



ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

“USO DEL PROTOCOLO MQTT BASADO EN LA NORMA IEC 61499 PARA LA INTEGRACIÓN DE UN ROBOT KUKA YUBOT HACIA LA NUBE”

TATIANA PAOLA ZAMBRANO VALVERDE

Trabajo de Titulación modalidad: Proyectos de Investigación y Desarrollo, presentado ante el Instituto de Posgrado y Educación Continua de la ESPOCH, como requisito parcial para la obtención del grado de:

**MAGÍSTER EN SISTEMAS DE CONTROL Y AUTOMATIZACIÓN
INDUSTRIAL**

Riobamba-Ecuador

Diciembre 2019

©2019, Tatiana Paola Zambrano Valverde

Se autoriza la reproducción total o parcial, con fines académicos, por cualquier medio o procedimiento, incluyendo la cita bibliográfica del documento, siempre y cuando se reconozca el Derecho de Autor.

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

CERTIFICACIÓN:

EL TRIBUNAL DEL TRABAJO DE TITULACIÓN CERTIFICA QUE:

El Trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo, denominado: “USO DEL PROTOCOLO MQTT BASADO EN LA NORMA IEC 61499 PARA LA INTEGRACIÓN DE UN ROBOT KUKA YUBOT HACIA LA NUBE”, de responsabilidad de la señorita Tatiana Paola Zambrano Valverde, ha sido minuciosamente revisado y se autoriza su presentación.

Ing. Oswaldo Geovanny Martínez Guashima, Msc.

PRESIDENTE

Ing. Mónica Andrea Zabala Haro, Msc.

DIRECTORA

Ing. Santiago Mauricio Altamirano Meléndez, Mg.

MIEMBRO DEL TRIBUNAL

Ing. Franklin Wilfrido Salazar Logroño, Msc.

MIEMBRO DEL TRIBUNAL



The image shows three handwritten signatures in blue ink, each on a horizontal line. The top signature is 'Oswaldo Geovanny Martínez Guashima', the middle one is 'Mónica Andrea Zabala Haro', and the bottom one is 'Santiago Mauricio Altamirano Meléndez'. The signature of Franklin Wilfrido Salazar Logroño is not visible in this image.

Riobamba, diciembre 2019

DERECHOS INTELECTUALES

Yo, Tatiana Paola Zambrano Valverde soy responsable de las ideas, doctrinas y resultados expuestos en este Trabajo de Titulación y el patrimonio intelectual del mismo pertenece a la Escuela Superior Politécnica de Chimborazo.

TATIANA PAOLA ZAMBRANO VALVERDE

No. Cédula: 060332441-9

DECLARACIÓN DE AUTENTICIDAD

Yo, Tatiana Paola Zambrano Valverde, declaro que el presente proyecto de investigación, es de mi autoría y que los resultados del mismo son auténticos y originales. Los textos constantes en el documento que provienen de otras fuentes están debidamente citados y referenciados.

Como autor, asumo la responsabilidad legal y académica de los contenidos de este Trabajo de Titulación de Maestría.

TATIANA PAOLA ZAMBRANO VALVERDE

No. Cédula: 060332441-9

CONTENIDO

	PAG.
RESUMEN	xii
ABSTRACT.....	xiii
CAPÍTULO I.....	1
1. INTRODUCCIÓN	1
1.1 Planteamiento del problema	2
1.1.1 Formulación del problema.....	2
1.1.2 Preguntas directrices o específicas de la investigación.....	3
1.1.3 Justificación de la investigación	3
1.2 Objetivos de la investigación.....	4
1.2.1 Objetivo general	4
1.2.2 Objetivos específicos.....	4
1.3 Hipótesis	5
1.3.1 Hipótesis general.....	5
1.3.2 Identificación de las variables	5
CAPÍTULO II	6
2. MARCO TEÓRICO.....	6
2.1 Antecedentes del problema.....	6
2.2 Bases Teóricas.....	8
2.2.1 Protocolo MQTT (Message Queuing Telemetry Transport).....	8
2.2.1.1 Modelo Publicador / Suscriptor.....	9
2.2.1.2 Arquitectura del protocolo MQTT.....	11
2.2.1.3 Paquetes de control MQTT	13
2.2.1.4 Tipos de paquetes de control MQTT	13
2.2.1.5 Niveles de Calidad de Servicio (QoS)	14
2.2.2 Norma IEC 61499	16
2.2.2.1 Objetivos de la Norma IEC 61499.....	16
2.2.2.2 Estructura de la Norma IEC 61499.....	17
2.2.2.3 Modelos de la Norma IEC 61499.....	17
2.2.2.4 Herramientas de desarrollo y de ejecución de la Norma IEC 61499.....	21
2.2.3 Grupo KUKA (Keller Und Knappich Ausburg)	26
2.2.3.1 Robótica KUKA.....	27
2.2.3.2 Kuka Youbot.....	27

2.3	Marco Conceptual	28
CAPÍTULO III		30
3.	MARCO METODOLÓGICO	30
3.1	Desarrollo de la metodología de la investigación	30
3.1.1	Alcance de la investigación	30
3.1.2	Técnicas de recolección de datos.....	31
3.1.3	Instrumentos para la recolección de datos	31
3.1.4	Instrumentos para procesar datos recopilados	32
3.2	Caso de estudio	32
3.2.1	Fase 1. Creación de librerías del MQTT PAHO	34
3.2.2	Fase 2. Creación de FORTE.....	34
3.2.3	Fase 3. Creación de Bloques de función.....	34
3.2.3.1	Bloque Suscriptor.....	34
3.2.3.2	Bloque Publicador.....	41
3.2.3.3	Bloque Controller.....	45
3.2.3.4	Carga de FBs en 4DIAC FORTE	47
3.2.4	Implementación del modelo	48
CAPÍTULO IV		50
4.	RESULTADOS Y DISCUSIONES	50
4.1	Pruebas de funcionamiento del sistema	50
4.2	Resultados obtenidos en la comunicación de datos	52
4.3	Comprobación de hipótesis	55
4.4	Medición de tiempos	57
4.4.1	Resultados obtenidos en los tiempos de transmisión.	58
4.5	CONCLUSIONES	60
4.6	RECOMENDACIONES	61
BIBLIOGRAFÍA		
ANEXOS		

ÍNDICE DE TABLAS

Tabla 2-1: Antecedentes del problema	6
Tabla 2-2: Herramientas de desarrollo y ejecución de la norma IEC 61499.....	21
Tabla 3-1: Operacionalización de variables	31
Tabla 4-1: Número de envíos desde el cliente MQTT.....	53
Tabla 4-2: Número de envíos desde el fuerte de la aplicación	54
Tabla 4-3: Total de éxitos y fracasos en el sistema de comunicación creado.	55
Tabla 4-4: Frecuencias de la variable de estudio	56
Tabla 4-5: Estadísticos de contraste	56
Tabla 4-6: Tiempos obtenidos en las etapas de publicación y suscripción de datos	58

ÍNDICE DE FIGURAS

Figura 2-1. MQTT protocolo de la capa de aplicación	9
Figura 2-2. Modelo Publicador / Suscriptor	10
Figura 2-3. Comunicación establecida entre un cliente y un bróker	11
Figura 2-4. MQTT cliente - bróker	12
Figura 2-5. Arquitectura del protocolo MQTT	13
Figura 2-6. QoS nivel 0.....	15
Figura 2-7. QoS nivel 1	15
Figura 2-8. QoS nivel 2.....	15
Figura 2-9. Modelo de Sistema en la norma IEC 61499.....	18
Figura 2-10. Modelo de Dispositivo en la norma IEC 61499	18
Figura 2-11. Modelo de recurso en la norma IEC 61499	19
Figura 2-12. Modelo de Aplicación en la norma IEC 61499.....	19
Figura 2-13. Representación gráfica de un bloque de función.....	20
Figura 2-14. Funcionalidad de un bloque funcional	20
Figura 2-15. Entorno de desarrollo FBDK	22
Figura 2-16. Entorno de desarrollo 4DIAC	23
Figura 2-17. Ejemplos de procesos donde Kuka tiene presencia	27
Figura 2-18. Componentes del Kuka Youbot.....	28
Figura 3-1. Caso de estudio.....	33
Figura 3-2. Fases de desarrollo del modelo de comunicación MQTT	33
Figura 3-3. Creación del FB youbot_subscribe	35
Figura 3-4. Interfaz inicial del youbot_subscribe	35
Figura 3-5. Entradas y salidas del youbot_subscribe	36
Figura 3-6. Arquitectura interna del youbot_subscribe	39
Figura 3-7. Wizard de exportación.....	40
Figura 3-8. Creación de archivos youbot_subscribe.cpp y youbot_subscribe.h	41
Figura 3-9. Creación del FB youbot_publish	41
Figura 3-10. Interfaz inicial del youbot_publish	42
Figura 3-11. Entradas y salidas del youbot_publish	42
Figura 3-12. Arquitectura interna del youbot_publish.....	44
Figura 3-13. Entradas y salidas del FB youbot_controller.....	45
Figura 3-14. Arquitectura interna del youbot_controller	46
Figura 3-15. Carga de los módulos creados	47

Figura 3-16. Compilación final del forte de la aplicación	48
Figura 3-17. Implementación del modelo.....	48
Figura 3-18. Modo de operación del modelo.....	49
Figura 4-1. Pruebas de funcionamiento del modelo	51
Figura 4-2. Envíos exitosos y de fracaso desde el cliente MQTT.....	53
Figura 4-3. Envíos exitosos y de fracaso desde el forte de la aplicación	54
Figura 4-4. Porcentajes obtenidos en las pruebas de Tx de datos	55
Figura 4-5. Medición de tiempos en la etapa de suscripción capturados desde Wireshark	57
Figura 4-6. Medición de tiempos en la etapa de publicación capturados desde el Wireshark....	58

ÍNDICE DE ANEXOS

Anexo A. Creación de librerías del MQTT PAHO

Anexo B. Creación de Forte

RESUMEN

Esta investigación presenta un modelo de comunicación Publicador/Suscriptor basado en el protocolo open source MQTT (Message Queue Telemetry Transport) cuya programación se la realiza a través de bloques de función adoptados por la nueva norma de automatización IEC 61499, la cual permite un desarrollo a través de eventos y datos de una arquitectura que comunica un sistema de control de un brazo robótico Kuka Youbot hacia un bróker en la nube. Construir soluciones de IIoT (Internet Industrial de las Cosas) permite avanzar en la nueva revolución industrial o mejor conocida como Industria 4.0. Diseñar arquitecturas que comuniquen dispositivos industriales hacia plataformas cloud (en la nube) abre las posibilidades de una verdadera colaboración en tiempo real. Los bloques de función (FB) desarrollados, corresponden a un suscriptor, publicador y un controlador de flujo que presentan como entrada de datos los correspondientes a los controles del robot Kuka Youbot desde donde se origina la información que se desea transmitir, y a los parámetros de conexión del servidor MQTT al cual se está apuntando. El modelo de comunicación fue desarrollado bajo el entorno de programación 4DIAC-IDE y el runtime 4DIAC-FORTE y validado mediante el uso de herramientas estadísticas que permitieron comprobar la hipótesis de investigación.

