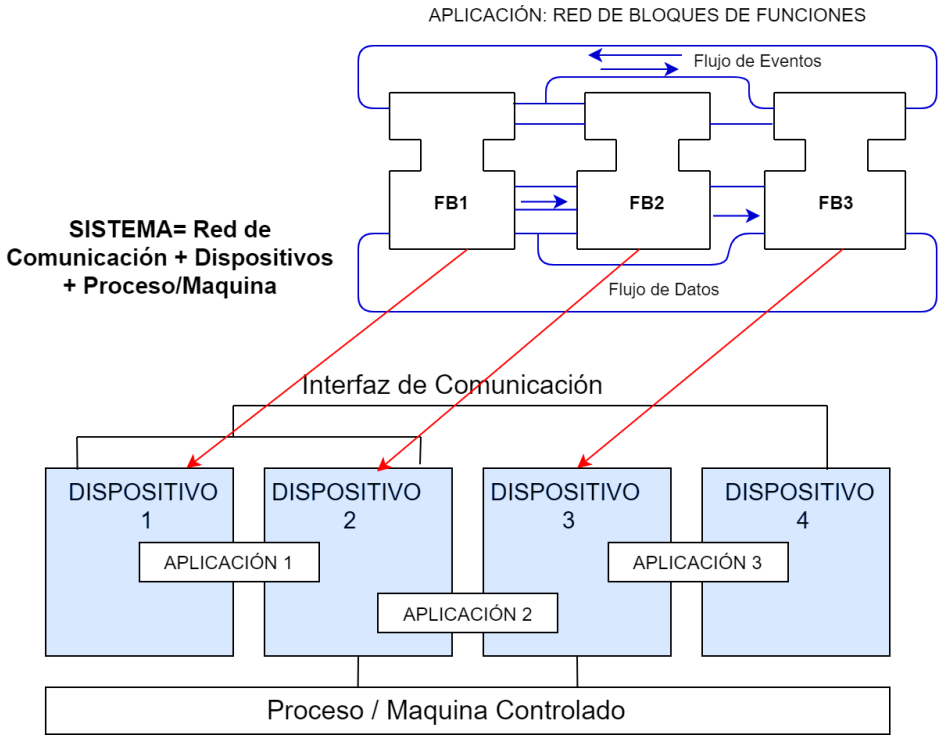


Figura 8: *Modelo de Distribución.* [11].

1.3.2 Arquitectura de Comunicación de la Norma IEC-61499

En la Norma IEC-61499, la interfaz de red está diseñada para integrar nuevos protocolos de comunicación, lo cual significa que la arquitectura es flexible y reconfigurable; está basada en el diseño de capas del modelo OSI. Esta capa de comunicación debe poder enviar y recibir información de cualquier Bloque de Función que esté realizando un proceso. La capa de red cuenta con una interfaz genérica la cual provee de las siguientes funciones básicas [20]:

- Abrir conexión
- Cerrar conexión
- Enviar y recibir datos
- Interrupciones genéricas

Como se había mencionado se basa en el modelo de capa OSI, en este caso se tiene tres secciones: capa superior, capa intermedia, capa inferior; a continuación, se detalla la función de cada una de estas [21]:

Capa superior: Esta capa se encarga de acceder a los puertos de datos directamente para indicar que estos datos están correctamente preparados para ser enviados de un Bloque de Funciones a otro.

Capa intermedia: En esta capa se agrega información adicional a un mensaje como, por ejemplo: información de sesión, codificación, comprensión, etc.

Capa inferior: La capa inferior obtiene los datos de forma asincrónica y accede directamente a la red.

En la Figura 9 se puede visualizar de forma general la arquitectura de la red de esta norma:

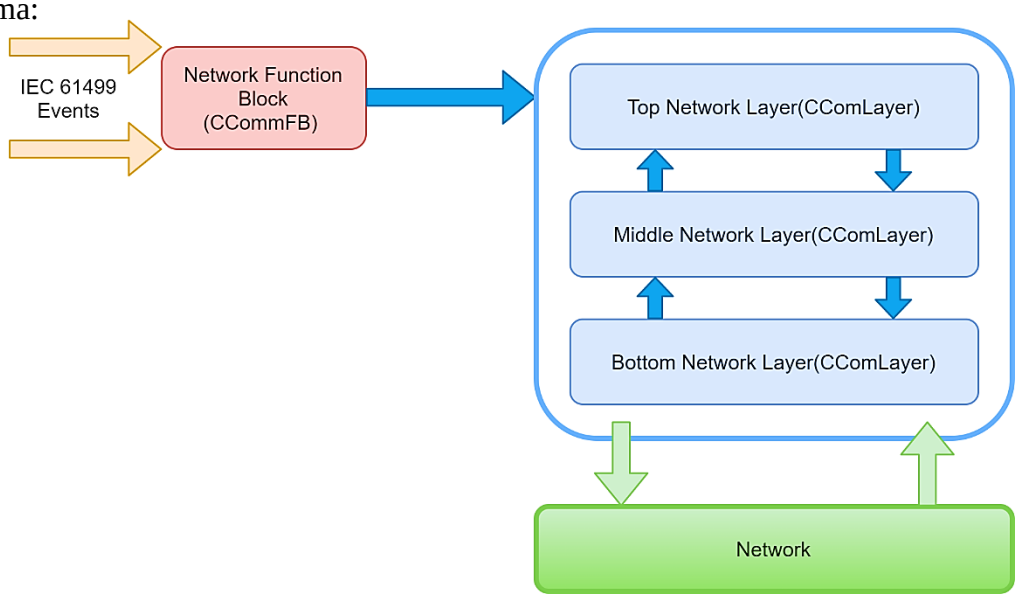


Figura 9: *Descripción general de la capa de red de la Norma IEC-61499.* [21].

Para que la capa pueda interactuar entre Bloques de Funciones necesita de las siguientes funciones [21]:

- **openConnection:** Se debe implementar por cada capa y por cada pila de red. Esta función establece una conexión.
- **closeConnection:** Esta función cierra la conexión y también debe implementarse por cada capa y pila de red.
- **endData:** Función que se encarga de enviar mensajes a través de la red y esta implementado por cada capa y pila de red.
- **recvData:** Debe ser implementado por cada capa, es quien recibe los mensajes nuevos de la red.

- **processInterrupt:** Cuando una capa necesita procesar información, esta función se debe llamar desde la pila de red.

En la Figura 10 se observa cómo se activa la conexión con la función openConnection, donde: primero obtiene el parámetro de inicialización y segundo el parámetro para crear la capa inferior. Se debe tomar en cuenta que cuando el Bloque de Funciones recibe un evento INIT+ esta inicializada la sesión, pero cuando llega un INIT- existe un error [21].

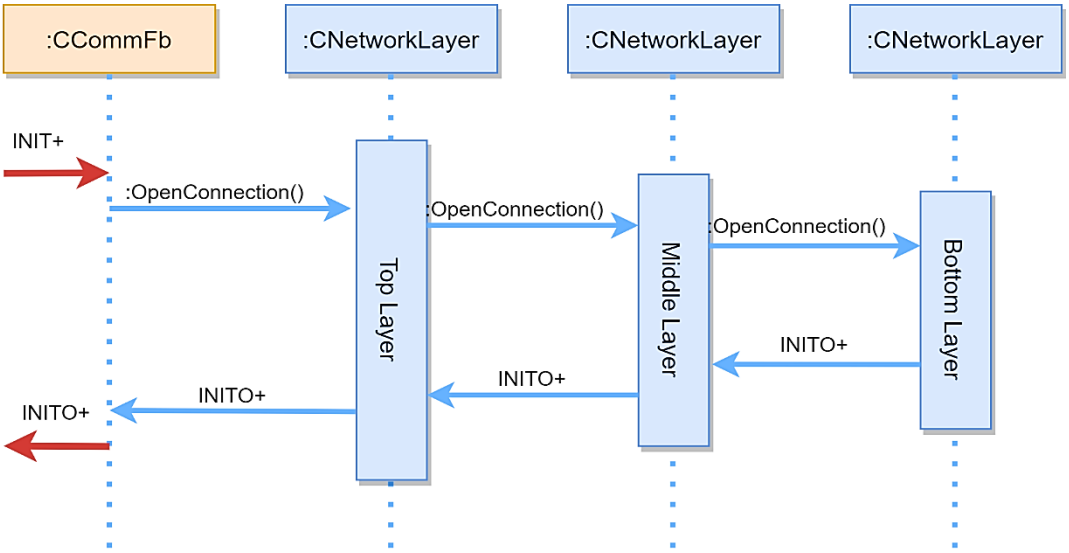


Figura 10: *Conexión exitosa.* [21].

En la Figura 11 se ilustra como la función closeConnection cierra las capas debajo de esta ya que depende de la función openConnection y espera la instrucción para poder terminar la conexión en orden cerrando consecutivo.

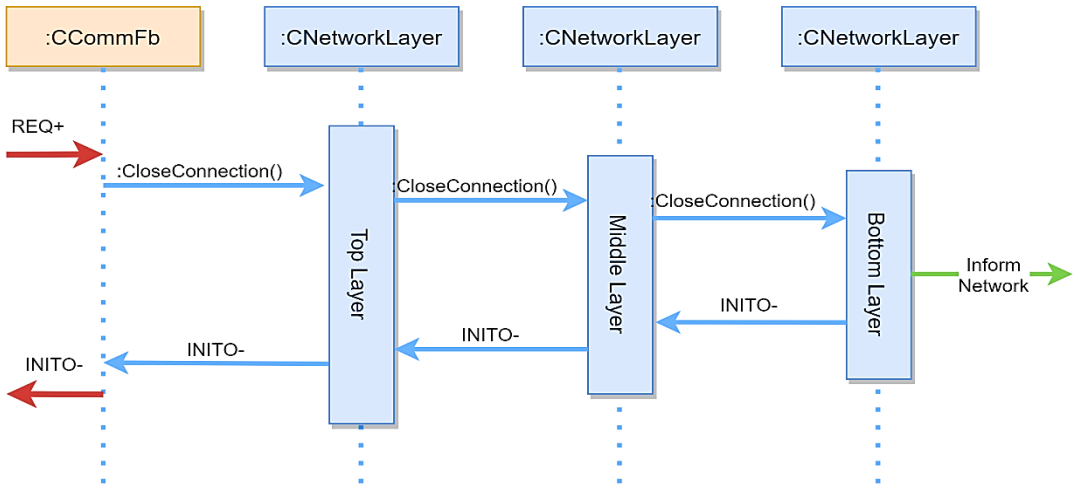


Figura 11: *Secuencia de cierre de conexión.* [21].

La Figura 12 muestra la función sendData la cual debe obtener datos de la capa superior posteriormente empaquetarlos y enviarlos a la capa inferior. El mensaje pasa de una capa a otra hasta llegar a la capa inferior donde esta capa envía el mensaje a la red y devuelve un evento que confirma al Bloque de Funciones que envió el mensaje.

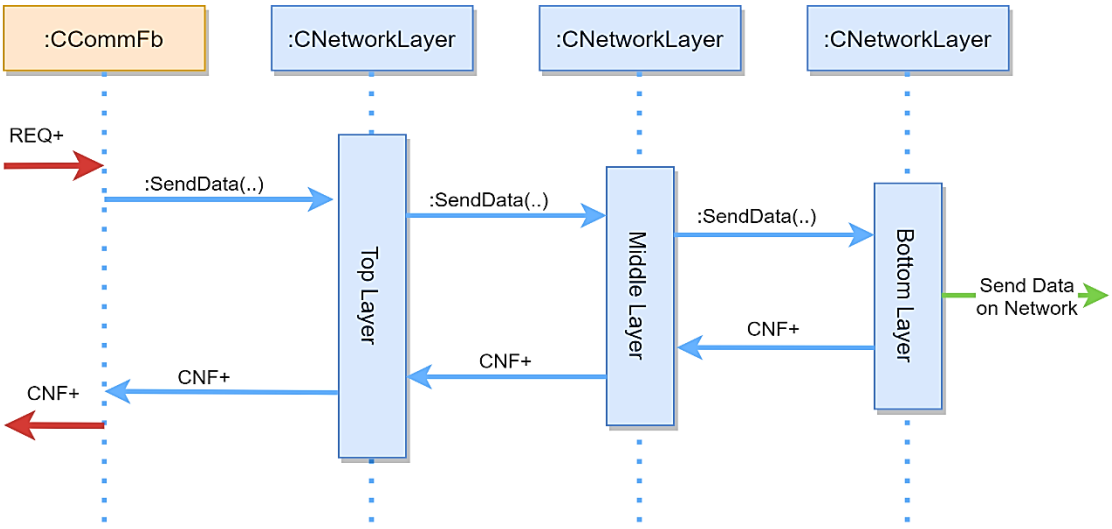


Figura 12: *Envío de datos en una conexión.* [21].

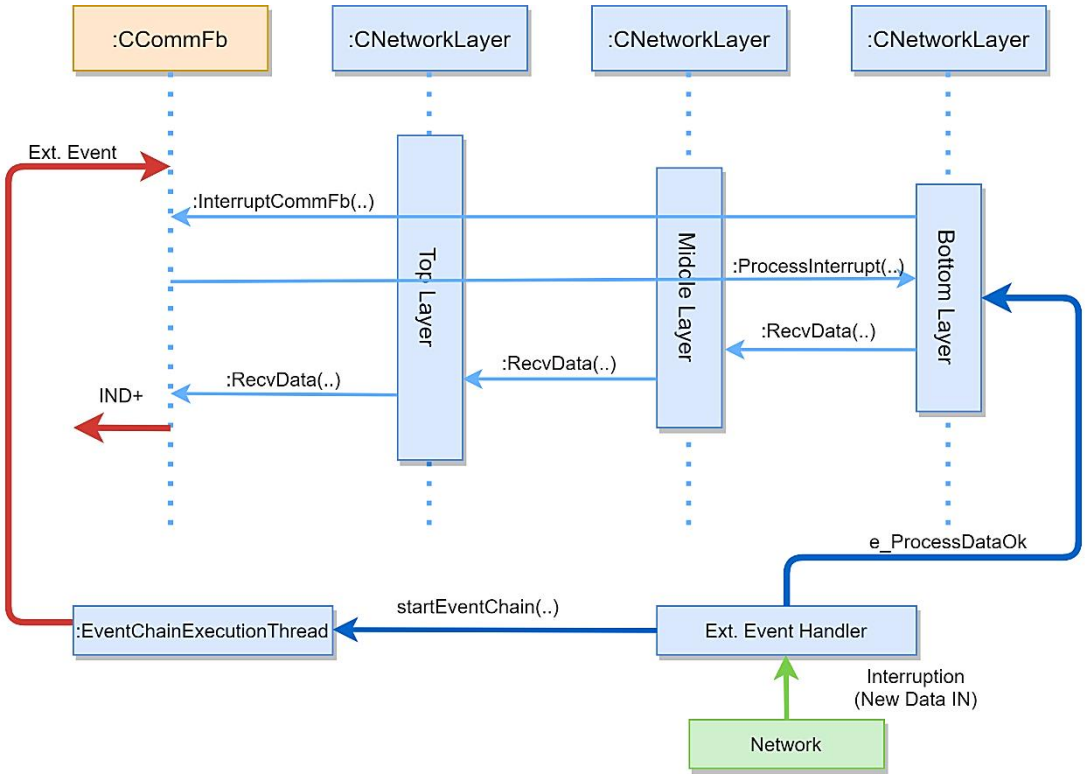


Figura 13: *Diagrama de recepción de datos con interrupciones en la conexión.* [21].

La función recvData es similar a la función sendData con una única diferencia en la capa inferior la cual necesita una discusión adicional. En la Figura 13 se puede observar cómo se recibe los datos mediante interrupciones las cuales son como un mensaje de la red, este mensaje es tomado como una llamada del sistema hasta que se reciban nuevos datos. Cuando se reciben eventos del Bloque de Funciones la función recvData realiza el proceso desde la capa inferior hacia la superior hasta que se reciba todos los datos y además emitiendo una señal CNF+ / IND+.

Cuando se realiza la recepción de datos sucede un caso de interrupción genérica que se utiliza en algunos casos para recibir errores en la conexión. La Figura 14 ilustra un diagrama al momento de enviar la función processInterrupt en la capa de red la cual devuelve un evento INIT si se pierde la conexión.

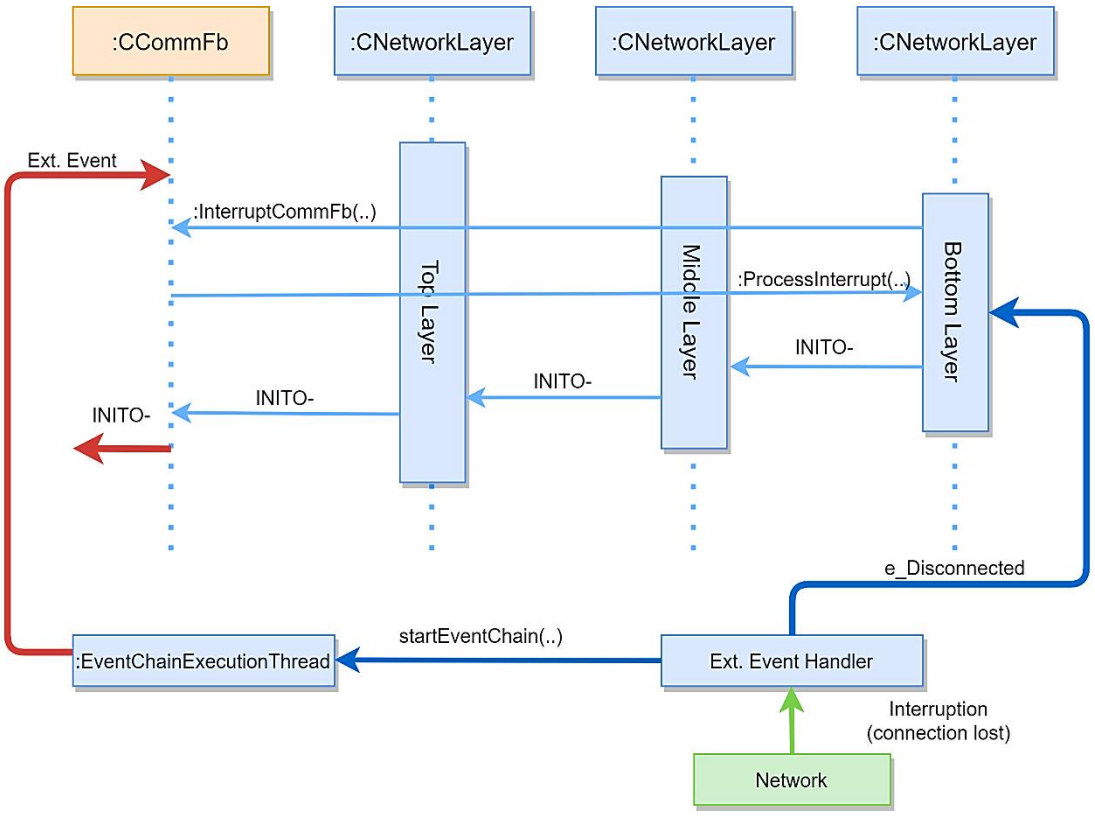


Figura 14:*Interrupción de la conexión en la interfaz de red*. [21].

En la siguiente Tabla 3 se especifican detalladamente los eventos enviados y recibidos por el Bloque de Función en la interfaz de red.

Tabla 3: *Eventos de Envío/Recibo de datos.* [21].

Evento	Entradas/Salidas	Dirección	Descripción
Init Positivo	INIT +	Entrada	Abre una conexión
Init Negativo	INIT -	Entrada	Cierra una conexión
Solicitud Positiva	REQ+ / RSP+	Entrada	Enviar datos
Solicitud Negativa	REQ - / RSP-	Entrada	Ignorar valor
Init Positivo	INITO +	Salida	Nueva conexión
Init Negativo	INITO -	Salida	Conexión cerrada
Confirmar Positivo	CNF+ / IND-	Salida	Nuevos datos disponibles
Confirmar Negativo	CNF- / IND-	Salida	Datos recibidos no validos / Error de conexión

Elaborado por: *La Investigadora.*

1.3.3 Herramientas de desarrollo y ejecución de la Norma IEC-61499

En la Norma IEC-61499 se sigue un diseño de aplicaciones para un sistema general que sea independiente con el hardware, cumpliendo con una red de Bloques de Funciones que se ejecuten en forma eventual. Puesto que la definición de Bloque de Función se puede configurar con varios dispositivos a través de herramientas de ingeniería de la norma se pueden utilizar los mismos comandos para reconfigurar cualquier sistema de forma online [22].

IDE (Entorno de Desarrollo Integrado) en la norma es un constructor de interfaz gráfica con un editor de código que permite desarrollar aplicaciones de control las cuales se pueden compilar y depurar. Por otra parte, también se tiene el entorno de ejecución o RTE (Entorno de Tiempo de Ejecución), esto es un estado de máquina virtual que provee servicios de software de programas mientras que un ordenador se está ejecutando [5].

Las herramientas de desarrollo que se pueden utilizar son las siguientes [7], [6], [23], [24], [25]:

FBDK (Functional Block Development Kit): Fue la primera herramienta en desarrollarse, es una aplicación Java que sirve para dibujar Bloques de Función, después de esto los creadores decidieron modificar esta herramienta para demostrar su viabilidad, añadiendo el intercambio de archivos XML (eXtensible Markup Language) y el testeo de los Bloques de Funciones.

Es un programa de libre distribución permitiendo que las aplicaciones se generen con una ingeniería centrada y se puedan descargar en diferentes dispositivos. Además, provee de una interfaz sencilla y poco amigable con el usuario, pero con el pasar del tiempo los distribuidores realizan actualizaciones para garantizar los cambios y mejoras constantes que realiza la norma [7], [24].

4DIAC-IDE (For Distributed Industrial Automation and Control): Es una herramienta que sirve para desarrollar aplicaciones con un conjunto de complementos para el entorno de desarrollo integrado de Eclipse. Este software de licencia pública de Eclipse sirve para crear aplicaciones de control distribuido y automatización. 4DIAC es compatible con los softwares FDBK y nxtOne [6].

nxtOne: Este software se basa en dos estándares importantes de la Industria: Estándar IEC-61131 y Estándar IEC-61499. Fue creado por la compañía nxtControl y dispone de un HMI que sirve como visualizador de cualquier proceso, también cuenta con funciones de debugging y test. nxtOne es compatible con las herramientas de desarrollo con excepción de ISaGRAF. Algunas de sus características son [23], [24]:

- Complejidad reducida.
- Solución sencilla para sistemas de control distribuido.
- HMI / SCADA perfectamente integrado.
- Automáticamente con rutas de comunicación.
- Ingeniería independiente de hardware y software.
- Sirve para cualquier tarea de automatización**.**

ISaGRAF: Este software también se rige a los estándares IEC-61131 y IEC-61499 lo cual permite que los desarrolladores puedan elegir el lenguaje de programación que

deseen adaptar a la aplicación de automatización. Fue creado por la empresa ICE Triplex para el control SoftPLC que sea portátil y robusto cuenta con un entorno de desarrollo de aplicaciones intuitivo [25].

A continuación se especifican las plataformas de ejecución con las que trabajan las herramientas de desarrollo [5], [24]:

FBRT (Funtional Block Runtime): Es una plataforma desarrollada en lenguaje Java para ejecutar modelos que se desarrollaron en el software FDBK. Contiene librerías y aplicaciones basada en la plataforma Java SE y es compatible con los programas: FDBK, 4DIAC-IDE y nxtOne [24].

FORTE: Es un entorno de ejecución portable de código abierto en lenguaje C++, este entorno puede ejecutarse en pequeños dispositivos embebidos y en sistemas de tiempo real. Además de ser compatible con los programas FDBK, 4DIAC y nxtOne. La configuración de este entorno tiene como ventaja que al ejecutar un proceso de baja prioridad no va a alterar la ejecución de un proceso de mayor prioridad [5], [24].

nxtRT61499F: Este entorno se basa en FORTE con la diferencia que este incluye funciones adicionales y otros servicios que se pueden configurar con ntxOne, 4DIAC y FDBK. Desarrollado por la compañía ntxOne [24].

ISaGRAF Runtime:

Es uno de los entornos que no cumple al cien por ciento con la norma IEC-61499 ya que funciona verificando de forma periódica los estados de las entradas, de manera que solo puede ser configurado con el programa ISaGRAF. Incluye [24], [25]:

- Herramientas de personalización para Workbench.
- Interfaz IXL.
- Interfaz de comunicación.
- Interfaz de Bloques de Funciones.
- Interfaz de Entradas y Salidas.
- Todo un Sistema de Interfaces

1.3.4 Dispositivos Runtime

Los dispositivos de Run Time (en tiempo de ejecución) deben contener una especificación en su entorno de trabajo de aplicaciones describiendo el contenido de bibliotecas o librerías básicas y también propiedades de máquina virtual. La máquina virtual viene a ser el software de simulación donde se ejecutan diferentes programas como si estuvieran en una computadora real. Se puede tener varias máquinas virtuales funcionando de forma simultánea, ya que se puede ejecutar un sistema operativo en cada una de ellas [26].

Run Time significa que cualquier programa desarrollado en cualquier lenguaje de programación se ejecute en un sistema. Después que el lenguaje se interpreta por consiguiente es almacenado en un fichero en modo de texto y el cual se puede editar y también modificar. Cuando se ejecuta este texto se utiliza un intérprete o Run Time el cual lee y ejecuta las instrucciones en cualquier dispositivo u ordenador [27].

Requisitos de un dispositivo Run Time [27]:

- CPU de 16 bits o 32 bits con una velocidad de reloj de 16MHz o más.
- 160KB ROM con espacio reservado para librerías y máquina virtual.
- Conectividad a red (inalámbrico o por cable).

1.3.5 Automatización de bajo costo

Hoy en día se puede automatizar y digitalizar diversos procesos con tecnologías de amplia disponibilidad y bajo costo, lo que permite a cualquier empresa ser competitiva e ir mejorando la calidad y eficiencia de su trabajo. Existen sistemas de control de bajo costo con softwares más avanzados que pueden ser utilizados por los ingenieros para automatizar los procesos industriales y de esta forma aumentar la flexibilidad de la producción y consecuentemente reducirá el costo de fabricación en una planta de trabajo [28], [29].

En la Industria 4.0 y en IoT (Internet de las cosas) se hace cada vez más visible la implementación de dispositivos de bajo costo con software libre. Estos dispositivos cuentan con microcontroladores de capacidades de cifrado en el hardware para reducir la sobrecarga en el software y también incrementan seguridad y bajo consumo. Para comprender esta relación se tiene como ejemplo: el cifrado AES (Advanced

Encryption Standard) mediante el hardware tiene como ventaja ser 10 veces mejor que el software. Los microcontroladores con estas capacidades son implementados en placas como: Arduino, Raspberry Pi, etc. La Figura 15 muestra algunos tipos de dispositivos de bajo costo [30].

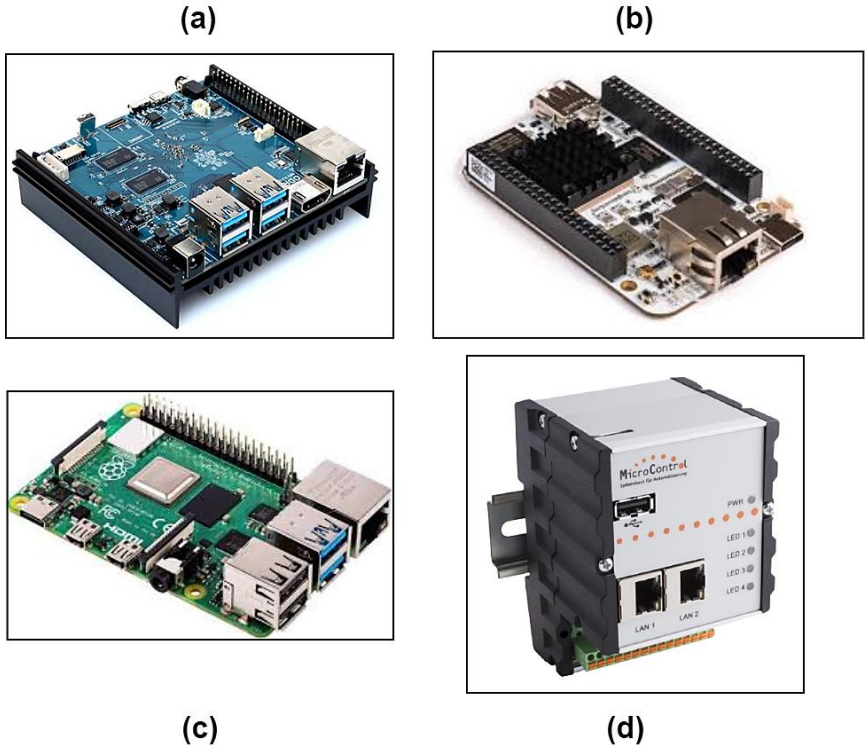


Figura 15: *Diferentes modelos de SBC (Single Board Computer). (a) ODROID XUA, (b) BEAGLEBONE BLACK, (c) RASPBERRY PI, (d) UMIC.200.* [30].

Los sistemas SBC están siendo fabricados para adaptarse a ambientes industriales y de esta forma soportar vibraciones, ruido eléctrico y humedad. Estos dispositivos son internacionalizados para tener un soporte amplio en cuanto a estandarización e implementación en cualquier tipo de empresas industriales. Además, cada vez se van creando sensores más precisos, conectores de comunicación, nuevos periféricos que sean compatibles con estas placas para que de esta forma se siga expandiendo la comercialización.

1.3.6 Sistemas Cyber Físicos de Producción (CPPS)

Los CPPS son un nexo con los nuevos prototipos de la Industria 4.0, unificando el mundo físico - virtual y la Industrial Internet of Things (IIoT), permitiendo la

interoperabilidad en las empresas de producción. En estas empresas el origen de toda esta virtualización son los productos inteligentes llamados también “Sistemas Cyber – Físicos (CPP)” que disponen de software embebido, conectividad y la parte electrónica; además que pueden interactuar con otros sistemas y también con humanos [9],[31].

Los CPS tienen softwares que les admite auto – gestionarse y de esta forma tomar decisiones descentralizadas, contienen sensores que receptan información del entorno en el que se encuentran y además proporcionan servicios inteligentes. La Figura 16 muestra que los CPPS son máquinas con capacidad de personalización, flexibilidad, comunicación y adaptación al entorno [31]:

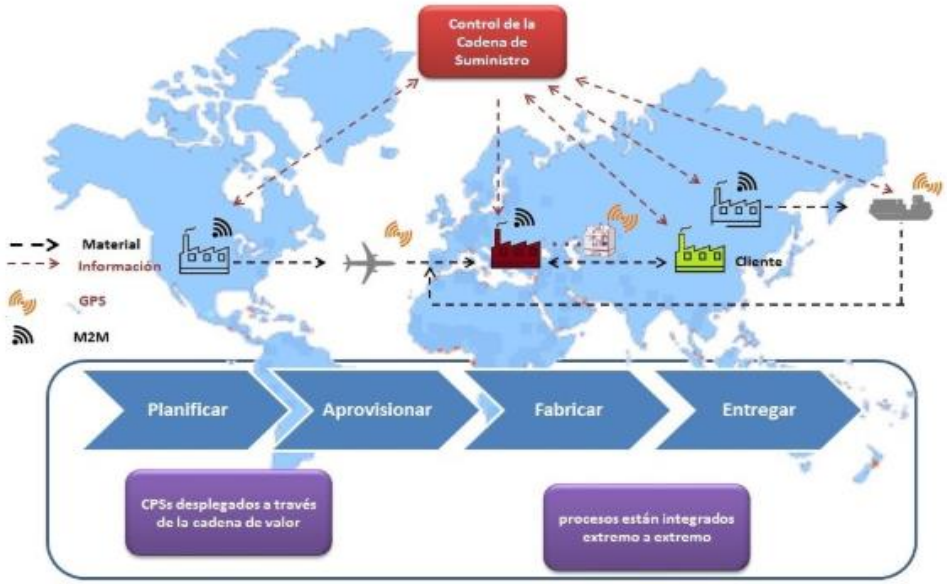


Figura 16: *Cadena de suministro inteligente y conectada a una central de Industria 4.0*. [31].

En la red de suministro inteligente, los CPS generan datos y proveen información sobre su estado y posición, la visibilidad de los movimientos que se generan en la red sirven para reconocer riesgos e ineficiencias de los procesos de producción con esto se podrá aumentar la robustez y capacidad de respuesta a cualquier incidente, por otro lado, la digitalización sirve para automatizar e identifica el producto a lo largo del proceso [31].

Cada Sistema Cyber Físico de Producción puede decidir su programa de producción con respecto a su tiempo de procesamiento, ya que son auto gestionables y se vuelven invisibles a los ojos de los operadores quienes solo se encargan del mantenimiento de

los CPPS. Entonces las empresas de producción se convierten en una red donde cada sistema tiene la capacidad de obtener información necesaria y tomar decisiones óptimas para la entrega de un producto satisfactorio [32].

En resumen, los CPPS actualmente se han convertido en sistemas autónomos con tecnología adaptable y de esta forma convergen con las nuevas tendencias de digitalización Smart. De hecho, la digitalización conlleva a la aparición de tareas complejas en los procesos de producción, para lo cual se requerirá de trabajadores con habilidades técnicas y de ingeniería. En la Figura 17 se muestra como se realiza la convergencia del mundo físico y virtual [33], [34]:

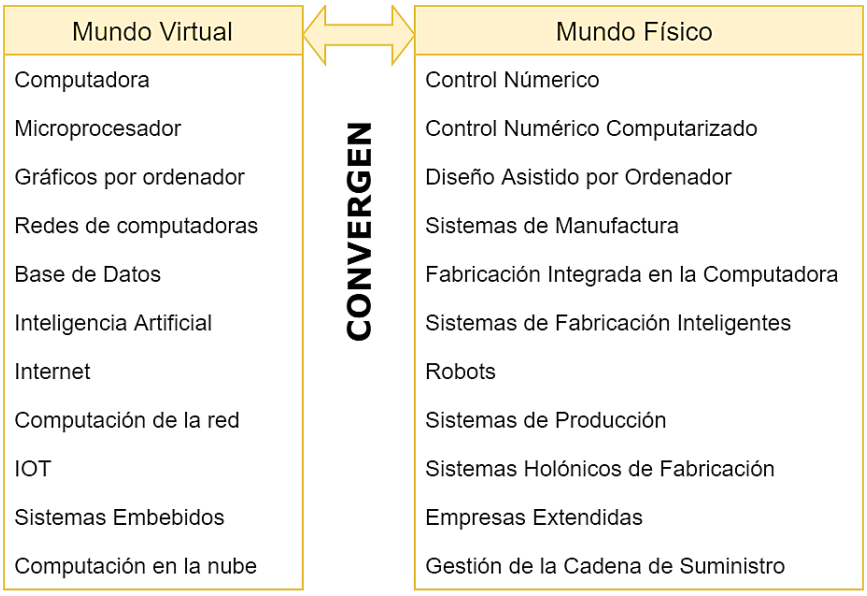


Figura 17: *Convergencia del mundo físico y virtual.*
Elaborado por: *La Investigadora.*

1.3.7 Industria 4.0

La Industria 4.0 se considera como la Cuarta Revolución industrial porque busca digitalizar procesos industriales, interconectar estos procesos mediante Internet, optimizar recursos, crear metodologías comerciales más efectivas; estableciendo cambios enfocados en asociar la “empresa común” con la Inteligencia Artificial dando como resultado una organización inteligente. Esta nueva organización incluye varios campos como: nanotecnología, impresión 3D, robótica, vehículos autónomos,

computación cuántica, entre otros. En la Figura 18 se puede observar cómo ha ido evolucionando la tecnología hasta esta nueva etapa [35], [36].

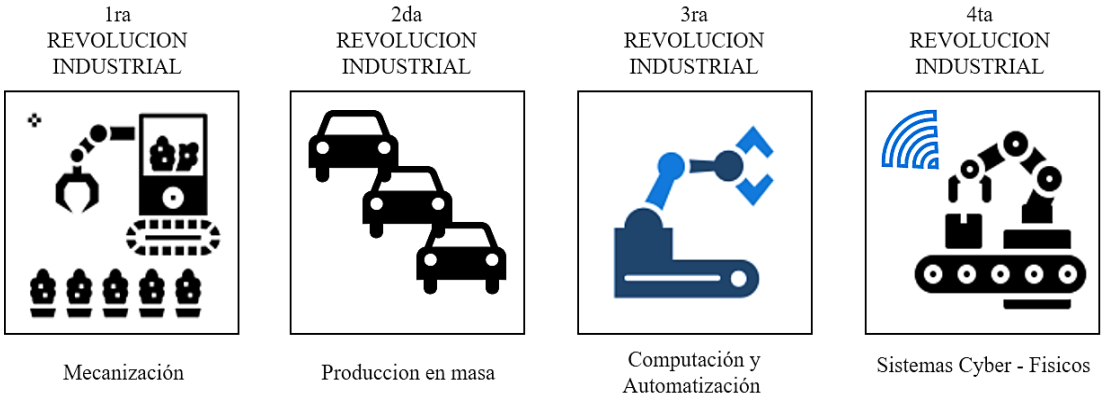


Figura 18: *Diagrama de la sucesión de la Revolución Industrial a lo largo del tiempo.*
Elaborado por: *La Investigadora.*

La nueva industria tiene aspectos importantes como se muestra en la Figura 19, las cuales son tecnologías que sustentaran el cambio radical de las empresas inteligentes [37]:

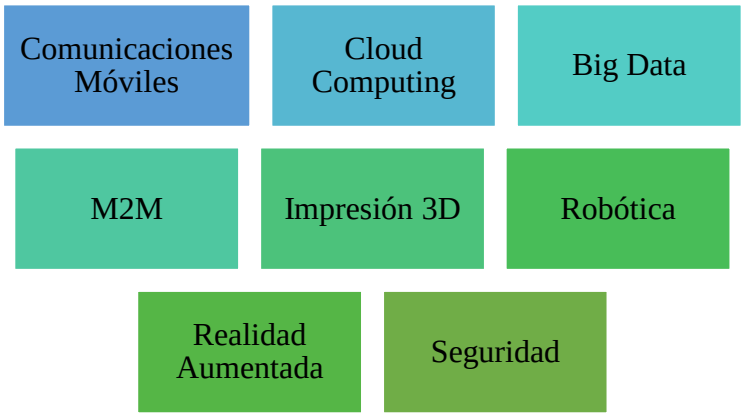


Figura 19: *Diagrama de las tecnologías claves en la Industria 4.0.*
Elaborado por: *La Investigadora.*

Comunicaciones móviles: El internet móvil son la base de la tecnología inteligente sirve para mantener conectados los sistemas y el entorno de producción en tiempo real y de forma difundida [37].

Cloud Computing (La nube): La nube ofrece servicios de almacenamiento donde se puede guardar y procesar la gran cantidad de datos que generan los sistemas de

producción esta información puede ser accesible de manera online, a través de redes públicas y privadas [37].

Big Data (Análisis de Datos): Permiten que se interprete y se gestione una gran cantidad de datos y de esta forma aprender los hábitos de los consumidores y así predecir eventos futuros que ayudaran a mejorar la producción [38].

M2M (Machine 2 Machine): La Comunicación de Máquina con Máquina sirve para el intercambio de datos de los sistemas inteligentes y los productos en el entorno industrial [37].

Impresión 3D: Son productos tridimensionales a base de modelos virtuales, se puede crear cualquier prototipo de productos por ejemplo joyas, calzado y hasta se pueden llegar a crear elementos internos de un automóvil haciendo así una fabricación altamente descentralizada [38].

Robótica: La robótica con el tiempo trabajará conjuntamente y de forma segura con los seres humanos, ya que la inteligencia artificial ha ayudado a crear robots cada vez más autónomos [37].

Realidad Aumentada: O realidad virtual aporta una manera distinta en la experiencia visual de las personas, combinando el mundo virtual con el real a través de un proceso informático [38].

Seguridad: Existen amenazas en la producción industrial ya que los sistemas están conectados entre sí y se utilizan protocolos de comunicación que deben ser protegidos contra ataques cibernéticos [37].

Por consiguiente, esta etapa revolucionaria podría significar que también existirá una Sociedad 4.0 que irá a la par con la Industria 4.0 mediante elementos, como la formación técnica, la globalización y la gestión electrónica siendo todo más constructivo, inteligente y beneficioso.

Ventajas de la Industria 4.0

- Los procesos automatizados tendrán menos errores y mayor eficiencia.
- La producción se hará ininterrumpidamente lo cual es una ventaja en las empresas que son estacionales.

- Se evita notablemente las interrupciones y los tiempos muertos durante la producción.
- Los niveles de calidad se optimizan.
- El personal en diferentes procesos será aventajado ya que evitaran trabajos forzados o a temperaturas altas y entornos peligrosos, su seguridad será primordial.
- El producto será adaptable en cuanto a los requerimientos de las empresas ya que la producción es más flexible.
- Se nota una mejora en la eficiencia de las organizaciones.
- Se tiene un gran potencial al interconectar variedad de personas a través de las redes sociales.
- Gracias a las redes de comunicación los datos de información tendrán un flujo constante [39].

Inconvenientes de la Industria 4.0

- Muchas de las empresas no se adaptan a los cambios tecnológicos y corren el riesgo de quedarse desactualizadas en corto tiempo.
- Puede existir desigualdades sociales por los grandes avances industriales.
- La Industria 4.0 depende enormemente de la tecnología.
- En cada uno de los países la nueva industrialización no puede ser satisfactoria ya que sus reglamentos o legislaciones no evolucionan de acuerdo a los avances tecnológicos.
- La industria inteligente requiere de personal más especializado y con esto una mayor remuneración.
- Existe como riesgo importante la obsolescencia tecnología.
- La empresa que quiera adatarse a la Industria 4.0 sufrirá un coste elevado en la inversión inicial, sin embargo, a largo o mediano plazo se recuperara dependiendo de la producción que realice.
- Se debe mantener actualizada la tecnología dentro de las industrias por el constante cambio tecnológico [39].

No obstante, las organizaciones inteligentes tienen como reto principal saber gestionar correctamente y aprovechar las oportunidades que genera el cambio hacia la Industria 4.0.

1.3.8 Protocolos de comunicación en la Industria 4.0

En la actualidad, la Industria 4.0 cuenta con la digitalización y la utilización de internet como elementos importantes en los centros de producción. Los departamentos de Tecnología de Información tienen una ardua tarea ya que no están familiarizados con los procesos de comunicación mediante protocolos industriales. Implica grandes retos en los procesos de producción ya que deben adaptarse a la interconectividad del cliente y a gestionar la trazabilidad multidimensional de extremo a extremo, también gestiona la coordinación de entornos industriales y garantiza la escalabilidad de estos cambios [10].

El estudio realizado por el Parlamento Europeo en el 2016 puntualiza que la Industria 4.0 tendrá éxito si se rige a la estandarización de sistemas, protocolos y plataformas, a cambios en la organización y pueda adaptarse a nuevos modelos de negocio, a una seguridad y protección digital. El entorno en el cual trabajarán los protocolos de comunicación se podrán visualizar en el modelo de capas sobre las que se apoyan servicios de software. En la Figura 20 se puede observar 5 capas [40]:

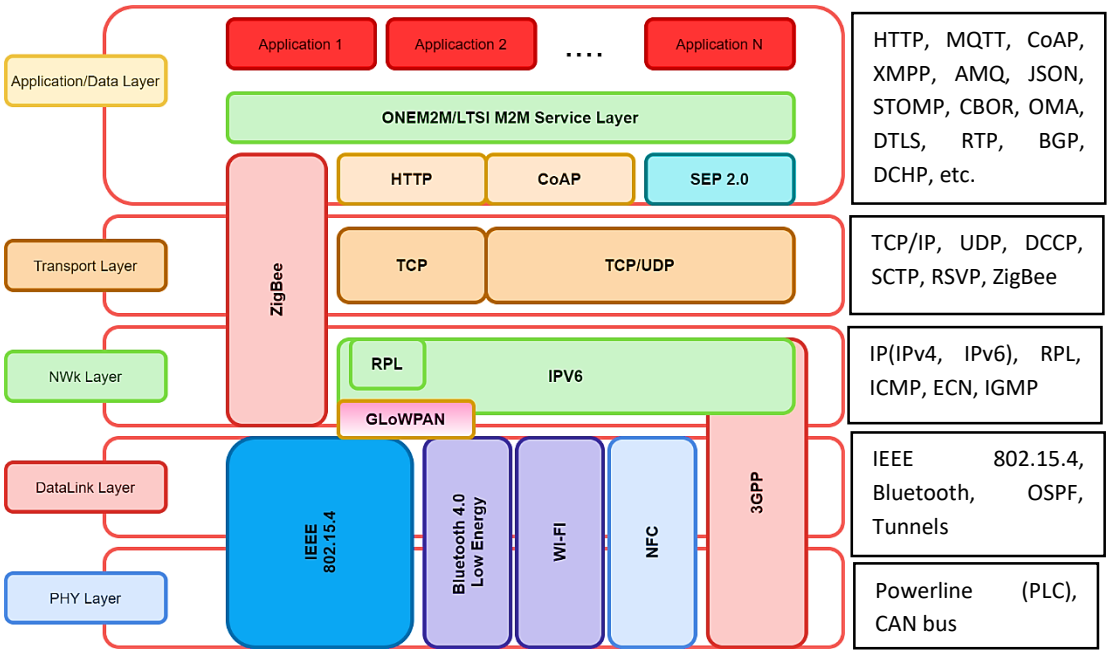


Figura 20: *Capa de comunicación Industria 4.0.* [40].

A continuación se detalla la función de cada una de las capas [41], [40]:

- Capa física: medio por donde viaja la información.
- Capa de enlace de datos: dispone de direcciones para acceder al medio físico.
- Capa de red: proporciona conectividad y selecciona la ruta de dos sistemas.
- Capa de transporte: transporta independientemente datos de la red utilizada.
- Capa de aplicación: ofrece a las aplicaciones la posibilidad de acceder a los servicios de las otras capas y además define los protocolos necesarios para el intercambio de información.

La capa de aplicación es el eje fundamental con respecto a los protocolos que se podrán aplicar en los procesos de Industria 4.0, ya que aquellos permiten la comunicación de los dispositivos entre sí. Dentro de estos protocolos se distingue los siguientes:

Los protocolos de Cliente/Servidor: Necesitan que los clientes se conecten al servidor y realizan una solicitud. El servidor es quien contiene la información y responde a las solicitudes de los clientes. El cliente debe saber dónde está el servidor para posteriormente establecer una conexión [40].

Los protocolos de Publicación/Suscripción: Sirven para que los dispositivos publiquen la información con un determinado asunto sobre un servidor con una dirección fija. En cambio, los suscriptores pueden conectarse al servidor y suscribirse a cualquier asunto, accediendo a la información que les interese. Este intercambio de información no requiere que el publicador conozca la dirección del suscriptor [40].

1.3.9 Internet Industrial de las Cosas (IIOT)

Los sectores industriales se unen a los avances tecnológicos aspirando que sus sistemas puedan comunicarse y coordinarse entre sí, aprovechando la información de los sensores para poder acoplar con otros datos de diferentes procedencias para analizar, mejorar y retroalimentar algún proceso. Los sistemas estarán conectados global y localmente prestando información a todos los usuarios posibles. La Industria 4.0 tiene

su propia filosofía para trabajar, pero con ayuda de la IIoT la competitividad global tendrá resultados positivos como se muestra en la Figura 21, [42], [43]:

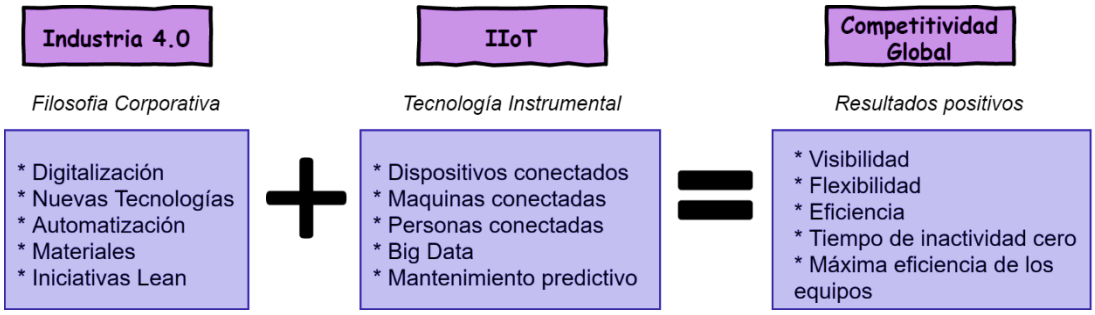


Figura 21: *Diagrama de la fusión de la Industria 4.0 e IIoT*
Elaborado por: *La Investigadora.*

El funcionamiento general de la IIoT tiene que ver con una red de dispositivos que en conjunto forman sistemas inteligentes con el fin de recopilar, intercambiar, monitorear, controlar y analizar información que permite a las industrias tomar decisiones mucho más ágiles. Para implementar un entorno IIoT se tendrá [43]:

- **Infraestructura** para el procesamiento de los datos.
- **Puerta de enlace**, permitirá la interconexión de los sensores a un nivel de comunicación superior mediante arquitecturas y protocolos.
- **Plataformas IIoT**, es el punto de acceso para los usuarios.
- **Dispositivos inteligentes**, son los sensores, actuadores, controladores entre otros.

Sin embargo, la IIoT no es simplemente conectar todo al Internet, puesto que interconectar un sin número de dispositivos requiere la ayuda de diferentes protocolos y estándares de comunicación. Entonces en la Tabla 4 se detalla algunos protocolos que se utilizan en diferentes capas de trabajo [44]:

Tabla 4: *Protocolos de Internet Industrial de las Cosas.*[44].

PROTOCOLOS IIOT	
PROTOCOLOS DE INFRAESTRUCTURA	
Protocolo	**Especificación**
IPv6	Conmutación de paquetes, proporcionando de un extremo a otro una transmisión de datagramas.
6LoWPAN	IPv6 Sobre Redes Inalámbricas de Área Personal de Baja Potencia.
UDP	Protocolo de Datagramas de Usuario.

QUIC	Conexiones Rápidas a Internet vía UDP.
Aeron	Multidifusión UDP con transporte de Comunicación Inter – Process (ICP).
ulP	Pila de código abierto, utilizado en microcontroladores de 8 y 16 bits. Pertenece al protocolo TCP/IP.
DTLS	Capa de Transporte de Datagramas.
ROLL/RPL	Enrutamiento IPv6 para Redes de Baja Potencia.
CCN	Red Centrada en Contenido es una arquitectura pensada para solucionar los desafíos de movilidad, seguridad y escalabilidad.
TSMP	Protocolo de Malla Sincronizada en Tiempo.
PROTOCOLOS DE DATOS	
Protocolo	**Especificación**
CoAP	Protocolo de Aplicación Restringido.
SMCP	Es una pila de CoAP de entradas/salidas asíncronas. SMCP.
STOMP	Protocolo de Mensajería Orientada a Texto Simple.
MQTT	Transporte de Telemetría del Servidor de Cola de Mensajes.
MQTT-SN	MQTT para redes de Sensores.
XMPP	Mensajería Extensible y Protocolo de Presencia.
XMPP-IoT	Comunicación entre Máquina a Máquina y Persona a Persona sea interoperable y de igual forma que XMPP en tiempo real .
AMQP	Protocolo Avanzado de Cola de Mensajería.
DDS	Servicio de Distribución de Datos para Sistemas en Tiempo Real.
SOAP	Protocolo Simple de Acceso a Objetos.
REST	Transferencia de Estado Representacional.
ONS	Servicio de Nombre de Objeto.
SSI	Interfaz de Sensor Simple.

Elaborado por: *La Investigadora.*

1.3.10 Buses de campo Industrial

En áreas industriales la comunicación entre dispositivos se rige por una jerarquía de niveles donde cada uno se conecta mediante buses de campo y de esta forma se permite el flujo de datos a lo largo del proceso de producción. Por lo cual, los buses de campo son redes digitales de comunicación multipunto y bidireccional entre dispositivos

inteligentes que conectan a sensores, actuadores, módulos de Entrada/Salida y controladores con los sistemas de control, permitiendo que un sin número de puntos analógicos y digitales se conecten al mismo tiempo reduciendo el número y la longitud de los cables [45].

Los buses de campo tienen como función gestionar eficientemente mensajes cortos, transmitir mensajes prioritarios y responder rápidamente los mensajes recibidos en la red, poseen mecanismos de detección y corrección de errores, tienen la capacidad de manejar el tráfico de datos y deben recuperarse inmediatamente ante incidentes en la conexión. Para este tipo de buses de campo existen varias normas que intentan integran su estandarización para poder utilizar este tipo de comunicación en las industrias, las más relevantes son: Fieldbus Foundation y Profibus, también: LonWorks, ControlNet, WORLDFIP, HART, MODBUS, DeviceNet, Interbus, FIP, entre otras [46].

En la Figura 22 se puede observar el diagrama de un sistema de control, donde el bus de campo conecta al PLC (Controlador Lógico Programable) con los dispositivos que realizan una tarea según las instrucciones del usuario u operador.

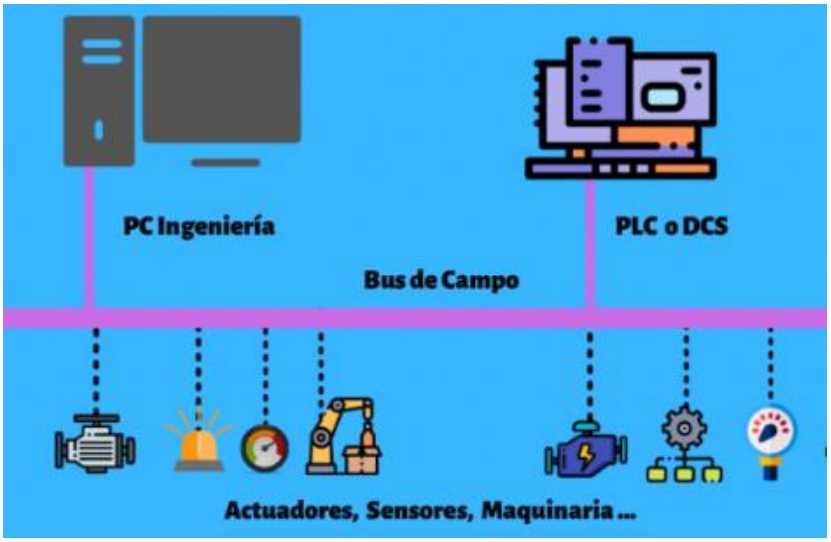


Figura 22: *Diagrama básico de un Bus de Campo*.[47].

Como se menciona anteriormente existen varios estándares impulsados por diferentes fabricantes, por lo cual también se diferencian entre sí por lo tanto se pueden utilizar cualquier de ellos dependiendo de la necesidad de la fábrica industrial. La Tabla 5 presenta algunas comparaciones de los buses utilizados en las industrias, con esto se tendrá más claras las características de cada uno de estos [45]:

Tabla 5: *Características de varios buses de comunicación.*[45].

Nombre	Comunicación	Topología	Maximo Dispositivo	Bps	Soporte
HART	Maestro/Esclavo	---------	15p/segm	1,2 K	Par trenzado
MODBUS	Maestro/Esclavo	Línea, Estrella, Árbol, Red con segmentos	1250p/segm	1,2 K a 115,2 K	Par trenzado, Coaxial, Radio
Device Net	Maestro/Esclavo, Multimaestro, Peer to Peer	Troncal, Puntual, Bifurcación	2018 nodos	500 K	Par trenzado, Fibra óptica
Interbus	Maestro/Esclavo	Segmentado	256 nodos	500 K	Par trenzado, Fibra óptica
LON Works	Maestro/Esclavo, Peer to Peer	Bus, Anillo, Estrella, Lazo	32768 dom	500 K	Par trenzado, Coaxial, Radio, Fibra óptica
Foundation Fieldbus	Single, Multimaestro	Estrella	240 p/segm, 32,768 sist	100 M	Par trenzado, Fibra óptica
Profibus DP	Maestro/Esclavo, Peer to Peer	Linea, estrella, anillo	127 segm	1,5 M y 12 M	Par trenzado, Fibra óptica
Profibus PA	Maestro/Esclavo, Peer to Peer	Linea, estrella, anillo	14400 segm	31,5 K	Par trenzado, Fibra óptica
Profibus FMS	Maestro/Esclavo, Peer to Peer	--------	127 segm	500 K	Par trenzado, Fibra óptica

Elaborado por: *La Investigadora.*

Ventajas del bus de campo

- **Flexibilidad:** Los sistemas podrán adaptarse a nuevos requisitos y de esta forma se podrán reutilizar en un futuro [47].
- **Velocidad:** Con la reducción de cableado la instalación y mantenimiento de la red de bus es más rápida [47].

- **Uniformidad:** Se puede intercambiar equipos de diferentes fabricantes porque los protocolos y normas unificarán técnicas [47].
- **Fiabilidad:** Aumenta la fiabilidad y disponibilidad en los trayectos de señal [47].

Desventajas del Bus de campo

- **Costo:** Aumentará considerablemente el costo de los equipos dependiendo de los diferentes estándares [47].
- **Complejidad:** Se requerirá un personal más cualificado para la manipulación de estas redes [47].
- **Averías en el Bus:** Se puede desconectar un sensor o actuador y en estos casos se debe utilizar redes de buses adicionales [47].

1.3.11 Sistemas distribuidos

Los sistemas distribuidos están relacionados con la evolución de los sistemas informáticos. debido a las necesidades que la tecnología ofrece. Son sistemas que ayudan a la obtención de resultados mediante dos o más máquinas, las cuales tienen como objetivo ser eficientes, escalables, fiables y tener disponibilidad en su trabajo. También es definido como un sistema que se comunica mediante una subred que interactúa y coordina acciones mediante mensajes; implícitamente realiza una funcionalidad jerarquizada. Existen tres modelos de comunicación en los sistemas distribuidos los cuales son [48], [49]:

- Modelo cliente – servidor.
- Modelo publicación – suscripción.
- Modelo productor – consumidor

Lo más interesante de implementar sistemas distribuidos es la utilización de herramientas en el diseño, la desintegración y programación, porque conlleva la coordinación, supervisión y control de un proceso. La Figura 23 detalla un diagrama básico de los Sistemas de Distribución [49]:

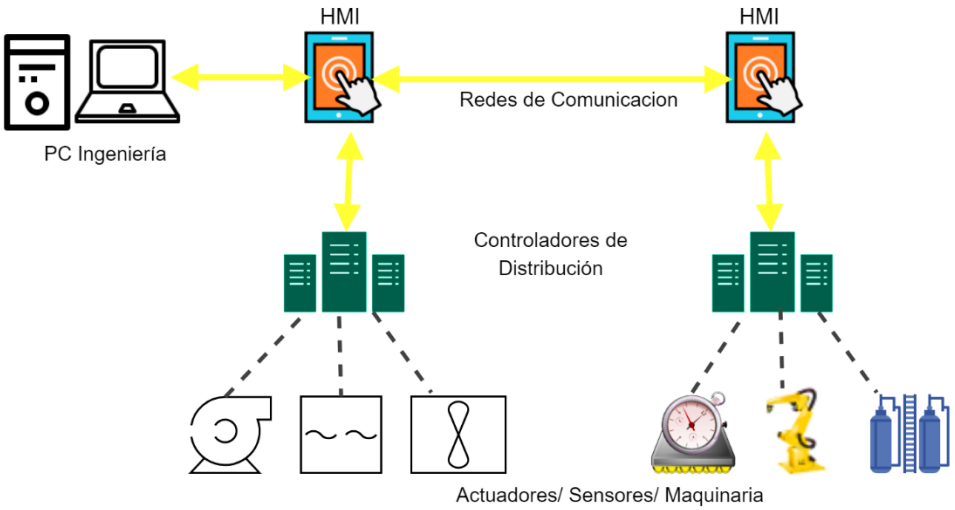


Figura 23: *Diagrama de un Sistema Distribuido.*
Elaborado por: *La Investigadora.*

Por otro lado, un sistema de control distribuido es un enlace de varios nodos distribuidos y enlazados a sensores como también a los actuadores mediante una red de comunicación, la cual intercambia información a través de mensajes que circulan por todos los nodos disponibles en la red [49].

Ventajas [6]**:**

- Con respecto a costos, los sistemas distribuidos proveen microprocesadores con una relación precio / rendimiento mejor que las computadoras centrales.
- En cuanto a velocidad tienen mayor procesamiento de cómputo que las computadoras centralizadas.
- Se puede implementar aplicaciones en máquinas remotas por lo que su distribución esta adjunta unas de otras.
- Tienen crecimiento proporcional ya que cada vez se requiere de mayor procesamiento en cuanto a los sistemas.

Desventajas [6]**:**

- Los softwares para los sistemas distribuidos están siendo perfeccionados.
- Aún existen problemas en la transmisión de datos.
- Se debe mejorar la seguridad en el acceso de información.

1.4 Objetivos

1.4.1 Objetivo general

Integrar la capa de comunicación de la Norma IEC-61499 con protocolos de la Industria 4.0, para lo cual, se puede configurar y añadir cualquier protocolo en la estructura estándar de comunicación que utiliza esta norma para controlar los sistemas descentralizados de las empresas industriales.

1.4.2 Objetivos específicos

Por lo tanto, para alcanzar el objetivo principal de la investigación es necesario realizar las siguientes actividades que son detalladas en cada uno de los objetivos específicos:

Definir los protocolos utilizados en la Industria 4.0 que se puedan integrar en la comunicación para el control de procesos industriales.

- Estudio de la norma IEC-61499 específicamente los postulados Parte 1 y 2.
- Análisis del modelo de comunicación que utiliza la Industria 4.0.
- Revisión de los protocolos más utilizados en la capa de aplicación de la Industria 4.0.

Diseñar la capa de comunicación basada en la Norma IEC-61499 con protocolos de la Industria 4.0 y el control de la Estación Festo MPS-200 empleando Bloques de Funciones.

- Utilización de los entornos de desarrollo 4DIAC establecidos por la Norma.
- Selección de dispositivos que trabajen en Run Time.
- Simulación de la capa de comunicación del control de la estación Festo MPS-200.
- Elaboración de los Bloques Funcionales Publicador/Suscriptor de la capa de comunicación.
- Análisis de la función de transferencia del controlador de la de la estación Festo MPS-200.

- Configuración de los controladores de posicionamiento de la estación Festo MPS-200.
- Exportación de los archivos de configuración correspondientes a los Bloques Funcionales diseñados en el entorno de desarrollo de 4DIAC.

Implementar en dispositivos de bajo costo los Bloques de Funciones para la comunicación y el control del funcionamiento de la estación Festo MPS-200.

- Puesta en marcha de los archivos de configuración en el dispositivo de Run Time.
- Edición de los archivos de configuración de las Bloques Funcionales diseñados anteriormente con las librerías correspondientes.
- Ejecución de los runtimes: Primero runtime de comunicación y Segundo runtime de control de estación.
- Implementación del dispositivo de Run Time conjuntamente con la estación Festo MPS-200.
- Pruebas de funcionamiento y corrección de errores en los diseños.

CAPÍTULO II

2 METODOLOGÍA

Para el diseño y desarrollo del presente proyecto de investigación es necesario recalcar los siguientes materiales: proyectos de investigación relacionados con el tema principal, artículos científicos publicados en revistas nacionales e internacionales y también se emplearon fuentes bibliográficas del repositorio que provee la Universidad Técnica de Ambato. Además, se solicitó las instalaciones de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial facilitando los Laboratorios de: "Robótica" y "Automatización e Hidráulica", puesto que se requiere de PLC´s y principalmente la Estación Festo MPS-200 para realizar pruebas de funcionamiento del sistema de control y de esta forma validar el proyecto de investigación.

2.1 Materiales para el desarrollo de la capa de comunicación

Los recursos necesarios para el desarrollo del proyecto son proporcionados por la Facultad de Ingeniería en Sistemas, Electrónica e Industrial y elementos propios de la autora, a continuación, se enlistan cada uno de los elementos:

- PLC S7-1200 (Control Lógico Programable)
  - Módulo CM 1243-5
- Estación Festo MPS-200 – Clasificador
  - Elementos del Clasificador
- PC con TIA Portal

2.1.1 PLC S7-1200 (Control Lógico Programable)

Un PLC es un dispositivo inteligente que sirve para automatizar cualquier proceso industrial, utiliza una memoria programable para almacenar instrucciones y ejecutarlas según la disposición de sensores/actuadores del proceso y utiliza varios lenguajes de programación comprensibles para el usuario. El PLC es parte del desarrollo del proyecto ya que funciona como conexión del proceso con el control virtual. Se conecta mediante Ethernet a la Raspberry Pi y con ayuda de uno de sus módulos CM 1243-5 se interconecta por medio de Profibus a la válvula GSD, asimismo en la CPU está cargado la configuración de reconocimiento del módulo con la válvula que funciona como entrada al mando de la Estación Festo MPS-200.

Específicamente el modelo S7-1200 brinda una potencia y flexibilidad para controlar cualquier dispositivo. En la CPU se incluye un microprocesador, circuitos de Entrada/Salida, puertos de Profinet y una fuente de alimentación. En la Tabla 6 se describen características técnicas del modelo S7-1200 CPU 1214C y en la Figura 24 se observa el modelo de PLC con su módulo CM 1243 [50]:

Figura 24: *Módulo de Profibus CM 1243-5 y PLC S7-1200 CPU 1214C. [49].*

Tabla 6: *Características técnicas del PLC S7-1200 CPU 1214C. [49].*

Función		**CPU 1214C**
Memoria de usuario	Trabajo	75 KB
	Carga	4 MB
	Remanente	10 Kb
E/S integradas locales	Digital	14 entradas / 10 salidas
	Analógica	2 entradas
Tamaño de la memoria imagen de proceso	Entradas (I)	1024 bytes
	Salidas (Q)	1024 bytes
Área de marcas (M)		8192 bytes
Ampliación con módulo de señales (SM)		8
Signal board (SB), Battery Board (BB) o Communication Board (CB)		1
Módulo de comunicación (CM)		3
Contadores rápidos	1 MHz	----
	100/80 KHz	De Ia.0 a Ia.5
	30/20 KHz	De Ia.6 a Ib.5
Salidas de impulsos	1 MHz	-----
	100 KHz	De Qa.0 a Qa.3
	20 KHz	De Qa.4 a Qb.1
Memory Card		SIMATIC Memory Card
Puerto de comunicación Ethernet Profinet		1

Elaborado por: *La Investigadora*

- **Módulo PROFIBUS CM 1243-5 (Función de Maestro)**

Es un módulo de comunicación que conecta un S7-1200 a un bus de campo Profibus permitiendo una periferia descentralizada para accionamientos de sensores y actuadores de varios fabricantes. En la Tabla 7 se describen características técnicas del modelo CM 1243-5 PROFIBUS [51]:

Tabla 7: *Características técnicas del PLC S7-1200 CPU 1214C.* [51]

Datos Técnicos	
Referencia	CM 1243-5 modelo maestro DP clase 1
Interfaces	
Conexión a PROFIBUS	Conexión hembra Sub-D de 9 polos
Consumo máximo de corriente	15mA a 5V (solo para la terminación de bus)
Tipo de Comunicación	
Áreas de datos del maestro DP	
Tamaño de las áreas de datos	max. 1024 bytes
Área de entrada	max. 512 bytes
Área de salida	max. 512 bytes

Elaborado por: *Investigadora*

2.1.2 Estación Festo MPS-200 – Clasificador

La Estación Festo es un Sistema de Producción Modular que se forma de tres partes y funcionan como subprocesos industriales los cuales pueden ser automatizados de forma interactiva, en la implementación de este proyecto solo se utilizó la parte de clasificación de piezas. El Módulo Clasificador sirve para ordenar piezas de colores en tres rampas según su color y material. El primer sensor óptico detecta la presencia de las piezas que se colocan al inicio de la banda transportadora, luego el segundo sensor detecta las características de las piezas según su color y el tercer sensor inductivo detecta el tipo de material de la pieza. Seguidamente se activan los brazos que empujarán la pieza a su respectiva rampa. Al inicio de cada rampa de clasificación se encuentra un sensor retro – reflectivo para saber si la pieza ya paso a su destino. Si la pieza activa el sensor por un tiempo prolongado quiere decir que, las rampas ya están llenas y consecuentemente se desocuparan para continuar el proceso. La Figura

25 muestra la Estación Festo de Clasificación indicando las partes principales del proceso [52]:

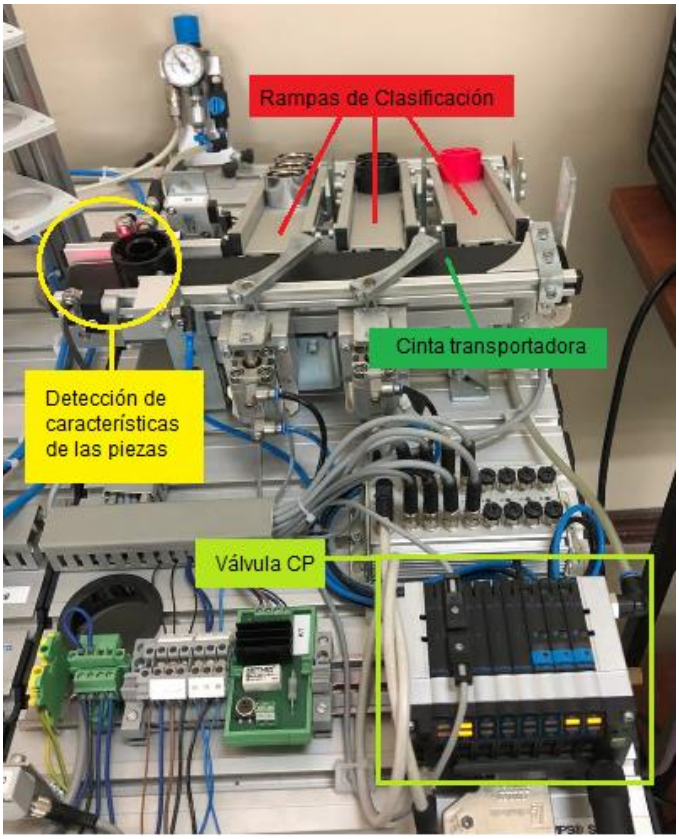


Figura 25: *Estación Festo MPS-200-Clasificador.*
Elaborado por: *La Investigadora.*

- **Elementos con los que trabaja el Módulo Clasificador**

El proceso de clasificación de piezas tiene sensores, actuadores, el bus de comunicación y una etapa neumática. En la Tabla 8, se enlistan cada uno de estos elementos con una corta explicación [52]:

Tabla 8: *Elementos de la Estación Festo MPS-200 módulo Clasificación. [51].*

Sección	Elemento	Función
Detección	2 sensores ópticos	Un sensor detecta la presencia de cualquier pieza por reflexión de la luz roja visible que el sensor emite. El otro sensor detecta la reflexión de la luz infrarroja emitida por el sensor, al chocar con la pieza roja o negra.
	Sensor inductivo	Detecta la presencia de metal en la pieza.
	Tope neumático con cilindro de carrera corta de simple efecto	Detiene las piezas hasta que el sensor óptico e inductivo identifiquen las características de color y material de las piezas y así activar la rampa que le corresponde.
Transporte	Módulo derivador neumático	Estos módulos se encuentran en los costados de la cinta transportadora para desviar las piezas a su destino final. Es un cilindro de doble efecto.
	Cinta transportadora	Se acciona mediante un motor dc de 24V DC.
Rampa de clasificación	Rampa	Almacena las piezas.
	Sensor de retro - reflexión	Detecta que la rampa está llena y ayuda al control de envío de piezas.
Terminal de válvulas CP con Profibus DP	Esclavo Profibus DP con 3 electroválvulas	Conecta todas las entradas y salidas del proceso para transportar la información al PLC mediante un bus de campo PROFIBUS - DP
	Módulo de entradas CP con 16 entradas con Profibus - DP	

Elaborado por: *La Investigadora.*

2.1.3 PC con TIA Portal

Como parte del desarrollo de todo el proyecto se requirió de una PC con el software TIA Portal. Este software se utiliza para el reconocimiento del PLC con la válvula GSD de la Estación y de esta forma exista conectividad entre estos dispositivos y también pueda tener compatibilidad de comunicación con la Raspberry.

2.2 Métodos

2.2.1 Modalidad de la investigación

El proyecto de investigación cuenta con una modalidad I+D (Investigativo y de Desarrollo) a consecuencia de que los conocimientos teóricos e investigados se complementan para implementar el proyecto, abriendo paso a un nuevo campo de aplicación en el entorno de programación acerca de sistemas descentralizados en las industrias. Además, se justifica su desarrollo mediante diferentes modalidades de investigación:

2.2.1.1 Investigación Aplicada

Los conocimientos adquiridos a lo largo de la malla curricular han sido de gran ayuda para el desarrollo del proyecto esencialmente los módulos de: PLC´s, comunicaciones avanzadas y programación ya que su información es importante para resolver la problemática específica de la investigación.

2.2.1.2 Investigación Bibliográfica

El proyecto se justificó y fundamentó con datos informáticos confiables provenientes de fuentes bibliográficas como: artículos científicos, libros, revistas nacionales e internacionales, proyectos de investigación y también tesis doctorales.

2.2.1.3 Investigación de Campo y Experimental

La manipulación de la estación FESTO MPS-200 sirvió para visualizar y analizar su funcionamiento y por consiguiente se realizaron pruebas y se validó la capa de comunicación para este sistema. Todo esto se realizó en los laboratorios de la Facultad

de Ingeniería en Sistemas Electrónica e Industrial de la Universidad Técnica de Ambato.

2.2.2 Recolección de información

La información requerida en el desarrollo del proyecto como ya se mencionó, es a partir de fuentes bibliográficas de los últimos años acerca de procesos industriales, programación descentralizada y protocolos de comunicación siempre y cuando estén basadas en estándares y normas de Industria 4.0, obteniendo así una investigación fidedigna.

2.2.3 Procesamiento y análisis de datos

En el procesamiento y análisis de datos se procedió de la siguiente forma:

- Analizar los puntos de vista de las investigaciones previas para discernir ideas estratégicas en el desarrollo del presente proyecto.
- Establecer una solución fiable al problema de investigación.
- Pruebas de funcionamiento.
- Interpretación y análisis de resultados.

2.2.4 Desarrollo del proyecto

Se llevó a cabo las siguientes actividades para cumplir con los objetivos planteados en el proyecto de investigación:

1. Estudio de la Norma IEC-61499 específicamente los postulados Parte 1 y 2.
2. Análisis del modelo de comunicación que utiliza la Industria 4.0.
3. Revisión de los protocolos más utilizados en la capa de aplicación de la Industria 4.0.
4. Utilización de los entornos de desarrollo 4DIAC establecidos por la Norma.

5. Selección de dispositivos que trabajen en Run Time.
6. Simulación de la capa de comunicación del control de la estación Festo MPS-200.
7. Elaboración de los Bloques Funcionales Publicador/Suscriptor de la capa de comunicación.
8. Análisis de la función de transferencia del controlador de la de la estación Festo MPS-200.
9. Configuración de los controladores de posicionamiento de la estación Festo MPS-200.
10. Exportación de los archivos de configuración correspondientes a los Bloques Funcionales diseñados en el entorno de desarrollo de 4DIAC
11. Puesta en marcha de los archivos de configuración en el dispositivo de Run Time.
12. Edición de los archivos de configuración de las Bloques Funcionales diseñados anteriormente con las librerías correspondientes.
13. Ejecución de los runtimes: Primero runtime de comunicación y Segundo runtime de control de estación.
14. Implementación del dispositivo de Run Time conjuntamente con la estación Festo MPS-200.
15. Pruebas de funcionamiento y corrección de errores en los diseños.
16. Elaboración de un informe final.

CAPÍTULO III

3 RESULTADOS Y DISCUSIÓN

3.1 Introducción de la Propuesta

La tecnología avanza a pasos agigantados y se debe ir a la par de la misma, por este motivo todas las empresas industriales deben estar aptas a los cambios que esto trae. Por lo que, la automatización de procesos está presente en la mayoría de estas empresas tratando de solucionar y mejorar el funcionamiento de su producción. Para que todo funcione en conjunto la comunicación entre estos procesos debe ser eficaz y eficiente. Otro motivo no menos importante que se lleva a cabo en el desarrollo de este proyecto, es la reutilización de código que permite a otros desarrolladores adaptar nuevos diseños de programación y continuamente mejorar la comunicación de cualquier proceso.

En tal contexto, se desarrolló una arquitectura de comunicación basada en la Norma IEC-61499 con el protocolo MQTT que está dentro de los estándares de Industria 4.0, de acuerdo a esto, abrir campo en el desarrollo de nuevos modelos de conexión para los sistemas distribuidos. Esta arquitectura fue diseñada en base a los módulos de comunicación de la Norma, por lo que, contiene una estructura Publicador/Suscriptor para comunicar las acciones de control de un proceso. Mediante el software CMAKE se configuró los archivos que contienen las librerías de Mosquitto con el protocolo MQTT y también la librería de SNAP7 con la cual se puede acceder al control del PLC. Seguidamente de la configuración se construye la aplicación de comunicación con sus respectivas librerías y de esta forma proporcionar una conexión segura y confiable.