PALABRAS CLAVE: <TECNOLOGÍA Y CIENCIAS DE LA INGENIERÍA>. <<REDES>> <INDUSTRIA 4.0>, <INTERNET INDUSTRIAL DE LAS COSAS (IIoT)>, <MESSAGE QUEUE TELEMETRY TRANSPORT (MQTT)>, <NORMA IEC 61499>, <KUKA YUBOT>



ABSTRACT

This research presents a communication model Publisher/Subscriber based on the open source protocol MQTT (Message Queue Telemetry Transport) whose programming is carried out through function blocks adopted by the new automation standard rule IEC 61499, which allows a development through events and data of an architecture that communicates a control system of a Kuka Youbot robotic arm towards a cloud broker. Building solutions of IIoT (Industrial Internet of Things) allows progress in the new industrial revolution or better known as Industry 4.0. Designing architectures that communicate industrial devices with cloud platforms (in the cloud) opens up the possibilities of real collaboration in real time. The function blocks (FB) developed, correspond to a subscriber, publisher and a flow controller that present as data entry those corresponding to the controls of the Kuka Youbot robot from which the information to be transmitted is originated, and to the parameters of MQTT server connection to which you are targeting. The communication model was developed under the 4DIAC IDE programming environment and the 4DIAC-FORTE runtime and validated through the use of statistical tools that allowed us to verify the research hypothesis.

KEYWORDS: <TECHNOLOGY AND SCIENCES OF ENGINEERING>, < NETWORKS > <INDUSTRY 4.0>, < INDUSTRIAL INTERNET OF THINGS (IIoT)>, <MESSAGE QUEUE TELEMETRY TRANSPORT (MQTT)>, < RULE IEC 61499>, <KUKA YOUNOT>



CAPÍTULO I

1. INTRODUCCIÓN

La implementación de arquitecturas de IoT interoperables es un verdadero desafío. Desde sensores y actuadores en el campo, hasta sistemas de back-end, exigen aspectos de soluciones integrales en las que es importante confiar en estándares.

El estándar IEC 61499 representa una solución de componentes para sistemas de automatización industriales distribuidos, destinados a la portabilidad, interoperabilidad, reutilización y la reconfiguración de aplicaciones distribuidas (Vyatkin, 2009). Además proporciona un modelo genérico para sistemas distribuidos el cual incluye procesos y redes de comunicación como un medio para dispositivos embebidos (Insaurralde, 2016). Existen varios modelos específicos que trabajan con este estándar, uno de los más empleados es el bloque de función FB, el cual permite metodologías de diseño modular (Zoitl & Vyatkin, 2009).

La nube puede ayudar a mejorar estos procesos de automatización, los avances emergentes y la creciente disponibilidad de trabajo en red en la "nube" sugieren nuevas aproximaciones donde los procesamientos son ejecutados de forma remota con acceso a la dinámica global del conjunto de datos para soportar un rango de funciones (Boschetti, Baglio, Ruiu, & Terzo, 2015).

Cloud automation es una infraestructura de nube que a través de los recursos que ofrece el internet se convierte en una potente herramienta que beneficia a los sistemas de automatización brindando ciertas ventajas como: Big Data, Cloud Computing y Human computation (B Kehoe S Patil, 2014).

A nivel de comunicaciones, los protocolos han tenido también un proceso de evolución gradual a medida que la tecnología electrónica ha avanzado especialmente en lo que se refiere a soluciones cloud.

Los estándares que se aplican son orientados a protocolos utilizados para implementar las comunicaciones dispositivo a dispositivo o dispositivo a servidor, además, protocolos de administración de dispositivos para permitir el control remoto de éstos y puertas de enlace de IoT.

Si bien los estándares abiertos son clave, también es importante poner a disposición implementaciones de código abierto de dichos estándares, para alentar su adopción tanto por los desarrolladores de IoT como por la industria de IoT en general.

Este tipo de protocolos combinados con las nuevas infraestructuras en la nube permiten crear soluciones que conecten dispositivos industriales con plataformas de TI optimizando el control y monitoreo de los procesos industriales desde cualquier punto.

1.1 Planteamiento del problema

La complejidad siempre creciente en los sistemas de automatización, así como la necesidad de agilidad, flexibilidad de uso, extensibilidad y comunicación, imponen la creación de nuevos procesos que aborden de manera efectiva las tendencias actuales de cambio dentro de la denominada revolución industrial o Industria 4.0.

Actualmente en las empresas manufactureras los sistemas de control y automatización generalmente se desarrollan utilizando procedimientos centrados en un solo dispositivo, cuyo control y monitoreo se lo realiza dentro del propio lugar de producción.

Esta realidad conduce por un lado a la escasa posibilidad de incorporar dentro de un mismo proceso dispositivos de múltiples marcas convirtiéndolo en un sistema poco escalable, y por otro, a la dificultad de controlar y manejar procesos remotamente.

El desafío de la nueva era industrial, es precisamente romper con todos estos paradigmas, donde todos los dispositivos industriales subsistan en un solo ecosistema y permitan la transmisión de datos hacia cualquier punto remoto.

1.1.1 Formulación del problema

- ¿Se logra establecer una transmisión de datos desde un robot industrial hacia la nube para potenciar un control remoto de éstos?

1.1.2 Preguntas directrices o específicas de la investigación

- ¿Se realiza un monitoreo remoto desde la nube de las señales generadas por robots manipuladores móviles?
- ¿El protocolo de comunicación MQTT sirve para transmitir los datos generados desde un robot Kuka Youbot hacia la nube?
- ¿La programación de bloques de función FBs bajo la norma IEC 61499 garantiza que el protocolo de comunicación transmita datos del robot Kuka Youbot hacia la nube?

1.1.3 Justificación de la investigación

El control y monitoreo de datos en un sistema de producción es clave para poder mejorar parámetros de calidad, ya que al conocer la información que se genera en éstos se pueden desarrollar estrategias que permitan optimizar sus procesos.

Conectar dispositivos industriales con tecnologías de información y plataformas IoT, significa dar un gran paso en la Industria 4.0 y avanzar así hacia el denominado Internet Industrial de las cosas IIoT el cual permite a través de una infraestructura de red conectar sistemas cyber físicos (CPS) que posibilitan la exploración de nuevos entornos de control y gestión de procesos de forma remota (Colombo, 2017).

Los CPS se perciben como los habilitadores fundamentales en tiempo real de la comunicación y colaboración de todos los participantes en la cadena de valor de un proceso industrial (Colombo, 2017), a través de la aplicación de protocolos de comunicación que permiten establecer la transmisión de datos a plataformas cloud (Semle, 2016).

De allí, surge la necesidad de desarrollar modelos de comunicación que permitan mediante protocolos adaptables al IoT conectar dispositivos industriales hacia la nube, haciendo uso de tecnologías avanzadas de software como la encapsulación de funcionalidad, diseño basado en componentes, ejecución dirigida por eventos y distribución, entre otros (Vlad, Popa, Pentiu, & Buzduga, 2014).

El proyecto propuesto busca desarrollar un modelo de comunicación utilizando el protocolo MQTT -que se destaca por ser ligero y apropiado para dispositivos de baja potencia como los que frecuentemente se tiene en IoT-, que mediante la programación de bloques de función (FB)

introducido por la norma IEC 61499 transmite y recibe datos desde un robot Kuka Youbot a un bróker en la nube, para el monitoreo y control de éstos.

Con la utilización de la norma IEC 61499 se garantiza que una aplicación industrial de este tipo modele su sistema de comunicación mediante la utilización de FBs interconectados por eventos y datos (Guellouz, Benzina, Khalgui, & Frey, 2016).

Los bloques de función están cada vez más presentes en la programación de sistemas de automatización y control industrial. Las características de éstos han hecho que sean vistos como una herramienta efectiva para que la comunicación sea flexible y reconfigurable.

No existe hasta el momento trabajos relacionados al tema propuesto, la transmisión de datos desde dispositivos industriales hacia la nube no ha sido desarrollada aún, es por ello que este tipo de investigación se convierte en un aporte importante a los objetivos que persigue la nueva era industrial, beneficiando a todos aquellos usuarios inmersos en procesos industriales que buscan elevar sus estándares de calidad mediante la aplicación de nuevos medios tecnológicos.

1.2 Objetivos de la investigación

1.2.1 Objetivo general

- Crear un modelo de comunicación que permita la transmisión de datos desde un robot Kuka Youbot hacia la nube utilizando el protocolo MQTT basado en la norma IEC 61499.

1.2.2 Objetivos específicos

- Diseñar la arquitectura de FBs basados en la norma IEC 61499 para establecer un modelo de comunicación hacia la nube.
- Programar bloques de función FBs para lograr la transmisión de los datos generados desde un robot Kuka Youbot aplicando el protocolo MQTT.
- Validar los bloques de función FBs para comunicación a la nube, aplicando herramientas de control estadístico.

1.3 Hipótesis

1.3.1 Hipótesis general.

El protocolo MQTT basado en la norma IEC 61499 permite establecer la comunicación de datos desde el robot Kuka Youbot hacia la nube.

1.3.2 Identificación de las variables

Variable independiente

- Integración del Robot kuka Youbot hacia la nube.

Variable dependiente

- Protocolo MQTT basado en la norma IEC 61499.

CAPÍTULO II

2. MARCO TEÓRICO

Este capítulo, inicia con la descripción de varios artículos científicos relacionados al tema de investigación que fueron tomados como antecedentes para el desarrollo del presente trabajo. Se aborda además las bases teóricas de los principales actores de esta investigación, como son: Protocolo MQTT, Norma IEC 61499 y Kuka Youbot, destacando su funcionamiento y características principales.

2.1 Antecedentes del problema

Como se mencionó en la justificación teórica del capítulo anterior (*numeral 1.1.4*), no existen trabajos similares al propuesto, sin embargo, lo que se encontró son publicaciones que relacionan de forma independiente a aplicaciones basadas en el protocolo MQTT, norma IEC 61499 y cloud computing. A continuación, se nombra algunas de éstas:

Tabla 2- 1: Antecedentes del problema

SECURE MQTT FOR INTERNET OF THINGS (IoT) (Singh, Ma, VI, & Balamuralidhar, 2015)		
Objetivo	Instrumentos de medición de datos	Resumen
Desarrollar una versión segura del protocolo MQTT, que garantice la confiabilidad de los datos al momento de transmitirlos de dispositivo a dispositivo.	Métodos analíticos y de simulación	En este artículo se analiza la viabilidad de CP / KP-ABE para permitir la seguridad de la comunicación de dispositivos IoT basados en la arquitectura Pub-Sub. Como parte de esto, se diseña e implementa protocolos MQTT seguros (SMQTT, SMQTTSN) con el nuevo comando de publicación segura "SPublish" que publica datos encriptados basados en el esquema CP / KP-ABE usando técnicas ECC ligeras optimizando parámetros y algoritmos de cómputo. Se estudia además el SMQTT bajo diferentes escenarios de ataque y también se propone la factibilidad de SMQTT para la arquitectura Pub-Sub distribuida de extremo a extremo. A través de métodos analíticos y de simulación, se demuestra la

		aplicación de SMQTT basado en CP / KP-ABE para diversos requisitos de IoT.
END-TO-END RESPONSE TIME OF IEC 61499 DISTRIBUTED APPLICATIONS OVER SWITCHED ETHERNET (Lindgren, Eriksson, Lindner, Lindner, & Tec, 2016)		
Objetivo	Instrumentos de medición de datos	Resumen
Proponer una técnica de implementación de baja complejidad que permita evaluar los tiempos de respuesta de extremo a extremo de cadenas de eventos que abarcan un conjunto de dispositivos en red, basados en la norma IEC 61499	Experimentos sobre entornos reales	En este artículo se discute los tiempos de respuesta de extremo a extremo para las aplicaciones IEC 61499 distribuidas que se comunican a través de redes Ethernet conmutadas. Para el análisis y la implementación, el sistema se traduce en tareas y recursos concurrentes en el lenguaje RTFM-core. Dichos modelos se expresan directamente en el lenguaje de núcleo RTFM derivado de las especificaciones del sistema IEC 61499 o cualquier mezcla de los mismos. La naturaleza estática del modelo de núcleo RTFM permite evaluar los tiempos de respuesta a nivel de dispositivo utilizando el análisis de planificación disponible. Para cada enlace, el número de mensajes (pendientes) en el sistema está enlazado con precisión, permitiendo así que el tiempo de bloqueo y cola sea estimado con seguridad. Para evaluar el impacto de los conmutadores en la red, se presenta un modelo genérico y se desarrolla una configuración experimental que permite caracterizar los conmutadores a mano. Los experimentos llevados a cabo en microcontroladores y conmutadores Ethernet comercialmente disponibles indican que los tiempos de respuesta de extremo a extremo bien inferiores a 1ms son factibles incluso sobre topologías de red de árbol (backbone) y en presencia de tráfico.
INTELLIGENT PARKING CLOUD SERVICES BASED ON IoT USING MQTT PROTOCOL (Dhar & Gupta, 2017)		
Objetivo	Instrumentos de medición de datos	Resumen
Crear una red de servicios en la nube para el manejo de un parqueadero vehicular utilizando el protocolo MQTT	Medición de QoS en entornos reales	Se trata de un sistema que ayuda a evitar el aumento de los problemas de tráfico en las áreas de estacionamiento pesado como centros comerciales y otras zonas con alta densidad vehicular, además de

		ahorrar el combustible de los vehículos y, por tanto, reducir la tasa de contaminación en el medio ambiente.
OPEN CLOUD SOLUTION FOR INTEGRATING ADVANCED PROCESS CONTROL INPLANT OPERATION (Chenaru et al., 2015)		
Objetivo	Instrumentos de medición de datos	Resumen
Integrar una biblioteca avanzada de control de procesos en un entorno basado en la nube.	Observación, pruebas en ambientes reales	Se trata de una aplicación basada en la nube que permite a un usuario acceder a diferentes estrategias de control usando una interfaz web beneficiándose además de los controladores virtualizados que ejecutan esos algoritmos y envían los comandos a diferentes dispositivos de planta. El enfoque innovador presentado en este trabajo viene de la virtualización de controladores de proceso y almacenamiento en bases de datos.
RECONFIGURABLE FUNCTION BLOCKS: EXTENSION TO THE STANDARD IEC 61499 (Guellouz et al., 2016)		
Objetivo	Instrumentos de medición de datos	Resumen
Optimizar el diseño de una red de bloques de funciones encapsulando varios escenarios de reconfiguración en un solo bloque de funciones. Definir los eventos que desencadenan la reconfiguración y adjuntar una probabilidad a cada evento para expresar su incertidumbre. Todo esto con el fin de verificar el sistema y evaluar sus resultados.	Modelación con una clase de redes Petri aplicado a una plataforma médica BROS a través de una herramienta de software desarrollado llamado ZiZo v3	Este artículo presenta una extensión a la norma IEC 61499 llamada Reconfigurable Function Block (RFB) que sirvió para diseñar el comportamiento de los sistemas de automatización distribuidos reconfigurables.

Fuente: Singh, Ma, VI, & Balamuralidhar, 2015; Lindgren, Eriksson, Lindner, Lindner, & Tec, 2016; Dhar & Gupta, 2017; Chenaru et al., 2015; Guellouz et al., 2016

Realizado por: Tatiana Zambrano. 2019

2.2 Bases Teóricas

2.2.1 Protocolo MQTT (Message Queuing Telemetry Transport)

Es un protocolo de mensajería que proporciona mecanismos para la comunicación asíncrona, en dispositivos con capacidades de memoria restringidas y potencia de procesamiento limitada (Bani Yassein, Shatnawi, Aljwarneh, & Al-Hatmi, 2017).

En un principio era un protocolo asociado a IBM, pero actualmente se trata de un protocolo abierto. Desde 2013 está en proceso de normalización y es supervisado por la Organización para el Avance de Estándares de Información Estructurada (OASIS, Organization for the Advancement of Structured Information Standards).

Sus principios de diseño son optimizar los recursos del dispositivo, reducir el consumo de ancho de banda y garantizar la seguridad y fiabilidad en la entrega de datos. MQTT es un protocolo de conectividad M2M (máquina a máquina) que se basa en el patrón de comunicación publicador / suscriptor, considerándose de esta forma como el protocolo de conexión más favorable para M2M dentro de la plataforma del Internet Industrial de las Cosas (Kang et al., 2017).

El protocolo MQTT posee puertos TCP/IP estandarizados para poder utilizarlo. El 1883 está reservado con IANA (Internet Assigned Numbers Authority) para su uso básico, y, por otro lado, tiene registrado el puerto 8883 para los casos de comunicaciones seguras mediante TLS/SSL (García Soria, 2017).

MQTT es un protocolo de la capa de aplicación que utiliza el protocolo de control de transmisión TCP para el transporte, proporcionando así una implementación simple y una transición flexible, la *figura 2-1* muestra lo mencionado (Bani Yassein et al., 2017).

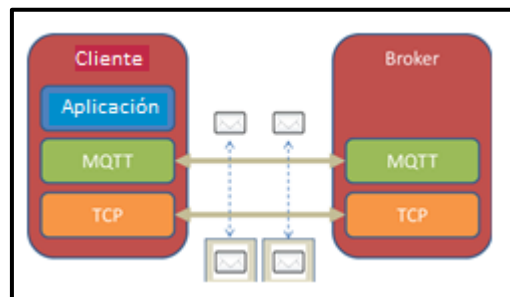


Figura 2-1. MQTT protocolo de la capa de aplicación
Fuente: Bani Yassein et al., 2017

2.2.1.1 Modelo Publicador / Suscriptor

Al tratarse de un protocolo basado en el modelo Publicador / Suscriptor, los clientes no se comunican directamente entre ellos, sino que publican mensajes -compuestos de un contenido y un tópico- sobre un bróker¹. Es decir, este protocolo requiere utilizar un gestor intermediario que

¹Gestor intermediario

funciona como hub o nodo central, el cual, gestiona la red y distribuye mensajes entre dispositivos o usuarios manteniendo activo el canal de comunicaciones, tal como se muestra en la *figura 2-2* (Semle, 2016).