Para validar esta propuesta y verificar la eficiencia de la capa de comunicación se implementó adicionalmente un algoritmo de control para la Estación Festo MPS-200-

Clasificador. Por lo cual, el funcionamiento de la Estación consiste en trasladar piezas de colores mediante una banda móvil a sus respectivas rampas según corresponda, un sensor detecta el color y el otro sensor el tipo de material, seguidamente un pistón deja pasar a la pieza mientras que su respectivo brazo lo desliza al lugar que le corresponde. Este algoritmo fue realizado con Bloques de Funciones en el software 4DIAC, y esta está conectado mediante Profibus desde el PLC hacia la válvula GSD de la Estación y mediante Ethernet desde el PLC hacia la Raspberry Pi. La Figura 26 ilustra de forma generalizada el proyecto final:

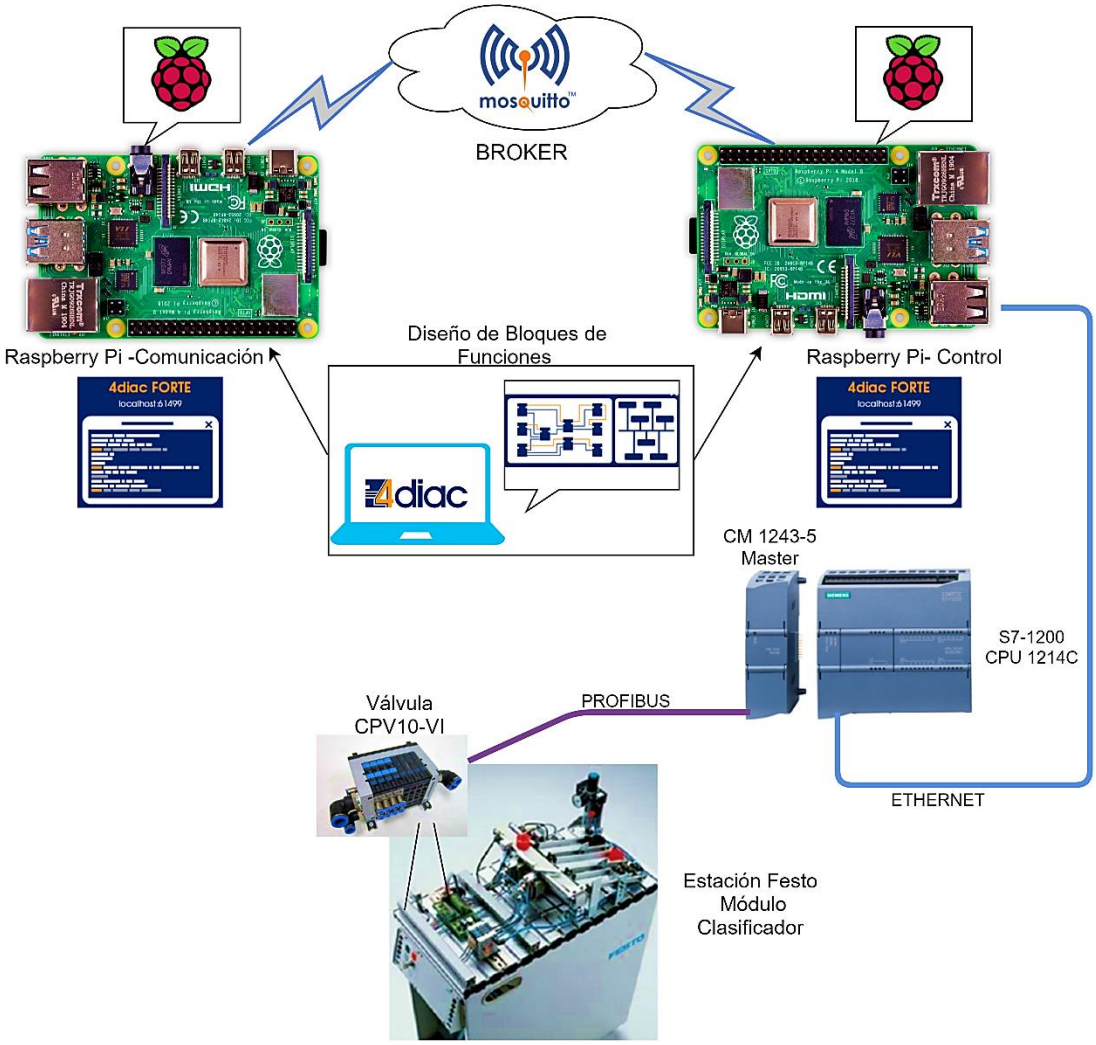


Figura 26: *Esquema general del proyecto.*
Elaborado por: *La Investigadora.*

3.2 Diagrama esquemático de la Arquitectura de Comunicación

La Figura 27 presenta el esquema del funcionamiento de la capa de comunicación, donde se puede observar que mediante 4DIAC se crea la aplicación de la capa basada en Bloques de Funciones Publicador/Suscriptor, posteriormente se ejecuta el programa para verificar su funcionamiento, además, la plataforma 4DIAC permite analizar y visualizar si los datos son enviados y recibidos mediante la opción mapear.

El programa que se generará en 4DIAC, son estructuras de C++ que contienen un cuerpo y una cabecera los cuales son exportados a FORTE para poder construir los archivos de configuración mediante CMAKE. Por lo cual, en el dispositivo Raspberry primero se construye la aplicación con las respectivas librerías estándar para luego editarlas y crear el nuevo módulo con las librerías de SNAP7, MQTT y Mosquitto para la comunicación de datos. Se debe respetar la estructura del cuerpo y la cabecera que generan las librerías estándar, para que en la ejecución del programa no existan inconvenientes. Entonces la capa de comunicación es reconfigurable para cualquier protocolo de la Industria 4.0

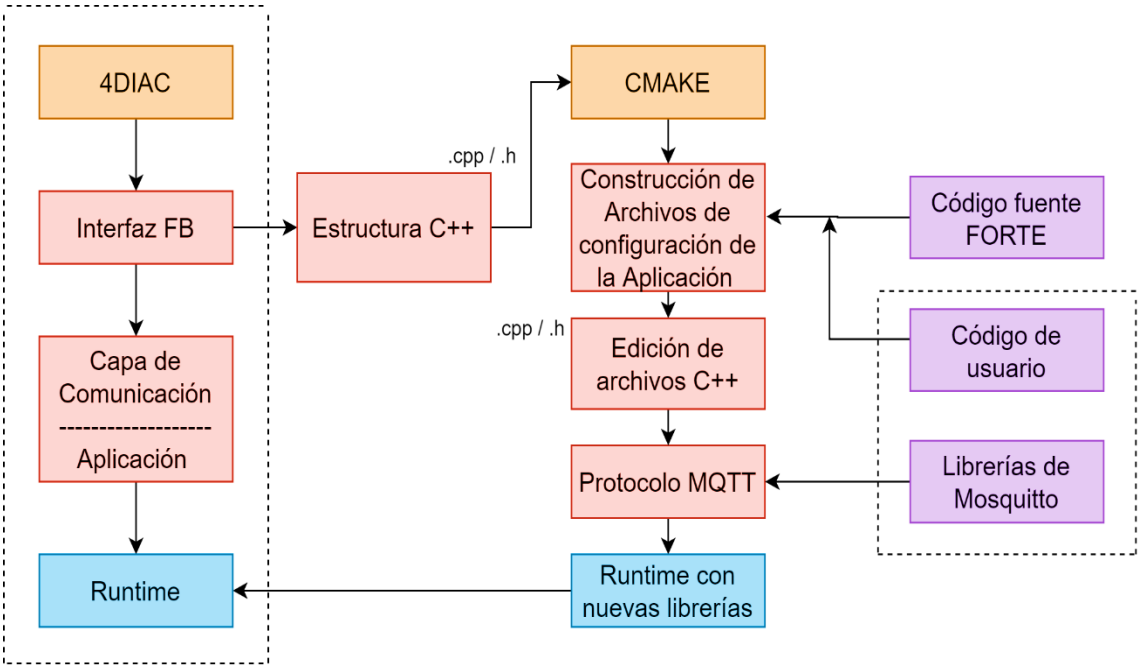


Figura 27: *Arquitectura de la capa de comunicación.*
Elaborado por: *La Investigadora.*

3.3 Esquema del Control de la Estación Festo MPS-200 – Clasificador

Para el control de la estación se requieren dos programas con la misma funcionalidad, el primero se realizará en 4DIAC con un diseño de Bloques de Funciones, el segundo será un archivo C++. Los dos controles tienen la finalidad de complementarse al momento de controlar el funcionamiento de la estación, puesto que, existen sensores analógicos entonces, el archivo en C++ contiene la configuración de lectura de tipo byte que se obtiene de los sensores, además el control en C++ sirve como guía para verificar el funcionamiento de la Estación. El diseño de FBs de control después de ser exportados a FORTE serán editados para incluir la configuración del archivo C++ y de esta forma se genere mediante CMAKE un FORTE que cumpla dichas funciones correctamente. La Figura 28 es un diagrama esquemático del control de la estación Módulo Clasificador.

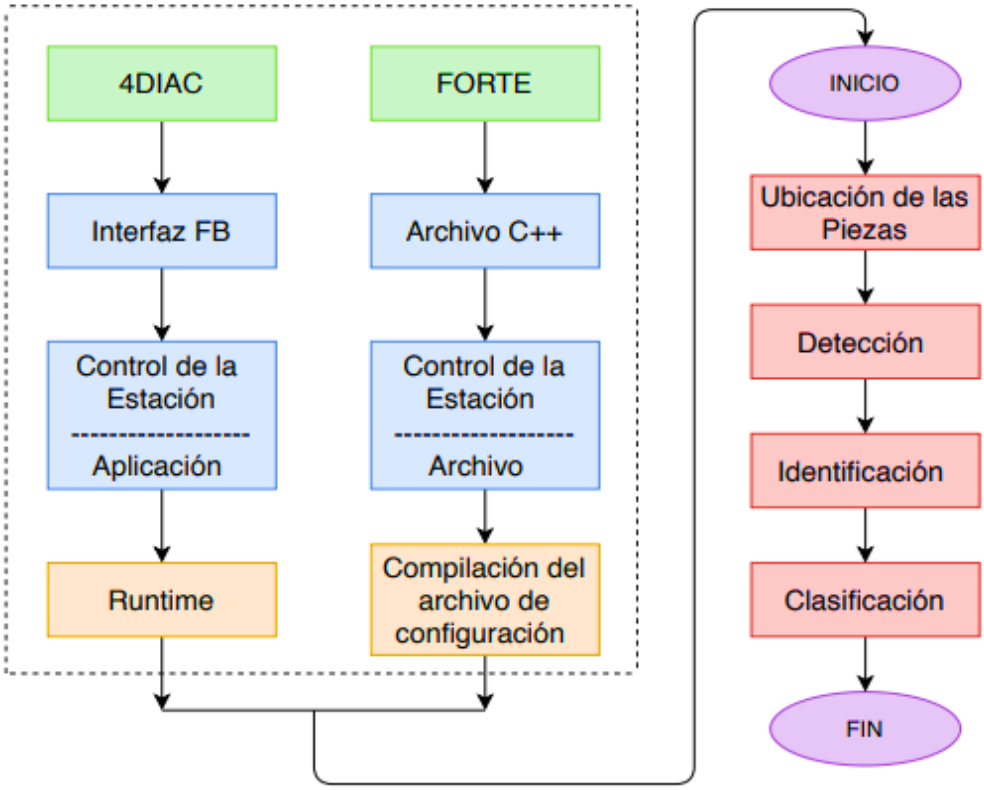


Figura 28: *Esquema de control de la Estación MPS-200 – Clasificador.*
Elaborado por: *La Investigadora.*

El funcionamiento lógico del programa consiste en dar inicio al sistema prendiendo la banda transportadora, luego se ubican las piezas manualmente. El programa detecta e identifica la pieza para posteriormente activar los brazos que son los encargados de la

clasificación de las mismas, las cuales son deslizadas a su rampa correspondiente, después el sensor retro – reflectivo detecta la llegada de la pieza y envía una señal como alarma para que se envíe otra.

3.4 Selección de los elementos para la implementación de la Capa de Comunicación.

Se va seleccionar elementos y/o dispositivos para la implementación de la capa de comunicación de acuerdo a la arquitectura de comunicación y el esquema de control de la estación presentados en las Figuras 27 y 28. Por lo cual, se realiza una comparación de opciones disponibles en cuanto a los dispositivos y también se plantea una vista general en cuanto a las características de los protocolos de comunicación que se utilizan en la Industria 4.0.

3.4.1 Software de Bloques de Funciones

Para el diseño de la capa de comunicación y el control de estación mediante Bloques de Funciones se optó por el software 4DIAC (For Distributed Industrial Automation and Control) basado en la Norma IEC-61499. Es un entorno IDE con herramientas para construir aplicaciones que sirvan para automatizar y controlar un proceso, estas aplicaciones se pueden descargar en dispositivos Runtime. Las características más relevantes de este software son las siguientes [5]:

- Es una herramienta de desarrollo basada en Eclipse bajo la Norma IEC-61499.
- Tiene diversos tipos de Bloques de Funciones, además se pueden crear nuevos modelos de FBs.
- Sus conexiones son mediante eventos y datos.
- Se puede ejecutar en plataformas como: Linux, Solaris y Windows.
- Sus datos también se basan en la Norma IEC-61131-3.

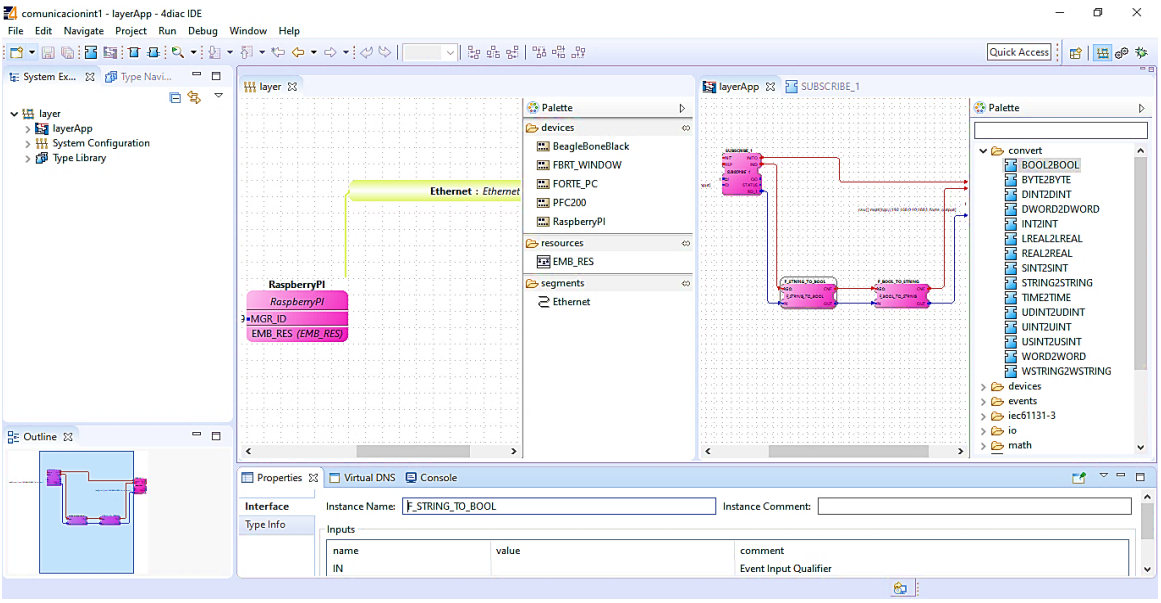


Figura 29: *Entorno IDE 4DIAC.*
Elaborado por: *La Investigadora.*

En la Figura 29 se ilustra el entorno del IDE 4DIAC en su versión más actual 1.12.2 que permite la prueba y depuración de las aplicaciones, además cuenta con la facilidad de fijar valores a las variables de forma remota.

3.4.2 Software para la Ejecución en Tiempo Real

En cuanto al software de ejecución se tiene a FORTE el cual es compatible con el IDE 4DIAC, además facilita una arquitectura escalable la cual permite acoplarse a cualquier tipo de aplicación. Implementa código abierto en C++ y ejecuta todos los tipos de Bloques de Funciones bajo la Norma IEC-61499. Su ejecución es óptima ya que un proceso de baja prioridad no interrumpirá el proceso con mayor prioridad, el funcionamiento está basado en la forma FIFO (primero entra / primero sale), de esta forma los eventos se ejecutan de forma ordenada. Es portable para dispositivos pequeños que utilizan sistemas embebidos de 16 o 32 bits, lo cual quiere decir que FORTE puede funcionar independientemente de la plataforma en la que se desarrolla la aplicación a ejecutarse.

3.4.3 Dispositivo Runtime

Un dispositivo Runtime sirve para ejecutar en tiempo real los programas que se configuraron y diseñaron con anterioridad en sus respectivas plataformas. Este dispositivo contendrá las librerías necesarias para que los programas funcionen correctamente e indistintamente, para lo cual, el dispositivo a elegir debe tener las características técnicas más optimas en comparación con los demás.

En la Tabla 9 se describen las características de algunos dispositivos que se utilizan como máquina virtual para ejecutar programas en tiempo real. De acuerdo a sus hojas técnicas se enlistan los dispositivos más relevantes del mercado de automatización de bajo costo.

El dispositivo Runtime apto para el desarrollo del proyecto es la tarjeta Raspberry Pi 3 Modelo B+, la cual funciona de forma rápida en cuanto a la ejecución de sus programas ya que su procesador es de cuatro núcleos a una velocidad de reloj de 1.4 GHz. La tarjeta cuenta con 4 puertos de USB versión 2.0 lo que permite la conexión de periféricos, tiene 1GB de memoria RAM, cuenta con conexión inalámbrica lo cual facilita su utilización en varios ámbitos de trabajo. Esto significa que se la puede utilizar de forma remota desde varios accesos mientras se encuentra en una misma red. Todas estas utilidades hacen que esta tarjeta sea la mejor opción en comparación con los demás dispositivos que se describe en la Tabla 10, además su tamaño es compacto y adaptable a cualquier ambiente de trabajo para lo cual, sirve como elemento principal en la realización de la capa de comunicación para el control de una estación industrial.

Tabla 9: *Tipos de dispositivos Runtime. ANEXOS 1,2,3,4,5.*

Dispositivo	**Imagen**	**Características técnicas**	
RASPBERRY PI 3 MODELO B		**Procesad or**	Broadcom BCM2837, Cortex-A53 (ARMv8) 64-bit SoC
		Frecuencia de reloj	1,2 GHz
		Memoria	1GB LPDDR2 SDRAM
		Conectividad inalámbrica	2.4GHz IEEE 802.11.b/g/n Bluetooth 4.1

		**Conectividad de red**	Fast Ethernet 10/100 Gbps
		Puertos	GP IO 40 pines HDMI 4 x USB 2.0 CSI (cámara Raspberry Pi) DSI (pantalla tácil) Toma auriculares / vídeo compuesto Micro SD Micro USB (alimentación)
		Sistema operativo	Raspbian, Linux, Ubuntu
RASPBERRY PI 3 MODELO B+		**Procesador**	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC
		Frecuencia de reloj	1,4 GHz
		Memoria	1GB LPDDR2 SDRAM
		Conectividad inalámbrica	2.4GHz / 5GHz IEEE 802.11.b/g/n/ac Bluetooth 4.2, BLE
		Conectividad de red	Gigabit Ethernet over USB 2.0 (300 Mbps de máximo teórico)
		Puertos	GPIO 40 pines HDMI 4 x USB 2.0 CSI (cámara Raspberry Pi) DSI (pantalla tácil) Toma auriculares / vídeo compuesto Micro SD Micro USB (alimentación) Power-over-Ethernet (PoE)
		Sistema operativo	Raspbian, Linux, Ubuntu
UMIC.200		**Procesador**	Procesador RISC de 32 bits Sitara ™ ARM® Cortex®-A8
		Frecuencia de reloj	1 GHz
		Memoria	512MB DDR3 RAM / 8GB Flash / Tarjeta MicroSD 32KB FRAM

		Conectividad de red	2 x Ethernet interface (IEEE 802.3) 2 x CAN interface (isolated) 2 x RS-232 interface (isolated)
		Puertos	Puerto USB 2.0 para la conexión de varios dispositivos USB. Ocho E / S digitales, entrada / salida configurable mediante software, máx.
		Sistema operativo	Linux, Kernal 4.1.18
BEAGLEBONE BLACK		**Procesador**	Sitara AM3359AZCZ100 1GHz, 2000 MIPS
		Frecuencia de reloj	1GHz.
		Memoria	512MB de memoria RAM DDR3. y la posibilidad de ampliarla mediante una memoria USB o una micro-SD.
		Conectividad de red	Tarjeta de red Fast Ethernet 10/100Mbps.
		Puertos	Salida de audio y vídeo a través del puerto mini-HDMI. 1 puerto USB 2.0.
		Sistema operativo	Debian, Ubuntu, LXDE
ODROID - XU4		**Procesador**	Samsung Exynos5422. 4 x Cortex-A7
		Frecuencia de reloj	1.4GHz
		Memoria	RAM: 2Gbyte LPDDR3 y MicroSD Card Slot hasta 64GByte
		Conectividad de red	Ethernet LAN 10/100Mbps y Gigabit
		Puertos	Vídeo: HDMI 1.4a Audio: solo por HDMI sin otro tipo de salida y USB2.0. 30 + 12 Pin : GPIO/IRQ/SPI/ADC
		Sistema operativo	Ubuntu, Android

Elaborado por: *La Investigadora.*

3.4.4 Protocolos de comunicación basados en Industria 4.0

La comunicación entre sistemas es de primordial importancia en las industrias de todo el mundo, por lo cual se crean constantemente protocolos y estandarizaciones para que estos sistemas puedan comunicarse entre ellos y sus operarios. Algunos de los protocolos más utilizados en la Industria 4.0 se muestran en la Tabla 10, aquí se enlistan sus características principales y de esta forma tener una visión general de la utilización de cada uno de estos protocolos ya que los campos de comunicación de los sistemas son de amplia variedad.

En vista que el proyecto se basa en una capa de comunicación para los sistemas industriales se eligió trabajar con el protocolo MQTT puesto que sirve como transporte de mensajes tipo Cliente/Servidor; esto quiere decir que, se puede publicar acciones y suscribirse para ver los resultados en el proceso. Este protocolo es eficiente ya que actualmente es el más utilizado en IIOT (Internet Industrial de la Cosas), puesto que no tiene pérdida de datos y es bidireccional, además el transporte de datos es optimizado y no produce tráfico en la red. MQTT es adecuado en este tipo de aplicaciones porque la cantidad de información que se envía no utiliza un amplio ancho de banda.

Tabla 10: *Algunos protocolos de comunicación en la Industria 4.0.* [44], [53],[54].

Características	**PROTOCOLOS**					
	AMQP	**CoAP**	**DDS**	**MQTT**	**XMPP**	**HTTP**
Nombre	Protocolo Avanzado de Cola de Mensajería	Protocolo de Aplicación Restringido	Servicio de Distribución de Datos para Sistemas en Tiempo Real	Transporte de Telemetría del Servidor de Cola de Mensajes	Mensajería Extensible y Protocolo de Presencia	Protocolo de Transferencia de HiperTexto
Transporte	TCP/IP	UDP/IP	UDP/IP TCP/IP	TCP/IP	TCP/IP	TCP/IP
Modelo	Intercambio de mensajes punto a punto	Petición/ Respuesta (REST)	Publicación/ Suscripción Petición/ Respuesta	Publicación/ Suscripción	Publicación/ Suscripción	Publicación/ Suscripción
Ámbito de aplicación	D2D D2C C2C	D2D	D2D D2D D2C C2C	D2C	D2C	D2C
Datos principales	Codificados	Codificados	Declarados codificados	No Definidos	No Definidos	Codificados
Seguridad	TLS	DTLS	TLS, DTLS, DDS	TLS	TLS, SASL	SSL, TLS
Prioridad de datos	-	-	Prioridad de transporte	-	-	-
Tolerancia de fallos	Se especifica en la implementación	Descentralizado	Descentralizado	El nodo central es el punto único de fallo (SPoF)	Se especifica en la implementación	Descentralizado

Elaborado por: *La Investigadora*

3.5 Sistema operativo y software de herramientas para Raspberry Pi

Los sistemas operativos funcionales que utiliza la tarjeta Raspberry Pi son: Linux, Ubuntu y Raspbian, se puede utilizar cualquiera de estos, pero es recomendable utilizar el de su propia familia ya que es el más optimizado para el hardware. Los detalles y especificaciones de las herramientas de software que se añaden a la tarjeta se muestran a continuación:

3.5.1 Sistema operativo Raspbian

Raspbian es un sistema operativo de la versión GNU/Linux y basado en Debian que fue desarrollado específicamente para la tarjeta Raspberry, su función es gestionar los recursos que necesita el usuario para comunicarse con la tarjeta. Al ser parte de Linux cualquier software que sea de código abierto puede compilarse en la tarjeta, además proporciona un repositorio en el cual se puede descargar distintos programas para trabajar en el dispositivo. También se tiene LXDE (Entorno de Escritorio X11 Ligero) como escritorio y herramientas como IDLE (Desarrollo Integrado y Ambiente Orientado) para trabajar con lenguajes de programación como Scratch y Python. Todas estas características hacen que el dispositivo Raspberry sea similar a un ordenador personal y además funcione como un microcontrolador de alta funcionalidad. En el Anexo 6 se describe los pasos para la instalación del sistema operativo Raspbian [55].

3.5.2 Constructor CMake

CMake es una herramienta que se ejecuta en varias plataformas y permite generar instrucciones o scripts de compilación de código principalmente para lenguaje de programación C++. Las instrucciones que realiza son [56]:

- Crea un ejecutable a base de los archivos de origen.
- Se agrega una ruta para el archivo en relación a la ruta de inclusión cuando se compilo el proyecto.

- Define variables que después el sistema necesitara y se encontraran el directorios y subdirectorios.
- Localiza cualquier biblioteca que este dentro de los archivos de origen.

La Figura 30 muestra el proceso que realiza CMake al momento de compilar el proyecto y en el Anexo 7 se describe su instalación.

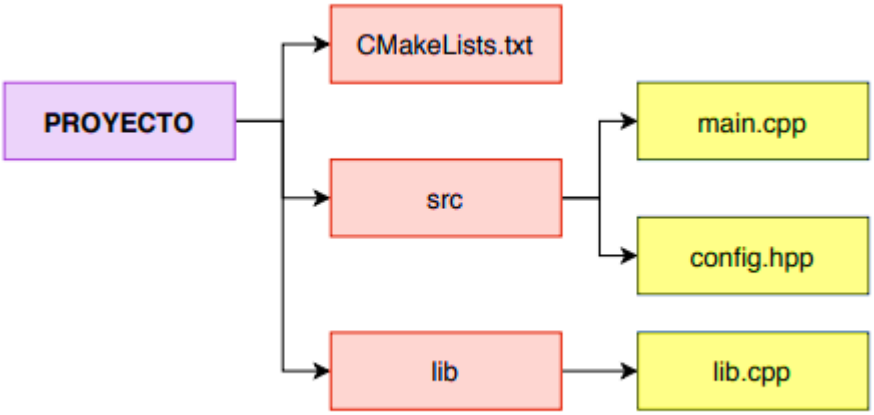


Figura 30: *Diagrama de bloques del proceso de construcción de archivos.*
Elaborado por: *La Investigadora.*

3.5.3 Bróker Mosquitto

Mosquitto es un proyecto de Eclipse que permite publicar y suscribir mensajes de código abierto, es práctico para mensajería de IoT (Internet de las Cosas) utilizando protocolo MQTT, además, el bróker Mosquitto contiene una librería de lenguaje C. El funcionamiento consiste en recibir y procesar datos que se envían desde los clientes para luego distribuir a quienes se suscriben al bróker. Los pasos para la instalación de Mosquitto se encuentran en el Anexo 8 [57].

3.5.4 SNAP7

SNAP7 es una plataforma de código abierto, funciona como un servidor, cliente y/o socio de los PLCs Siemens de las siguientes series: S7 300 y 400, también con los S7 1200 Y 15000. Fue diseñado para trabajar en la transferencia de grandes cantidades de información en un ambiente industrial. Algunas de sus características son [58]:

- SNAP7 no necesita de librerías externas.

- Utilizado en multiplataformas (Linux, Mac OS, Windows, etc) de 32 y 64 bits.
- Se pueden utilizar los siguientes lenguajes de programación: Pascal, Python, C/C++, NetMono, LabView.

La Raspberry Pi es una de las varias tarjetas microcontroladoras con las que trabaja SNAP7, en el Anexo 9 se puede visualizar la instalación.

3.6 Configuración para la interconexión física y virtual del PLC con la Estación MPS-200-Clasificador

Para que una red Profibus funcione con un autómata es necesario utilizar los archivos GSD de cualquier elemento físico que sea parte de la red; estos archivos son textos ASCII que incluye las especificaciones y en la mayoría de casos la imagen del dispositivo. Para este proyecto se necesitó el archivo GSD de la válvula CPV10-VI-DI0-8 y la instalación de los archivos GSD en TIA Portal, esto se encuentra en el Anexo 10. A continuación se describe como se realiza la interconexión virtual:

En la Figura 31 se muestra la selección del PLC que se está ocupando en la realización de este proyecto.

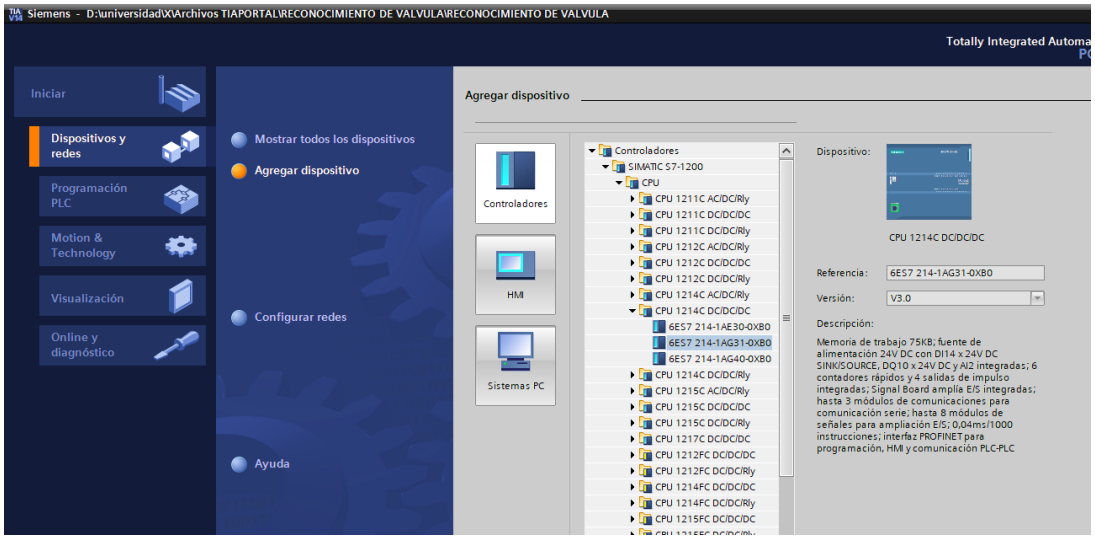


Figura 31: *Selección del modelo de PLC que se utiliza en el proyecto.*
Elaborado por: *La Investigadora.*

Después, en la sección de configuración de dispositivo específicamente, en la sección de “Catálogo” se coloca el tipo de módulo de comunicación, en este caso el Master

CM 1243-5. Luego se ubica el dispositivo junto al PLC y así de esta forma se emparejan los dos dispositivos. En la Figura 32 se puede observar el proceso.

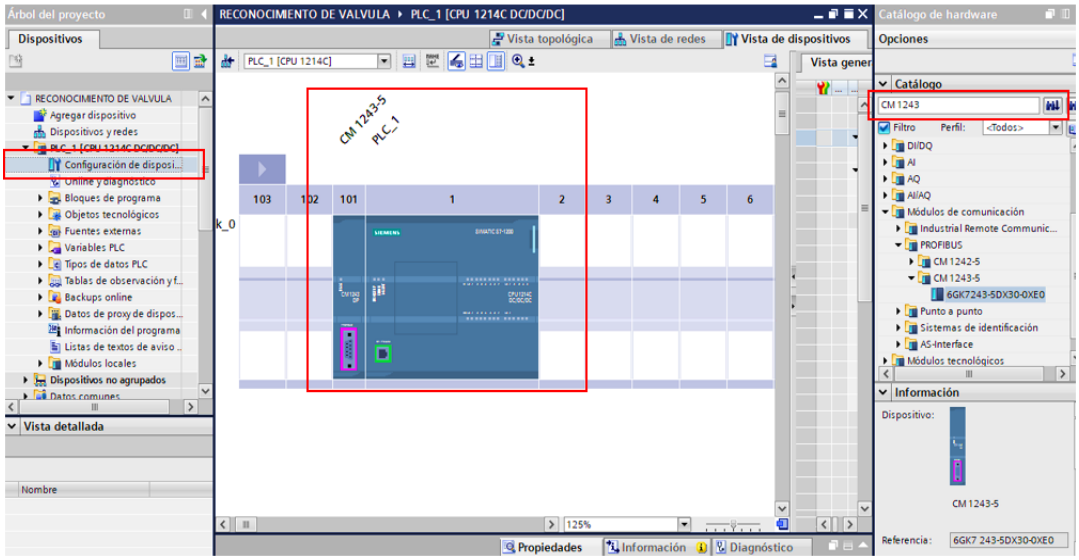


Figura 32: *Proceso para agregar el módulo de comunicación Master CM 1243-5..*
Elaborado por: *La Investigadora.*

Como último paso, en la pestaña de vista de redes, en la sección de "Catálogo" se coloca la válvula que interconecta el PLC con la Estación MPS-200. El modelo CPV10 DI0 se ubica en el área de topología y se conecta con el Master CM 1243-5. Finalmente, se manda a cargar la configuración al PLC y esta lista la interconexión física y virtual, esto permite conectarlo con la Raspberry Pi. En la Figura 33 se ilustra el proceso.

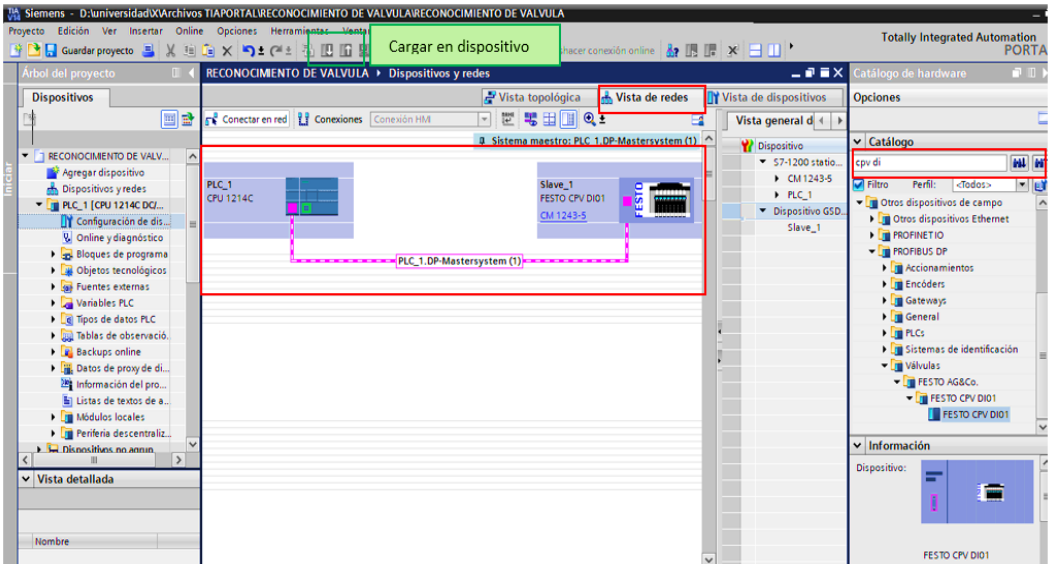


Figura 33: *Proceso para agregar el modelo de la válvula de la estación MPS-200.*
Elaborado por: *La Investigadora.*

3.7 Control de la Estación Festo MPS-200 - Clasificador utilizando C++

Para controlar el funcionamiento de la Estación Festo con la Raspberry por medio del PLC se utilizó la librería de SNAP7 la cual permiten conectar, leer y escribir datos en el microcontrolador. El archivo <controlplc.cpp> sirve como guía para obtener los valores analógicos de la Estación Módulo Clasificador, también, se puede escribir datos en las direcciones de salida del PLC. A continuación, se enlistan las funciones más relevantes de este archivo de control y en el Anexo 11 se encuentra la configuración completa:

- Librerías del sistema.

```
#include "snap7.h"
#include <iostream>
#include <string>
#include <curses.h>
```

- Instancia de lectura del sensor20 (Sensor de Presencia).

```
byte LeerSensorI20(){
int leer_sensor=Client->EBRead(2, 16, &Buffer_S_20)
byte byte_sensor20 = GetByteAt(Buffer_S_20, 0);//
printf("\nNumero de sensor20 Byte=:  %d\n", byte_sensor20);
return byte_sensor20;
}
```

- Instancia de escritura del actuador35 (Banda Transportadora).

```
bool BandaEstado(int bmem, int bposi, bool estado_banda){
  bool estado_banda_salida=estado_banda;
  if(estado_banda==true){
    int leer_banda=Client->ABRead(bmem, 16, &Buffer_Q_35[0]);
    Buffer_Q_35[0]=(byte)SetBitVal(Buffer_Q_35[0], bposi, true);
    int write_banda=Client->ABWrite(bmem, 16, &Buffer_Q_35[0]);
    int leer_banda1=Client->ABRead(bmem, 16, &Buffer_Q_35[0]);
    bool estado_banda_salida = GetBitAts(Buffer_Q_35, 0, bposi);
    printf("\nESTADO Q3.5 BANDA(1 == ON - 0 == OFF )=:   %d\n",
estado_banda_salida);
    printf("\nBANDA ENCEDIDA\n");
  }else{
    int leer_banda=Client->ABRead(bmem, 16, &Buffer_Q_35[0]);
    Buffer_Q_35[0]=(byte)SetBitVal(Buffer_Q_35[0], bposi, false);
    int write_banda=Client->ABWrite(bmem, 16, &Buffer_Q_35[0]);
    int leer_banda1=Client->ABRead(bmem, 16, &Buffer_Q_35[0]);
    bool estado_banda_salida = GetBitAts(Buffer_Q_35, 0, bposi);
    printf("\nESTADO Q3.5 BANDA(1 == ON - 0 == OFF )=:   %d\n",
estado_banda_salida);
    printf("\nBANDA APAGADA\n");
  }
```

De acuerdo a lo mencionado anteriormente, el archivo <controlplc.cpp> sirve para incluir los algoritmos de programación de lectura y escritura de datos a los archivos de configuración de los Bloques de Funciones exportados de 4DIAC. Para luego ser ejecutados con el nuevo módulo de comunicación que incluye sus respectivas librerías.

3.8 Diseño de los Bloques de Funciones para el Control y la Comunicación de la Estación Festo.

4DIAC-IDE permite el diseño de aplicaciones de control distribuido a través de Bloques de Funciones. Para el desarrollo de este proyecto se crearon nuevos FBs ya que se deben agregar variables que requieran el control de la Estación Festo y de esta forma no exista limitaciones al usar los FBs que vienen por defecto en el software de diseño. Seguidamente se detalla las características de cada uno de los Bloques de Funciones utilizados en este proyecto.

3.8.1 FB Compuesto: SENSORS_FB (Bloque de Funciones para los Sensores).

SENSORS_FB como se muestra en la Figura 34, es un FB de tipo Compuesto, lo que significa que se puede generar una red compuesta de FBs internamente. La función de este Bloque es receptar los valores analógicos que envían los sensores desde la Estación.

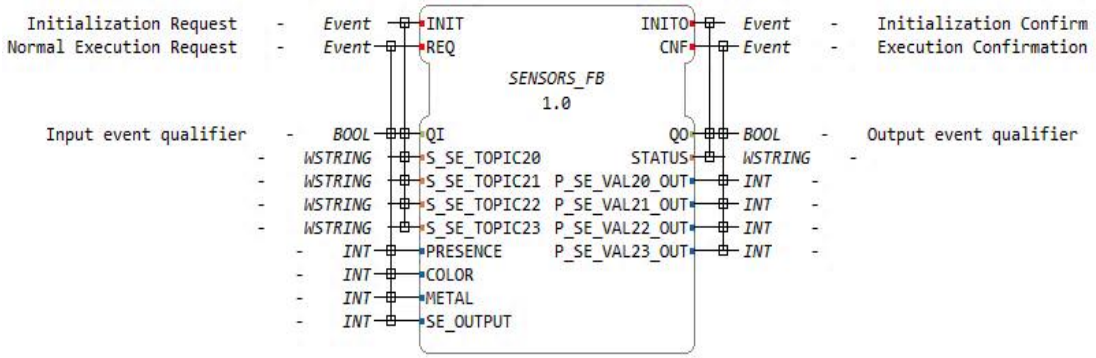


Figura 34: *SENSORS_FB (Bloque de Funciones para los Sensores).*
Elaborado por: *La Investigadora.*

Los valores analógicos que recibe este Bloque son emitidos de los siguientes sensores ubicados en la Estación:

- Sensor de Presencia: “Óptico”.
- Sensor de Color: “Óptico”.
- Sensor de Material: “Inductivo”.
- Sensor de Salida: “Retro-Reflectivo”.

A continuación, en las Tabla 11 y 12 se puede observar las características de configuración del FB:

Tabla 11: *Características de los eventos de SENSORS_FB.*

EVENTOS DE ENTRADA		**EVENTOS DE SALIDA**	
Variable	**Especificación**	**Variable**	**Especificación**
INIT	Este evento de entrada se utiliza para inicializar el FB, también sirve para comprobar la comunicación mediante un Test. Con este evento se puede comenzar a tomar los datos del sensor.	INITO	Es un evento de salida que sirve como confirmación de haber inicializado el FB.
REQ	La función de este eventos es inicializar las variables de entrada.	CNF	Tiene la finalidad de indicar cuando se finalizó la etapa que se estaba ejecutando.