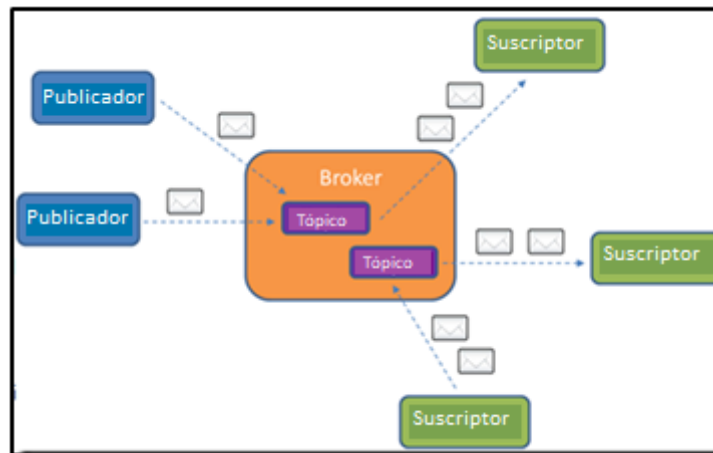


Figura 2-2. Modelo Publicador / Suscriptor
Fuente: Semle, 2016

MQTT usa un tópico para definir un canal de comunicación entre los equipos. Aquellos que deseen recibir datos necesitan suscribirse bajo ese tópico en el bróker y así se los relacionará con un canal de comunicación, mientras que los dispositivos que deseen enviar datos necesitarán publicarlos bajo el tópico que relaciona al canal de comunicación deseado. En este protocolo, cuando un nuevo dato está disponible, su bróker simplemente envía los nuevos datos a cualquier dispositivo suscrito en éste (Osathanukul, Hantrakul, Pramokchon, Khoenkaw, & Tantitharanukul, 2017).

Los tópicos en MQTT tienen una estructura jerárquica, donde los clientes deciden el nivel al que desean suscribirse (García Soria, 2017).

Esta solución tiene la ventaja de permitir que muchos clientes se comuniquen incluso si no están conectados al mismo tiempo con el bróker. La comunicación se realiza intercambiando un conjunto de paquetes de control MQTT (Houimli, Kahloul, & Benaoun, 2017).

Una comunicación MQTT puede dividirse en las siguientes etapas: conexión, autenticación, suscripción, anulación de suscripción, publicación y desconexión, como se indica en la *figura 2-3* (García Soria, 2017).

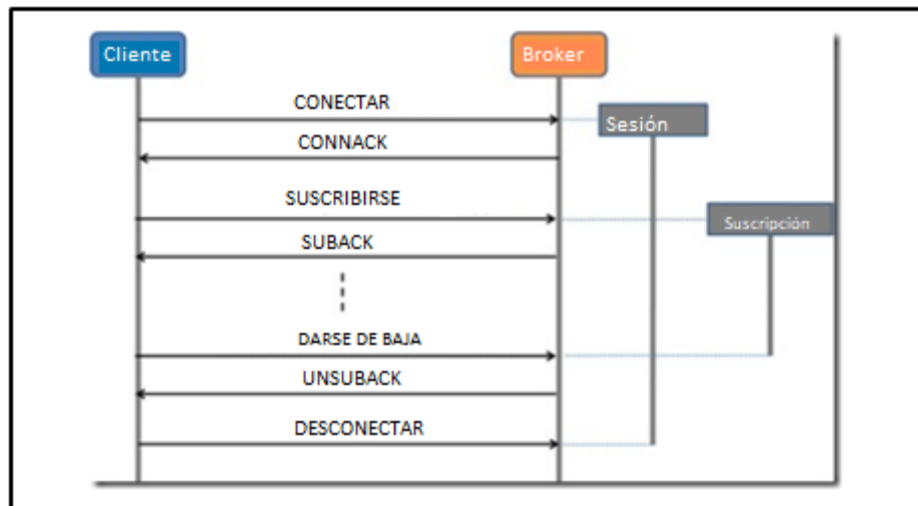


Figura 2-3. Comunicación establecida entre un cliente y un bróker
Fuente: García Soria, 2017

A diferencia de otros protocolos, éste no necesita enviar mensajes constantes de solicitud para refrescar nuevos datos, por tanto, maneja una carga inferior de paquetes lo que lo hace más rápido (Houimli et al., 2017).

MQTT está diseñado para entornos de IoT donde se tiene recursos limitados y responde a las siguientes necesidades (Yokotani & Sasaki, 2017):

- Está adaptado especialmente para utilizar un ancho de banda mínimo.
- Es ideal para utilizarlo en redes inalámbricas, ya que consume muy poca energía.
- Es muy rápido y posibilita un tiempo de respuesta superior al resto de protocolos web actuales.
- Permite una gran fiabilidad y una distribución eficiente de la información en los receptores.
- Requiere pocos recursos procesadores y memorias.

2.2.1.2 *Arquitectura del protocolo MQTT (TITULO 4 NIVEL SIN NEGRITA Y CURSIVA)*

MQTT Client: Puede ser cualquier objeto IoT que envíe o reciba datos, no solo dispositivos. Y, cualquier dispositivo puede ser un cliente, por ejemplo, un microcontrolador, servidor, etc. El tipo

de cliente MQTT depende de su función en el sistema, es decir, puede tomar el rol de suscriptor o publicador (Ding Yi, Fan Binwen, Kong Xiaoming, & Ma Qianqian, 2016).

MQTT Broker: Es un dispositivo central que funciona como hub. Las principales responsabilidades del bróker de MQTT son procesar la comunicación entre los clientes de MQTT y distribuir los mensajes entre ellos en función de sus temas de interés (Mektoubi, Hassani, Belhadaoui, Rifi, & Science, 2016). El bróker puede manejar miles de dispositivos conectados al mismo tiempo. Al recibir el mensaje, éste debe buscar y encontrar todos los dispositivos que poseen una suscripción a ese tema (Ding Yi et al., 2016). La *figura 2-4* indica lo mencionado.

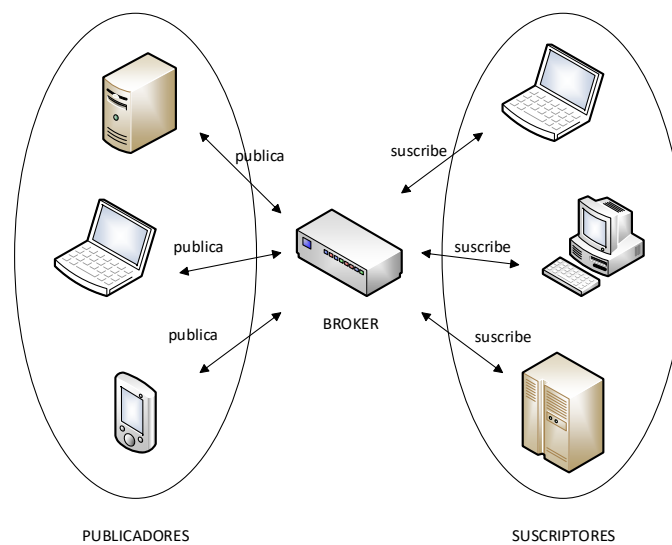


Figura 2-4. MQTT cliente - bróker
Fuente: Ding Yi et al., 2016

De esta forma, la arquitectura MQTT contiene tres componentes: un publicador, un bróker y un suscriptor como se observa en la *figura 2-5*. El dispositivo que está interesado en un tema específico se registra en el bróker como un suscriptor para estar informado cuando los editores publican sus temas en éste. Luego, el publicador transfiere la información a los suscriptores a través del bróker, es decir, las entidades interesadas (Ding Yi et al., 2016).

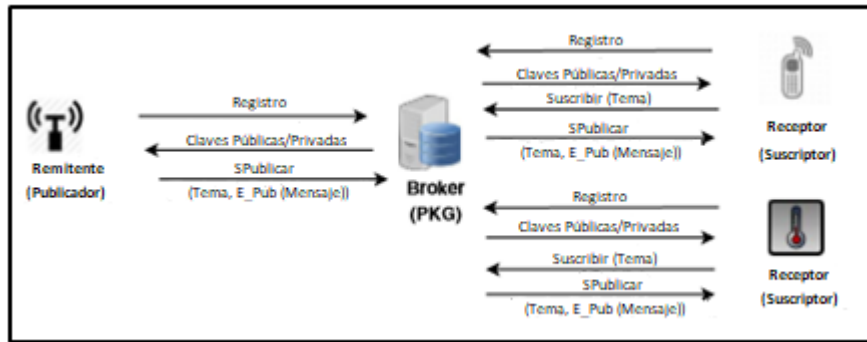


Figura 2-5. Arquitectura del protocolo MQTT
Fuente: Ding Yi et al., 2016

2.2.1.3 Paquetes de control MQTT

Consta de hasta tres partes, siempre en el siguiente orden (Ding Yi et al., 2016):

- Encabezado Fijo (todos los paquetes de control lo tienen obligatoriamente, 2 bytes)
- Encabezado Variable (sólo algunos paquetes de control MQTT lo poseen, ya que varía según el tipo de paquete)
- Carga útil o Payload (limitada a 256MB de información)

2.2.1.4 Tipos de paquetes de control MQTT

El protocolo MQTT funciona intercambiando una serie de paquetes de control MQTT que se describen brevemente a continuación (Houimli et al., 2017):

- *CONNECT* es el primer paquete que el cliente debe enviar al servidor después del establecimiento de una red solicitando una conexión. Si un segundo *CONNECT* es recibido por el servidor, este último debe ser procesado como una violación de protocolo y el cliente debe desconectarse.
- *CONNACK* es un paquete de solicitud de confirmación de conexión enviada por el servidor en respuesta a un paquete *CONNECT*. Es el primer paquete enviado desde el servidor al cliente. Si el tiempo de recepción del paquete *CONNACK* excede una cantidad razonable de tiempo, el cliente debe cerrar la conexión de red.
- *PUBLISH* es el paquete que puede ser enviado por ambos servidor y cliente para transportar un mensaje de aplicación.

- *PUBACK* es un paquete de reconocimiento PUBLISH con el nivel de calidad de servicio (QoS)
- *PUBREC* es la respuesta a un paquete PUBLISH con QoS
- *PUBREL* es la respuesta a un paquete PUBREC
- *PUBCOMP* es la respuesta a un paquete PUBREL.
- *SUBSCRIBE* puede ser enviado por un cliente para crear una o más suscripciones que registran el interés de un cliente en uno o más tópicos. El paquete SUBSCRIBE también especifica (para cada suscripción) la máxima QoS con la cual el servidor puede enviar mensajes de aplicación al cliente.
- *SUBACK* es un paquete de acuse de recibo SUBSCRIBE que confirma la recepción y el procesamiento de un paquete SUBSCRIBE.
- *UNSUBSCRIBE* puede ser enviado por un cliente para darse de baja de los tópicos.
- *UNSUBACK* es un paquete de reconocimiento UNSUBSCRIBE que confirma la recepción de un paquete UNSUBSCRIBE
- *PINGREQ* puede ser enviado por un cliente cuando no hay un paquete de control que se enviará al servidor. Este paquete indica al servidor que el cliente está vivo. También se usa para indicar que la red está activa.
- *PINGRESP* puede ser enviado por el servidor en respuesta a un paquete PINGREQ, éste indica que el servidor está activo.
- *DISCONNECT* es el último paquete de control enviado por un cliente al servidor para indicar la desconexión de éste.