Elaborado por: *La Investigadora.*

Tabla 12: *Características de los datos de SENSORS_FB.*

DATOS DE ENTRADA			**DATOS DE SALIDA**		
Variable	**Tipo**	**Especificación**	**Variable**	**Tipo**	**Especificación**
QI	BOOL	La entrada QI funciona con el evento INIT como indicador de inicio o final de ejecución.	QO	BOOL	La salida QO sirve para indicar que se completó el servicio de algún evento de entrada.

S_SE_TOPIC20	WSTRING	En esta entrada va ir una cadena de información que necesita el FB para comunicarse con los otros FBs del proceso por medio de la red interna. El Topic es: sensor20.	STATUS	WSTRING	Se utiliza esta variable para visualizar el estado de ejecución de un evento en específico.
S_SE_TOPIC21	WSTRING	En esta entrada va ir una cadena de información que necesita el FB para comunicarse con los otros FBs del proceso por medio de la red interna. El Topic es: sensor21.	P_SE_VAL20_OUT	INT	Toma el valor de la entrada por medio del publicador interno para enviar el valor del sensor 20 a su posterior FB de proceso.
S_SE_TOPIC22	WSTRING	En esta entrada va ir una cadena de información que necesita el FB para comunicarse con los otros FBs del proceso por medio de la red interna. El Topic es: sensor22	P_SE_VAL21_OUT	INT	Toma el valor de la entrada por medio del publicador interno para enviar el valor del sensor 21 a su posterior FB de proceso.
S_SE_TOPIC23	WSTRING	En esta entrada va ir una cadena de información que necesita el FB para comunicarse con los otros FBs del proceso por medio de la red interna. El Topic es: sensor23	P_SE_VAL22_OUT	INT	Toma el valor de la entrada por medio del publicador interno para enviar el valor del sensor 22 a su posterior FB de proceso.
PRESENCE	INT	Recepta valores de tipo entero para saber la presencia o ausencia de piezas en el proceso.	P_SE_VAL23_OUT	INT	Toma el valor de la entrada por medio del publicador interno para enviar el valor del sensor 23 a su posterior FB de proceso.
COLOR	INT	Recepta valores de tipo entero para saber el color de pieza que está pasando por la banda.			
METAL	INT	Recepta valores de tipo entero para saber si la pieza que pasó es de tipo metal.			
SE_OUTPUT	INT	Recepta valores de tipo entero para saber si la pieza paso a la rampa de destino.			

Elaborado por: *La Investigadora.*

El FB SENSORS_FB al ser de tipo compuesto internamente contiene una red de Bloques de Funciones de tipo básico para procesar los datos que recoge desde la Estación. En la Figura 35 se ilustra cómo está estructurado un FB básico para cada sensor y en la Figura 36 se puede observar la conexión interna, donde se encuentran los FBs: PUBLISH_PRESENCE, PUBLISH_COLOR, PUBLISH_METAL, y PUBLISH_SE_OUTPUT, conectados entre sí.

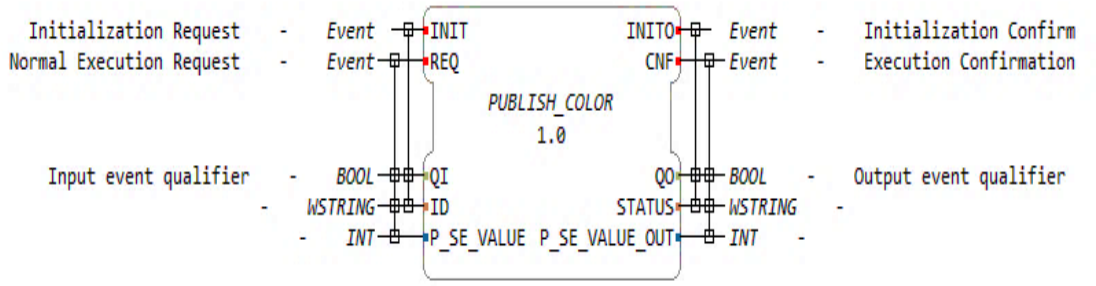


Figura 35: *PUBLISH_COLOR (Bloque de Funciones para Publicar el valor del sensor de color).*
Elaborado por: *La Investigadora.*

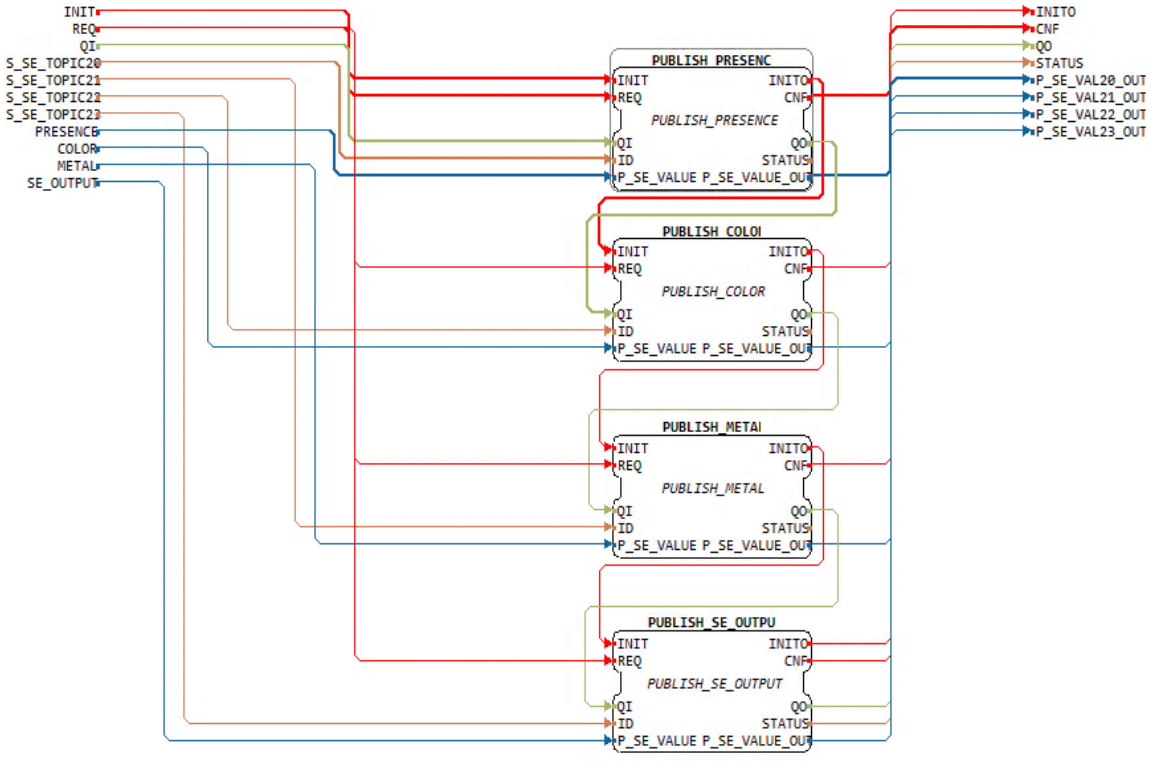


Figura 36: *Red interna del FB SENSORS_FB.*
Elaborado por: *La Investigadora.*

Los FBs PUBLISH sirven para publicar el dato de cada sensor correspondientemente a la salida del Bloque principal de su red. En las Tablas 13 y 14 se describen las características que comparten los 4 FBs.

Tabla 13: *Características de los eventos de los FBs PUBLISH.*

EVENTOS DE ENTRADA		EVENTOS DE SALIDA	
Variable	**Especificación**	**Variable**	**Especificación**
INIT	Este evento de entrada se utiliza para inicializar el FB, también sirve para comprobar la comunicación mediante un Test. Con este evento se puede comenzar a tomar los datos de cada uno de los sensores, según corresponda su Publicador.	INITO	Es un evento de salida que sirve como confirmación de haber inicializado el FB.
REQ	La función de este eventos es inicializar las variables de entrada.	CNF	Tiene la finalidad de indicar cuando se finalizó la etapa que se estaba ejecutando.

Elaborado por: *La Investigadora.*

Tabla 14: *Características de los datos de los FBs PUBLISH.*

DATOS DE ENTRADA			DATOS DE SALIDA		
Variable	**Tipo**	**Especificación**	**Variable**	**Tipo**	**Especificación**
QI	BOOL	La entrada QI funciona con el evento INIT como indicador de inicio o final de ejecución.	QO	BOOL	La salida QO sirve para indicar que se completó el servicio de algún evento de entrada.
ID	WSTRING	En esta entrada del Publicador se recibe la cadena de información del Bloque principal para crear la conexión con el resto de Bloques de Funciones. Los tópicos son: sensor 20, sensor21, sensor22, sensor23	STATUS	WSTRING	Se utiliza esta variable para visualizar el estado de ejecución de un evento en específico.
P_SE_VALUE	INT	Permite receptar el dato de entrada del sensor (sensor 20, sensor 21, sensor 22, sensor 23) para luego enviarlo hacia la salida del Bloque principal de la red.	P_SE_VAL20_OUT	INT	Remite el valor que recibió de la entrada hacia la salida del Bloque Principal de la red.

Elaborado por: *La Investigadora.*

3.8.2 FB Básico: CONTROL (Bloque de Funciones para el Control de la Estación).

El FB CONTROL es de tipo Básico, por lo cual se añaden variables de acuerdo al algoritmo que realiza el Bloque; toma los valores desde el FB de Comunicación de entrada y empieza a gestionar acciones de acuerdo al funcionamiento automático que debe realizar la Estación. Por ejemplo, si se recibe un valor entero del sensor de presencia, el CONTROL envía un valor booleano hacia el FB de comunicación de salida de datos para que el Bloque de Actuadores encienda o apague la banda transportadora. Este proceso se repite con todos los sensores ya que la información de cada uno de ellos genera una acción en los Actuadores. La Figura 37 muestra la estructura del FB CONTROL, además, en las Tablas 15 y 16 se detallan las características de los eventos y datos de entrada/salida del Bloque de Control.

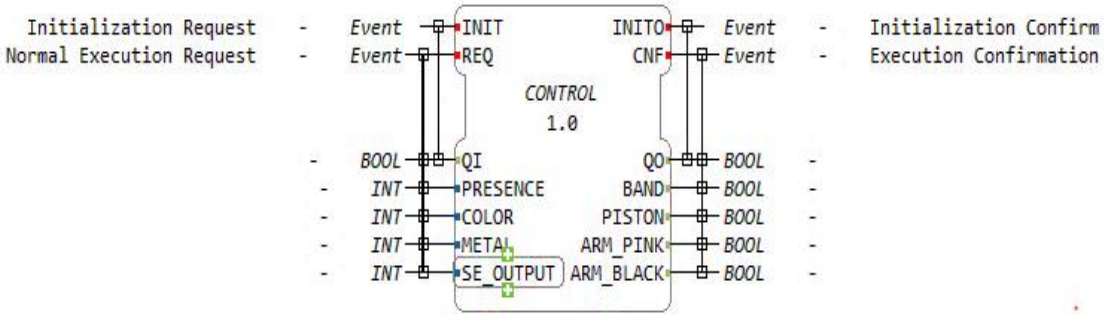


Figura 37: *CONTROL (Bloque de Funciones para el control de la Estación).*
Elaborado por: *La Investigadora.*

Tabla 15: *Características de los eventos del CONTROL.*

EVENTOS DE ENTRADA		EVENTOS DE SALIDA	
Variable	**Especificación**	**Variable**	**Especificación**
INIT	Este evento de entrada se utiliza para inicializar el FB, también sirve para comprobar la comunicación mediante un Test. Con este evento se puede comenzar a recibir los datos del Bloque de Sensores por medio del FB PUBLISH_CONTROL.	INITO	Es un evento de salida que sirve como confirmación de haber inicializado el FB.
REQ	La función de este eventos es inicializar las variables de entrada.	CNF	Tiene la finalidad de indicar cuando se finalizó la etapa que se estaba ejecutando.

Elaborado por: *La Investigadora.*

Tabla 16: *Características de los datos de CONTROL.*

DATOS DE ENTRADA			DATOS DE SALIDA		
Variable	**Tipo**	**Especificación**	**Variable**	**Tipo**	**Especificación**
QI	BOOL	La entrada QI funciona con el evento INIT como indicador de inicio o final de ejecución.	QO	BOOL	La salida QO sirve para indicar que se completó el servicio de algún evento de entrada.
PRESENCE	INT	Valores de tipo entero que se publicaron anteriormente para saber la presencia o ausencia de piezas en el proceso.	BAND	BOOL	Valor de tipo booleano que se envía a suscribir y así activar o desactivar la Banda.
COLOR	INT	Valores de tipo entero que se publicaron anteriormente para saber el color de pieza que está pasando por la banda.	PISTON	BOOL	Valor de tipo booleano que se envía a suscribir y así activar o desactivar el pistón.
METAL	INT	Valores de tipo entero que se publicaron anteriormente para saber si la pieza que paso es de tipo metal.	ARM_PINK	BOOL	Valor de tipo booleano que se envía a suscribir y así activar o desactivar el brazo de las piezas rosadas.
SE_OUTPUT	INT	Valores de tipo entero que se publicaron anteriormente para saber si la pieza pasó a la rampa de destino.	ARM_BLACK	BOOL	Valor de tipo booleano que se envía a suscribir y así activar o desactivar el brazo de las negras.

Elaborado por: *La Investigadora.*

3.8.3 FB Compuesto: ACTORS_FB (Bloque de Funciones para los Actuadores).

El Bloque de Funciones ACTORS_FB como se ilustra en la Figura 38, es un FB de tipo Compuesto al igual que el de sensores tiene una red compuesta de FBs internamente. Su objetivo es enviar valores booleanos hacia los actuadores y de esta forma realicen una acción en la Estación. Los valores booleanos que envía el Bloque Suscriptor es para activar o desactivar los siguientes actuadores ubicados en la Estación:

- Banda Transportadora.
- Pistón.
- Brazo para las Piezas de Color Rosado.
- Brazo para las Piezas de Color Negro.

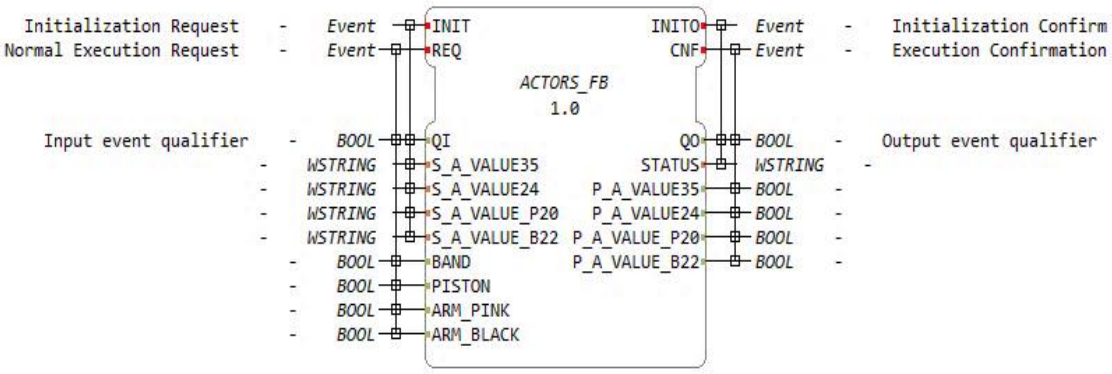


Figura 38: *ACTORS_FB (Bloque de Funciones para los Actuadores).*
Elaborado por: *La Investigadora.*

A continuación, en las Tabla 17 y 18 se puede observar las características de configuración del FB:

Tabla 17: *Características de los eventos de ACTORS_FB.*

EVENTOS DE ENTRADA		**EVENTOS DE SALIDA**	
Variable	**Especificación**	**Variable**	**Especificación**
INIT	Este evento de entrada se utiliza para inicializar el FB, también sirve para comprobar la comunicación mediante un Test. Con este evento se puede comenzar a tomar los datos del suscriptor del control.	INITO	Es un evento de salida que sirve como confirmación de haber inicializado el FB.
REQ	La función de este eventos es inicializar las variables de entrada.	CNF	Tiene la finalidad de indicar cuando se finalizó la etapa que se estaba ejecutando.

Elaborado por: *La Investigadora.*

Tabla 18: *Características de los datos de ACTORS_FB.*

DATOS DE ENTRADA			**DATOS DE SALIDA**		
Variable	**Tipo**	**Especificación**	**Variable**	**Tipo**	**Especificación**
QI	BOOL	La entrada QI funciona con el evento INIT como indicador de inicio o final de ejecución.	QO	BOOL	La salida QO sirve para indicar que se completó el servicio de algún evento de entrada.

S_A_VALUE35	WSTRING	Entrada que contiene una cadena de información que necesita el FB para comunicarse con los otros FBs del proceso por medio de la red interna. El Topic es: actuador35.	STATUS	WSTRING	Se utiliza esta variable para visualizar el estado de ejecución de un evento en específico.
S_A_VALUE24	WSTRING	Entrada que contiene una cadena de información que necesita el FB para comunicarse con los otros FBs del proceso por medio de la red interna. El Topic es: actuador24.	P_A_VALUE35	BOOL	Obtiene el valor de la entrada por medio del suscriptor interno para enviar el valor booleano al actuador35 en la Estación.
S_A_VALUE_P20	WSTRING	Entrada que contiene una cadena de información que necesita el FB para comunicarse con los otros FBs del proceso por medio de la red interna. El Topic es: actuador20.	P_A_VALUE24	BOOL	Obtiene el valor de la entrada por medio del suscriptor interno para enviar el valor booleano al actuador24 en la Estación.
S_A_VALUE_B22	WSTRING	Entrada que contiene una cadena de información que necesita el FB para comunicarse con los otros FBs del proceso por medio de la red interna. El Topic es: actuador22.	P_A_VALUE_P20	BOOL	Obtiene el valor de la entrada por medio del suscriptor interno para enviar el valor booleano al actuador20 en la Estación.
BAND	BOOL	Recibe valores de tipo booleano para saber la presencia o ausencia de piezas en el proceso.	P_A_VALUE_B22	BOOL	Obtiene el valor de la entrada por medio del suscriptor interno para enviar el valor booleano al actuador22 en la Estación.
PISTON	BOOL	Recibe valores de tipo booleano para saber el color de pieza que está pasando por la banda.			
ARM_PINK	BOOL	Recibe valores de tipo booleano para saber si la pieza que paso es de tipo metal.			
ARM_BLACK	BOOL	Recibe valores de tipo booleano para saber si la pieza pasó a la rampa de destino.			

Elaborado por: *La Investigadora.*

ACTORS_FB es un FB de tipo compuesto por lo cual contiene una red interna de Bloques de Funciones de tipo básico para ejecutar acciones en la Estación. La Figura 39 muestra la estructura de los FBs: SUSCRIBE_BAND, SUSCRIBE_PISTON, SUSCRIBE_PINK, SUSCRIBE_BLACK. Mientras que en la Figura 40 se observa la red interna del Bloque Principal.

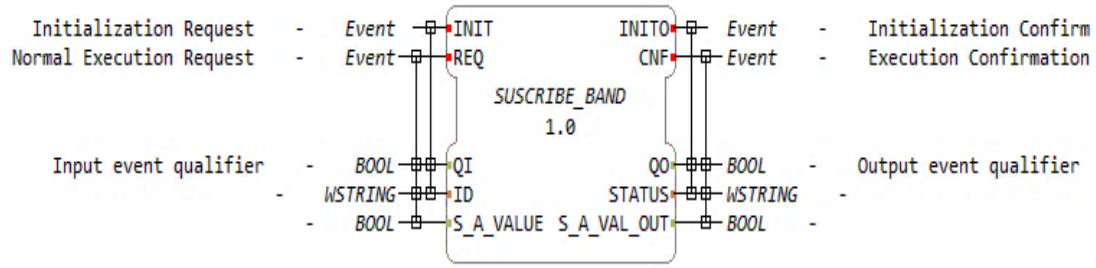


Figura 39: *SUSCRIBE_BAND (Bloque de Funciones para Suscribir el valor del actuador banda).*
Elaborado por: *La Investigadora.*

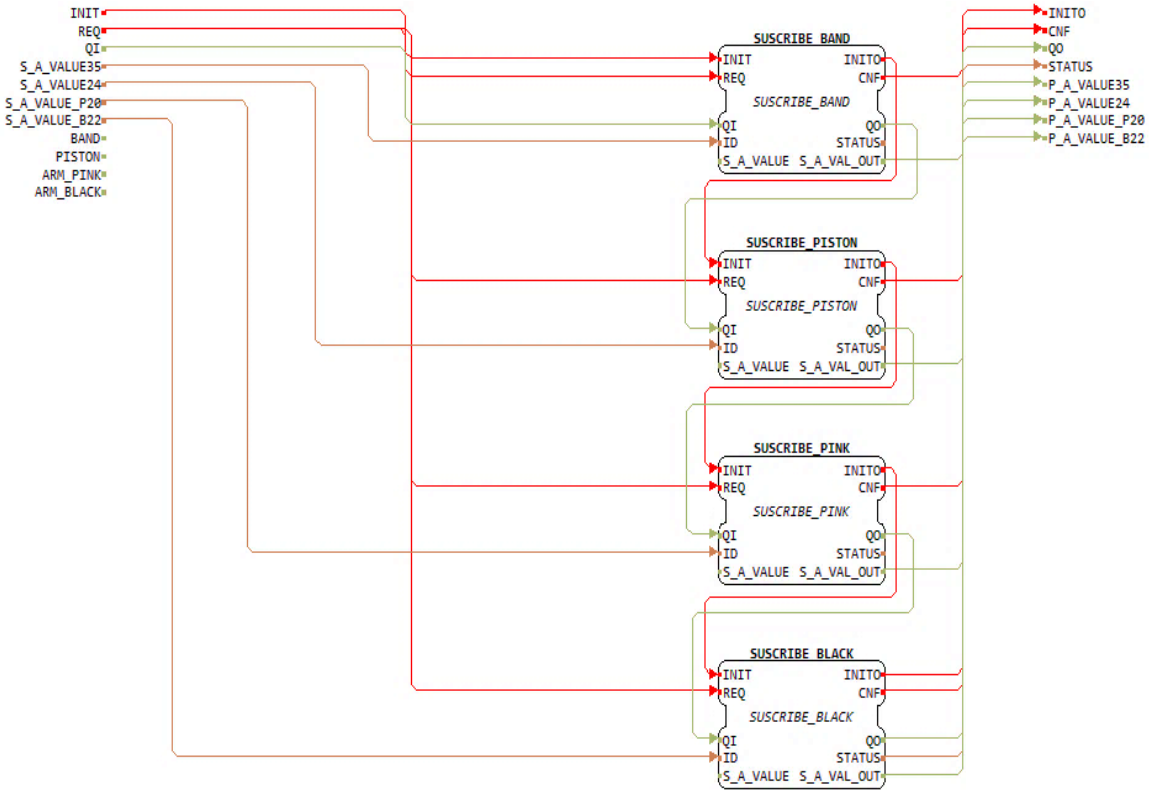


Figura 40: *Red interna del FB ACTORS_FB.*
Elaborado por: *La Investigadora.*

Los FBs SUSCRIBE se utilizan para suscribir el dato que se obtuvo en la entrada del Bloque principal de la red y de esta forma enviar una respuesta hacia la Estación. Las

Tablas 19 y 20 describen las características que comparten los 4 FBs internos del Bloque de Funciones ACTORS_FB.

Tabla 19: *Características de los eventos de los FBs PUBLISH.*

EVENTOS DE ENTRADA		EVENTOS DE SALIDA	
Variable	**Especificación**	**Variable**	**Especificación**
INIT	Este evento de entrada se utiliza para inicializar el FB, también sirve para comprobar la comunicación mediante un Test. Con este evento se puede comenzar a tomar los datos enviados como respuesta desde el FB de Control hacia cada uno de los actuadores, según corresponda su Suscriptor.	INITO	Es un evento de salida que sirve como confirmación de haber inicializado el FB.
REQ	La función de este eventos es inicializar las variables de entrada.	CNF	Tiene la finalidad de indicar cuando se finalizó la etapa que se estaba ejecutando.

Elaborado por: *La Investigadora*

.

Tabla 20: *Características de los datos de los FBs SUSCRIBE.*

DATOS DE ENTRADA			DATOS DE SALIDA		
Variable	**Tipo**	**Especificación**	**Variable**	**Tipo**	**Especificación**
QI	BOOL	La entrada QI funciona con el evento INIT como indicador de inicio o final de ejecución.	QO	BOOL	La salida QO sirve para indicar que se completó el servicio de algún evento de entrada.
ID	WSTRING	Entrada del Suscriptor que recibe la cadena de información del Bloque principal para crear la conexión con el resto de Bloques de Funciones. Los tópicos son: actuador35, actuador24, actuador20, actuador22.	STATUS	WSTRING	Se utiliza esta variable para visualizar el estado de ejecución de un evento en específico.
S_A_VALUE	BOOL	Recepta la respuesta del FB de Control, el dato booleano de los actuadores (actuador35, actuador 24, actuador 20, actuador22) son enviados hacia la salida del Bloque principal de la red.	S_A_VAL_OUT	BOOL	Remite el valor que recibió de la entrada hacia la salida del Bloque Principal de la red.

Elaborado por: *La Investigadora.*

3.8.4 FBs Básicos: PUBLISH_CONTROL y SUSCRIBE_ CONTROL (Bloques de Funciones para la comunicación de datos).

Los Bloques de Funciones de comunicación son utilizados en la transferencia de información entre los Bloques de control, estos FBs son de tipo básico. Así de esta forma se puedan incluir variables necesarias que sirven para almacenar y reenviar datos a través del proceso de funcionamiento de la Estación.

FB PUBLISH_CONTROL

El Bloque de Funciones PUBLISH_CONTROL se utiliza de intermediario entre el SENSORS_FB y el FB CONTROL de manera que, recibe los datos de los sensores y envía esta información para que se comparen los valores y se obtenga las respuestas requeridas por el proceso. La Figura 41 ilustra el FB PUBLISH_CONTROL y en las Tablas 21 y 22 se enlistan los detalles de sus variables.

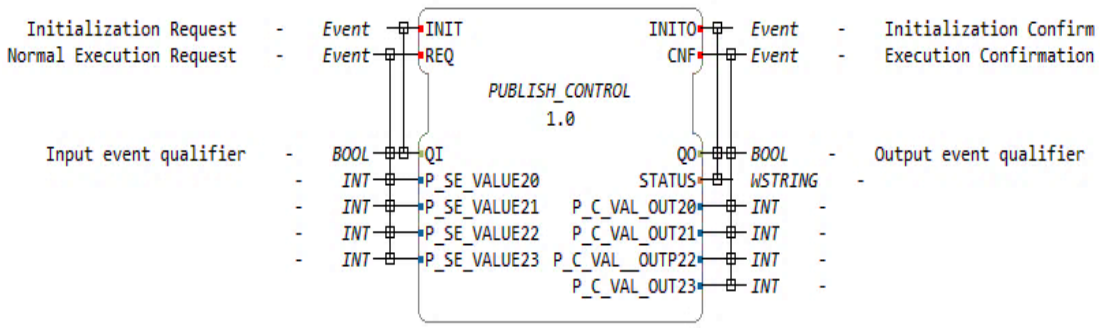


Figura 41: *PUBLISH_CONTROL (FB de Comunicación para el Control de la Estación).*
Elaborado por: *La Investigadora.*

Tabla 21: *Características de los eventos del FB PUBLISH_CONTROL.*

EVENTOS DE ENTRADA		EVENTOS DE SALIDA	
Variable	**Especificación**	**Variable**	**Especificación**
INIT	Evento de entrada que se utiliza para inicializar el FB, sirve también para comprobar la comunicación mediante un Test.	INITO	Es un evento de salida que sirve como confirmación de haber inicializado el FB.
REQ	La función de este eventos es inicializar las variables de entrada.	CNF	Tiene la finalidad de indicar cuando se finalizó la etapa que se estaba ejecutando.

Elaborado por: *La Investigadora.*

Tabla 22: *Características de los datos de los FB PUBLISH_CONTROL.*

DATOS DE ENTRADA			DATOS DE SALIDA		
Variable	**Tipo**	**Especificación**	**Variable**	**Tipo**	**Especificación**
QI	BOOL	La entrada QI funciona con el evento INIT como indicador de inicio o final de ejecución.	QO	BOOL	La salida QO sirve para indicar que se completó el servicio de algún evento de entrada.
P_SE_VALUE20 P_SE_VALUE21 P_SE_VALUE22 P_SE_VALUE23	INT	Recepta el dato sensor20, sensor21, sesnor22, sensor23 que envía el FB de sensores para después enviar hacia la salida del Bloque.	STATUS	WSTRING	Se utiliza esta variable para visualizar el estado de ejecución de un evento en específico.
			P_C_VAL_OUT20 P_C_VAL_OUT21 P_C_VAL__OUTP22 P_C_VAL_OUT23	INT	Remite el valor que recibe de la entrada hacia el FB de control correspondientemente.

Elaborado por: *La Investigadora.*

FB SUSCRIBE_CONTROL

SUSCRIBE_CONTROL es un Bloque de Funciones que sirve de intermediario entre el FB CONTROL y ACTORS_FB de manera que, recibe los datos desde el control para después enviar las respuestas al FB de actuadores y de este modo se genere una acción correspondiente. En la Figura 42 se muestra el FB SUSCRIBE_CONTROL, además, en la Tablas 23 se detalla las características de sus variables, los eventos de este FB realizan las mismas funciones que el FB PUBLISH_CONTROL.

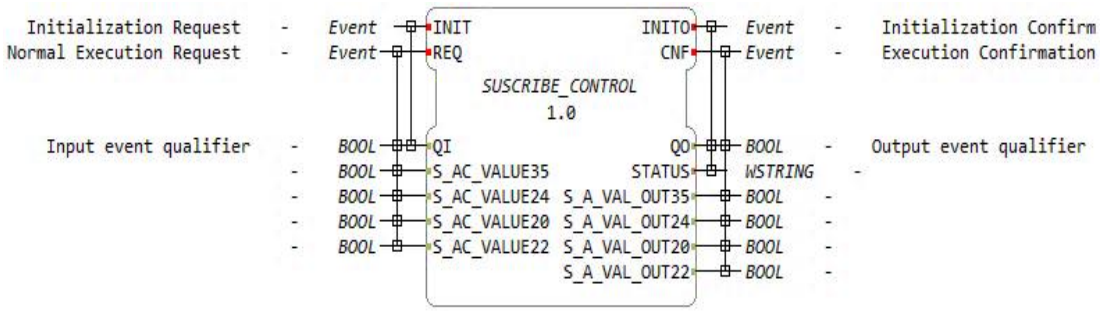


Figura 42: *SUSCRIBE_CONTROL (FB de Comunicación para el Control de la Estación).*
Elaborado por: *La Investigadora.*

Tabla 23: *Características de los datos de los FB PUBLISH_CONTROL.*

DATOS DE ENTRADA			DATOS DE SALIDA		
Variable	**Tipo**	**Especificación**	**Variable**	**Tipo**	**Especificación**
QI	BOOL	La entrada QI funciona con el evento INIT como indicador de inicio o final de ejecución.	QO	BOOL	La salida QO sirve para indicar que se completó el servicio de algún evento de entrada.
S_AC_VALUE35 S_AC_VALUE24 S_AC_VALUE20 S_AC_VALUE22	BOOL	Recepta el dato de: BAND, PISTON, ARM_PINK, ARM_BLACK, luego se envía al FB de actuadores mediante la salida del Bloque.	STATUS	WSTRING	Se utiliza esta variable para visualizar el estado de ejecución de un evento en específico.
			S_A_VAL_OUT35 S_A_VAL_OUT24 S_A_VAL_OUT20 S_A_VAL_OUT22	BOOL	Remite el valor que recibe de la entrada hacia el FB de actuadores según corresponde.

Elaborado por: *La Investigadora.*

3.9 Diseño de la Aplicación del Control y la Comunicación para el Proceso de la Estación.

Después de la creación de los FBs se procede a conectarlos según corresponda su función, la distribución final se muestra en la Figura 43. La Raspberry Pi de comunicación trabajara con los Bloques de Funciones de color verde y la Raspberry Pi de control con los FBs de color naranja. La aplicación de control se encarga de leer, procesar y escribir información en el PLC por medio del dispositivo de ejecución para lo cual físicamente la Válvula CP recibe señales de los sensores y envía instrucciones a los actuadores de la Estación y de esta forma realizar su proceso correctamente. Por otro lado, la Raspberry Pi de comunicación ejecuta los Bloques de Función Publicador/Suscriptor para gestionar la petición de instrucciones del proceso. Conjuntamente los dos dispositivos funcionan en tiempo real.

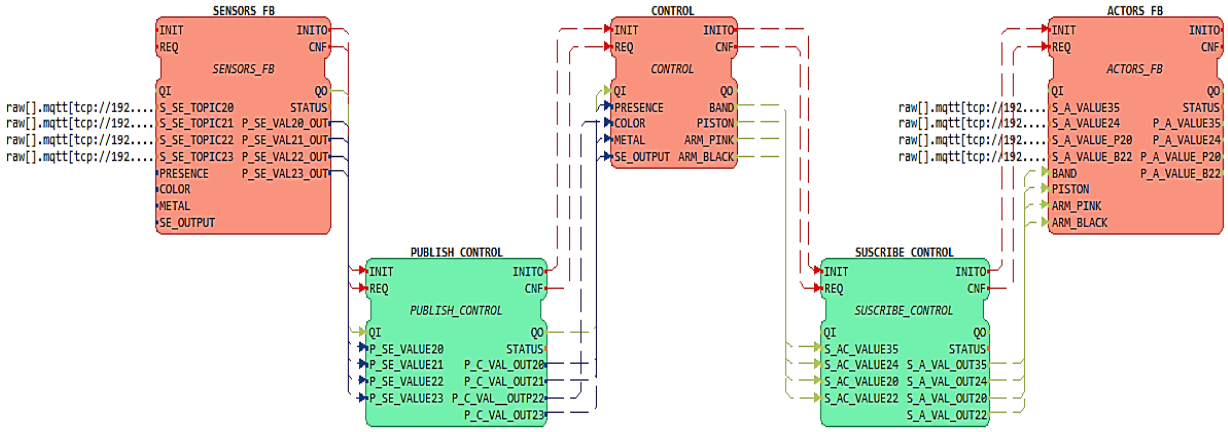


Figura 43: *Aplicación de Control y Comunicación en 4DIAC.*
Elaborado por: *La Investigadora.*

Se debe tomar en cuenta el identificador de los FBs de Sensores y Actuadores porque de esta forma se enlaza con el Bróker Mosquitto. A continuación, se muestra el formato del ID:

El tema para subscribir y publicar

Puerto del servidor:1883

```
raw[].mqtt[tcp://IP:Port,ClientID,topic]
```

IP del Broker Nombre del cliente

Identificadores de los Sensores:

```
raw[].mqtt[tcp://192.168.4.4:1883, comunication, sensor20]
raw[].mqtt[tcp://192.168.4.4:1883, comunication, sensor21]
raw[].mqtt[tcp://192.168.4.4:1883, comunication, sensor22]
raw[].mqtt[tcp://192.168.4.4:1883, comunication, sensor23]
```

Identificadores de los Actuadores:

```
raw[].mqtt[tcp://192.168.4.4:1883, comunication, actuador35]
raw[].mqtt[tcp://192.168.4.4:1883, comunication, actuador24]
raw[].mqtt[tcp://192.168.4.4:1883, comunication, actuador20]
raw[].mqtt[tcp://192.168.4.4:1883, comunication, actuador22]
```

3.9.1 Diseño virtual de la Configuración del Sistema.

La Figura 44 muestra el diseño virtual del sistema, se conectan virtualmente los dos dispositivos mediante Ethernet. Cada una de las Raspberry´s deben tener una IP única e incluir el puerto de la Norma IEC-61499.

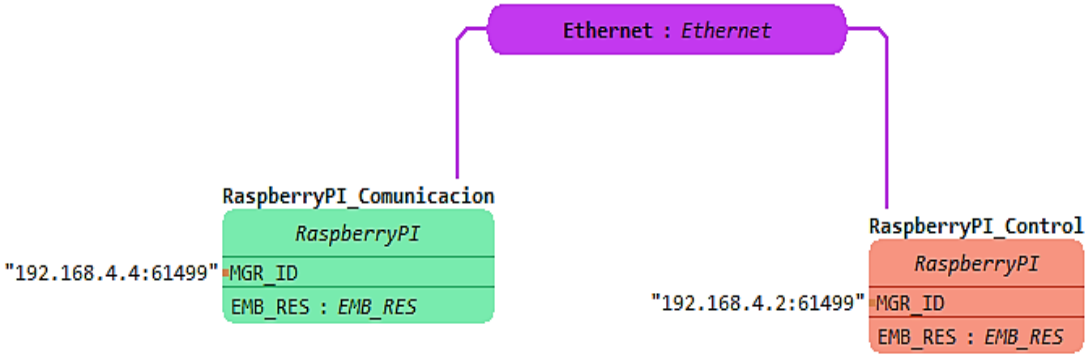
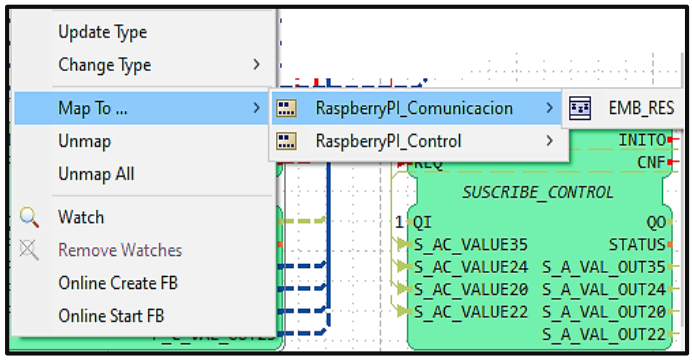


Figura 44: *Sistema embebido de comunicación en 4DIAC..*
Elaborado por: *La Investigadora.*

La conexión virtual del sistema sirve para ejecutar las funciones en sus correspondientes dispositivos. Cuando se mapea los Bloques de Control con el recurso de la RaspberryPI_Control automáticamente se genera el color para sus Bloques, de igual forma pasa cuando se mapea los FB´s de Comunicación, la finalidad de colorear los Bloques de Funciones es orientar la ejecución de segmentos del sistema. La Figura 45 muestra el procedimiento para mapear los FB´s de la aplicación con sus respectivos dispositivos.

(a)



(b)

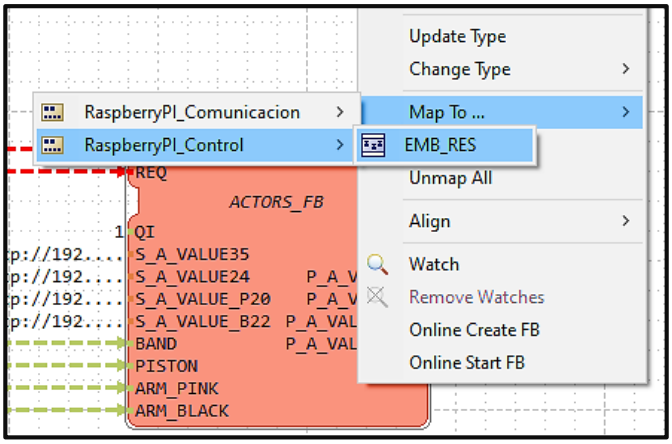


Figura 45: *(a): Mapeo de los FB´s de Comunicación, (b): Mapeo de los FB´s de Control.*
Elaborado por: *La Investigadora.*

3.9.2 Desarrollo y Configuración de la Capa de Comunicación.

El desarrollo de la capa de comunicación se basa en el protocolo MQTT, este tipo de protocolo hace referencia al modelo Publicador/Suscriptor de mensajes que son enviados o recibidos desde un cliente. La trasferencia de datos requiere de un intermediario, en este caso Mosquitto, el cual gestiona y distribuye los mensajes del cliente de acuerdo a un tópico. Se mantienen conectados los clientes con cualquier tipo de proceso debido al Bróker. En la Figura 46 se ilustra lo mencionado en el apartado anterior.

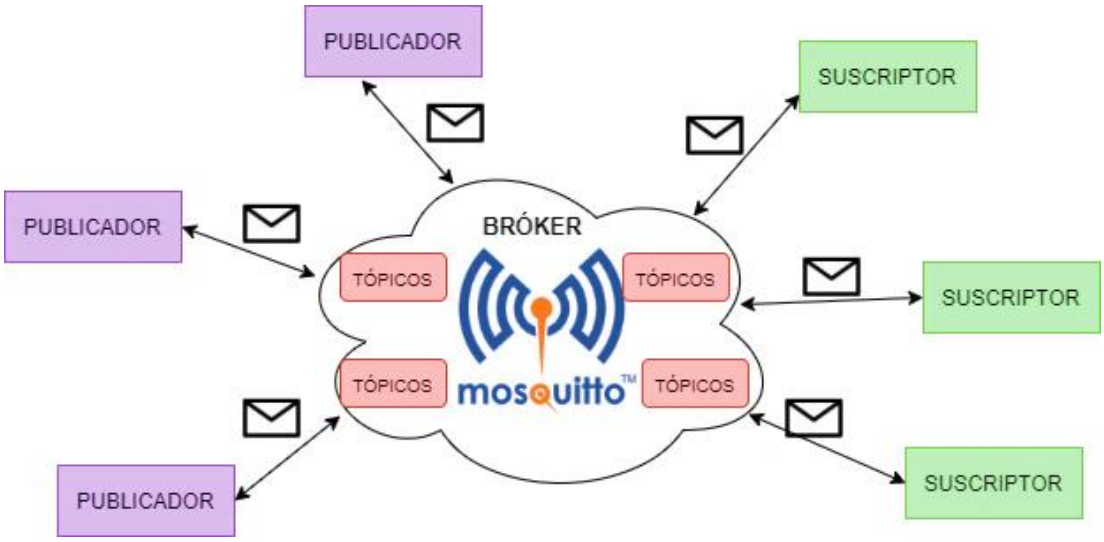


Figura 46: *Modelo de Publicador/Suscriptor.*
Elaborado por: *La Investigadora.*

En comparación con otros protocolos, MQTT trabaja con recursos limitados y aun así es eficiente y de respuesta rápida. La arquitectura que emplea MQTT es: MQTT_Client y MQTT_Broker, para lo cual, la Figura 47 describe el proceso de conexión entre el cliente y el bróker.

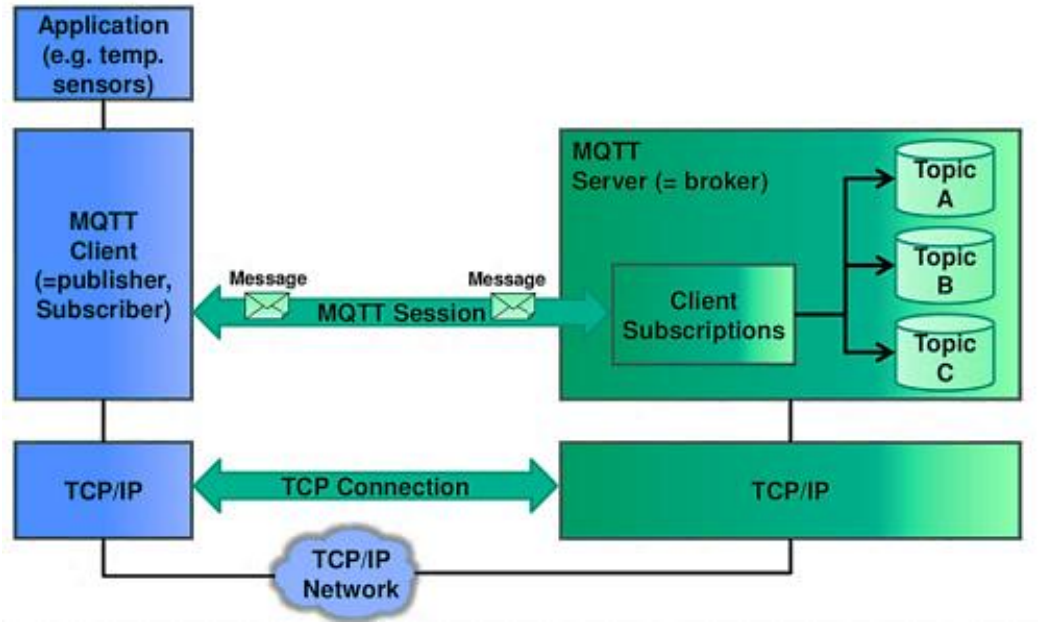


Figura 47: *Arquitectura MQTT.* [59].

De acuerdo a lo señalado anteriormente se procede a diseñar los archivos de configuración de la capa de comunicación. Para lo cual, se tiene el fichero "cabecera", que sirve para declarar las variables de la arquitectura; aquí se encuentran las librerías de Mosquitto y el encapsulamiento de mensajes con el protocolo MQTT. Por otro lado, está el fichero "cuerpo", que se utiliza para realizar las funciones de comunicación, adicionalmente en el archivo es necesario incluir la librería SNAP7. El Anexo 12 contiene el diseño completo de la configuración de los archivos < al_mqtt.cpp > y < al_mqtt.h >, seguidamente se detallan las funciones más sobresalientes de la capa de comunicación.

- Clase instanciada de la librería Mosquitto.

Método de comparación de Tópicos.

```
mosqpp::topic_matches_sub("sensor20",message->topic,&se20);
mosqpp::topic_matches_sub("sensor21",message->topic,&se21);
mosqpp::topic_matches_sub("sensor22",message->topic,&se22);
mosqpp::topic_matches_sub("sensor23",message->topic,&se23);
mosqpp::topic_matches_sub("actuador35",message->topic,&ac35);
mosqpp::topic_matches_sub("actuador24",message->topic,&ac24);
mosqpp::topic_matches_sub("actuador20",message->topic,&ac20);
mosqpp::topic_matches_sub("actuador22",message->topic,&ac22);
```

3.9.3 Configuración del archivo CMakeLists para la creación del Módulo Forte MOSQUITTO_MQTT_SNAP.

Para poder ejecutar los FBs de la aplicación de acuerdo con los requerimientos del proyecto de investigación es necesario crear un nuevo Módulo Forte de Comunicación. Para esto se crea una carpeta llamada mosquitto_mqtt_snap en la Raspberry de Control, la cual almacena los archivos exportados de la aplicación 4DIAC. También se incluyen las librerías de Mosquitto, el cual se enlaza al bróker de comunicación. Además, se agrega la librería de SNAP7, que sirve para la conexión con el PLC, adicional a esto los archivos de configuración de la capa de comunicación con el protocolo MQTT. En la Figura 48 se muestran todos los archivos que debe contener la carpeta mosquitto_mqtt_snap:

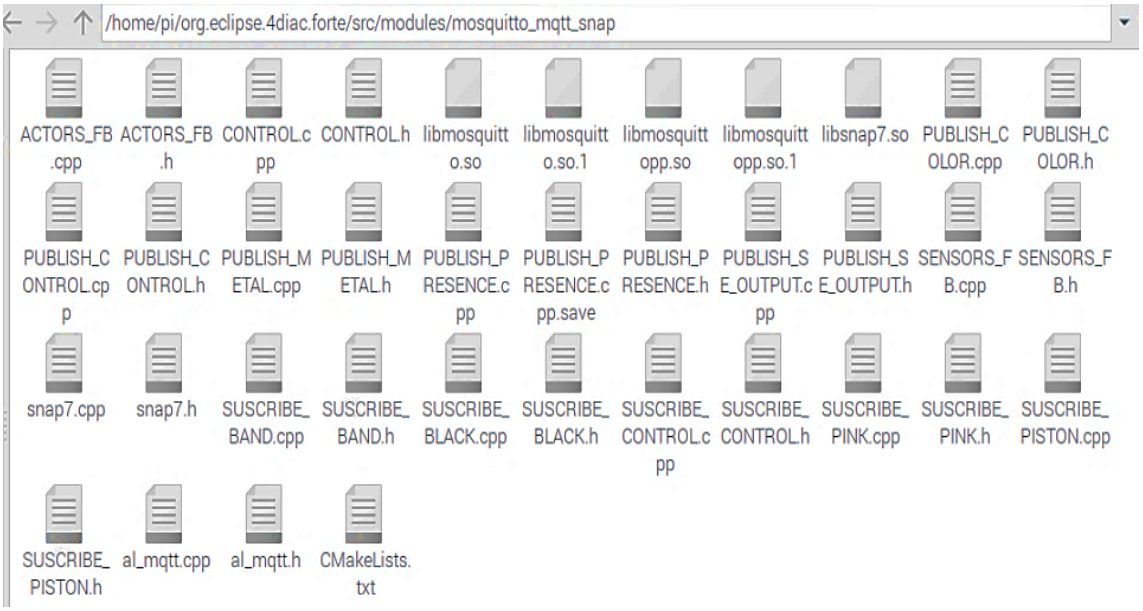


Figura 48: *Archivos de configuración para el nuevo Módulo de Forte.*
Elaborado por: *La Investigadora.*

El fichero CMakeLists.txt es muy importante puesto que, contiene información sobre todos los archivos de la carpeta mosquitto_mqtt_snap y permite aparecer el nuevo Módulo Forte de Comunicación en el software CMake y de esta forma construir la configuración de todos los archivos en una nueva carpeta llamada GENERADOS para luego recompilar estos archivos y generar un único archivo <./forte > que ejecuta la aplicación de comunicación y control del proceso. En el Anexo 13 se encuentra el

archivo completo de CMakeLists.txt y la Figura 49 muestra el nuevo Módulo Forte de Comunicación en el software CMake.

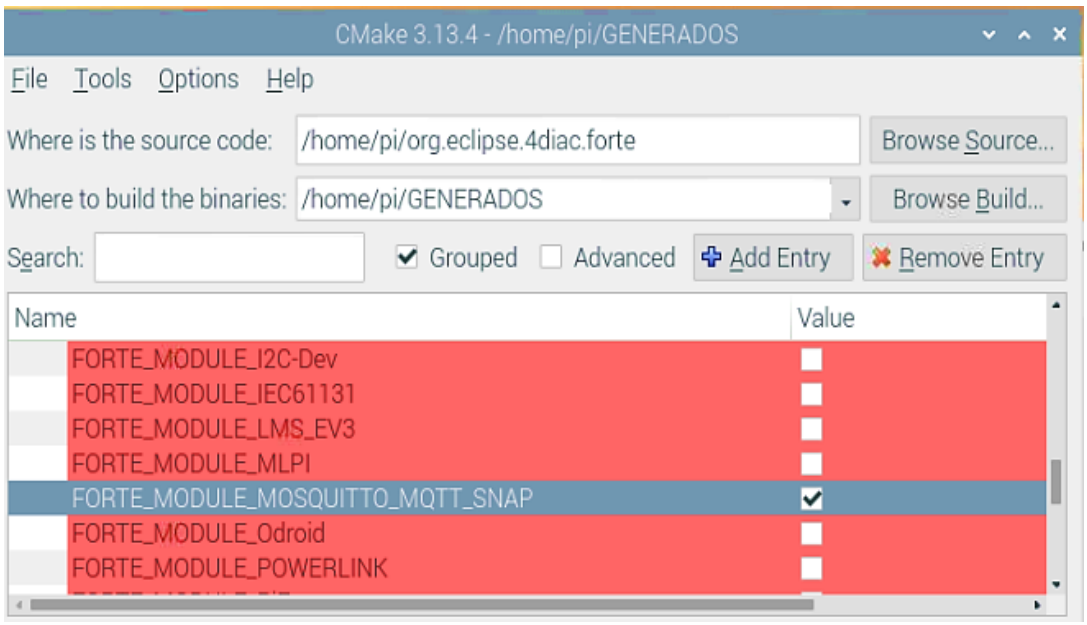


Figura 49: *Modulo Forte de Comunicación MOSQUITTO_MQTT_SNAP.*
Elaborado por: *La Investigadora.*

3.9.4 Exportación, Edición y Ejecución de los Bloques de Funciones.

Para que la aplicación se ejecute en las Raspberry´s es necesario exportar todos los FBs que se crearon en 4DIAC-IDE y construirlos con el nuevo Módulo Forte de Comunicación, para lo cual en la Figura 50 se ilustra la forma que deben ser exportados los FBs. Los archivos que se generan desde 4DIAC son <.cpp y .h > y estos archivos se deben pegar en la carpeta mosquitto_mqtt_snap. Tomar en cuenta que solo la Raspberry Pi de control realiza todo este procedimiento, puesto que, en la Raspberry de comunicación se copia el archivo <./forte > y se ejecuta indistintamente del dispositivo de control.

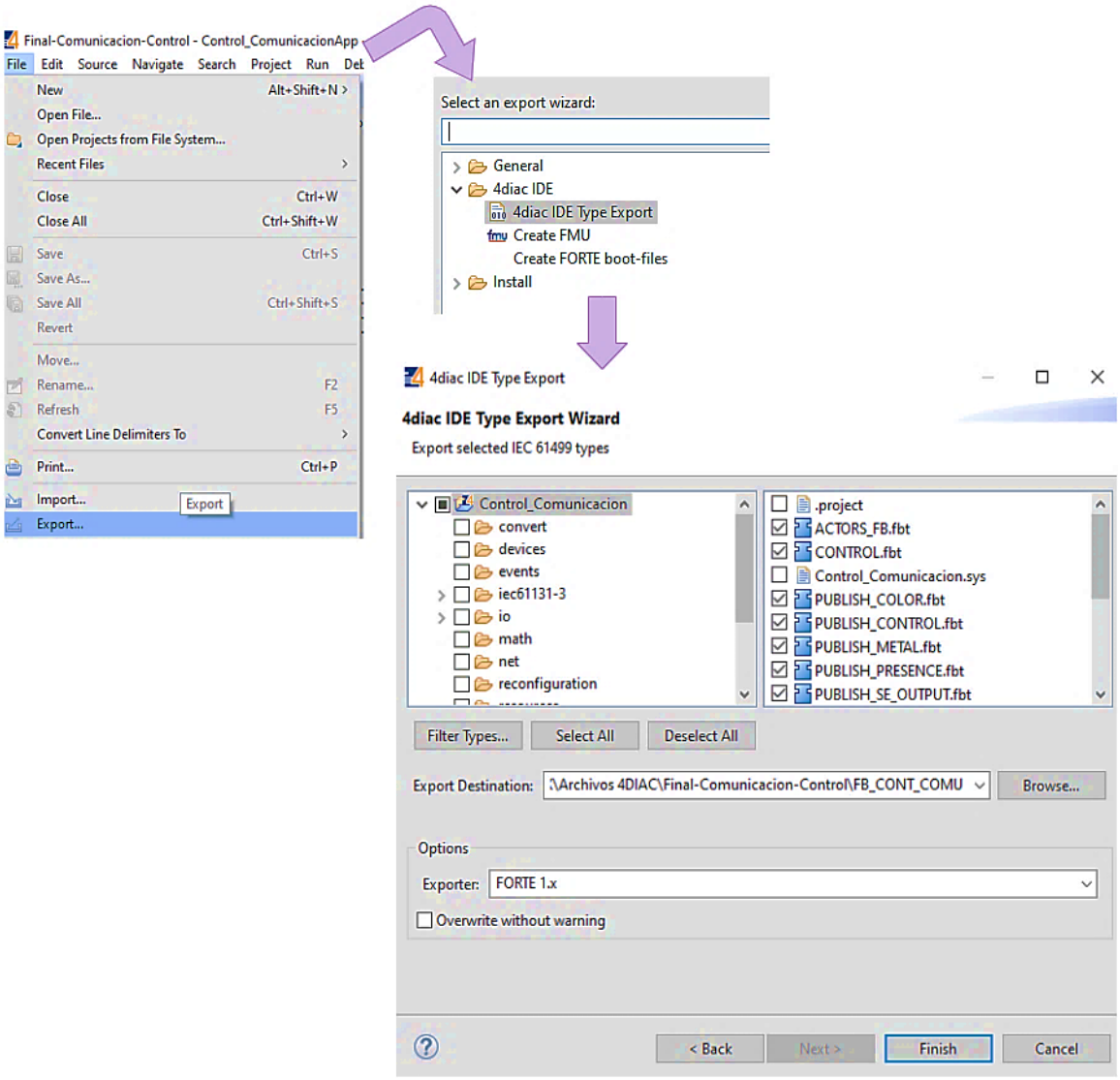


Figura 50: *Procedimiento de exportación de FBs desde 4DIAC.*
Elaborado por: *La Investigadora.*

Seguidamente en la Raspberry Pi con ayuda del software CMake se procede a construir los archivos con el nuevo Módulo Forte de Comunicación y las nuevas librerías, en el Anexo 14 se encuentran los pasos para construir los ficheros de configuración. Posteriormente, en la carpeta GENERADOS desde la terminal se ejecuta el comando <sudo make >, el cual recompila los archivos de configuración creando un fichero único llamado < ./forte >, el cual está listo para ser ejecutado y de esta forma dar inicio al Control y Comunicación de la Estación Festo MPS-200- Clasificador. En la Figura 51 se puede observar la re-compilación del archivo < ./forte >.

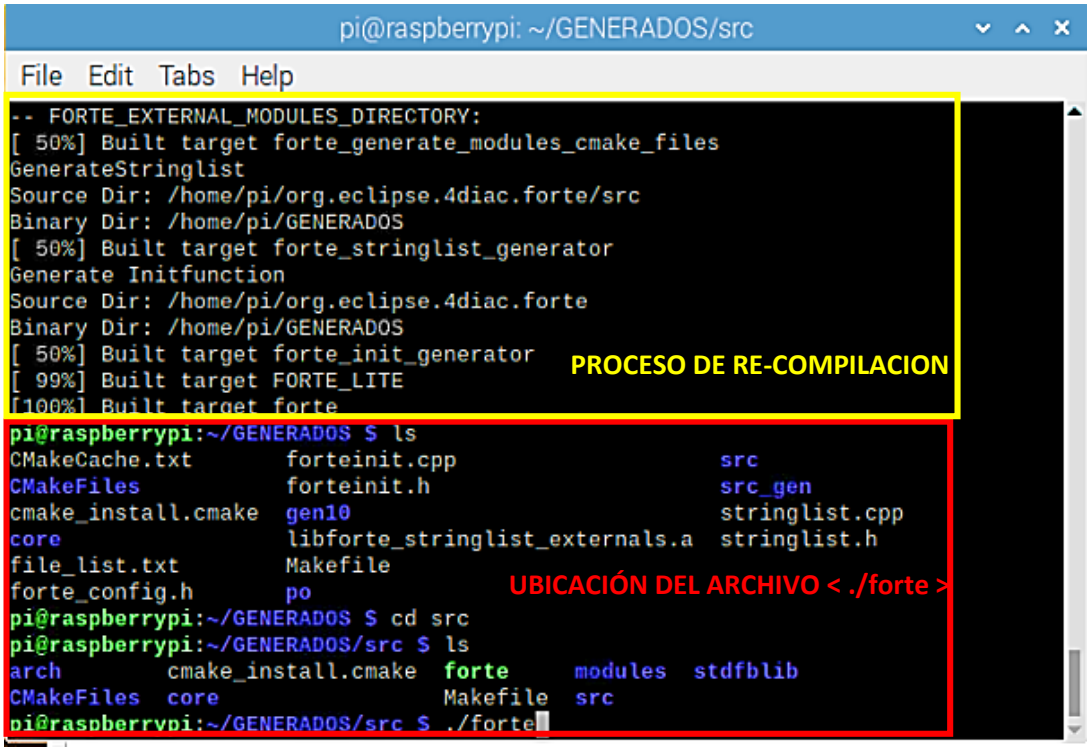


Figura 51: *Procedimiento de re-compilación del archivo < ./forte > .*
Elaborado por: *La Investigadora.*

3.10 Pruebas de Funcionamiento

3.10.1 Verificación de Publicación y Suscripción.

Para verificar que los clientes puedan publicar mensajes y suscribirse al Bróker se activa por la terminal de la Raspberry de Comunicación, un cliente, el cual puede enviar los tópicos y saber el estado de cualquier variable del control. La Figura 52 muestra las respuestas que recibe el cliente de acuerdo a las peticiones que hace al Bróker. Se puede leer el estado de los sensores y ver el control que se realiza a los actuadores con una respuesta de su etapa.

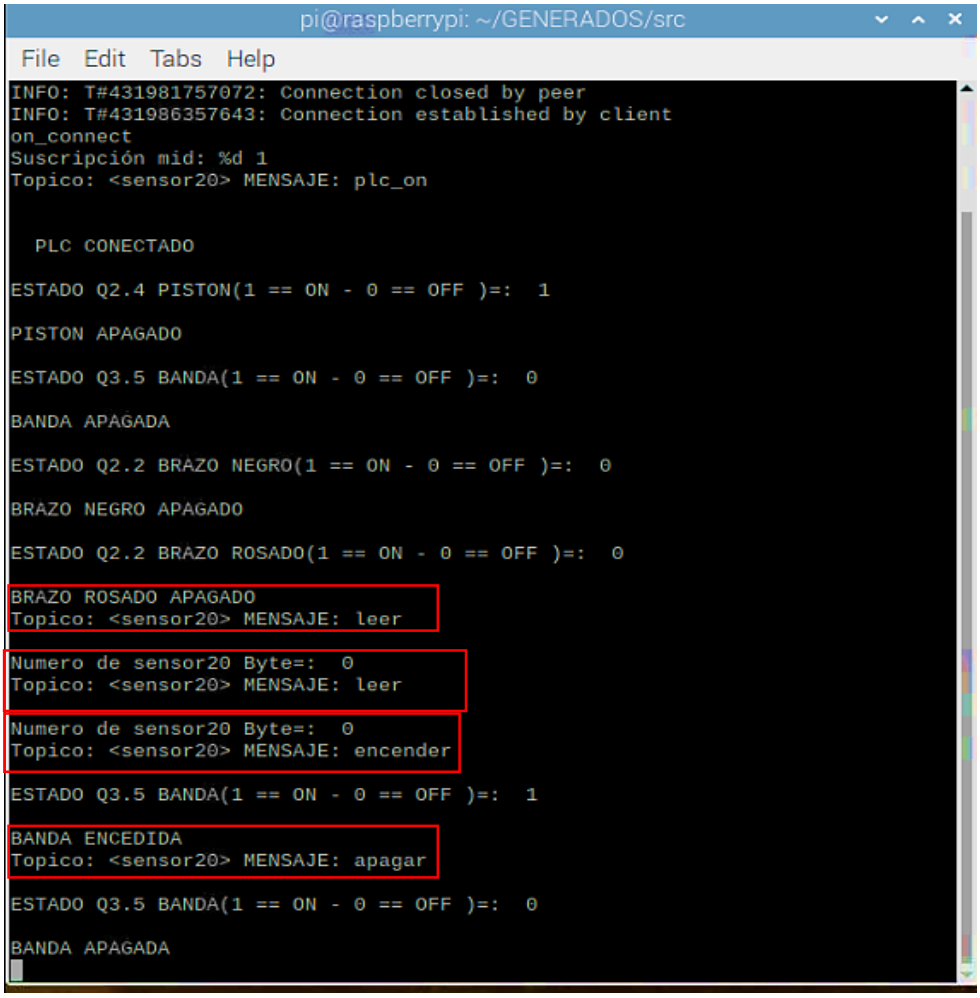


Figura 52: *Cliente enviando Tópicos al Bróker para verificar el funcionamiento de la Estación..*
Elaborado por: *La Investigadora.*

El Bróker Mosquitto se debe estar ejecutando constantemente para que pueda admitir las peticiones de un cliente remoto, el servidor espera los tópicos de acuerdo a esto distribuir la información al cliente. En la Figura 53 se puede observar que el Mosquitto recibe los datos desde la Raspberry de control y comunicación.

Figura 53: *Bróker Mosquitto ejecutándose en tiempo real..*
Elaborado por: *La Investigadora.*

El ejecutable < ./forte > de las dos Raspberry`s también comprueban el correcto funcionamiento del proceso. En la Figura 54 se puede observar cómo trabaja < ./forte > en la Raspberry de Control y en la Figura 55 se muestra como se ejecuta el archivo < ./forte_comunication > en la Raspberry de Comunicación.

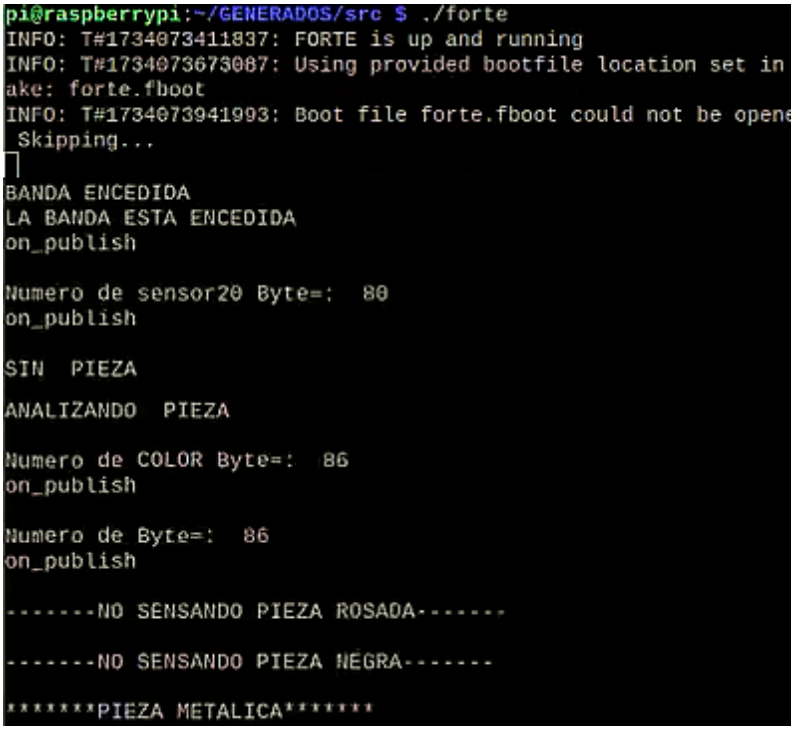


Figura 54: *Raspberry de Control ejecutando el archivo < ./forte >.*
Elaborado por: *La Investigadora.*

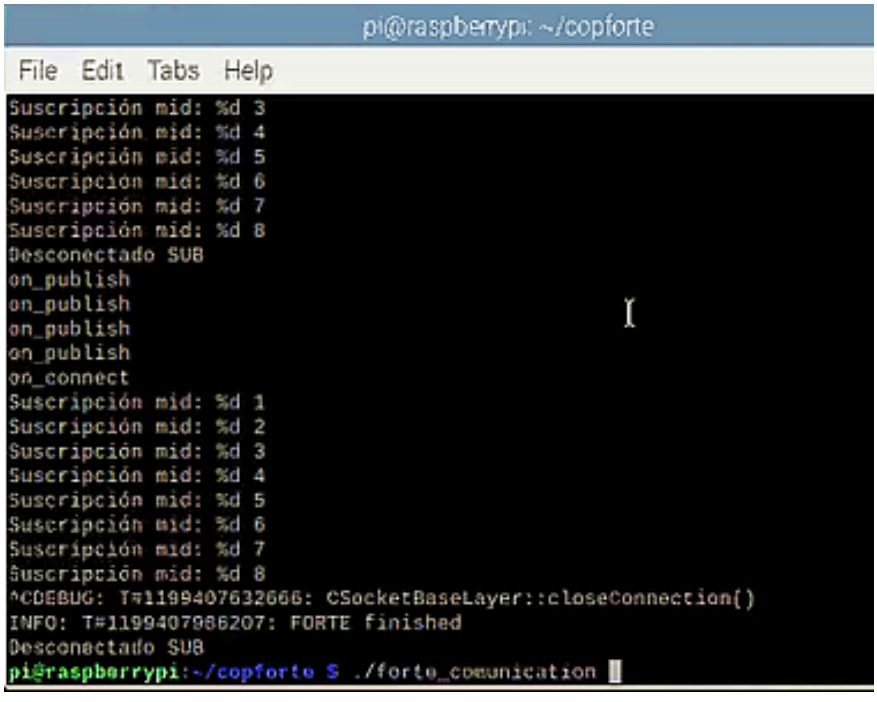


Figura 55: *Raspberry de Comunicación ejecutando el archivo < ./forte_comunication >.*
Elaborado por: *La Investigadora.*

La capa de comunicación utiliza el protocolo MQTT para enviar y recibir los mensajes o tópicos a través de la red y de esta forma se controla la Estación, este proceso se puede verificar mediante el software Wireshark, el cual analiza los protocolos que se utilizan en una red y captura el protocolo con la IP con la que se está enviando el mensaje encapsulado. En la Figura 56 se puede visualizar los paquetes que se envían mediante el protocolo MQTT.

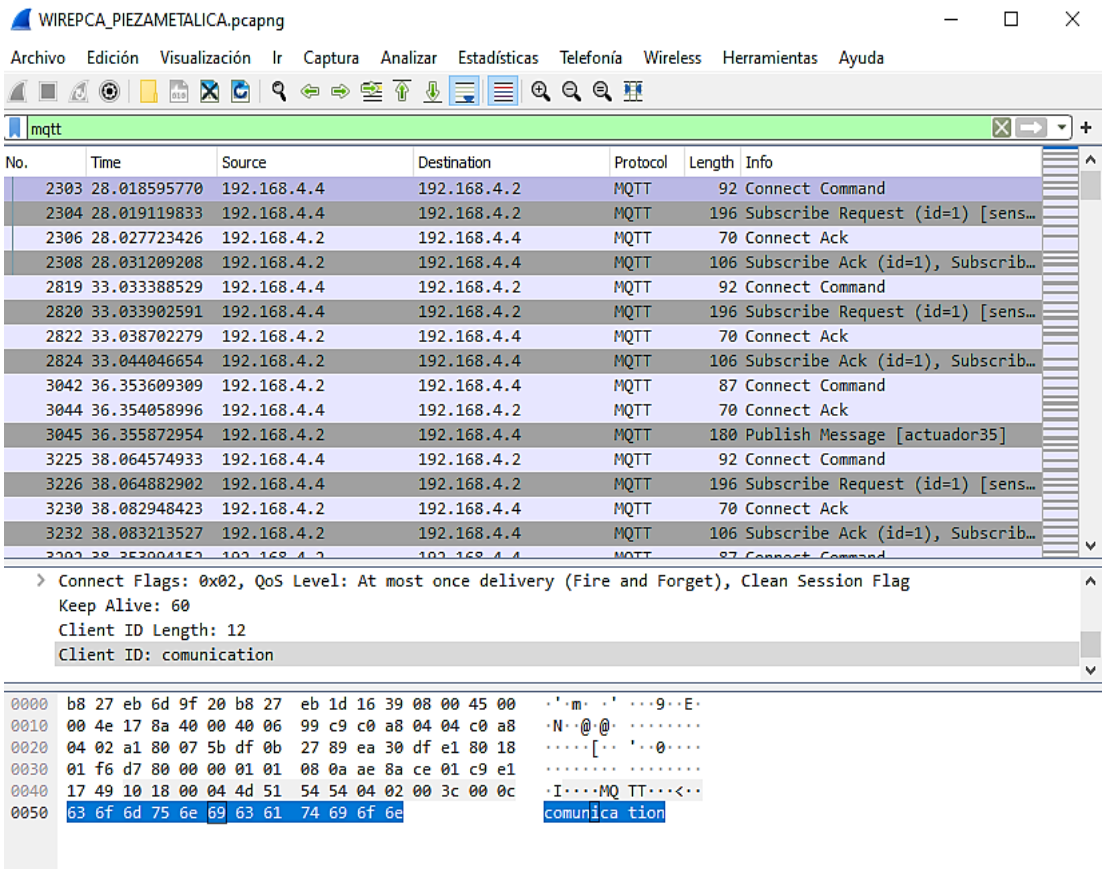


Figura 56: *Wireshark capturando datos del protocolo MQTT.*
Elaborado por: *La Investigadora.*

3.10.2 Pruebas de recepción y envió de datos.

El control de la Estación es automático, por lo cual para comprobar que todos los datos son enviados y recibidos se clasifica el funcionamiento en tres etapas, puesto que, la Estación clasifica tres piezas diferentes y cada una de estas piezas debe cumplir un proceso completo de publicación y suscripción de mensajes. Para las pruebas se envió 10 piezas en cada etapa.

En la nueva capa de comunicación se utiliza el nivel QoS – 0 del protocolo MQTT, este nivel no garantiza la recepción del mensaje, pero al no crear cola de paquetes en el bróker se considera un nivel eficiente, por lo que se procede a comprobar el porcentaje de mensajes enviados por el sistema. En las Tablas 24, 25 y 26 se enlista los tópicos del proceso con el número de envíos hacia el Bróker de cada etapa. Y las Figuras 57, 58 y 59 muestran el diagrama de flujo de datos capturado por Wireshark.

- **Etapa de funcionamiento de la Pieza Rosada.**

Tabla 24: *Número de envíos hacia el Bróker con la Pieza Rosada.*

PIEZA ROSADA			
Referencia sensores/actuadores	**Tópico**	**Éxito**	**Perdidos**
Presencia	sensor20	10	0
Colores	sensor21	10	0
Metálica	sensor22	-	-
Salida	sensor23	9	1
Banda	actuador35	10	0
Pistón	actuador24	10	0
Brazo-rosadas	actuador20	10	0
Brazo- negras	actuador22	-	-

Elaborado por: *La Investigadora.*

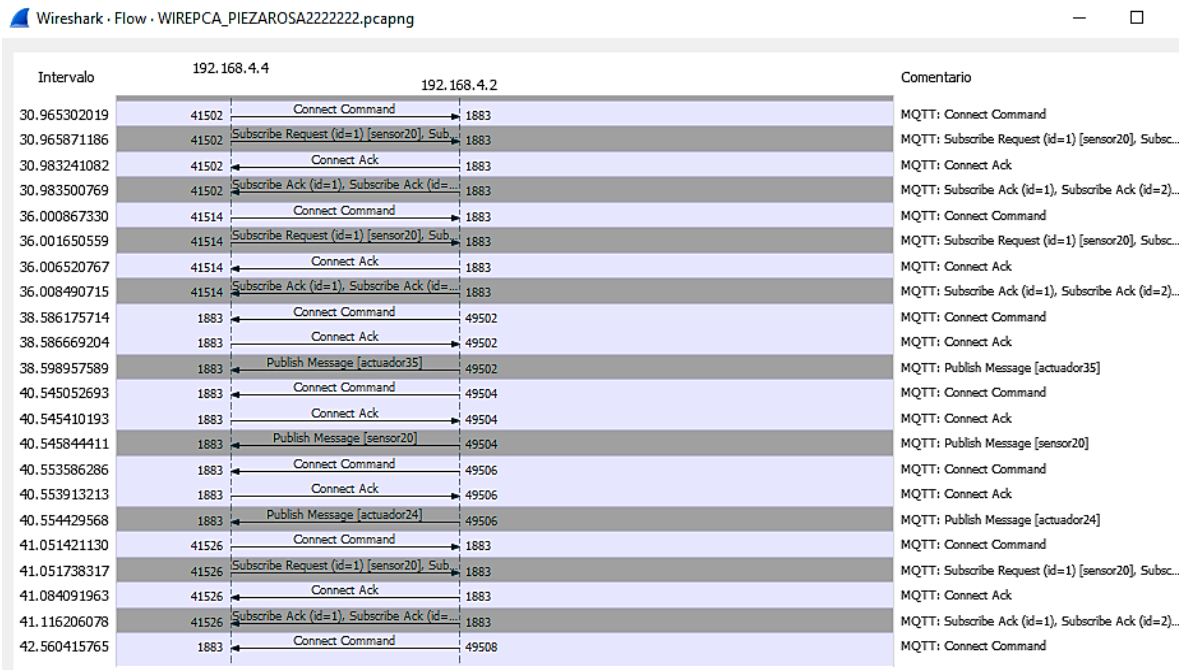


Figura 57: *Wireshark capturando datos de la Pieza Rosada.*
Elaborado por: *La Investigadora.*

- **Etapa de funcionamiento de la Pieza Negra.**

Tabla 25: *Número de envíos hacia el Bróker con la Pieza Negra.*

PIEZA NEGRA			
Referencia sensores/actuadores	**Tópico**	**Éxito**	**Perdidos**
Presencia	sensor20	10	0
Colores	sensor21	10	0
Metálica	sensor22	-	-
Salida	sensor23	9	1
Banda	actuador35	10	0
Pistón	actuador24	10	0
Brazo-rosadas	actuador20	-	-
Brazo- negras	actuador22	9	1

Elaborado por: *La Investigadora.*

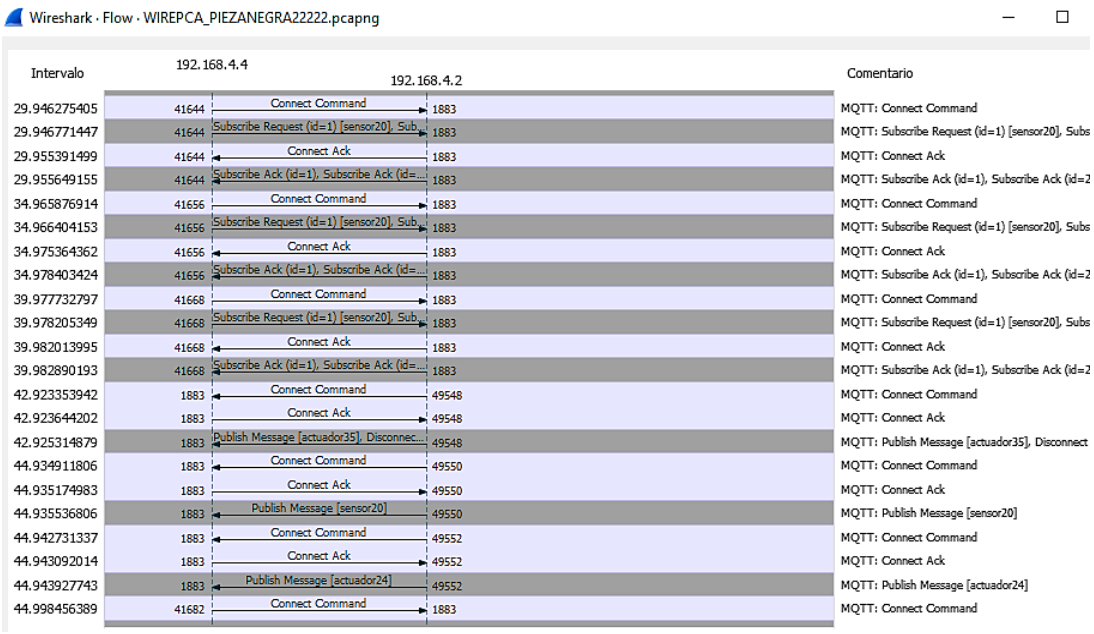


Figura 58: *Wireshark capturando datos de la Pieza Rosada.*
Elaborado por: *La Investigadora.*

- **Etapa de funcionamiento de la Pieza Metálica.**

Tabla 26: *Número de envíos hacia el Bróker con la Pieza Metálica.*

PIEZA NEGRA			
Referencia sensores/actuadores	**Tópico**	**Éxito**	**Perdidos**
Presencia	sensor20	10	0
Colores	sensor21	-	-
Metálica	sensor22	10	0
Salida	sensor23	9	1
Banda	actuador35	10	0
Pistón	actuador24	10	0
Brazo-rosadas	actuador20	-	-
Brazo- negras	actuador22	-	-

Elaborado por: *La Investigadora.*

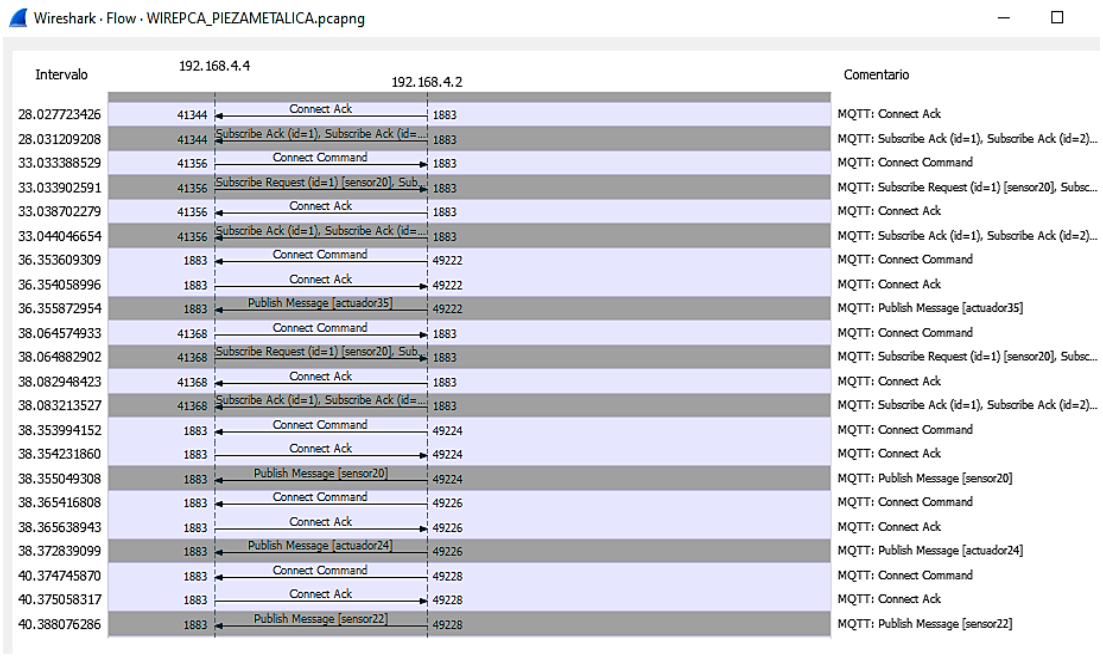


Figura 59: *Wireshark capturando datos de la Pieza Metálica.*
Elaborado por: *La Investigadora.*

3.11 Resultados

Finalmente, en la Tabla 27 se observa el total de mensajes enviados con éxito y los que se perdieron en la comunicación de todo el funcionamiento de la Estación Módulo Clasificador, seguidamente en la Figura 60 se puede ver el porcentaje de eficiencia de conexión en el Publicador y en el Suscriptor.