2.2.1.5 Niveles de Calidad de Servicio (QoS)

Hay tres niveles de calidad de servicio (QoS) para mantener la confiabilidad de los mensajes transmitidos en MQTT (Bani Yassein et al., 2017).

La entrega de un mensaje de aplicación puede ser desde un único remitente a un solo receptor o desde un servidor a más de un cliente. MQTT entrega mensajes de aplicación según los niveles de calidad de servicio (QoS) definidos (Andrew Banks and Rahul, 2017).

- *QoS 0*: Se envía un mensaje “máximo una vez” y no se requiere confirmación de la recepción, como se ilustra en la *figura 2-6*.

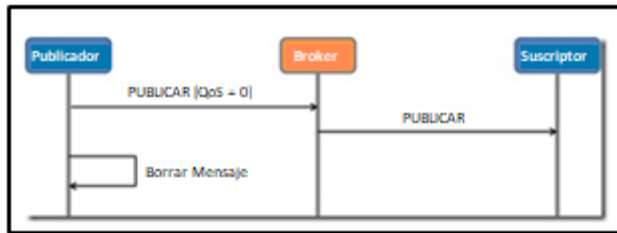


Figura 2-6. QoS nivel 0
Fuente: Bani Yassein et al., 2017

- *QoS 1*: Cada mensaje se entrega “al menos una vez” y se requiere una confirmación de recibir un mensaje, como se muestra en la *figura 2-7*.

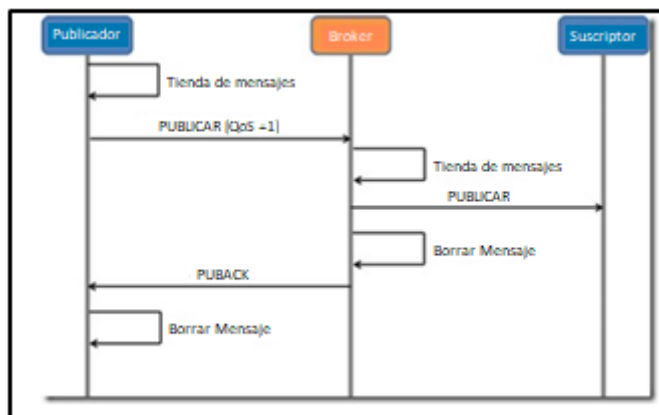


Figura 2-7. QoS nivel 1
Fuente: Bani Yassein et al., 2017

- *QoS 2*: Se envía un mensaje “exactamente una vez”, la entrega de la información se realiza en dos duplas de mensajes, así lo indica la *figura 2-8*.

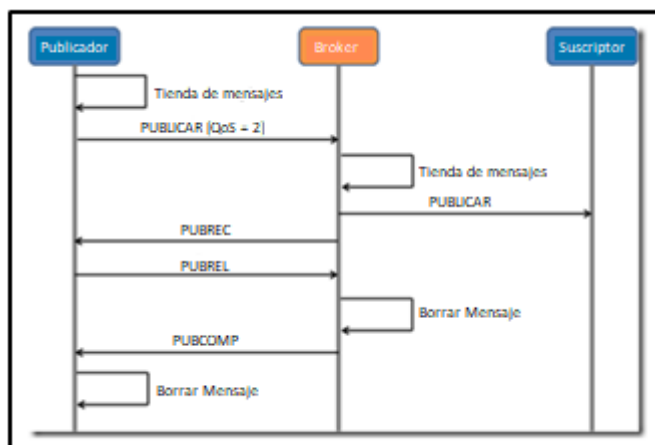


Figura 2-8. QoS nivel 2
Fuente: Bani Yassein et al., 2017

2.2.2 Norma IEC 61499

Se ha desarrollado para permitir una auténtica automatización donde la inteligencia se descentraliza y se integra de forma inequívoca en componentes de software, que se pueden distribuir libremente entre dispositivos en red (Vyatkin, 2011).

Se ha considerado como una arquitectura de referencia de la tecnología de información de automatización industrial (II), ya que está diseñada para facilitar el desarrollo de aplicaciones de control con lógica descentralizada, donde la unidad estructural básica de sus modelos son los bloques de función (Dai & Vyatkin, 2010).

Con la aparición de herramientas de software y docenas de plataformas de hardware, IEC 61499 está obteniendo reconocimiento en la industria ya que facilita el uso de la inteligencia de automatización distribuida. Las áreas de aplicación prometedoras de IEC 61499 incluyen sistemas flexibles de manejo de materiales, automatización de manufactura reconfigurable flexible, redes inteligentes de distribución de energía y SmartGrid, así como la amplia gama de sistemas integrados en red (Vyatkin, 2011).

2.2.2.1 Objetivos de la Norma IEC 61499

El propósito general de la norma es el de cambiar los paradigmas de los esquemas en los sistemas actuales de automatización convirtiéndolos de homogéneos a heterogéneos a través del cumplimiento de:

- *Portabilidad:* Se trata de que los componentes de software y las configuraciones del sistema puedan aceptar e interpretar correctamente las herramientas creadas por diferentes equipos de desarrollo (Lindgren et al., 2016).
- *Configurabilidad:* Cualquier dispositivo con su respectivo software puede ser configurado por las herramientas desarrolladas por diferentes proveedores (Lindgren et al., 2016).
- *Re-configurabilidad:* Poder cambiar sobre la marcha la programación de los controladores.
- *Interoperabilidad:* Los dispositivos multimarca pueden trabajar juntos para realizar las funciones necesarias de las aplicaciones distribuidas (Lindgren et al., 2016).

IEC 61499 define un lenguaje de modelado específico para desarrollar soluciones de control industrial distribuido, mejorando la encapsulación de componentes de software para una mayor reutilización, con un formato independiente del proveedor y simplificando el soporte para la comunicación entre dispositivos. Su funcionalidad de distribución y el soporte inherente para la reconfiguración dinámica proporcionan la infraestructura necesaria para las aplicaciones de la Industria 4.0 y el IIoT.

2.2.2.2 Estructura de la Norma IEC 61499

Se encuentra distribuida en cuatro partes, que se resumen a continuación:

- *PARTE 1_ Arquitectura:* Define la arquitectura de referencia para el desarrollo, reutilización e implementación de los bloques de función. Se define además los requisitos generales, definiciones y modelos de referencia (Lindgren et al., 2016).
- *PARTE 2_Requisitos de herramienta software:* Define los requisitos que deben cumplir las herramientas de desarrollo para poder desplegar las arquitecturas propuestas. Además, se definen los esquemas XML para el intercambio de ficheros entre las distintas herramientas (Lindgren et al., 2016).
- *PARTE 3_Manual Informativo:* Contiene información cuyo objetivo es mostrar las funcionalidades y ejemplos de aplicación de la norma IEC 61499 en un amplio conjunto de dominios (Lindgren et al., 2016).
- *PARTE 4_Reglas y Perfiles de Conformidad:* Define perfiles de compilación en los dispositivos compatibles y las herramientas de desarrollo (Lindgren et al., 2016).

2.2.2.3 Modelos de la Norma IEC 61499

- *Modelo de Sistema*

Se trata del elemento de mayor nivel en la estructura jerárquica de esta arquitectura, contiene a los demás modelos y en él se encuentran los dispositivos y las diferentes aplicaciones que conforman el sistema de control, así como la relación existente entre éstos conforme a lo definido en el estándar. Las aplicaciones pueden ser distribuidas sobre uno o varios recursos que existen en los diferentes dispositivos (Jose Carlos Mercé Godoy, 2015). La *figura 2-9* grafica lo descrito respecto a este modelo.

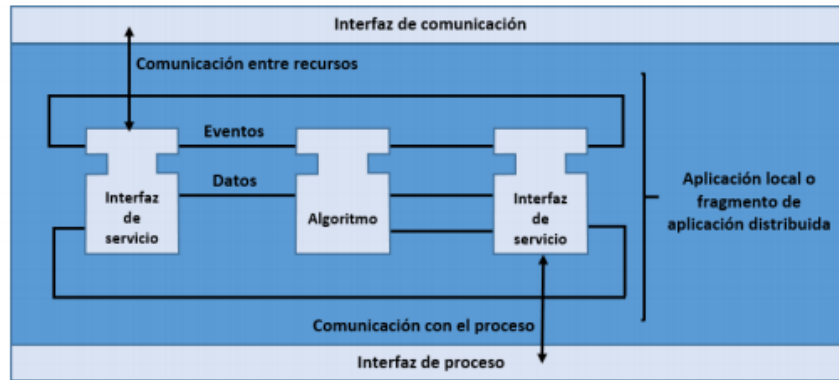


Figura 2-11. Modelo de recurso en la norma IEC 61499
Fuente: Insaurralde, 2016

- *Modelo de Aplicación*

Una aplicación es un conjunto de bloques de función interconectados entre sí y unidos por flujos de señales y datos (Insaurralde, 2016). Las aplicaciones se pueden distribuir entre diferentes recursos, del mismo dispositivo o de dispositivos diferentes. La *figura 2-12* indica lo descrito.