Tabla 27: *Total de mensajes enviados y perdidos en la Comunicación.*

COMUNICACIÓN					
PUBLISH			**SUSCRIBE**		
Tópicos	**Éxito**	**Perdidos**	**Tópicos**	**Éxito**	**Perdidos**
sensor20	30	0	actuador35	30	0
sensor21	30	0	actuador24	30	0
sensor22	30	0	actuador20	30	0
sensor23	28	2	actuador22	29	1

Elaborado por: *La Investigadora.*

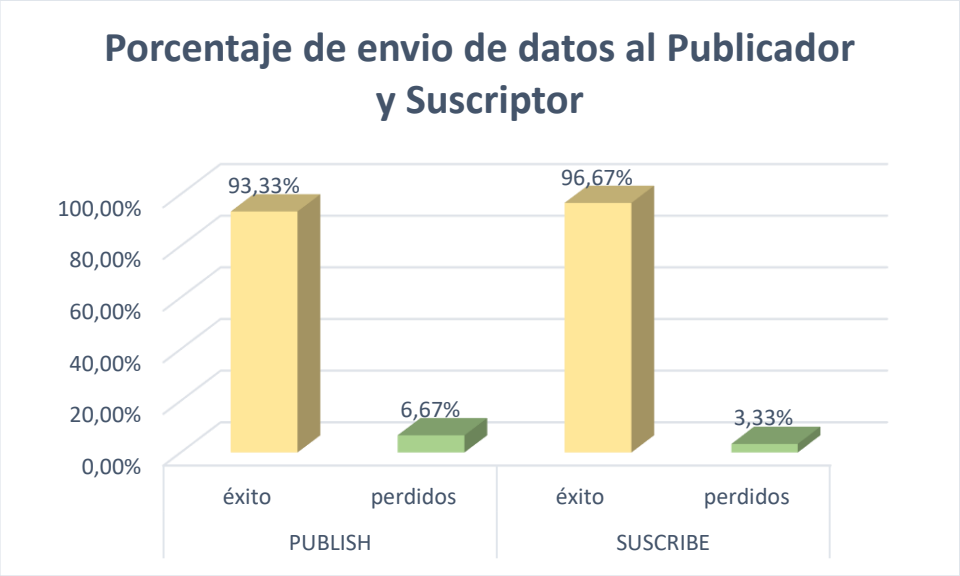


Figura 60: *Porcentaje de paquetes enviados la capa de comunicación.*
Elaborado por: *La Investigadora.*

Mediante los porcentajes obtenidos en la Figura 60 y la media aritmética de los dos resultados se concluye que la eficiencia de la capa de comunicación es del 95%. Entonces posterior a las pruebas se resume la funcionalidad de la nueva Arquitectura de Conexión.

Por medio de un nuevo Módulo Forte de Comunicación la Arquitectura de Conexión cumple cada aspecto importante de la Norma IEC-61499. Se debe tener en cuenta que el Protocolo MQTT utiliza un mecanismo de calidad del servicio QoS de tres niveles, este mecanismo sirve para gestionar la robustez de conectividad del cliente con el proceso. Para el desarrollo de este proyecto se utilizó el nivel QoS-0 porque la

arquitectura permite él envio de mensajes una única vez mediante el publicador y suscriptor para no cargar datos innecesarios al sistema.

El diagrama de secuencia de la Figura 61 ilustra el procedimiento de recepción y envio de datos internamente en la Arquitectura de Comunicación. Donde, los Bloques PUBLISH y SUSCRIBE conectan y devuelven la conexión con el Bróker a través de las instanciaciones de los archivos de configuración del Módulo Forte de Comunicación y de esta forma se pueda acceder a la publicación y suscripción de los tópicos con sus respectivos mensajes. El Bloque de CONTROL se encuentra en un ciclo loop llamando y retornando datos que se subscribieron y publicaron desde los Bloques de Sensores y Actuadores.

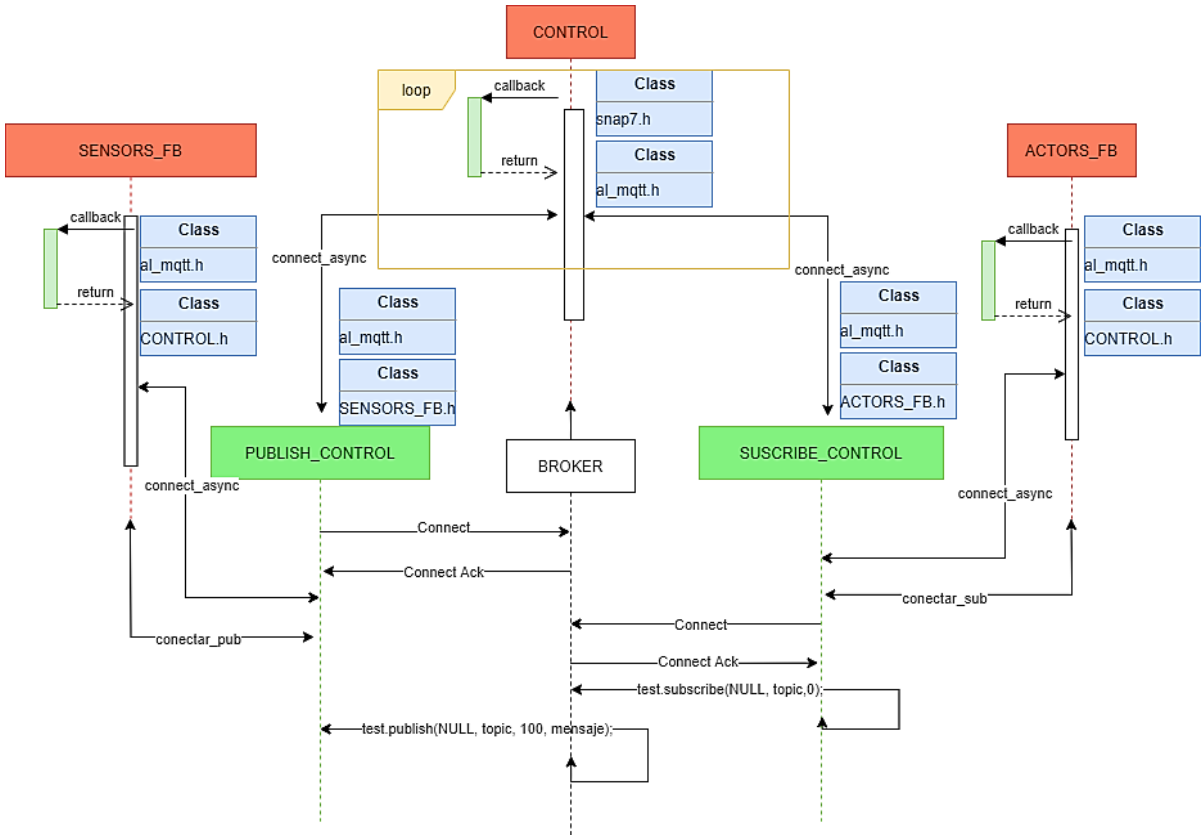


Figura 61: *Procedimiento de re-compilación del archivo < ./forte > .*
Elaborado por: *La Investigadora.*

Diagrama de clases de los Sensores

En la Figura 62 se presentan las clases utilizadas en el Bloque de sensores, en el cual constan los atributos, métodos y el tipo de relación que existe entre los demás Bloques que utiliza la aplicación de control y la arquitectura de comunicación. La clase

principal FORTE_SENSORS_FB utiliza una conexión de tipo composición que describe la relación entre un conjunto de partes y una parte de este todo, lo que significa que, si el sistema elimina el todo (la composición o conjunto de partes) también destruye todas las partes o sub clases. Y también utiliza la conexión de tipo dependencia, la cual es una relación entre la clase principal y las sub clases. Esta relación dirigida describe que un elemento depende de otro. La descripción de cada clase se presenta en la Tabla 28.

Tabla 28: *Descripción de las clases del Bloque de sensores.*

Clase	Descripción
<<Interface>> FORTE_SENSOR_FB	Clase principal para acceder a los métodos de entrada y salida del Bloque de Funciones de la aplicación. Los métodos: S_SE_TOPIC20, S_SE_TOPIC21, S_SE_TOPIC22 y S_SE_TOPIC23 permiten inicializar la cadena compuesta con la dirección IP, el puerto, el cliente y un tópico. Adicionalmente, los métodos: PRESENCE, COLOR, METAL y SE_OUTPUT son los métodos que permiten el ingreso de los valores de lectura de cada uno de los sensores. Finalmente, los datos de lectura de tipo entero son enviados a los métodos de salida, para este proceso se utiliza dos funciones getValue() y setValue().
FORTE_PUBLISH_PRESENCE	Sub clase que permite obtener el valor de lectura de entrada P_SE_ VALUE del sensor de presencia para posteriormente enviarlo al método de salida P_SE_VALUE_OUT perteneciente a la clase principal FORTE_SENSOR_FB. Este proceso se realiza a través de la clase mqtt_test, la cual se encarga de la comunicación.
FORTE_PUBLISH_COLOR	Sub clase que permite obtener el valor de lectura de entrada del sensor óptico el cual nos retorna un valor de tipo entero para distinguir el color negro o rosado de la pieza.
FORTE_PUBLISH_METAL	Sub clase que permite obtener el valor de lectura de entrada del sensor inductivo determinando si la pieza sensada es de tipo metálica. Todas las operaciones de las sub clases se las realiza por el método alg_normalOperation(void) de tipo privado ya que solo se ejecutará si fue instanciado por la clase principal.

FORTE_PUBLISH_SE_OUTPUT	Sub clase que permite obtener el valor de lectura de entrada del sensor retro – reflectivo determinando si el proceso finalizó con la pieza sensada, a través de un valor de tipo entero.
mqtt_test	Clase que permite la comunicación virtual entre los diferentes Bloques de Funciones de la aplicación y físicamente permite la recepción y envio de datos entre los diferentes dispositivos runtime. Esta clase contiene los métodos: on_connect() que permite la conexión entre la aplicación y el Broker, on_disconnect() el cual cierra las conexiones, on_publish() método en el que se publica el mensaje en su respectivo tópico, on_sucribe() método en el cual se suscribe los tópicos de la cadena de conexión. Adicionalmente se crean dos métodos para la suscripción de los sensores y actuadores con sus respectivos publicadores. Para esta clase se utiliza los atributos: dato_sensor20, dato_sensor21, dato_sensor22 y dato_sensor23 de acceso público, de tipo estático entero para poder ser actualizado.
FORTE_PUBLISH_CONTROL	Clase intermediaria entre FORTE_SENSOR_FB y FORTE_CONTROL, la cual permite el paso de los datos entre los diferentes dispositivos runtime a través de sus métodos de entrada y salida.
FORTE_CONTROL	Clase principal de la aplicación que permite obtener los valores de los sensores y mediante condiciones activar los actuadores. Los métodos de entrada de tipo entero: PRESENCE, COLOR, METAL y SE_OUTPUT, actualizan sus valores que se obtuvieron en la clase FORTE_SENSOR_FB. Además, se crea un método llamado “ConectarPlc”, el cual establece la dirección IP del PLC al que se desea conectar. Por otro lado, se tiene los métodos: LeerSensorI20, LeerSensorI21, LeerSensorI22 y LeerSensorI23, los cuales obtienen un valor de tipo byte de la lectura obtenida desde el PLC.
TS7Client	Clase que permite utilizar funciones externas para poder leer y escribir los estados de un PLC. Esta clase pertenece a la librería SNAP7.

Elaborado por: *La Investigadora.*

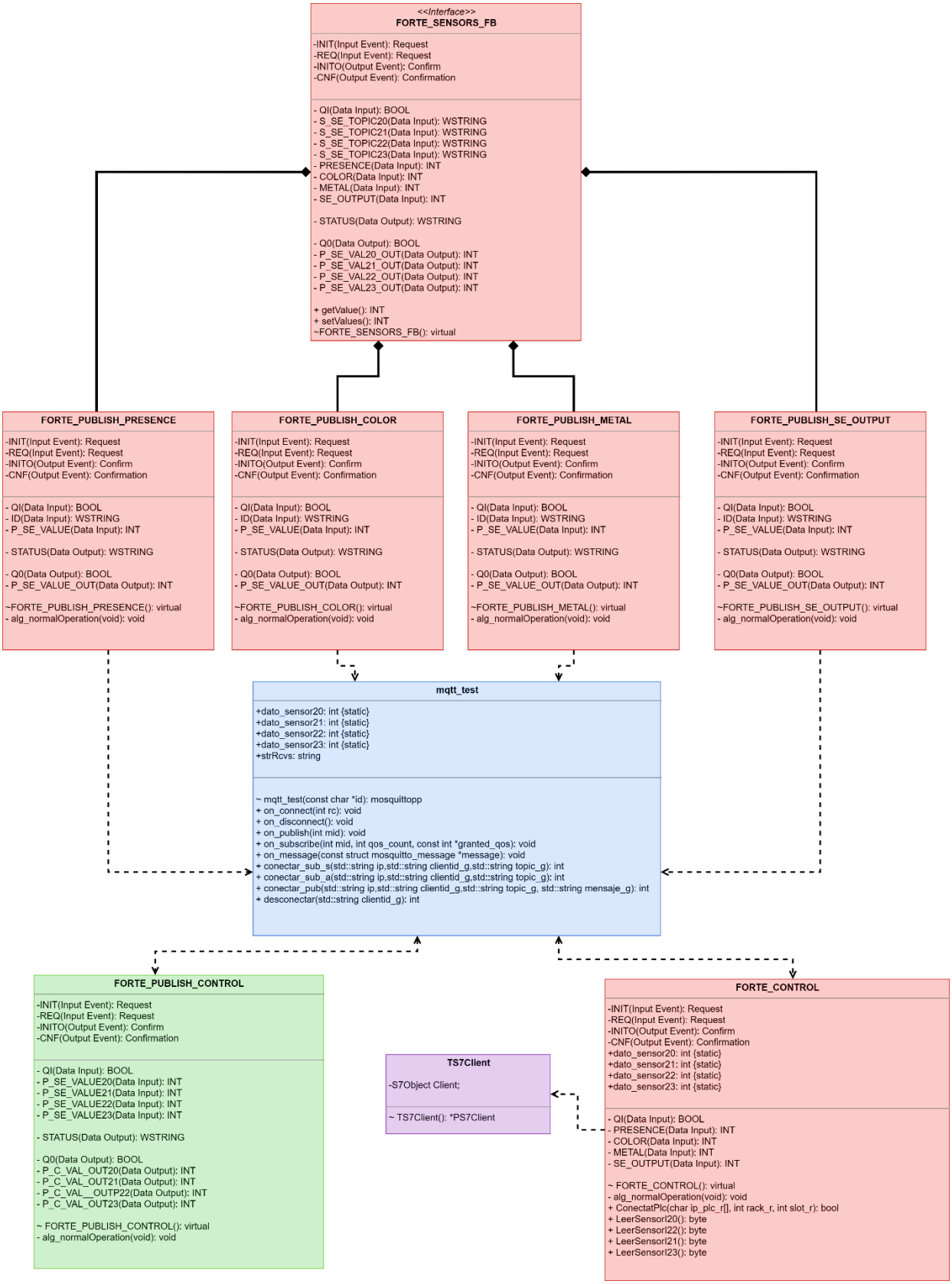


Figura 62: *Diagrama de clases de los Sensores.*
Elaborado por: *La Investigadora.*

Diagrama de clases de los Actuadores

El diagrama de la Figura 63 presentan las clases utilizadas en el Bloque de actuadores, el cual constan de atributos, métodos y tipo de relación como se muestra en el diagrama de Bloque de sensores. La clase principal FORTE_ACTORS_FB utiliza una conexión de tipo composición y también utiliza la conexión de tipo dependencia. Se describe cada clase en la Tabla 29.

Tabla 29: *Descripción de las clases del Bloque de actuadores..*

Clase	Descripción
<<Interface>> FORTE_ACTORS_FB	Clase principal para acceder a los métodos de encendido y apagado del Bloque de Funciones de la aplicación. Los métodos: S_A_VALUE35, S_A_VALUE24, S_A_VALUE_P20 y S_A_VALUE_B22, permiten inicializar la cadena conexión. Adicionalmente, se crea los métodos: BAND, PISTON, ARM_PINK y ARM_BLACK de tipo de dato de retorno bool, que permiten la activación y desactivación de los actuadores.
FORTE_SUSCRIBE_BAND	Sub clase que permite encender y apagar los estados correspondientes de los métodos P_A_VALUE35 y P_SE_VALUE_OUT. Este proceso se realiza a través de la clase mqtt_test, la cual se encarga de la comunicación.
FORTE_SUSCRIBE_PISTON	Sub clase que permite activar y desactivar el pistón el cual, bajo una condición permite el paso de las diferentes piezas. Retornando un valor de tipo de booleano para saber su estado (true / false).
FORTE_SUSCRIBE_PINK	Sub clase que permite la activación del brazo rosado, después de haber sido analizado por los sensores de lectura. Todas las operaciones de las sub clases se las realiza por el método alg_normalOperation(void).
FORTE_SUSCRIBE_BLACK	Sub clase que permite la activación del brazo negro, después de haber sido analizado por los sensores de lectura, actualizando sus métodos.

mqtt_test	Clase que permite la comunicación virtual y física de la aplicación. Esta clase contiene los métodos: on_connect(), on_disconnect(), on_publish(), on_sucribe(), estos métodos son propios de la librería Mosquitto. El método on_message() permite una llamada interna de tipo estructura de la clase Mosquitto para que se pueda publicar el mensaje al tópico correspondiente. Adicionalmente se crean dos métodos para la suscripción de los sensores y actuadores con sus respectivos publicadores. Para esta clase se utiliza los atributos: dato_estado_banda, dato_estado_piston, dato_brazo_rosado y dato_brazo_negro de acceso público, de tipo estático booleano para poder ser actualizado.
FORTE_SUSCRIBE_CONTROL	Clase intermediaria que permite pasar el estado de cada actuador entre la clase FORTE_CONTROL y FORTE_ACTORS_FB.
FORTE_CONTROL	Clase principal de la aplicación que permite la activación y desactivación de los actuadores según corresponda la lectura de cada sensor. Los métodos de salida de tipo booleano: BAND, PISTON, ARM_PINK y ARM_BLACK actualizan sus valores que se obtuvieron en la clase FORTE_ACTORS_FB. Por otro lado, se tiene los métodos: dato_estado_banda, datos_estado_piston, dato_brazo_rosado y dato_brazo_negro, los cuales instancian a la librería SNAP7 que se encuentra en la clase TS7Cliente para realizar su respectiva operación.

Elaborado por: *La Investigadora.*

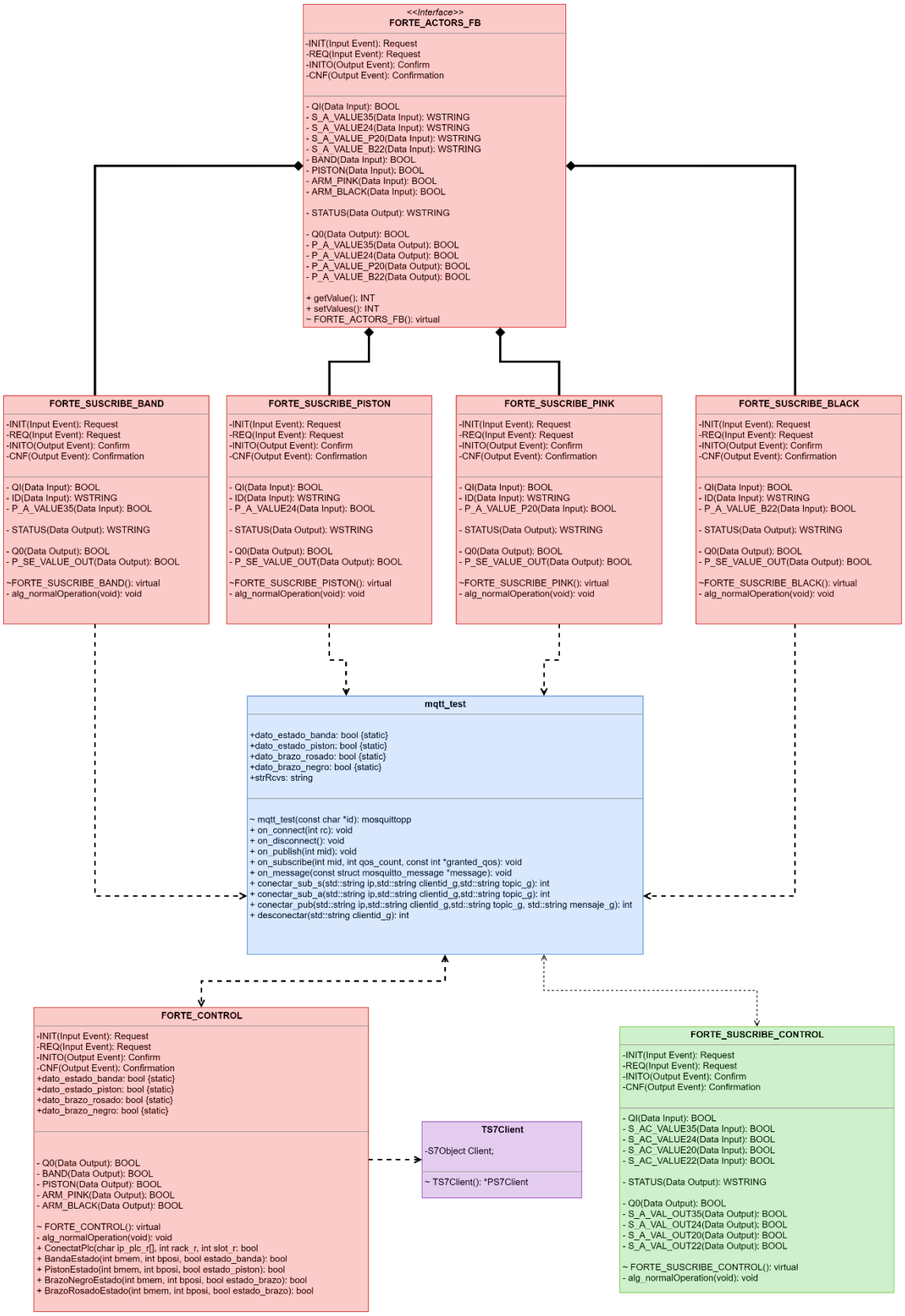


Figura 63: *Diagrama de clases de los Actuadores.*
Elaborado por: *La Investigadora.*

Seguidamente se muestra que el software 4DIAC-IDE permite visualizar la secuencia del funcionamiento de los FBs como se ilustra en la Figura 64. El monitoreo de los FBs permite verificar que los Bloques funcionan correctamente, es importante saber que cada mapeo del proceso se realiza con su respectiva Raspberry.

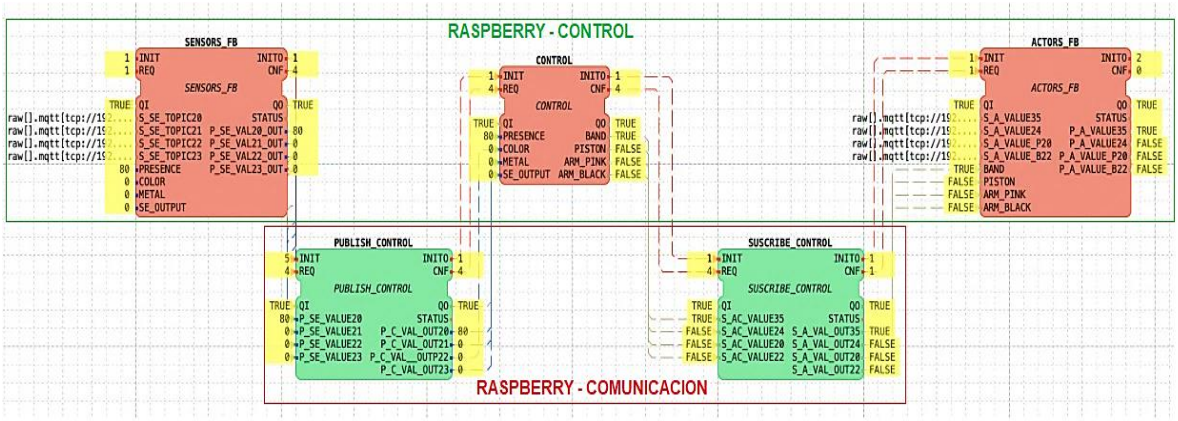


Figura 64: *Monitoreo de los FBs de control.*
Elaborado por: *La Investigadora.*

La aplicación de FB's físicamente controla la Estación Festo MPS-200-Clasificador, por lo cual sus respectivas conexiones se pueden observar en la Figura 65.

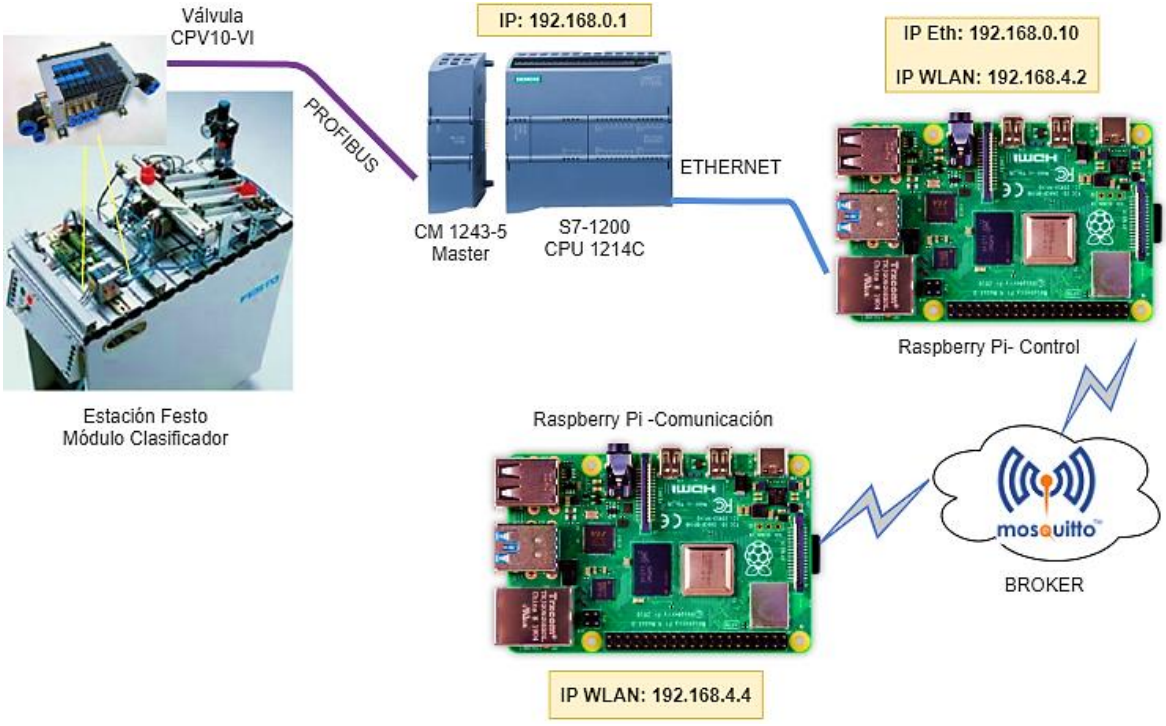


Figura 65: *Conexiones del funcionamiento físico del proyecto.*
Elaborado por: *La Investigadora.*

A continuación, se muestra en las Figura 66, 67 y 68 el funcionamiento físico del control de la Estación cuando detectan los sensores y de acuerdo a esto responden los actuadores.

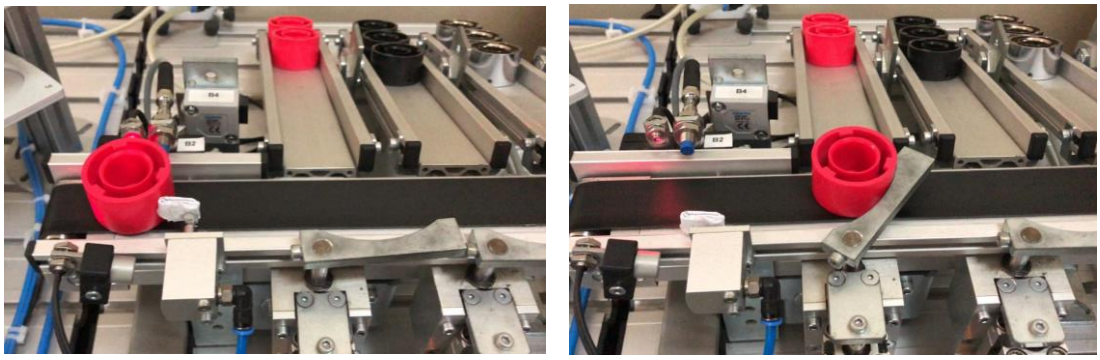


Figura 66: *Detección de la Pieza Rosada.*
Elaborado por: *La Investigadora.*

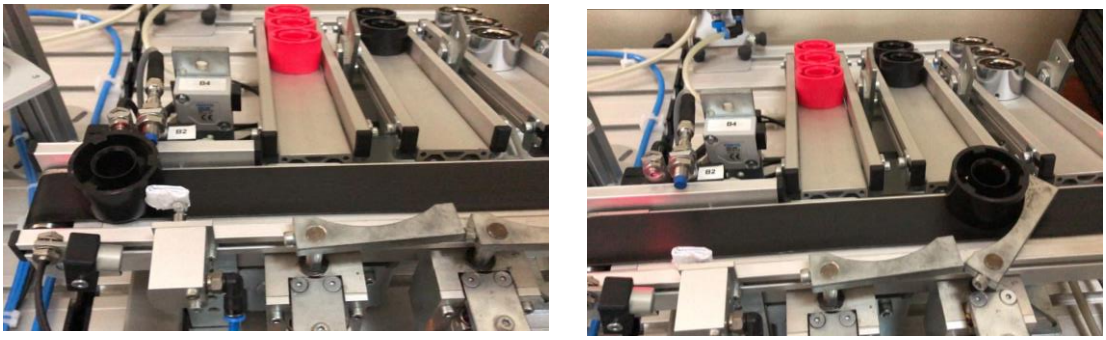


Figura 67: *Detección de la Pieza Negra.*
Elaborado por: *La Investigadora.*

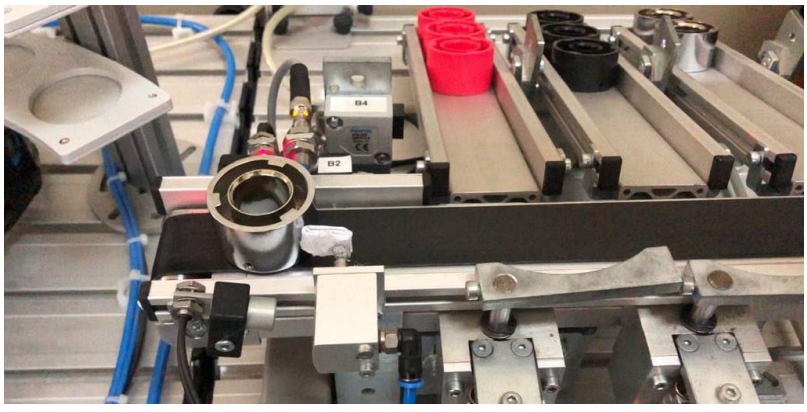


Figura 68: *Detección de la Pieza Metálica.*
Elaborado por: *La Investigadora.*

CAPÍTULO IV

4 CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- ✓ La nueva Arquitectura de Comunicación cumple con los requerimientos establecidos por la Norma IEC-61499, además, su estructura de programación se puede reutilizar para incluir cualquier tipo de protocolo utilizado en la Industria 4.0. El desarrollo de la capa de comunicación utilizó el protocolo MQTT que funciona con las librerías de Mosquitto, las cuales se encargan de gestionar la recepción y envió de datos hacia los Publicadores y Suscriptores del proceso.

- ✓ La capa de comunicación tiene una eficiencia del 95% en la transmisión de datos del proceso de control de la Estación Festo MPS-200-Clasificador. Se obtiene este porcentaje debido a que el protocolo MQTT trabaja con un nivel de QoS – 0. Pero aun así, el 95% se considera optimo ya que soporta el transporte de datos bidireccionalmente y adicional a esto se basa en el protocolo TCP/IP que lo hace más robusto. Asimismo, este protocolo es el más utilizado en las IIOT (Internet Industrial de la Cosas) porque permite el acceso a varias conexiones remotas.

- ✓ Las plataformas de diseño y ejecución de 4DIAC, permiten implementar variedad de algoritmos de control, mediante Bloques de Funciones ya que utilizan lenguajes de programación de código abierto, con el fin de promover la investigación en la automatización industrial, además facilita el monitoreo de la secuencia de los FB´s lo cual sirve para evidenciar el correcto funcionamiento de los mismos.

- ✓ El control de la Estación Festo MPS-200-Clasificador comprobó la viabilidad del proyecto a través de las pruebas de funcionamiento, debido a que se realizaron 30 intentos de comunicación, al enviar 10 veces cada pieza se perdieron 2 mensajes en el publicador y 1 en el suscriptor, esto a causa de las latencias en la conexión

con un error del 5% en la arquitectura de comunicación, obteniendo un sistema fiable y flexibles.

4.2 Recomendaciones

A quienes continúen con trabajos futuros en base a este proyecto de investigación, se recomienda lo siguiente:

- ✓ Se sugiere para este tipo de controles linealizar y/o calibrados los sensores del proceso para que de esta forma se facilite la recolección de datos.

- ✓ Se recomienda utilizar dispositivos de bajo costo porque son adaptables a varios tipos de sistemas operativos en los cuales su programación es compartida para que cualquier desarrollador pueda contribuir con ideas, modificar o incluir nuevas configuraciones.

- ✓ Para que el desempeño de dispositivo Raspberry Pi mejore, se sugiere utilizar un ventilador externo con el fin de evitar el recalentamiento de la tarjeta cuando se está ejecutando los procesos internamente, además se debería colocar en una carcasa compacta para que resista a un entorno industrial.

- ✓ Se sugiere complementar el presente proyecto añadiendo más clientes con plataformas webs y móviles con las cuales se pueda visualizar y controlar el proceso que realiza la Estación Festo MPS-200 Módulo Clasificador. Estas plataformas sirven como un entorno amigable para los usuarios.

4.3 Bibliografía

[1] G. J. Caiza and M. V. García, “Implementación de sistemas distribuidos de bajo costo bajo norma IEC-61499, en la estación de clasificación y manipulación del MPS 500,” *Ingenius,* no. 18, p. 40, Jul. 2017.

[2] M. Minhat, V. Vyatkin, X. Xu, S. Wong, and Z. Al-Bayaa, “A novel open CNC

architecture based on STEP-NC data model and IEC 61499 function blocks," *Robot. Comput. Integr. Manuf.*, vol. 25, no. 3, pp. 560–569, 2009.

[3] M. García, E. Irisarri, and F. Pérez, "Integración Vertical en plantas industriales utilizando OPC UA e IEC-61499," *Enfoque UTE*, p. 287, Feb. 2017.

[4] H. Derhamy, D. Drozdov, S. Patil, J. Van Deventer, J. Eliasson, and V. Vyatkin, "Orchestration of Arrowhead services using IEC 61499: Distributed automation case study," *IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA*, vol. 2016-Novem, 2016.

[5] E. Castellanos and C. Garcia, "Tema : Control distribuido en procesos industriales utilizando sistemas empotrados de bajo costo, para verificar la aplicabilidad de la Norma IEC-61499," p. 140, 2017.

[6] G. M. Mafla, "Desarrollo de un sistema distribuido bajo la Norma IEC-61499 para control de Robot Kuka modelo YouBot," Escuela Superior Politecnica de Chimborazo, 2019.

[7] S. P. Bustos, "Desarrollo de un sistema de control industrial basado en el Estandar IEC-61499," Universidad Tecnica de Ambato, 2017.

[8] I. C. Office, "International Standards and Conformity Assessment for all electrical, electronic and related technologies," 2018. [Online]. Available: https://www.iec.ch/about/globalreach/?ref=menu. [Accessed: 17-Jan-2020].

[9] M. Felix and D. Dario, "Arquitectura Inteligente CPPS Integrada en el Uso de la Norma IEC-61499, con Bloque de Funciones Altamente Adaptables en la Industria 4.0," in *KnE Engineering*, 2018, vol. 3, no. 1, p. 502.

[10] A. Ortiz, "De Industria 4.0, necesidades de conectividad y guerras de protocolos y estándares," *Martes, 24 de septiembre*, 2019. [Online]. Available: http://www.automaticaeinstrumentacion.com/es/notices/2018/11/de-industria-4.0-necesidades-de-conectividad-y-guerras-de-protocolos-y-estandares-45153.php#.XYlHMChKjIU. [Accessed: 23-Sep-2019].

[11] M. Garcia, "Implementación de Sistemas Empotrados de Control Distribuidos bajo el Estándar IEC-61499," Universidad del Pais Vasco, 2013.

[12] K. Thramboulidis, “IEC 61499 vs. 61131: A Comparison Based on Misperceptions,” p. 12, 2014.

[13] E. Querol, J. A. Romero, A. M. Estruch, and F. Romero, “Norma iec-61499 para el control distribuido. aplicación al cnc.,” pp. 3–5, 2014.

[14] A. Zoitl and V. Vyatkin, “IEC 61499 architecture for distributed automation: The ‘glass half full’ view,” *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 7–22, 2009.

[15] C. Catalan, “Modelos y plataforma IEC 61499 adaptados al control distribuido de máquinas herramienta en sistemas de fabricación ágil,” Universidad de Zaragoza, 2016.