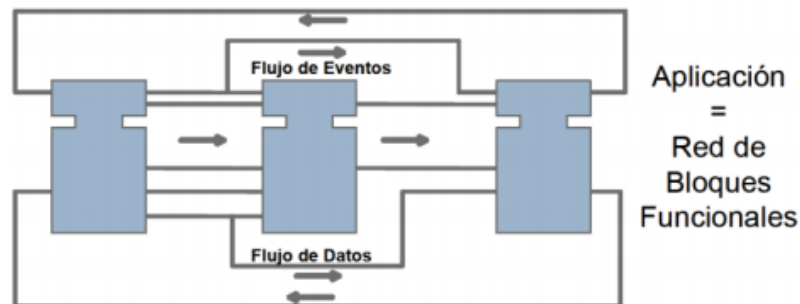


Figura 2-12. Modelo de Aplicación en la norma IEC 61499
Fuente: Insaurralde, 2016

- *Modelo de Bloque de Función*

Se define como una estructura a nivel de diseño que encapsula los algoritmos industriales y los datos en los que operan estos algoritmos, los cuales pueden ser programados en lenguajes de alto nivel como: C, C++, Java o incluso Delphi. Un FB es el elemento más pequeño en un sistema de control distribuido, representa la unidad funcional de más bajo nivel a partir de la cual se construirá el resto (De Sousa, 2010). La *figura 2-13* es una representación gráfica de un bloque de función.

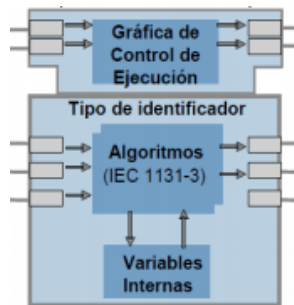


Figura 2-13. Representación gráfica de un bloque de función
Fuente: De Sousa, 2010

Un FB activa su funcionalidad cuando inicialmente le llega un evento de entrada, a partir de allí las entradas de datos relacionadas con el evento entrante se actualizan y dicho evento pasa al cuadro de control de ejecución ECC (Execution Control Chart) que procesa entrada y salida de eventos. Dependiendo del tipo de FB y del control de ejecución, la funcionalidad interna se activa para la ejecución y luego finaliza proporcionando nuevos datos de salida, éstos a su vez se actualizan y finalmente se envía un evento de salida. Este proceso de activación del FB se lo muestra en la *figura 2-14*.

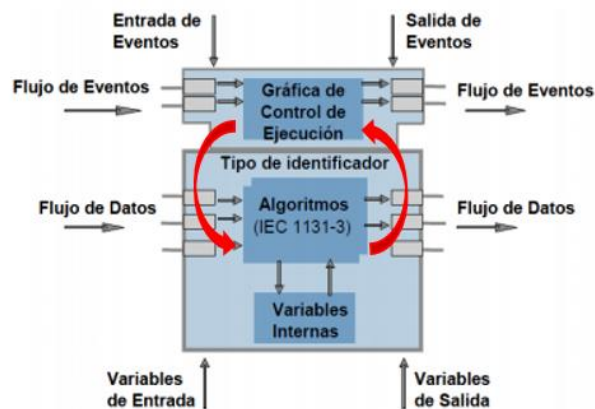


Figura 2-14. Funcionalidad de un bloque funcional
Fuente: De Sousa, 2010

Tipos de bloques de función:

Bloque funcional Básico: Unidad más pequeña de programación. Consta de dos partes: ECC y algoritmos.

Bloque funcional Compuesto: Son bloques funcionales compuestos de bloques funcionales básicos. Así, dentro de cada bloque compuesto se puede encapsular una cantidad ilimitada de bloques simples. La interfaz de cada bloque es idéntica a la de un bloque simple.

Bloque funcional de interfaz de servicio (SIFB): Son los encargados de gestionar las comunicaciones entre los bloques funcionales y el mundo exterior. Se encargan de proveer los servicios que los bloques necesitan, como por ejemplo, la escritura y la lectura de entradas y salidas y la comunicación entre aplicaciones y recursos diferentes.

- *Modelo de Subaplicación*

Se trata de un modelo de bloque funcional compuesto distribuible. Donde una subaplicación nunca puede estar contenida dentro de un bloque compuesto, solo puede pertenecer a aplicaciones o a otras subaplicaciones. La asociación de señales es muy similar a la de los bloques compuestos pero con menos restricciones (Insaurralde, 2016).

2.2.2.4 Herramientas de desarrollo y de ejecución de la Norma IEC 61499

- *Entorno de desarrollo IDE (Integrated Development Environment):* se trata de un constructor de interfaz gráfica (GUI) con editor de código, compilador y depurador que permiten el desarrollo de las aplicaciones de control.
- *Entorno de ejecución RTE (Runtime Environment):* es un estado de máquina virtual que brinda los servicios de software de procesos o programas mientras un ordenador está ejecutándose.

En la parte dos (2) de la norma *Requisitos Herramienta Software*, se especifica las características que deben cumplir las herramientas de desarrollo y ejecución para lograr cumplir con uno de los objetivos de la norma que es la interoperabilidad (Dai & Vyatkin, 2010). A continuación, se presenta las principales herramientas de software y sus entornos de ejecución bajo licencia de código abierto que se utilizan actualmente.

Tabla 2- 2: Herramientas de desarrollo y ejecución de la norma IEC 61499

ENTORNO DE DESARROLLO (IDE)	ENTORNO DE EJECUCIÓN (RTE)
FUNCTION BLOCK DEVELOPMENT KIT (FBDK)	FUNCTION BLOCK RUN TIME (FBRT)
4DIAC-IDE	FORTE
NXTONE	NXTRT61499F
ISAGRAF WORKBENCH	ISAGRAF RUNTIME

Fuente: Dai & Vyatkin, 2010

Realizado por: Tatiana Zambrano

- *Function Block Development Kit (FBDK)*

Se trata del primer entorno de desarrollo que se creó y se lo muestra en la *figura 2-15*, justamente por uno de los miembros creadores del IEC 61499. Su principal objetivo fue permitir la visualización de los FB, pero actualmente permite además representar, testear los bloques e intercambiarlos en forma de XML como se define en la parte dos (2) de la norma.

La herramienta, se encuentra desarrollada en java y posee una interfaz simple y poco amigable para el usuario. La aplicación aún tiene soporte y se distribuye de forma gratuita por la empresa Holoblocs. Inc. Sus actualizaciones son continuas y reflejan los cambios y mejoras en la norma. Es capaz de exportar e importar desde 4DIAC y nxtStudio (Thramboulidis, 2009).

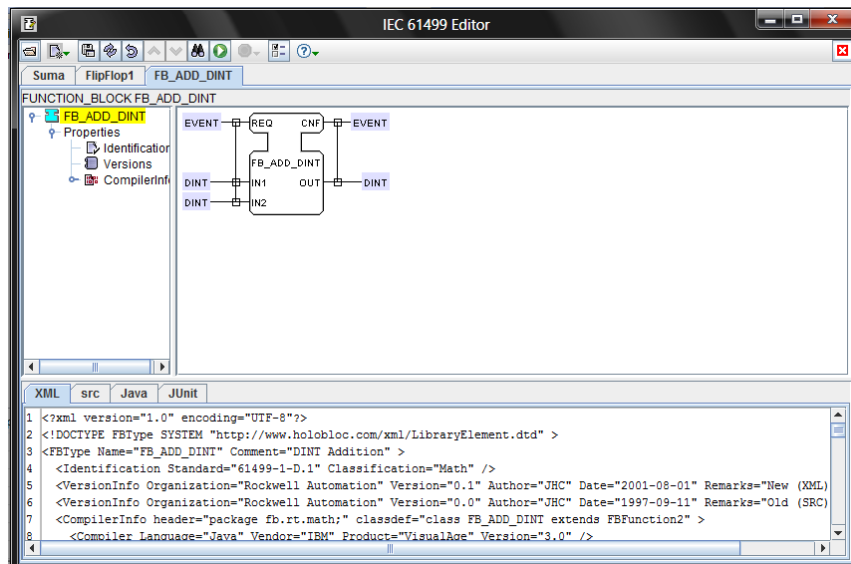


Figura 2-15. Entorno de desarrollo FBDK

Fuente: Thramboulidis, 2009

- *Function Block Run Time (FBRT)*

Este runtime fue el primero que se desarrolló junto a FBDK para poder testear y hacer las primeras demostraciones usando la norma IEC 61499. Está desarrollado en java y tiene dos versiones diferentes.

La primera se desarrolló en java embebido para poder ser utilizada en los dispositivos embebidos, no obstante cuando aparecieron otros runtimes se dejó de desarrollar, entre otras cosas, por la dificultad de lograr un sistema tiempo real en java.

La otra versión no-embebida estaba desarrollada en java estándar y se puede ejecutar en cualquier dispositivo con JVM. Esta última versión sigue siendo evolucionada y presenta la ventaja respecto a otras de que tiene un HMI que permite visualizar los resultados. Es por esto que se sigue utilizando FBRT, especialmente para simular y depurar aplicaciones. Además de FBDK, FBRT también puede ser configurado y utilizado por 4DIAC-IDE y nxtOne (Thramboulidis, 2009).

- *4DIAC-IDE*

Estructura para la Automatización y Control Industrial Distribuida (Distributed Industrial Automation and Control), es una herramienta de ingeniería open source basada en la plataforma de Eclipse, es muy robusto e intuitivo, tiene soporte oficial y saca dos actualizaciones del entorno por año (Zoiti S. a., 2011). Permite la creación de bloques funcionales y tiene características de debug, test e integración de otros plug-ins. Además, se ha utilizado en el desarrollo de muchas aplicaciones en componentes de diferentes vendedores lo que lo hace atractivo. Las aplicaciones modeladas se pueden descargar a dispositivos de campo distribuidos según los medios definidos por la norma IEC 61499 así como también a plataformas en la nube, además de ser compatible con la mayoría de las herramientas actualmente existentes (Wenger & Zoiti, 2012). La *figura 2-16* muestra su entorno.

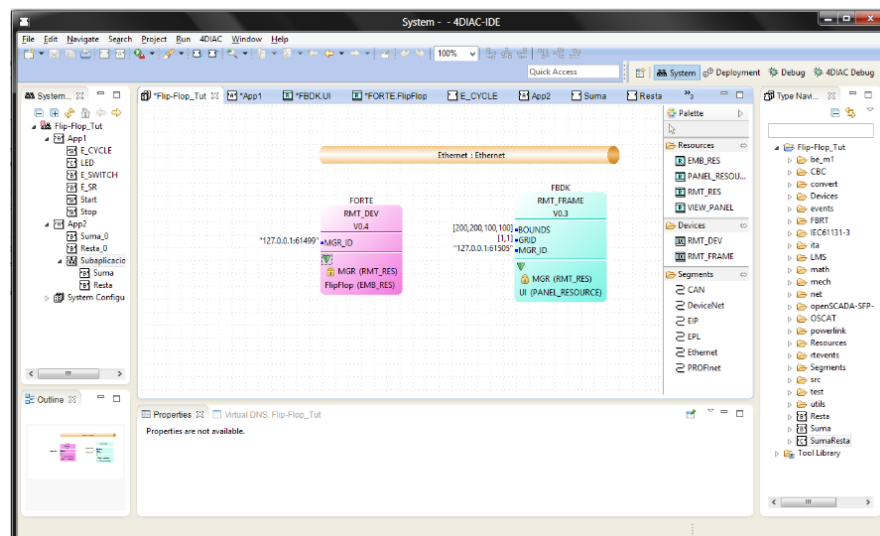


Figura 2-16. Entorno de desarrollo 4DIAC

Fuente: Wenger & Zoiti, 2012

El editor es un entorno de desarrollo integrado (IDE) cuyo objetivo es proporcionar herramientas conforme al estándar que permiten establecer un entorno de automatización y control, basado en los objetivos de reconfiguración, portabilidad e interoperabilidad. Éste proporciona una base común para desarrollo e investigación de IEC 61499, suministra un paquete que contiene un

entorno runtime para diferentes plataformas de control que a su vez extienden la norma hacia más dispositivos con sistemas embebidos, tiene conexiones de eventos y datos, configuración de comandos (crear, escribir, iniciar de acuerdo con la norma), FBs de comunicación (Cliente/Servidor, Publicador/Suscriptor) para Ethernet, ejecución de bloques de función de tipo FB básicos, FB compuestos, FB interfaces de servicio SIFBs y además se ejecuta bajo las plataformas de Windows, Linux y Solaris.

El IDE de 4DIAC se destaca entre los demás gracias a sus últimas características que lo vuelven la herramienta de software más utilizada; en su última versión se cuenta con la capacidad de establecer el valor de las variables de manera remota además de permitir la depuración y prueba de aplicaciones de control distribuido en línea.

- *4DIAC-RTE (FORTE)*

Se trata de una pequeña implementación de un entorno runtime conforme a IEC 61499, es portable para múltiples plataformas debido a que está implementado en C++ (Zoitl S. a., 2011). Fue desarrollado por el mismo equipo de 4DIAC y está especialmente pensado para ejecutarse en pequeños sistemas embebidos y sistemas de tiempo real, ya que tiene los mecanismos necesarios para poder asegurar la ejecución a tiempo y la gestión de prioridades. Otro de los puntos importantes de FORTE es que ha sido diseñado para ser independiente de la plataforma sobre la que se ejecute y posee una arquitectura escalable que le permite adaptarse al diseño de la aplicación. En la actualidad FORTE se ha llevado a diferentes sistemas operativos como POSIX, windows32, threadX y eCos. Y además de que puede ser configurado con 4DIAC también lo puede ser en otros entornos como FBDK o nXtOn.

En el entorno de ejecución FORTE, se utiliza un disparador de eventos para la planificación de FBs. Es decir, todos los eventos de entrada van a los FBs destino en orden FIFO (primero entra-primero sale). El disparador de eventos desacopla la ejecución de envío de eventos del bloque receptor, de ese modo crea el periodo de bloqueo de un FB independiente de la topología de red (Zoitl S. a., 2011).

FORTE cambia según la optimización de ejecución e implementación de interfaz de comunicaciones. Es así como puede funcionar independientemente de la plataforma que se emplea. La versión actual de FORTE soporta Windows(Win32), eCos, POSIX, Lego Mindstorms, NXT controller.

- *NxtStudio*

Es una aplicación comercial desarrollada por la compañía nxtControl, que ofrece soporte para la norma IEC 61499. Se ha utilizado en diversos dispositivos y máquinas de diferentes vendedores. Al igual que 4DIAC, es compatible con el resto de herramientas de desarrollo a excepción de ISaGRAF, incluye además funciones de debugging y de test. Posee un HMI para visualizar la ejecución e incluye una funcionalidad CAT (Compound Automation Types) que aporta un conjunto de elementos de control prediseñados (Thramboulidis, 2009).

- *NxTRT61499F*

Esta plataforma de ejecución se basa en FORTE, ha sido modificada para incluir algunas funcionalidades adicionales y así hacerla más eficiente. Pertenece a nxtControl y puede ser configurada desde nxtOne o desde 4DIAC y FBDK. Se ha implantado en dispositivos de diferentes vendedores entre los que destacan SIEMENS y BOSCH.

- *IsaGraft Workbench*

Al igual que nxtOne es una herramienta comercial desarrollada por la compañía ICE Triplex que pertenece a Rockwell automation. Esta herramienta soporta la IEC 61499 conjuntamente con la IEC 61131. Al igual que 4DIAC y nxtOne, tiene funcionalidades de debug y test y soporte oficial. Se ha utilizado en el desarrollo de muchas aplicaciones en componentes de diferentes vendedores. Uno de los grandes inconvenientes de esta herramienta es su incompatibilidad con el resto de plataformas de desarrollo (Garc & Castellanos, 2018).

- *IsaGraft Run Time*





A diferencia de las otras plataformas de ejecución, este runtime no se ha desarrollado con el perfil de compilación propuesto por Holobloc, que es el que se ha adoptado como estándar, sino que usa su propio perfil de compilación. En consecuencia, ISaGRAF Runtime sólo se puede utilizar desde ISaGRAF y no existe interoperabilidad con las otras plataformas de ejecución (Garc & Castellanos, 2018).

Por otro lado, tiene ciertas características de ejecución que no cumple estrictamente con la IEC 61499, entre otras cosas no utiliza el modelo de ejecución basado en eventos. Es decir, en lugar de ejecutar un bloque solamente cuando llega la señal de entrada, ISaGRAF runtime utiliza un

modelo más cercano a las subrutinas de 61131, y hace una verificación continua del estado de la entrada. Este comportamiento supone una diferencia de ejecución respecto a las otras plataformas.

2.2.3 Grupo KUKA (Keller Und Knappich Ausburg)

El grupo KUKA de Alemania es uno de los proveedores líderes en el mundo de robots y sistemas de producción automatizados. Los clientes de KUKA son en su mayoría de la industria en general. En el campo de la robótica mundial se encuentra dentro del Top 3 y en la industria automotriz es el número uno en Europa y Norteamérica. KUKA es la aplicación de la experiencia adquirida a lo largo de más de 100 años en la industria y 42 años en el campo robótico para desarrollar soluciones de automatización innovadoras y que hoy en día también sirve a otros segmentos como por ejemplo, robots para tecnología médica, industria solar y la industria aeroespacial, así como la investigación en universidades alrededor del mundo (Lozada, 2018). A continuación, se presenta la *figura 2-17* donde se puede observar ejemplos donde el grupo Kuka tiene presencia.

 <ul style="list-style-type: none">• Proceso de embotellamiento	 <ul style="list-style-type: none">• Proceso de empaquetado.
 <ul style="list-style-type: none">• Proceso de corte y suelda	 <ul style="list-style-type: none">• Proceso de ensamble

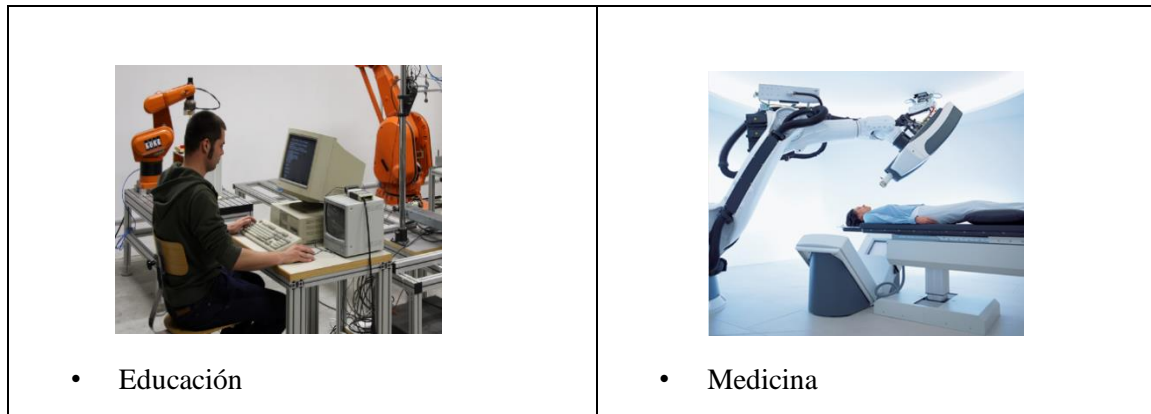


Figura 2-17. Ejemplos de procesos donde Kuka tiene presencia

Fuente: Lozada, 2018

2.2.3.1 Robótica KUKA

El robot KUKA es del tipo antropomórfico debido a que su movimiento se asemeja a los de un brazo humano. Permite realizar tareas repetitivas de una manera exacta y óptima (Mirelez-Delgado, Morales-Díaz, Ríos-Cabrera, & Pérez-Villeda, 2015).

2.2.3.2 Kuka Youbot

El Kuka Youbot es un sistema robótico abierto, ampliable y modular, especialmente desarrollado para fines de investigación con énfasis en la robótica. Es usado únicamente para aplicaciones de laboratorio mas no para aplicaciones industriales (Faber, Diaz, Eduardo, Rojas, & Morales, 2011).

El Youbot consiste en una base omnidireccional, un brazo con cinco grados de libertad (Degrees Of Freedom DOF) y una pinza de dos dedos. La base omnidireccional tiene una carga útil de 20 kg y está impulsada por cuatro ruedas omnidireccionales. La base del brazo (base frame) contiene la conexión de entrada de 24VDC y la de comunicación de datos RJ45. Las ruedas y las articulaciones de los brazos son accionadas por motores DC sin escobillas (también conocido como motores EC) con caja de engranajes incorporada, codificador relativo y cojinete de articulación. Todo el brazo en conjunto puede levantar 0.5 kg usando la pinza. Todos los motores (4 ruedas, 5 articulaciones del brazo, 2 dedos) se pueden controlar con EtherCAT. El nivel más bajo de comando es modulación por ancho de pulso (PWM). Los comandos de control de nivel superior son: control de corriente, control de velocidad y control de posición (Locomotec, 2011). La *figura 2-18*. indica sus principales componentes.