[16] A. Zoitl, C. Sünder, and I. Terzic, “Dynamic reconfiguration of distributed control applications with reconfiguration services based on EEC 61499,” *Proc. - DIS 2006 IEEE Work. Distrib. Intell. Syst. - Collect. Intell. Its Appl.*, vol. 2006, pp. 109–114, 2006.

[17] C. Eclipse, “4diac IDE – IEC 61499 Compliant Development Environment,” 2017. [Online]. Available: https://www.eclipse.org/4diac/en_ide.php. [Accessed: 23-Sep-2019].

[18] M. Garcia, “Metodologías Para El Diseño De Sistemas De Control Distribuido Bajo El Estándar Iec 61499 Aplicados Al Control De Procesos,” Universidad del Pais Vasco, 2018.

[19] A. Romero, “Diseño e implementación de un sistema de control distribuido en el banco de pruebas neumático de la Escuela de Ingeniería Mecánica de la Universidad del Valle con base en la Norma IEC 61499,” Universidad de Valle, 2016.

[20] J. L. Correa Guambuguete, “Estudio y diseño de la reorganización y ampliación de planta externa, de ANDINATEL S.A. mediante nodos de acceso externo de nueva generación, con capacidad de servicios de banda ancha, en Cubaya sector la de servicios de banda ancha, en Cumbayá sector la ,” Universidad de las Fuerzas Armadas ESPE, 2007.

[21] F. Eclipse, “Communication Architecture,” 2018. [Online]. Available: https://www.eclipse.org/4diac/documentation/html/development/forte_communicationArchitecture.html. [Accessed: 02-Mar-2020].

[22] P. Holger, “4Diac (Fortiss) - FED4SAE,” 2018. [Online]. Available: https://fed4sae.eu/advanced-platforms/advanced-technologies/4diac-fortiss/. [Accessed: 04-Mar-2020].

[23] M. Brown, “nxtControl: Engineering-Tool nxtSTUDIO,” *22 marzo*, 2018. [Online]. Available: https://www.nxtcontrol.com/. [Accessed: 05-Mar-2020].

[24] J. C. Mercé, “Desarrollo de un Sistema de Control en Red mediante el Estandar IEC-61499,” Universitat Jaume I, 2015.

[25] R. Automation, “Tecnología ISaGRAF,” 2015. [Online]. Available: https://www.rockwellautomation.com/es_EC/detail.page?pagetitle=Isagraf&content_type=tech_data&docid=209076c017d6dd586c895e9e3a4856e4&redirect_type=tld&redirect_url=www.isagraf.com. [Accessed: 05-Mar-2020].

[26] F. Aristeguieta, “CODESYS Runtime,” 2019. [Online]. Available: https://larraioz.com/codesys/productos/runtime. [Accessed: 08-Oct-2019].

[27] SoftPLC, “CODESYS Runtime acerca los dispositivos a la Industria 4.0,” *20 febrero*, 2019. [Online]. Available: https://www.infoplc.net/noticias/item/106222-codesys-runtime-industria-4-0. [Accessed: 08-Oct-2019].

[28] P. Juan, “Automatización de bajo costo – J KEY.” [Online]. Available: https://j-key.com.ar/automatizacion-y-digitalizacion-de-bajo-costo/. [Accessed: 11-Mar-2020].

[29] C. Rexroth a Bosh, “Reducción de costos con un sistema de automatización integrado - infoPLC,” *junio*, 2017. [Online]. Available: https://www.infoplc.net/documentacion/6-industria-de-automatizacion/1000-reduccion-de-costos-con-un-sistema-de-automatizacion-integrado. [Accessed: 11-Mar-2020].

[30] INCIBE-CERT, “Automatización de bajo coste,” *22/03*, 2018. [Online].

Available: https://www.incibe-cert.es/blog/automatizacion-coste. [Accessed: 11-Mar-2020].

[31] J. del Val, “Industria 4.0: la transformación digital de la industria | Revista Ingeniería,” *18 marzo*, 2016. [Online]. Available: https://revistaingenieria.deusto.es/industria-4-0-la-transformacion-digital-de-la-industria/. [Accessed: 24-Mar-2020].

[32] S. Thiede, M. Juraschek, and C. Herrmann, “Implementing Cyber-physical Production Systems in Learning Factories,” *Procedia CIRP*, vol. 54, pp. 7–12, 2016.

[33] T. (CARSA) Lalanne, S. Asanova, L. Azcona, and K. Dervojeda, “Cyber-Physical Productio Systems,” no. 7, pp. 1–27, 2017.

[34] L. Monostori, “Cyber-physical production systems : Roots , expectations and R & D challenges,” *Procedia CIRP*, vol. 17, pp. 9–13, 2014.

[35] I. Calle, “Industria 4.0 la cuarta revolución industrial inteligente,” *01/14*, 2017. [Online]. Available: https://www.cic.es/industria-40-revolucion-industrial/. [Accessed: 27-Mar-2020].

[36] X. Lopez, “Industria 4.0 para la monitorizacion de un proceso industrial,” Universidad Tecnica de Ambato, 2019.

[37] J. L. del Val Román, “Industria 4.0. La Transformación Digital de la Industria Española,” *Coddiinforme*, p. 120, 2012.

[38] E. Carrera, “Bienvenido a la Industria 4.0, la cuarta revolución industrial,” 2017. [Online]. Available: https://www.michaelpage.es/advice/empresas/desarrollo-profesional-y-retención-de-talento/bienvenido-la-industria-40-la-cuarta. [Accessed: 28-Mar-2020].

[39] A. Grupo, “Industria 4.0 Qué es Ventajas e Inconvenientes para Implantarlo Paso a Paso,” 2017. [Online]. Available: http://www.aldakin.com/industria-4-0-que-es-ventajas-e-inconvenientes/. [Accessed: 28-Mar-2020].

[40] M. C. de Ciberdefensa, “Caminar con éxito hacia la Industria 4.0: Capítulo 12 – Infraestructuras (II) Protocolos,” *junio*, 2019. [Online]. Available: https://ticnegocios.camaravalencia.com/servicios/tendencias/caminar-con-exito-hacia-la-industria-4-0-capitulo-12-infraestructuras-ii-protocolos/. [Accessed: 23-Sep-2019].

[41] R. Blanco, J. Fontrodona, and C. Poveda, “La industria 4.0: El estado de la cuestión,” *Econ. Ind.*, no. 406, pp. 151–164, 2017.

[42] M. Vallenar, “La Internet Industrial de las Cosas,” *13 agosto*, Chile, 2016.

[43] M. Artiaga, “IIoT, ¿qué es el Internet Industrial de las Cosas?,” *29 enero*, 2019. [Online]. Available: https://enviraiot.es/iiot-que-es/. [Accessed: 02-Apr-2020].

[44] O. Serra, “Internet de las Cosas, a nivel industrial,” *10 octubre*, 2018. [Online]. Available: https://blog.techdata.com/ts/latam/internet-de-las-cosas-a-nivel-industrial. [Accessed: 03-Apr-2020].

[45] C. A. Salazar Serna and L. C. Correa Ortiz, “Buses de campo y protocolos en redes industriales,” *Vent. Inform.*, no. 25, pp. 83–109, 2016.

[46] Universidad de Valencia, “Redes de comunicación industriales,” *Estud. Gen. Val.*, vol. I, pp. 39–60, 2015.

[47] J. Titos, “Buses de campo, la renovación en las comunicaciones industriales.,” *20 noviembre*, 2019. [Online]. Available: https://revistadigital.inesem.es/gestion-integrada/buses-de-campo/. [Accessed: 10-Apr-2020].

[48] H. Lanzarote, “Arquitecturas de Sistemas Distribuidos y conceptos de Software,” pp. 1–23, 2011.

[49] M. Castro, G. Diaz, F. Mur, R. Fernandez, E. Ruiz, and J. Blanes, *Comunicaciones Industriales: Sistemas Distribuidos y Aplicaciones*, UNED. Madrid, 2007.

[50] A. Siemens, “Controlador programable S7-1200 - Datasheet,” 2014.

[51] Siemens AG, “CM 1243-5 - PROFIBUS - Datasheet,” 2011.

[52] A. Siemens, “Estaciones MPS®: Descripción del sistema - MPS® Sistema de Producción Modular,” *19 enero*, 2017. [Online]. Available: https://www.festo-didactic.com/es-es/productos/mps-sistema-de-produccion-modular/descripcion-del-sistema/estaciones-mps-sistemas-mecatronicos-para-campeones-mundiales.htm?fbid=ZXMuZXMuNTQ3LjE0LjE4LjY0NC43NjMy. [Accessed: 20-May-2020].

[53] I. Porro, “IoT: protocolos de comunicación, ataques y recomendaciones,” *febrero 17*, 2019. [Online]. Available: https://www.incibe-cert.es/blog/iot-protocolos-comunicacion-ataques-y-recomendaciones. [Accessed: 14-Jul-2020].

[54] A. Jesús González García, J. López, V. Xavi, and V. Guillen, “IoT: Dispositivos, tecnologías de transporte y aplicaciones,” Catalunya, 2017.

[55] E. López, *Raspberry Pi Fundamentos y Aplicaciones - Eugenio López Aldea*, RA-MA-Edit. Madrid, 2017.

[56] R. Remigio and R. Bast, *CMake Cookbook: Building, testing, and packaging modular software with modern Cmake*, Safis Edit. Birmingham, 2018.

[57] F. Eclipse, “Eclipse Mosquitto,” *18 junio*, 2018. [Online]. Available: https://mosquitto.org/. [Accessed: 15-Jul-2020].

[58] S. Preeker and M. Küsel, “Step7 Open Source Ethernet Communication Suite,” *31 agosto*, 2017. [Online]. Available: http://snap7.sourceforge.net/.

[59] I. M. Castro, “Internet de las Cosas-Sistemas Embebidos Avanzados,” Madrid, 2018.

ANEXOS

ANEXO 1 - DATASHEET - RASPBERRY PI 3 MODELO B

Farnell element14 — http://uk.farnell.com/buy-raspberry-pi

Newark element14 — http://www.newark.com/buy-raspberry-pi

RASPBERRY PI 3 MODEL B

Technical Specification:

Processor

- Broadcom BCM2387 chipset.
- 1.2GHz Quad-Core ARM Cortex-A53 (64Bit)

802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)

- IEEE 802.11 b / g / n Wi-Fi. Protocol: WEP, WPA WPA2, algorithms AES-CCMP (maximum key length of 256 bits), the maximum range of 100 meters.
- IEEE 802.15 Bluetooth, symmetric encryption algorithm Advanced Encryption Standard (AES) with 128-bit key, the maximum range of 50 meters.

GPU

- Dual Core Video Core IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated Open VG, and 1080p30 H.264 high-profile decode.
- Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure

Memory

- 1GB LPDDR2

Operating System

- Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT

Dimensions

- 85 x 56 x 17mm

Power

- Micro USB socket 5V1, 2.5A

ANEXO 2 - DATASHEET - RASPBERRY PI 3 MODELO B+

Overview

Specifications

Processor:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memory:	1GB LPDDR2 SDRAM
Connectivity:	■ 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE ■ Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps) ■ 4 × USB 2.0 ports
Access:	Extended 40-pin GPIO header
Video & sound:	■ 1 × full size HDMI ■ MIPI DSI display port ■ MIPI CSI camera port ■ 4 pole stereo output and composite video port
Multimedia:	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support:	Micro SD format for loading operating system and data storage
Input power:	■ 5V/2.5A DC via micro USB connector ■ 5V DC via GPIO header ■ Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature, 0–50°C
Compliance:	For a full list of local and regional product approvals, please visit www.raspberrypi.org/products/raspberry-pi-3-model-b+
Production lifetime:	The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.

ANEXO 3 – DATASHEET - UMIC.200

Automation Controller

µMIC.200

Most robust, fast, secure.

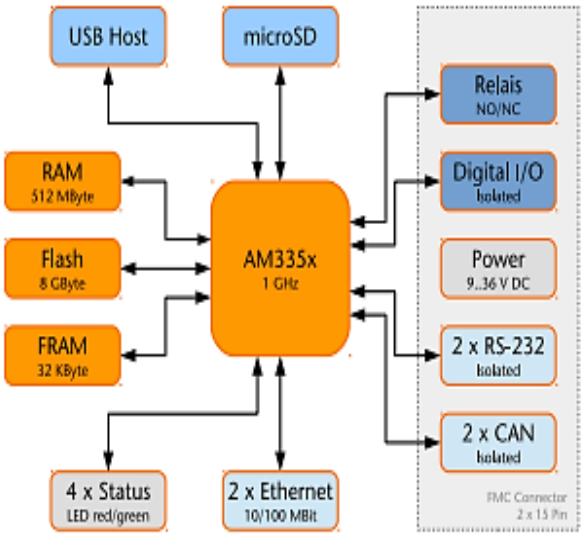
This freely programmable Automation Controller µMIC.200 is equipped with an ARM Controller which may be clocked to a maximum of 1 GHz. The module provides 512 MByte RAM as well as 8 GByte Flash memory which may be extended by a microSD-Slot. Its maximum power consumption is 4 Watts.

The Automation Controller can communicate with peripheral devices via double Ethernet, CAN or serial interfaces. Additional peripherals (e.g. WLAN adapter) may be conntected via USB.

Equipped with eight configurable digital in-/outputs, a relay, four freefly controllable LEDs and a real-time clock the µMIC.200 may be used in a multitude of different applications.

Features

- 2 x Ethernet interface (IEEE 802.3)
- 2 x CAN interface (isolated)
- 2 x RS-232 interface (isolated)
- Real-time clock (buffered without battery)
- Cortex-A8 CPU with 1 GHz clock frequency
- 8 GByte Flash and 512 MByte DDR3 RAM
- FRAM memory for remanent data
- Memory extendable via microSD card
- Power supply voltage 9..36 V DC
- Operating temperature -40 °C..+85°C
- Max. power consumption max. 4 Watt
- Real-time Linux operating system (Kernel 4.1.18)
- Integrated CANopen Master functionality



ANEXO 4 - DATASHEET - BEAGLEBONE BLACK

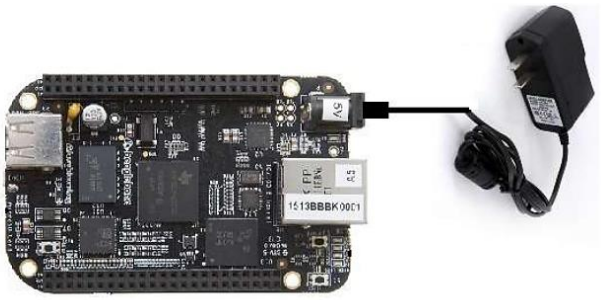


Figure 13. External DC Power

4.2 BeagleBone Black Features and Specification

This section covers the specifications and features of the board and provides a high level description of the major components and interfaces that make up the board.

Table 2 provides a list of the features.

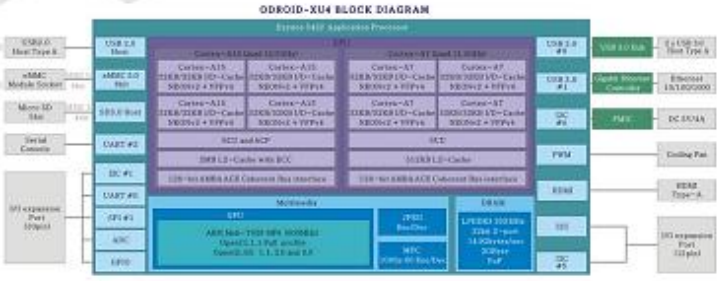
Table 2. BeagleBone Black Features

	Feature	
Processor	Sitara AM3359AZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 606MHZ	
Onboard Flash	2GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(65), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 3 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

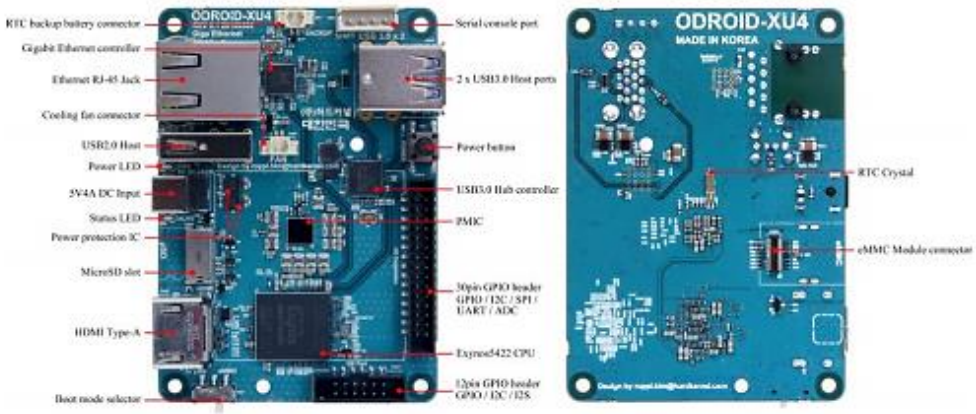
ANEXO 5 - DATASHEET - ODROID - XU4

Block Diagram

The following diagram illustrates conceptually how the components of the XU4 fit together:



XU4 Block Diagram and Annotated Board Image



Processor	Samsung Exynos5 Octa ARM Cortex™-A15 Quad 2Ghz and Cortex™-A7 Quad 1.3GHz CPUs
Memory	2Gbyte LPDDR3 RAM at 933MHz (14.9GB/s memory bandwidth) PoP stacked
3D Accelerator	Mali-T628 MP6(OpenGL ES 3.0/2.0/1.1 and OpenCL 1.1 Full profile)
Video	supports 1080p via HDMI cable(H.264+AAC based MP4 container format)
Video Out	Standard Type A HDMI connector
Audio	Instead of On-board Audio codec, There is I2S Expansion Port(CON11)
USB3.0 Host	SuperSpeed USB standard A type connector x 2 port, Max Load: total 2Amp for two USB 3.0 host ports
USB2.0 Host	High Speed standard A type connector x 1 ports, Max Load: 500mA/port
Display	HDMI monitor
Storage (Option)	MicroSD Card Slot, eMMC module socket : eMMC 5.0 HS4000 Flash Storage
Gigabit Ethernet LAN	10/100/1000Mbps Ethernet with RJ-45 Jack (Auto-MDIX support)
Serial console port	Connecting to a PC gives access to the Linux console. You can see the log of the boot, or to log in to the C1 to change the video or network settings. Note that this serial UART uses a 1.8 volt interface. We recommend the USB-UART module kit from Hardkernel. Molex 5268-04a(2.5mm pitch) is mounted on the PCB. Its mate is Molex 50-37-5043 Wire-to-Board Crimp Housing.
RTC (Real Time Clock) backup battery connector	If you want to add a RTC functions for logging or keeping time when offline, just connect a Lithium coin backup battery (CR2032 or equivalent). All of the RTC circuits are included on the ODROID-XU4 by default. Molex 53398-0271 1.25mm pitch Header, Surface Mount, Vertical type (Mate with Molex 51021-0200)
WiFi (Option)	USB WIFI Modules
HDD/SSD SATA interface (Optional)	SuperSpeed USB (USB 3.0) to Serial ATA3 adapter for 2.5"/3.5" HDD and SSD storage
Power (included)	5V 4A Power
Case(Option)	Mechanical case & cooler (90 x 59 x 28 mm approx.)
PCB Size	83 x 58 x 20 mm approx.

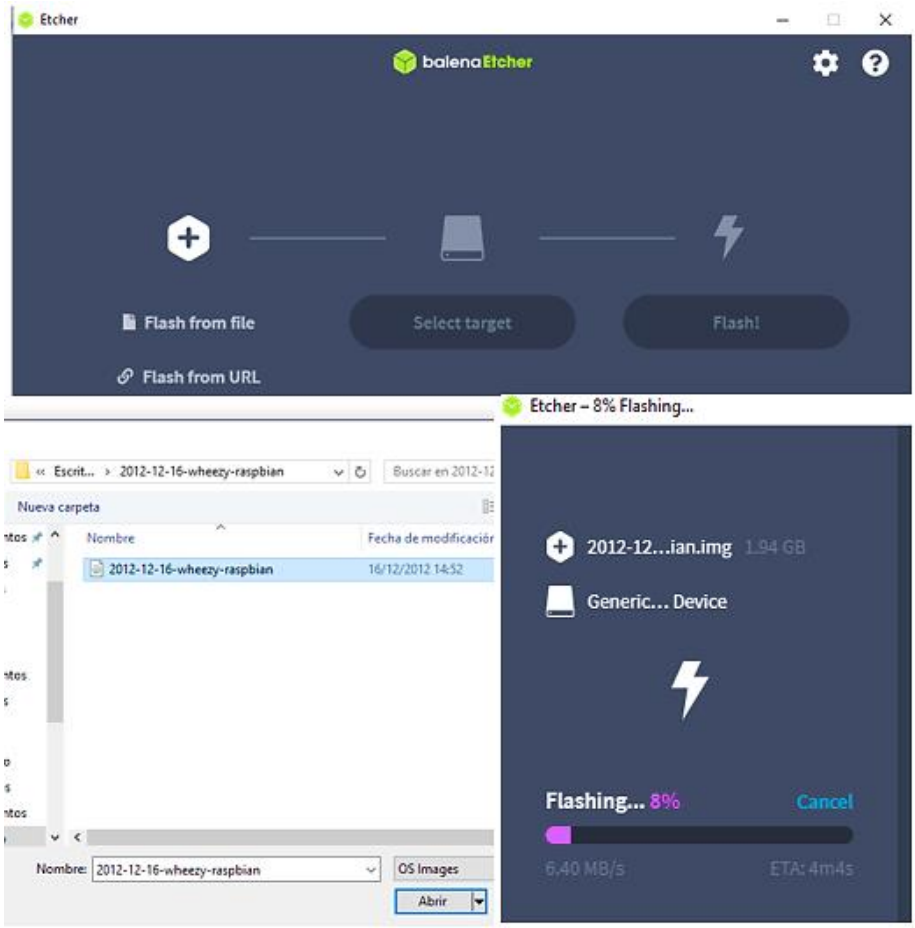
ANEXO 6 – Instalación de Raspbian

Se debe acceder a la página oficial de Raspberry Pi para poder descargar la imagen .iso del sistema operativo Raspbian.

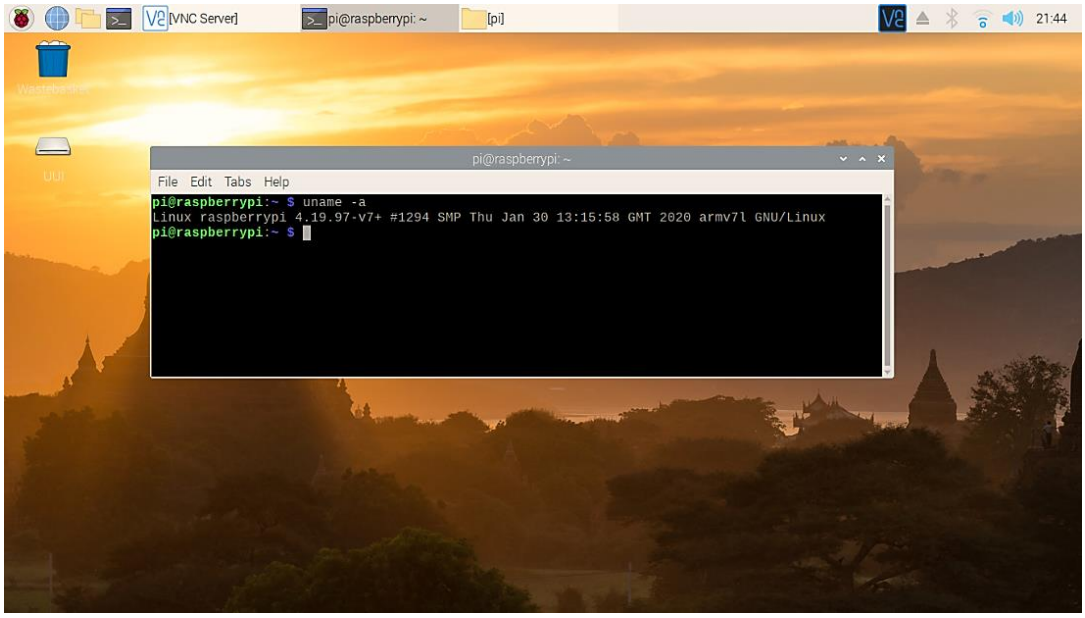
Sistema Operativo para Raspberry.

A continuación, se debe obtener una memoria Micro SD de 32 GB la cual trabajará como disco duro de arranque del sistema operativo seleccionado. Por lo tanto, el software balenaEtcher permitirá la conversión de una imagen .iso a un sistema de archivos de arranque. Estos archivos permitirán inicializar la Raspberry Pi con el sistema operativo cargado en la Micro SD.

Una vez encendido el dispositivo se podrá observar la interfaz del sistema operativo Raspbian, con ayuda de los periféricos conectados a la Raspberry tendremos acceso a las funciones del sistema y como primer punto se habilita el puerto SSH para poder acceder de forma remota desde cualquier ordenador que se encuentren en la red. Además, se puede conectar inalámbricamente seleccionando y colocando la red que se tiene disponible y de esta forma se tendrá más funcionalidades en cuanto a la utilización de la Raspberry.



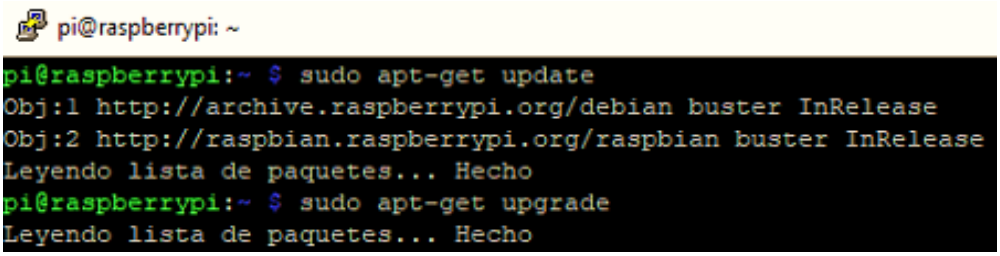
balenaEtcher convirtiendo la imagen .iso en archivos de arranque para la Raspberry Pi.



Interfaz gráfica del escritorio de la Raspberry Pi.

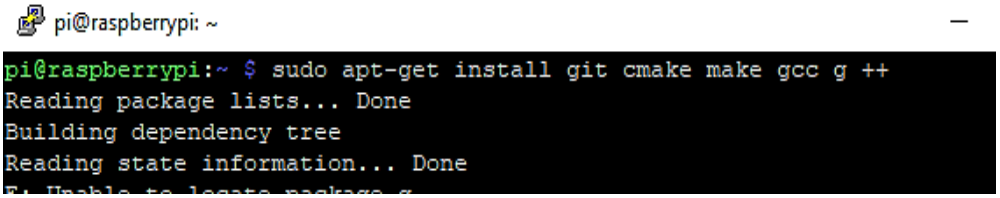
ANEXO 7 – Instalación de CMake

Para continuar instalando las herramientas necesarias para el proyecto primero se debe actualizar los repositorios de la Raspberry Pi, para lo cual se ejecutan en consola los siguientes comandos:



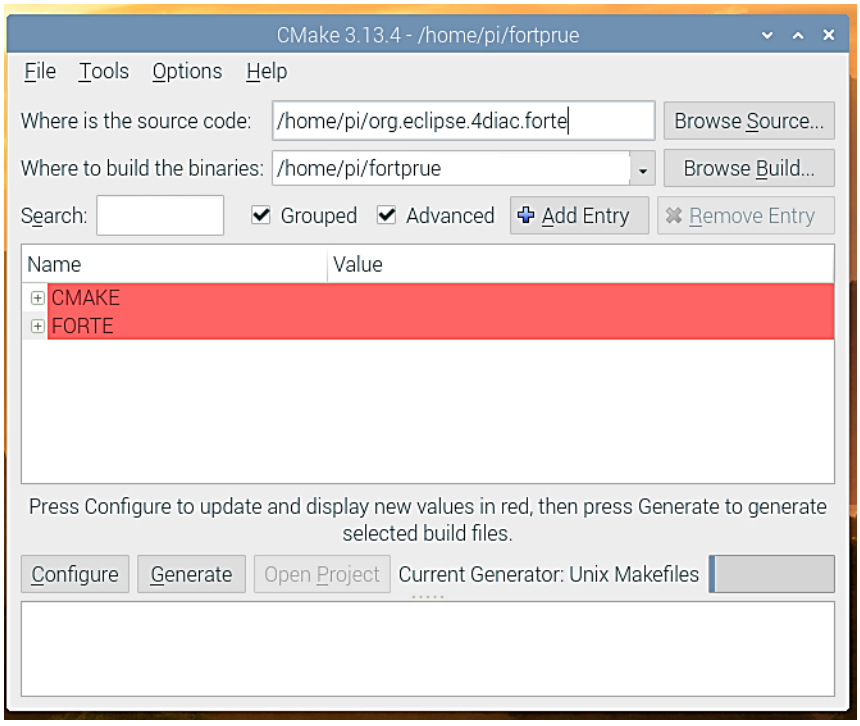
Actualización de Raspberry Pi.

Después de que se actualice el dispositivo se debe mandar a ejecutar el siguiente comando para que se instale CMake. En la siguiente figura se observa la interfaz gráfica del CMake.



Comando para instalar CMake..

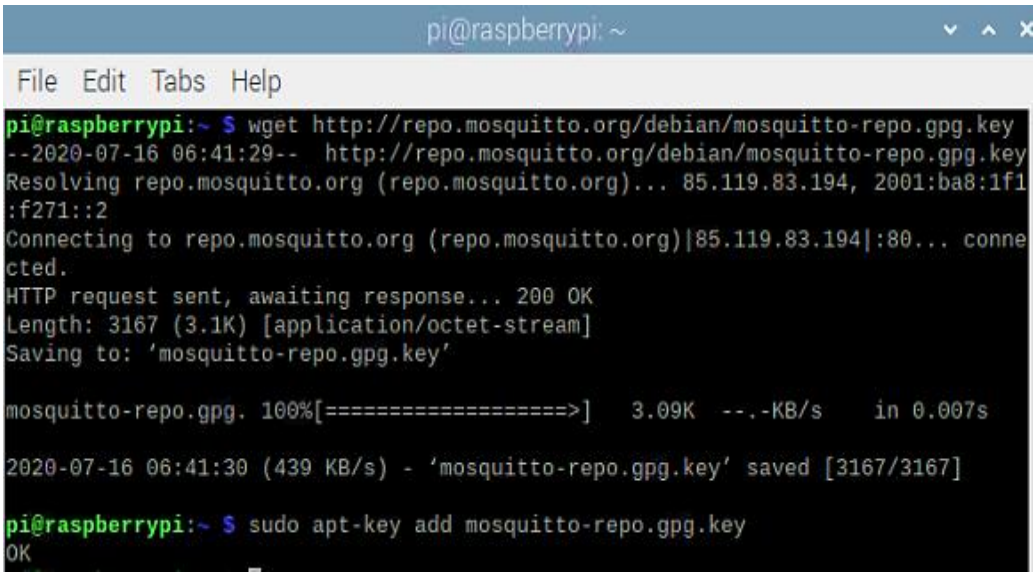
.



Interfaz gráfica de CMake.

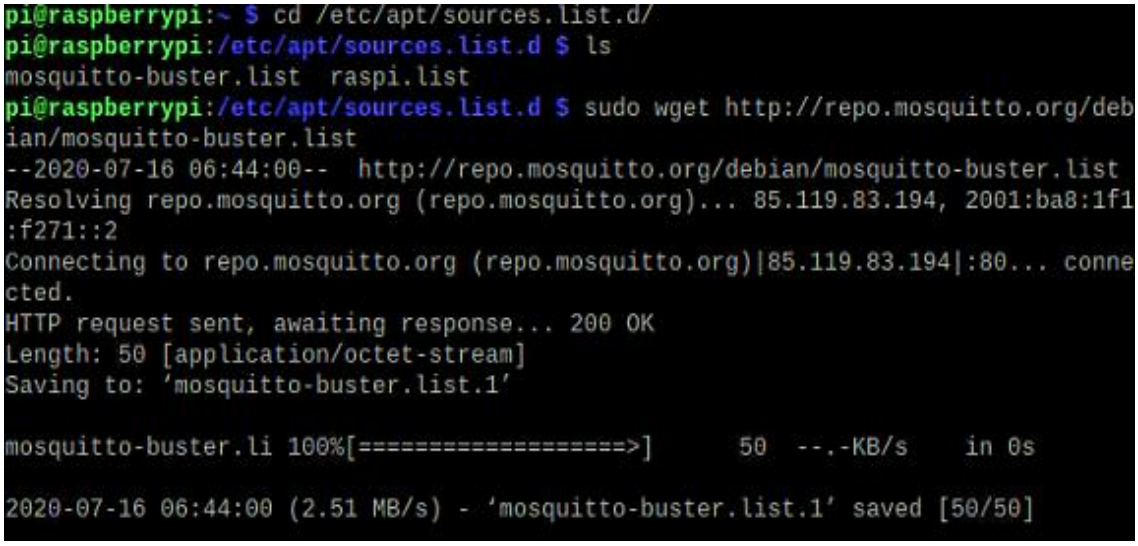
ANEXO 8 – Instalación de Mosquitto

Previamente la Raspberry Pi está actualizada entonces se procede a la instalación de los repositorios de Mosquitto. En la consola se coloca el comando que se visualiza en la imagen y de esta forma se accede a la llave del repositorio.



Comando para descargar Mosquitto.

Luego con el comando sources.list.d se puede visualizar en la Figura los repositorios, se debe escoger la versión Buster y luego mandar a ejecutar en la consola lo siguiente:



Descarga de la versión Buster de Mosquitto.

También se debe instalar los servicios de clientes de Mosquitto con el comando que se muestra a continuación:

```
pi@raspberrypi:~/mosquitto $ sudo apt install mosquitto mosquitto-clients
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libmosquitto1
The following packages will be upgraded:
  libmosquitto1 mosquitto mosquitto-clients
3 upgraded, 0 newly installed, 0 to remove and 198 not upgraded.
Need to get 332 kB of archives.
After this operation, 7,168 B of additional disk space will be used.
N: Ignoring file 'mosquitto-buster.list.1' in directory '/etc/apt/sources.list.d
/' as it has an invalid filename extension
Do you want to continue? [Y/n]
```

Descarga de los Clientes de Mosquitto.

Finalmente se reinicia la Raspberry se activa el Mosquitto y se puede observar el estado con el siguiente comando:

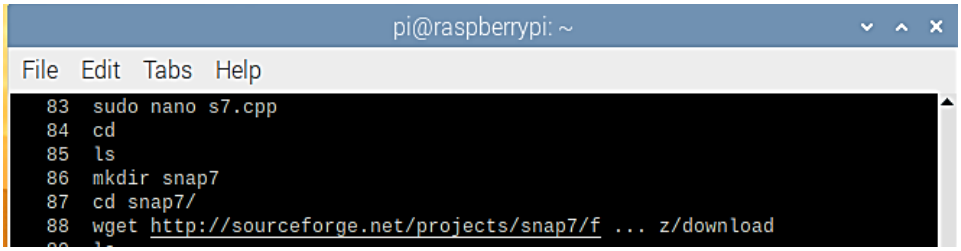
```
File  Edit  Tabs  Help
pi@raspberrypi:~ $ sudo systemctl enable mosquitto
Synchronizing state of mosquitto.service with SysV service script with /lib/syst
emd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mosquitto
pi@raspberrypi:~ $ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset
   Active: active (running) since Thu 2020-07-16 06:22:50 BST; 25min ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
 Main PID: 419 (mosquitto)
    Tasks: 1 (limit: 2200)
   Memory: 2.2M
   CGroup: /system.slice/mosquitto.service
           └─419 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Jul 16 06:22:49 raspberrypi systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Brok
Jul 16 06:22:50 raspberrypi systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broke
lines 1-13/13 (END)
```

Estado de Mosquitto en la Raspberry.

ANEXO 9 – Instalación de SNAP7

Para instalar SNAP7 se coloca el comando:

```
wget http://sourceforge.net/projects/snap7/files/1.2.1/snap7-full-1.2.1.tar.gz/download
```



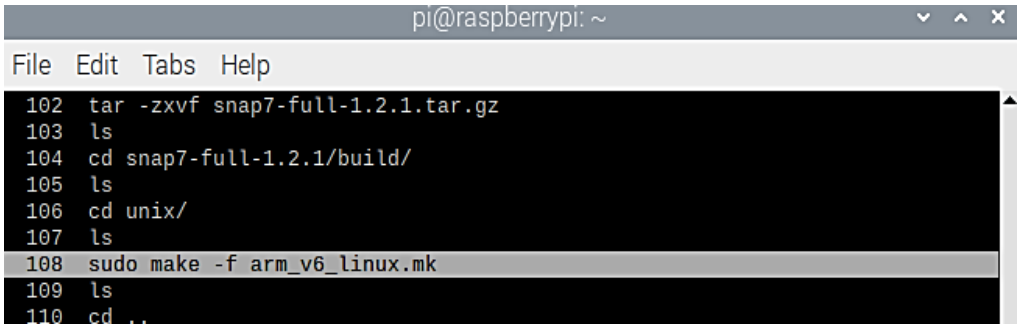
Comando para descargar los repositorios de SNAP7.

Se espera hasta que se descargue los repositorios y luego se coloca el comando tar –zxvf para descomprimir los archivos del SNAP7, por consiguiente, con el comando cd se dirige a la carpeta donde se descomprimió los ficheros. Todo este proceso se visualiza en la siguiente imagen.

Comandos para descomprimir los repositorios.

Después de estar en la carpeta de SNAP7 se manda el siguiente comando para que se compile los directorios. Luego con cp copiamos a las librerías del SNAP7.

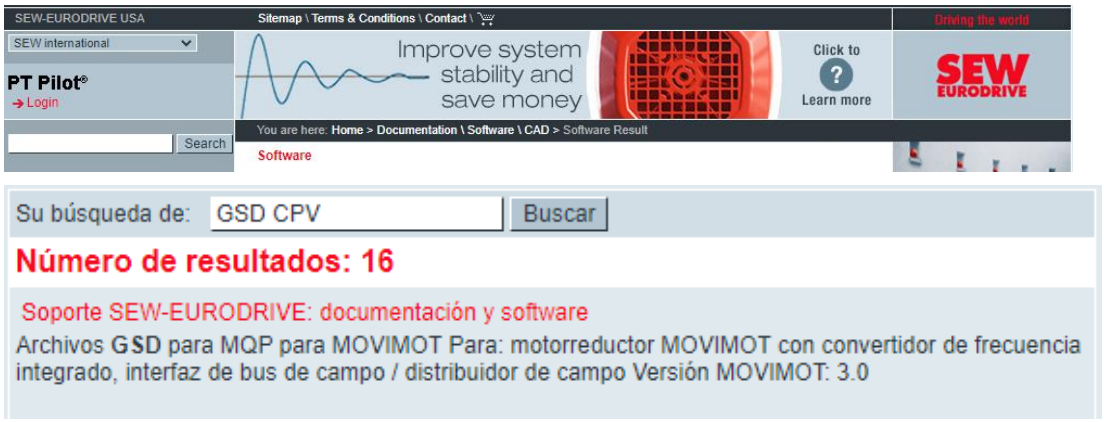
```
sudo cp ../bin/arm_v6-linux/libsnap7.so /usr/lib/libsnap7.so
sudo cp ../bin/arm_v6-linux/libsnap7.so /usr/local/lib/libsnap7.so
```



Código para compilar las bibliotecas de SNAP7.

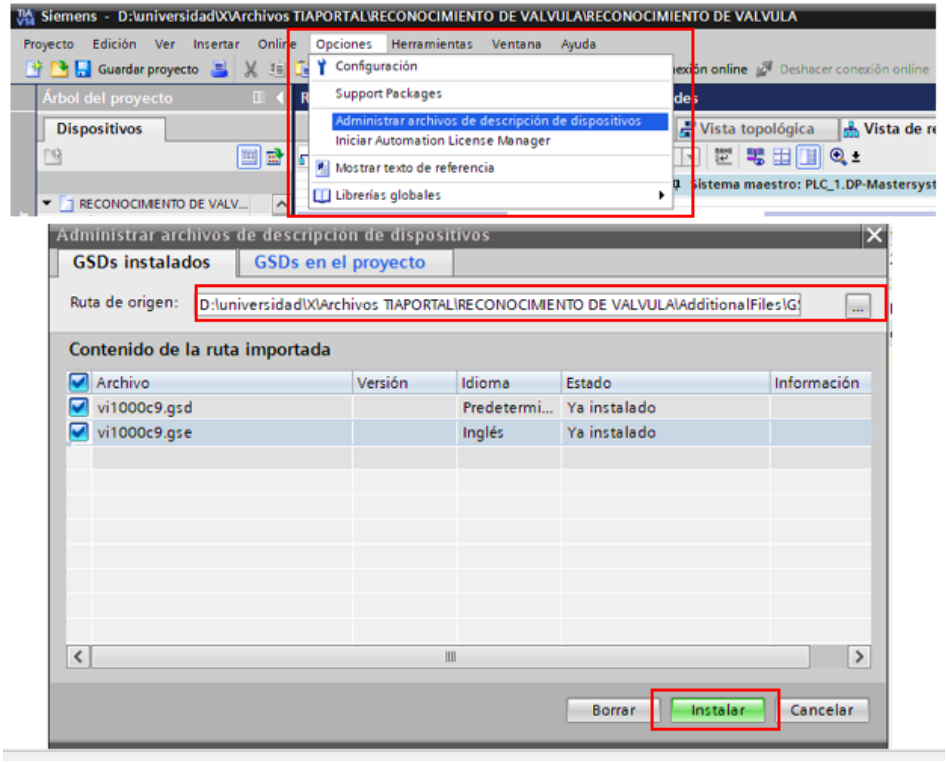
ANEXO 10 – Instalación de archivos GSD en TIA Portal

Para agregar los archivos GSD se debe primero descargar los modelos, es recomedable ingresar en la pagina que muestra la imagen y descargar el siguiente modelo.



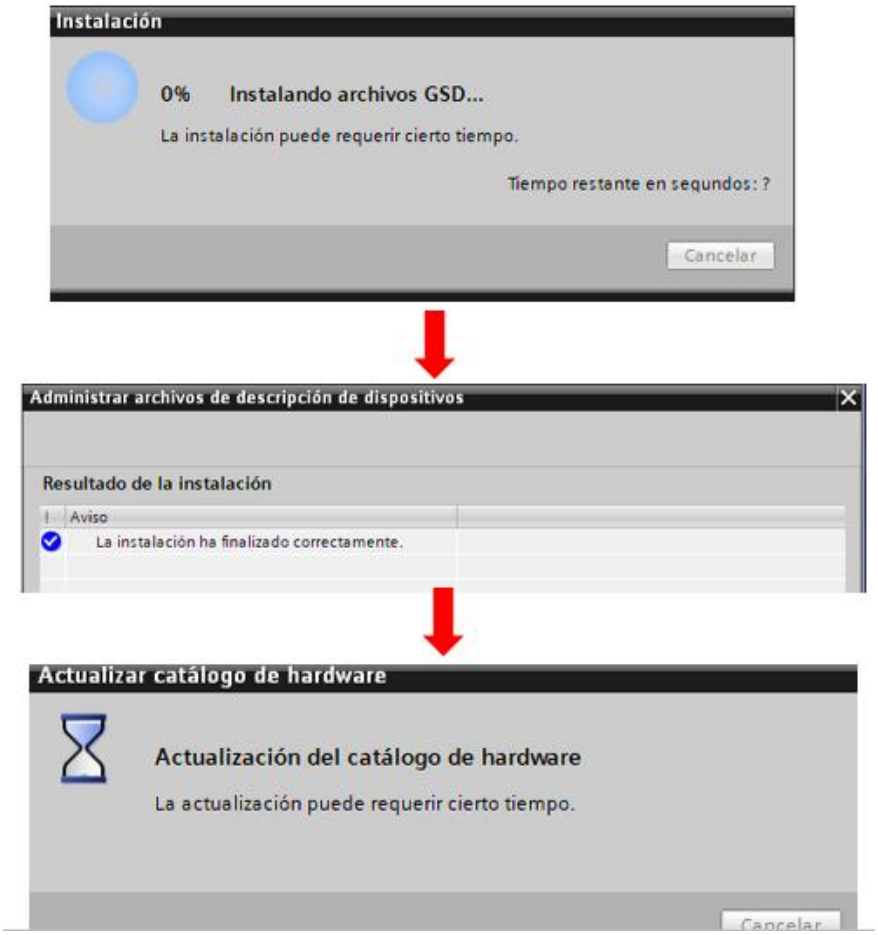
Página donde se puede descargar los módulos GSD.

Despues de la descarga descomprimimos en una carpeta y se prosigue ah abrir TIA Portal y en la pestaña “Opciones” seleccionamos “Administrar archivos de descripcion de dispositivos”. Luego en la ventana que se despliega ubicamos donde se tiene los archivos descomprimidos y colocamos “Instalar”. A continuacion se muestra el proceso.



Proceso para la instalación de los archivos GSD.

Se espera que se instalen los archivos, luego se reinicia el TIA Portal y ya estan listos los nuevos modulos para ser utilizados en este proyecto. Los ultimos pasos de la instalacion son:



Pasos finales de la instalación de los archivos GSD.

ANEXO 11 – Código para el Control de Estación en C++

```
#include "snap7.h"
#include <iostream>
#include <string>
#include <curses.h>

using namespace std;

TS7Client *Client;
char ip_plc[] = "192.168.0.1";
int rack=0;
int slot=0;
char ch; //or 'int ch;' (it
doesn't really matter)

//Buffer por sensor
//Buffer de salida
byte Buffer_S_20[256];
//Buffer de llegada
byte Buffer_S_23[256];
//Buffer pieza negra y rosada
byte Buffer_S_22_NR[256];
//Buffer pieza metalica
byte Buffer_S_21_M[256];
//Buffer de banda
byte Buffer_Q_35[256];
//Buffer de piston
byte Buffer_Q_24[256];
//Buffer de brazo negro
byte Buffer_Q_24_R[256];
//Buffer de brazo negro
byte Buffer_Q_22[256];
//Buffer de brazo rosado
byte Buffer_Q_20[256];

typedef byte       *pbyte;
bool GetBitAts(void *Buffer,
int Pos, int Bit)
{
byte Mask[] =
{0x01,0x02,0x04,0x08,0x10,0x20,
0x40,0x80};
if (Bit < 0) Bit = 0;
if (Bit > 7) Bit = 7;
return (*(pbyte(Buffer)+Pos) &
Mask[Bit]) != 0;
    }
void SetBitAt(void *Buffer, int
Pos, int Bit, bool Value)
  {
byte Mask[] =
{0x01,0x02,0x04,0x08,0x10,0x20,
0x40,0x80};
pbyte p = pbyte(Buffer)+Pos;
if (Bit < 0) Bit = 0;
if (Bit > 7) Bit = 7;
(Value) ? *p |= Mask[Bit] : *p
&= ~Mask[Bit];
    }
byte GetByteAt(void *Buffer,
int Pos)
  {
return *(pbyte(Buffer)+Pos);
    }
int SetBitVal(int Num, int
iIndex, bool b)
{
 if(b)
  return (Num | (1 << iIndex))
;
 else {
  int mask = ~(1 << iIndex);
  return Num & mask;
 }
}
//CLASES DEL CONTROL
//CONECTAR PLC
//clases de ayuda

int ConectatPLc(char
ip_plc_r[], int rack_r, int
slot_r){

  Client= new TS7Client();

  int conectar= Client-
>ConnectTo(ip_plc_r,rack_r,slot
_r);

  if(conectar == 0){
          printf("\n\n  PLC
CONECTADO\n"); //success

        }
  else{
          printf("\n  PCL NO
CONECTADO\n"); //success

          delete Client;

          return 0;
  }
  return 0;
}

bool PistonEstado(int bmem, int
bposi, bool estado_piston){
  bool
estado_piston_salida=estado_pis
ton;
  if(estado_piston==true){

    //PRIMERA OPCION Q24
    int leer_piston=Client-
>ABRead(bmem, 16,
```

```
&Buffer_Q_24[0]);

Buffer_Q_24[0]=(byte)SetBitVal(
Buffer_Q_24[0], bposi, false);
    int write_piston=Client-
>ABWrite(bmem, 16,
&Buffer_Q_24[0]);
    int leer_piston1=Client-
>ABRead(bmem, 16,
&Buffer_Q_24[0]);
    bool estado_piston_salida =
GetBitAts(Buffer_Q_24, 0,
bposi);
    printf("\nESTADO Q2.4
PISTON(1 == ON - 0 == OFF )=:
%d\n", estado_piston_salida);

    printf("\nPISTON
ENCEDIDO\n");
  }else{
    int leer_piston=Client-
>ABRead(bmem, 16,
&Buffer_Q_24[0]);

Buffer_Q_24[0]=(byte)SetBitVal(
Buffer_Q_24[0], bposi, true);
    int write_piston=Client-
>ABWrite(bmem, 16,
&Buffer_Q_24[0]);
    int leer_piston1=Client-
>ABRead(bmem, 16,
&Buffer_Q_24[0]);
    bool estado_piston_salida =
GetBitAts(Buffer_Q_24, 0,
bposi);
    printf("\nESTADO Q2.4
PISTON(1 == ON - 0 == OFF )=:
%d\n", estado_piston_salida);

    printf("\nPISTON
APAGADO\n");
  }

  return estado_piston_salida;
}

bool BandaEstado(int bmem, int
bposi, bool estado_banda){

  bool
estado_banda_salida=estado_band
a;
  if(estado_banda==true){
    //PRIMERA OPCION Q24
    int leer_banda=Client-
>ABRead(bmem, 16,
&Buffer_Q_35[0]);

Buffer_Q_35[0]=(byte)SetBitVal(
Buffer_Q_35[0], bposi, true);
    int write_banda=Client-
>ABWrite(bmem, 16,
&Buffer_Q_35[0]);
    int leer_banda1=Client-
>ABRead(bmem, 16,
&Buffer_Q_35[0]);
    bool estado_banda_salida =
GetBitAts(Buffer_Q_35, 0,
bposi);
    printf("\nESTADO Q3.5
BANDA(1 == ON - 0 == OFF )=:
%d\n", estado_banda_salida);

    printf("\nBANDA
ENCEDIDA\n");
  }else{
    int leer_banda=Client-
>ABRead(bmem, 16,
&Buffer_Q_35[0]);

Buffer_Q_35[0]=(byte)SetBitVal(
Buffer_Q_35[0], bposi, false);
    int write_banda=Client-
>ABWrite(bmem, 16,
&Buffer_Q_35[0]);
    int leer_banda1=Client-
>ABRead(bmem, 16,
&Buffer_Q_35[0]);
    bool estado_banda_salida =
GetBitAts(Buffer_Q_35, 0,
bposi);
    printf("\nESTADO Q3.5
BANDA(1 == ON - 0 == OFF )=:
%d\n", estado_banda_salida);
    printf("\nBANDA
APAGADA\n");
  }


  return estado_banda_salida;
}
bool BrazoNegroEstado(int bmem,
int bposi, bool estado_brazo){

  //printf("\nEjecuto::
0=EXITOSO %d\n", leer);
  bool
estado_brazo_salida_n=estado_br
azo;
  if(estado_brazo==true){

    //PRIMERA OPCION Q22
    int leer_brazo=Client-
>ABRead(bmem, 16,
&Buffer_Q_22[0]);

Buffer_Q_22[0]=(byte)SetBitVal(
Buffer_Q_22[0], bposi, true);
    int write_banda=Client-
>ABWrite(bmem, 16,
```

```
&Buffer_Q_22[0]);
    int leer_banda1=Client-
>ABRead(bmem, 16,
&Buffer_Q_22[0]);
    bool estado_brazo_salida_n
= GetBitAts(Buffer_Q_22, 0,
bposi);
    printf("\nESTADO Q2.2 BRAZO
NEGRO(1 == ON - 0 == OFF )=:
%d\n", estado_brazo_salida_n);

   printf("\nBRAZO NEGRO
ENCEDID0\n");
  }else{
    int leer_brazo=Client-
>ABRead(bmem, 16,
&Buffer_Q_22[0]);

Buffer_Q_22[0]=(byte)SetBitVal(
Buffer_Q_22[0], bposi, false);
    int write_banda=Client-
>ABWrite(bmem, 16,
&Buffer_Q_22[0]);
    int leer_banda1=Client-
>ABRead(bmem, 16,
&Buffer_Q_22[0]);
    bool estado_brazo_salida_n
= GetBitAts(Buffer_Q_22, 0,
bposi);
    printf("\nESTADO Q2.2 BRAZO
NEGRO(1 == ON - 0 == OFF )=:
%d\n", estado_brazo_salida_n);

    printf("\nBRAZO NEGRO
APAGADO\n");
  }



  return estado_brazo_salida_n;
}

bool BrazoRosadoEstado(int
bmem, int bposi, bool
estado_brazo){

    bool
estado_brazo_salida_r=estado_br
azo;
  if(estado_brazo==true){

    //PRIMERA OPCION Q20
    int leer_brazo=Client-
>ABRead(bmem, 16,
&Buffer_Q_20[0]);

Buffer_Q_20[0]=(byte)SetBitVal(
Buffer_Q_20[0], bposi, true);
    int write_banda=Client-
>ABWrite(bmem, 16,
&Buffer_Q_20[0]);
    int leer_banda1=Client-
>ABRead(bmem, 16,
&Buffer_Q_20[0]);
    bool estado_brazo_salida_r
= GetBitAts(Buffer_Q_20, 0,
bposi);
    printf("\nESTADO Q2.2 BRAZO
ROSADO(1 == ON - 0 == OFF )=:
%d\n", estado_brazo_salida_r);


    printf("\nBRAZO ROSADO
ENCEDIDO\n");
  }else{
    int leer_brazo=Client-
>ABRead(bmem, 16,
&Buffer_Q_20[0]);

Buffer_Q_20[0]=(byte)SetBitVal(
Buffer_Q_20[0], bposi, false);
    int write_banda=Client-
>ABWrite(bmem, 16,
&Buffer_Q_20[0]);
    int leer_banda1=Client-
>ABRead(bmem, 16,
&Buffer_Q_20[0]);
    bool estado_brazo_salida_r
= GetBitAts(Buffer_Q_20, 0,
bposi);
    printf("\nESTADO Q2.2 BRAZO
ROSADO(1 == ON - 0 == OFF )=:
%d\n", estado_brazo_salida_r);

    printf("\nBRAZO ROSADO
APAGADO\n");
  }

  return estado_brazo_salida_r;
}

byte LeerSensorI20(){
  //COMPROBAR SENSOR 1 I2.0
  int leer_sensor=Client-
>EBRead(2, 16,
&Buffer_S_20);//LEEE TODAS LAS
I2 PARA G$

  byte byte_sensor20 =
GetByteAt(Buffer_S_20, 0);//
  printf("\nNumero de sensor20
Byte=:  %d\n", byte_sensor20);


return byte_sensor20;
}
byte LeerSensorI22(){
  //COMPROBAR SENSOR 1 I2.2
  int leer_sensor22=Client-
>EBRead(2, 16,
```

```
&Buffer_S_22_NR);//LEEE TODAS
LAS I2 PARA G$

  byte byte_sensor22 =
GetByteAt(Buffer_S_22_NR, 0);//
  printf("\nNumero de COLOR
Byte=:  %d\n", byte_sensor22);


return byte_sensor22;
}
byte LeerSensorI21(){
  //COMPROBAR SENSOR 1 I2.2
  int leer_sensor21=Client-
>EBRead(2, 16,
&Buffer_S_21_M);//LEEE TODAS
LAS I2 PARA G$
  byte byte_sensor21 =
GetByteAt(Buffer_S_21_M, 0);//
  printf("\nNumero de Byte=:
%d\n", byte_sensor21);


return byte_sensor21;
}
byte LeerSensorI23(){
  //COMPROBAR SENSOR 1 I2.2
  int leer_sensor23=Client-
>EBRead(2, 16,
&Buffer_S_23);//LEEE TODAS LAS
I2 PARA G$

  byte byte_sensor23 =
GetByteAt(Buffer_S_23, 0);//
  printf("\nNumero de llegada
Byte=:  %d\n", byte_sensor23);


return byte_sensor23;
}
int ApagarNegro()
{
        //Q2.2 BRAZO NEGRO
          Client->ABRead(2, 16,
&Buffer_Q_24_R);

      SetBitAt(&Buffer_Q_24_R,
2, 2, false);//Q2.2 BRAZO NEGRO
            Client->ABWrite(2,
16, &Buffer_Q_24_R);
             return 0;
}

int main(){


  int opcion;
  int banda_memAddr,
banda_posicion;
  int bandera;
  do {
    /* code */
    printf( "\n   1. Conectar
PLC", 163 );
    printf( "\n   2. Control
PLC", 163 );
    printf( "\n   3. Salir." );
    printf( "\n\n   Introduzca
opc (1-3): ", 162 );
    scanf( "%d", &opcion );
    /* Inicio del anidamiento
*/
    switch (opcion) {
      case 1: printf( "\n   1.
Conectar PLC", 163 );{

ConectatPLc(ip_plc,rack,slot);

          bool
ver_estado_piston=PistonEstado(
2,4,false);//descitivo el
piston


          bool
ver_estado_banda=BandaEstado(3,
5,false);
          bool
ver_estado_brazo_negro=BrazoNeg
roEstado(2,2,false);
          bool
ver_estado_brazo_rosado=BrazoRo
sadoEstado(2,0,false);

int negro=ApagarNegro();


      }
      break;
      case 2: printf( "\n   2.
Control PLC", 163 );{
        printf( "\n   ENCIENDE
BANDA Q3.5\n ");


        printf( "\n     QX.=
");
        scanf( "%d",
&banda_memAddr );
        int
bmemoria=banda_memAddr;

        printf( "\n     QX.X=
");
        scanf( "%d",
&banda_posicion );
        int
bposicion=banda_posicion;

        bool
```

```
ver_estado_banda=BandaEstado(bm
emoria,bposicion,true);

        if
(ver_estado_banda==1) {
          printf("\nESTADO Q3.5
%d\n", ver_estado_banda);

          do {

            byte sensor20 =
LeerSensorI20();//


            if
((sensor20==81)||(sensor20==82)
||(sensor20==84)||(sensor20==85
)) {


              printf("\nCON
PIEZA\n");
              bool
ver_estado_piston=PistonEstado(
2,4,true);//activo el piston
              printf("\nESTADO
Q2.4 %d\n", ver_estado_piston);


              if
(ver_estado_piston==1) {

printf("\nANALIZANDO
PIEZA\n");
                sleep(2);
                          byte
sensor22=LeerSensorI22();

printf("\nESTADO
sssssssssssssssssss %d\n",
sensor22);
                        byte
sensor21=LeerSensorI21();

sleep(1);


                          if
(sensor22==85) {

printf("\n*******PIEZA
ROSADA*******\n");
                    bool
ver_estado_brazo_rosado=BrazoRo
sadoEstado(2,0,true);

                    if
(ver_estado_brazo_rosado==1) {
                     bool
ver_estado_piston=PistonEstado(
2,4,false);//desactivo

                    }else{


                                 }
                  }else{
                    printf("\n-
------NO SENSANDO PIEZA ROSADA-
------\n");

                  }
                  if
(sensor22==84) {

printf("\n*******PIEZA
NEGRA*******\n");
                    bool
ver_estado_brazo_negro=BrazoNeg
roEstado(2,2,true);

                    if
(ver_estado_brazo_negro==1) {
                     bool
ver_estado_piston=PistonEstado(
2,4,false);//desactivo
                    }else{


                                 }
                  }else{
                    printf("\n-
------NO SENSANDO PIEZA NEGRA--
-----\n");
                  }

                  if
((sensor21==86)||(sensor20==82)
) {

printf("\n*******PIEZA
METALICA*******\n");
                    sleep(1);
                    bool
ver_estado_piston=PistonEstado(
2,4,false);//desactivo

                  }else{
                    printf("\n-
------NO SENSANDO PIEZA
METALICA-------\n");

                  }

sleep(4);
                          byte
sensor23=LeerSensorI23();

      printf("\nESTADO ale222
%d\n", sensor23);
```

```
        if
((sensor23==96)||(sensor23==88)
) {

              sleep(2);

              printf(
"\n\n************ LLEGO
PIEZA************\n\n" );//



                  bool
ver_estado_brazo_negro=BrazoNeg
roEstado(2,2,false);

                  bool
ver_estado_brazo_rosado=BrazoRo
sadoEstado(2,0,false);

                  int
negro=ApagarNegro();


              }else{

                  printf(
"\n\nNO LLEGO PIEZA\n\n" );//

              }

              }else{

                printf("\nNO
ANALIZANDO -- PIEZA\n");


              }

            }else{

                printf("\nSIN
PIEZA\n");


      // bool
ver_estado_piston=PistonEstado(
2,4,false);//desactivo


                }

                  if
((sensor20==96)) {

                  bool
ver_estado_brazo_negro=BrazoNeg
roEstado(2,2,false);

                  bool
ver_estado_brazo_rosado=BrazoRo
sadoEstado(2,0,false);

                  int
negro=ApagarNegro();


              }else{


              }

//printf("\nSensor_2.0 en
Byte=:  %d\n", sensor20);


          }
while(ver_estado_banda==1);

        }else{
          printf("\nESTADO ELSE
Q3.5 %d\n", ver_estado_banda);
        }


      }
      break;

    }//CIERRA EL switch


  } while(opcion != 3);

delete Client;
return 0;

}
```

ANEXO 12 – Código de las librerías de la Capa de Comunicación en C++

CABECERA

```
#include <mosquittopp.h>
#include <cstdio>
#include <string>
#include <iostream>
#include "mosquitto.h"
#include "mosquittopp.h"
#include <cstring>


#define MAX_PAYLOAD 50
#define DEFAULT_KEEP_ALIVE 60


class mqtt_test:public mosqpp::mosquittopp
{
public:
    mqtt_test(const char *id):mosquittopp(id){}
    void on_connect(int rc) {std::cout<<"on_connect"<<std::endl;}
    void on_disconnect() {std::cout<<"on_disconnect"<<std::endl;}
    void on_publish(int mid) {std::cout<<"on_publish"<<std::endl;}
    void on_subscribe(int mid, int qos_count, const int
*granted_qos);
    void on_message(const struct mosquitto_message *message);
    int conectar_sub(std::string ip,std::string
clientid_g,std::string topic_g);
    int conectar_pub(std::string ip,std::string
clientid_g,std::string topic_g);
    int desconectar(std::string clientid_g);
};
```

CUERPO

```
#include "al_mqtt.h"
#include <string>
#include <iostream>
#include "mosquitto.h"
#include "mosquittopp.h"
#include <cstring>
#include "snap7.h"
#include "CONTROL.h"
#include "PUBLISH_CONTROL.h"
#include "SUSCRIBE_CONTROL.h"
#include "SENSORS_FB.h"
#include "ACTORS_FB.h"
#include "PUBLISH_COLOR.h"
#include "PUBLISH_METAL.h"
#include "PUBLISH_PRESENCE.h"
#include "PUBLISH_SE_OUTPUT.h"
#include "SUSCRIBE_BAND.h"
#include "SUSCRIBE_PISTON.h"
#include "SUSCRIBE_BLACK.h"
#include "SUSCRIBE_PINK.h"
```

```
using namespace std;

void mqtt_test::on_subscribe(int mid, int qos_count, const int
*granted_qos)
{
    std::cout<<"Suscripción mid: %d "<<mid<<std::endl;
}
int salida_estado=0;

int  mqtt_test::dato_sensor20=0001;
int  mqtt_test::dato_sensor21=0001;
int  mqtt_test::dato_sensor22=0001;
int  mqtt_test::dato_sensor23=0001;


bool mqtt_test::dato_estado_banda=false;
bool mqtt_test::dato_estado_piston=false;
bool mqtt_test::dato_brazo_rosado=false;
bool mqtt_test::dato_brazo_negro=false;
//static  std::string strRcv;
void mqtt_test::on_message(const struct mosquitto_message *message)
{
    bool se20,se21,se22,se23=false;
    bool ac35,ac24,ac20,ac22=false;


    std::string strRcv=(char *)message->payload;
//    int value = std::stoi(record[strRcv]);

//int i80 = std::stoi(strRcv);
//dato_sensor20=i3;
    mosqpp::topic_matches_sub("sensor20",message->topic,&se20);
    mosqpp::topic_matches_sub("sensor21",message->topic,&se21);
    mosqpp::topic_matches_sub("sensor22",message->topic,&se22);
    mosqpp::topic_matches_sub("sensor23",message->topic,&se23);
    mosqpp::topic_matches_sub("actuador35",message->topic,&ac35);
    mosqpp::topic_matches_sub("actuador24",message->topic,&ac24);
    mosqpp::topic_matches_sub("actuador20",message->topic,&ac20);
    mosqpp::topic_matches_sub("actuador22",message->topic,&ac22);
    std::cout<<"Topico: <"<<message->topic<<"> MENSAJE:
"<<strRcv<<std::endl;


      if (se20) {
        /* code */
        std::cout<<"sensor20"<<std::endl;
        int i80 = std::stoi(strRcv);
        dato_sensor20=i80;

       if (strRcv=="leer") {
        std::cout<<"sensor20"<<std::endl;
        /* code */


        }
      }
      if (se21) {
        /* code */
         std::cout<<"sensor21"<<std::endl;
```

```
   int i80 = std::stoi(strRcv);
  dato_sensor21=i80;

  if (strRcv=="leer") {
  /* code */

  }

}
if (se22) {
  /* code */
   std::cout<<"sensor22"<<std::endl;

      int i80 = std::stoi(strRcv);
  dato_sensor22=i80;
  if (strRcv=="leer") {
  /* code */

  }

}
if (se23) {
  /* code */
   std::cout<<"sensor23"<<std::endl;
    int i80 = std::stoi(strRcv);
    dato_sensor23=i80;
  if (strRcv=="leer") {
  /* code */

  }
}
if (ac35) {

  if (strRcv=="encender") {
    /* code */

    dato_estado_banda=true;
  }

  if (strRcv=="apagar") {
    /* code */

    dato_estado_banda=false;
  }
}
if (ac24) {

  if (strRcv=="encender") {
    /* code */

     dato_estado_piston=true;
    }

  if (strRcv=="apagar") {
    /* code */

        dato_estado_piston=false;
  }
}
if (ac20) {
```

```
        /* code */
        if (strRcv=="encender") {
          /* code */
         dato_brazo_rosado=true;
        }
        if (strRcv=="apagar") {
          /* code */
         dato_brazo_rosado=false;
        }
      }
      if (ac22) {
        /* code */
        if (strRcv=="encender") {
          /* code */
          dato_brazo_negro=true;
        }
        if (strRcv=="apagar") {
          /* code */
          dato_brazo_negro=false;
        }
      }

//return strRcv;
}
//char buf2[255];
int mqtt_test::conectar_pub(std::string ip,std::string
clientid_g,std::string topic_g,std::string mensaje_g)
{

mqtt_test test2(clientid_g.c_str());
int rc2;
rc2 = test2.connect(ip.c_str());
rc2 = test2.publish(NULL, topic_g.c_str(), 100, mensaje_g.c_str());
rc2 = test2.loop_start();
test2.disconnect();
mosqpp::lib_cleanup();
return 0;

}
int mqtt_test::desconectar(std::string clientid_g){

mosqpp::lib_init();
mqtt_test test(clientid_g.c_str());
test.disconnect();
mosqpp::lib_cleanup();
return 0;
}


int mqtt_test::conectar_sub_s(std::string ip,std::string
clientid_g,std::string topic_g)
{
    int aux=salida_estado;
    mosqpp::lib_init();
    mqtt_test test(clientid_g.c_str());
    int rc;
    test.connect_async(ip.c_str(),1883,60);
    test.loop_start();
    //test.subscribe(NULL,topic_g.c_str(),1);
    test.subscribe(NULL,"sensor20",0);
```

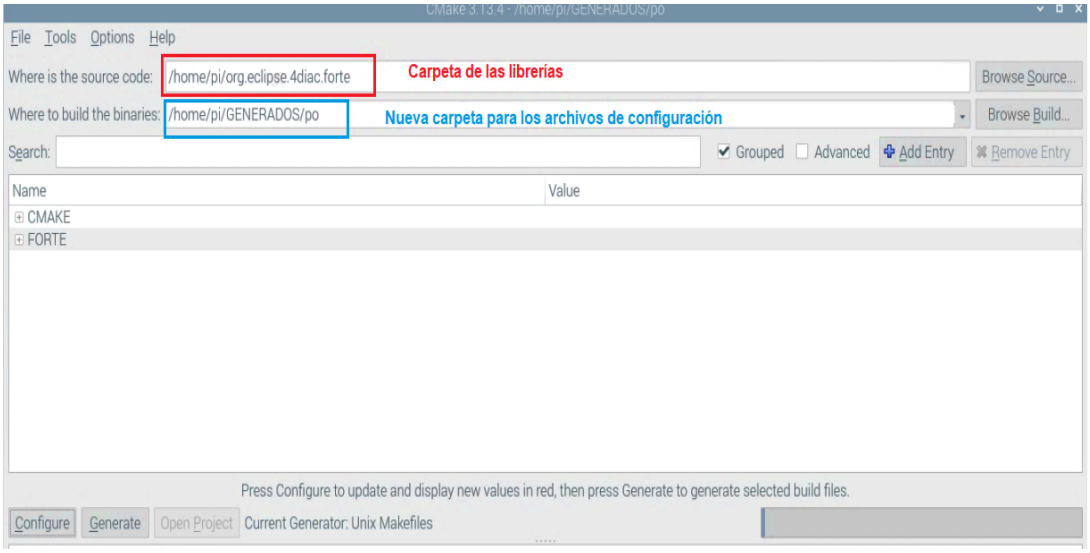
```
    test.subscribe(NULL,"sensor21",0);

    test.subscribe(NULL,"sensor22",0);

    test.subscribe(NULL,"sensor23",0);

    test.subscribe(NULL,"actuador35",0);

    test.subscribe(NULL,"actuador24",0);

    test.subscribe(NULL,"actuador20",0);

    test.subscribe(NULL,"actuador22",0);


    test.loop_start();
    sleep(5);
    std::cout << "Desconectado SUB" << std::endl;
    test.disconnect();
    mosqpp::lib_cleanup();

    return aux;
}
int mqtt_test::conectar_sub_a(std::string ip,std::string
clientid_g,std::string topic_g)
{
    int aux=salida_estado;
    mosqpp::lib_init();
    mqtt_test test(clientid_g.c_str());
    int rc;
    test.connect(ip.c_str());
    test.loop_start();
    //test.subscribe(NULL,topic_g.c_str(),1);
    test.subscribe(NULL,"sensor20",0);

    test.subscribe(NULL,"sensor21",0);

    test.subscribe(NULL,"sensor22",0);

    test.subscribe(NULL,"sensor23",0);

      test.subscribe(NULL,"actuador35",0);

    test.subscribe(NULL,"actuador24",0);

    test.subscribe(NULL,"actuador20",0);

    test.subscribe(NULL,"actuador22",0);


    test.loop_start();

    std::cout << "Desconectado SUB" << std::endl;
    test.disconnect();
    mosqpp::lib_cleanup();


    return aux;
}
```

ANEXO 13 – Creación del archivo CMakeLists.txt

```
#####################################################################
# SNAP AND MOSQUITTO FBs
#####################################################################
forte_add_include_directories(${CMAKE_CURRENT_SOURCE_DIR})
forte_add_module(MOSQUITTO_MQTT_SNAP "Interacting with control and
communication")

forte_add_sourcefile_h(snap7.h)
forte_add_sourcefile_hcpp(snap7)

forte_add_link_library(libsnap7.so)
forte_add_link_library(libmosquittopp.so.1)

forte_add_sourcefile_h(al_mqtt.h)
forte_add_sourcefile_hcpp(al_mqtt)

forte_add_sourcefile_hcpp(ACTORS_FB)
forte_add_sourcefile_h(ACTORS_FB.h)

forte_add_sourcefile_hcpp(SENSORS_FB)
forte_add_sourcefile_h(SENSORS_FB.h)

forte_add_sourcefile_hcpp(CONTROL)
forte_add_sourcefile_h(CONTROL.h)

forte_add_sourcefile_hcpp(PUBLISH_PRESENCE)
forte_add_sourcefile_h(PUBLISH_PRESENCE.h)

forte_add_sourcefile_hcpp(PUBLISH_COLOR)
forte_add_sourcefile_h(PUBLISH_COLOR.h)

forte_add_sourcefile_hcpp(PUBLISH_METAL)
forte_add_sourcefile_h(PUBLISH_METAL.h)

forte_add_sourcefile_hcpp(PUBLISH_SE_OUTPUT)
forte_add_sourcefile_h(PUBLISH_SE_OUTPUT.h)

forte_add_sourcefile_hcpp(SUSCRIBE_BAND)
forte_add_sourcefile_h(SUSCRIBE_BAND.h)

forte_add_sourcefile_hcpp(SUSCRIBE_BLACK)
forte_add_sourcefile_h(SUSCRIBE_BLACK.h)

forte_add_sourcefile_hcpp(SUSCRIBE_PINK)
forte_add_sourcefile_h(SUSCRIBE_PINK.h)

forte_add_sourcefile_hcpp(SUSCRIBE_PISTON)
forte_add_sourcefile_h(SUSCRIBE_PISTON.h)

forte_add_sourcefile_hcpp(PUBLISH_CONTROL)
forte_add_sourcefile_h(PUBLISH_CONTROL.h)

forte_add_sourcefile_hcpp(SUSCRIBE_CONTROL)
forte_add_sourcefile_h(SUSCRIBE_CONTROL.h)
```

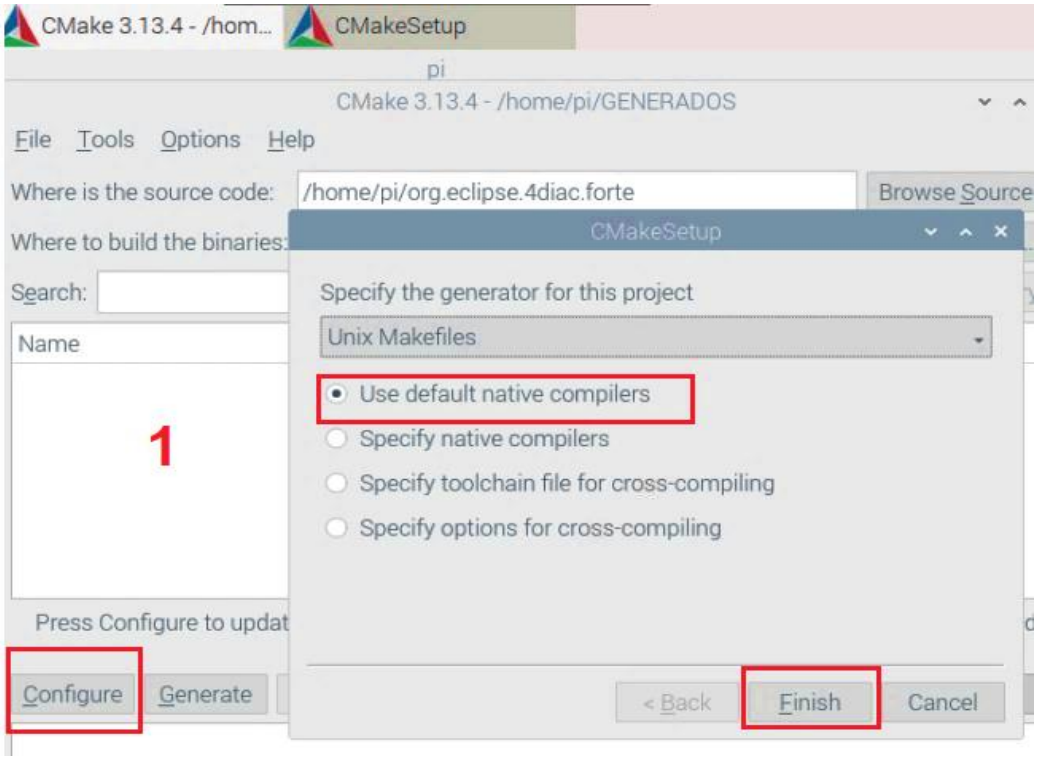
ANEXO 14 – Construcción del archivo < ./forte > con las librerías de Mosquitto-MQTT-SNAP7

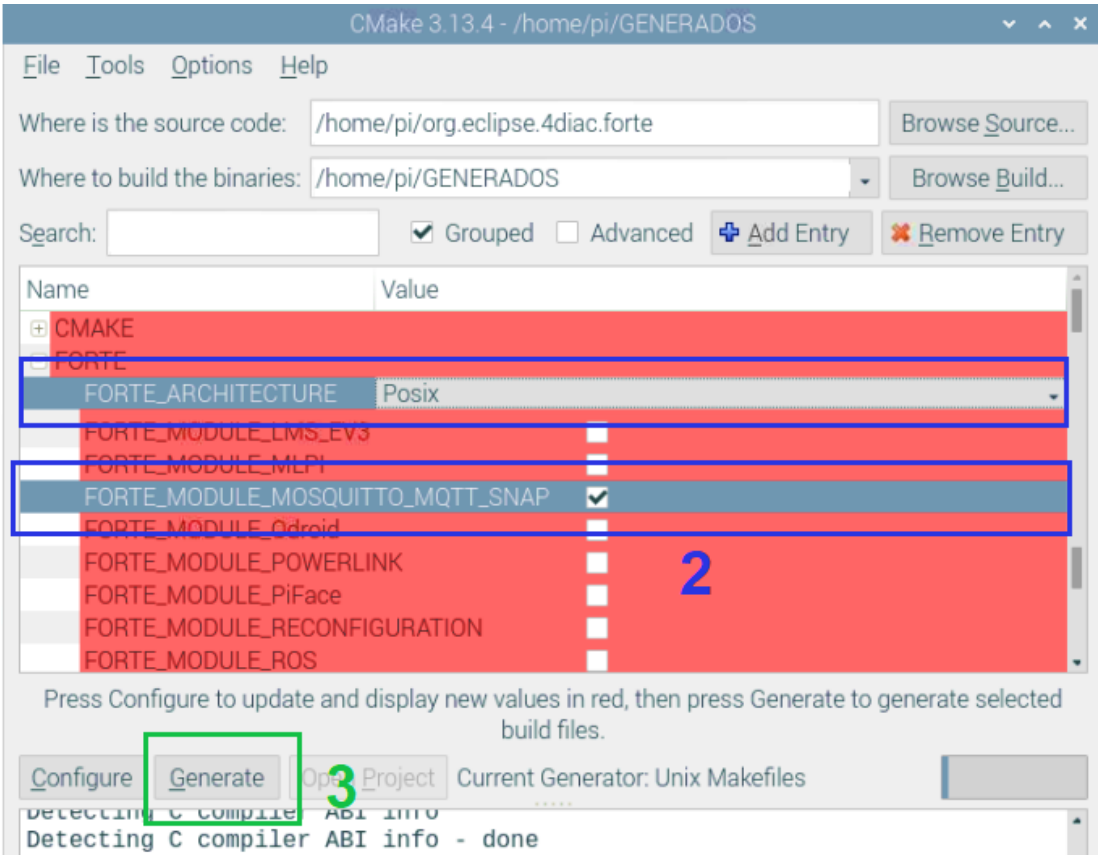
Primero se deben colocar correctamente las direcciones de las carpetas en el software CMake. La carpeta GENERADOS fue creada para almacenar los archivos de configuración.



Direcciones de las carpetas para la construcción de archivos de configuración.

Seguidamente se realiza los siguientes pasos: 1- Configure, 2- Selección del Módulo: Mosquitto_MQTT_SNAP, Arquitectura: Posix, 3- Generate.





Pasos para construir los archivos de configuración para la aplicación.