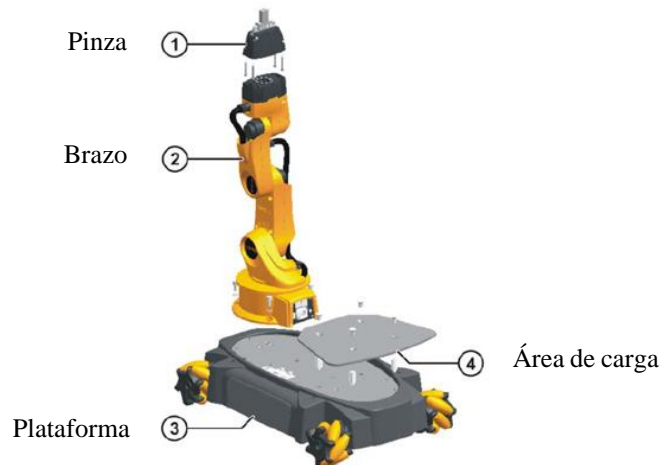


Figura 2-18. Componentes del Kuka Youbot
Fuente: Locomotec, 2011

2.3 Marco Conceptual

En esta sección se conceptualizan de manera resumida, los términos más sobresalientes utilizados durante el desarrollo de la presente investigación.

CLOUD AUTOMATION: Es un término amplio que se refiere a los procesos y herramientas que utiliza una organización para reducir los esfuerzos manuales asociados con el aprovisionamiento y la administración de cargas de trabajo de computación en la nube. Los equipos de TI pueden aplicar esto mediante el uso de diversas herramientas de automatización de software que se instalan directamente en la plataforma o software de virtualización y se controlan a través de una interfaz intuitiva (Boschetti et al., 2015).

CLOUD COMPUTING: Se encarga de obtener servicios de computación a través de la red, normalmente Internet permitiendo que una empresa solo utilice los recursos necesarios en el momento preciso, mejorando los costes y optimizando los presupuestos necesarios en una era tan cambiante (B Kehoe S Patil, 2014).

INDUSTRIA 4.0: Llamada también cuarta revolución industrial, es la tendencia a la automatización y el intercambio de los datos en las tecnologías de fabricación mediante tres elementos principales: el Internet de las Cosas, los Sistemas Cibernéticos Físicos y la Computación en la nube (Dumitrache, 2016).

INTERNET DE LAS COSAS (IoT): Es un sistema que permite enriquecer diferentes dispositivos con informática integrada y conectarlos usando tecnologías estándar. Esto permite que diferentes equipos se comuniquen e interactúen tanto entre ellos como con controladores más centralizados (Hermann, Pentek, & Otto, 2016).

INTERNET INDUSTRIAL DE LAS COSAS (IIoT): Es el uso de las tecnologías del Internet de las Cosas (IoT) en la manufactura (Wan et al., 2016). Se trata de una vasta cantidad de sistemas industriales conectados que se comunican y coordinan análisis de datos y acciones para mejorar el desempeño industrial, fomentando la optimización de la eficacia operativa y la producción industrial. El IIoT permite reducir el tiempo de inactividad de las máquinas y conseguir que los sistemas estén 100% disponibles. Manteniendo los principios del Internet of Things: conectando las máquinas a Internet y permitiendo una monitorización remota de los datos que cada dispositivo ofrece, entrando en juego términos como el Cloud Computing o el Big Data (Jayaram, 2016).

SISTEMA CIBER FÍSICO DE PRODUCCIÓN INDUSTRIAL (CPPs): Son percibidos como el eje fundamental para una nueva era de comunicación y colaboración en tiempo real basada en Internet entre los participantes en la cadena de valor, por ejemplo, dispositivos, sistemas, organizaciones y seres humanos. Los CPS industriales difuminan el tejido de los mundos cibernético (empresarial) y físico y ponen en marcha una era de colaboración a nivel de sistema e interacciones impulsadas por la información entre todas las partes interesadas de la cadena de valor (Colombo, 2017). Las características esenciales de los sistemas ciber-físicos son: la capacidad de relacionarse con los objetos físicos para monitorizar y/o controlar y; la utilización de la información disponible en el mundo virtual, pudiendo tener en algunos casos capacidad de aprender y evolucionar.

SMART FACTORY: Se trata de una fábrica que permite la comunicación entre los sistemas ciber físicos, el internet de las cosas y la computación en la nube para ayudar a personas y máquinas en la ejecución de sus tareas, haciendo los procesos de fabricación y producción más sencillos, ágiles y rentables (Cuevas, López, & García, 2012).

CAPÍTULO III

3. MARCO METODOLÓGICO

En este capítulo se presenta la metodología aplicada en el caso de estudio describiendo las fases de desarrollo del modelo de comunicación hasta llegar a su implementación.

3.1 Desarrollo de la metodología de la investigación

En esta investigación se desarrollaron contenidos referentes a la forma de diseño y programación de bloques de función bajo la norma IEC 61499 a través de los entornos de programación y ejecución 4DIAC IDE y FORTE, con la finalidad de generar una programación en un lenguaje compatible al API de la plataforma del Kuka Youbot que permita la comunicación hacia un servidor MQTT.

El enfoque fue de tipo *experimental* debido a que se verificó si la hipótesis planteada tenía una validez en el marco de la investigación propuesta.

Para cumplir con los objetivos trazados, se llevaron a cabo las siguientes tareas de investigación:

- Identificación de los parámetros a ser transmitidos desde el robot kuka Youbot.
- Búsqueda de información para el desarrollo de la programación de bloques de función bajo la norma IEC 61499.
- Diseño de la arquitectura del modelo de comunicación a través de los bloques de función, conforme al estándar IEC 61499.
- Integración del modelo de comunicación a una plataforma cloud.
- Pruebas de transmisión de datos.
- Valoración de resultados.

3.1.1 Alcance de la investigación

Esta investigación hace referencia exclusivamente a la creación de un modelo de comunicación MQTT, cuyo objetivo es transmitir y recibir datos generados desde un robot Kuka Youbot hacia un bróker en la nube.

Dado que se trata de una aplicación dentro de IIoT, el uso del protocolo MQTT es determinante, ya que es uno de los principales pilares de IoT por su sencillez y ligereza, ambos, condicionantes importantes para los dispositivos industriales que generalmente tienen limitaciones de potencia, consumo y ancho de banda.

3.1.2 Técnicas de recolección de datos

Para ello se basó en el cuadro de operacionalización de variables, que se muestra en la *tabla 3-1*.

Tabla 3- 1: Operacionalización de variables

V.D: Protocolo MQTT basado en la norma IEC 61499			
Conceptualización	Dimensión	Indicadores	Técnicas e Instrumentos
Se trata de un protocolo de comunicación tipo publish/subscribe, que comunica dispositivos dentro del IIoT, basado en la programación de bloques funcionales de acuerdo a la norma de automatización IEC 61499	Datos transmitidos	Transmisión de datos	Observación, revisión bibliográfica, diseño y programación de bloques funcionales
V.I: Integración del robot kuka YUBOT hacia la nube			
Conceptualización	Dimensión	Indicadores	Técnicas e Instrumentos
Es un robot manipulador móvil, cuyas señales que genera serán transmitidas hacia la nube	Datos recibidos	Validación de datos enviados vs datos recibidos	Observación, revisión bibliográfica, diseño y programación de bloques funcionales

Fuente: Zambrano, 2019

Realizado por: Tatiana Zambrano

3.1.3 Instrumentos para la recolección de datos

Básicamente se utilizaron los siguientes:

- *Experimentos:* se definió que es el instrumento más preciso para la recopilación de información debido a que el tema de estudio tiene muy pocas fuentes informativas, y el desarrollo del modelo es netamente experimental.
- *Técnicas clásicas de recopilación de información (libros, artículos científicos, etc.):* las cuales permitieron dar nuevas aristas a la investigación, el apoyo en estas fuentes informativas fue importante para el desarrollo del estudio del arte, lo cual permitió

avizorar de una forma anticipada el modelo planteado para el presente proyecto de titulación.

3.1.4 Instrumentos para procesar datos recopilados

- *Excel*: Utilizado para realizar una tabulación organizada de los datos recolectados.
- *SPSS*: Utilizado para el análisis estadístico de los datos obtenidos en la investigación.

3.2 Caso de estudio

El caso de estudio propuesto describe el diseño de un modelo de comunicación utilizando el protocolo MQTT a través de la programación de bloques de función introducido por la norma IEC 61499, que permite la transmisión de datos generados desde un robot kuka youbot hacia una plataforma cloud y desde esta última hacia el robot kuka Youbot.

El presente proyecto se desprende de un proyecto de investigación de la Universidad Técnica de Ambato denominado *CLOUD AUTOMATION PARA ROBOTS MANIPULADORES MÓVILES Y SU USO EN SMART FACTORIES*, proyecto que comprende tres desarrollos:

- *Tesis 1*: Desarrollo de un sistema distribuido bajo la norma IEC 61499 para control de robot Kuka modelo Youbot.
- ***Tesis 2, que se refiere al presente trabajo: Uso del protocolo MQTT basado en la norma IEC 61499 para la integración de un robot Kuka Youbot hacia la nube.***
- *Tesis 3*: Desarrollo de un entorno de simulación virtual para la operación de un robot Kuka Youbot desde la nube.

El objetivo principal del proyecto fue articular los tres temas mencionados aplicando conceptos de cloud automation para el control y comunicación de robots manipuladores móviles.

Como se puede observar en la *figura 3-1*, esta tesis fue la encargada de establecer la comunicación de datos entre los bloques de función de control de movimientos del robot Kuka Youbot (desarrollados en la *tesis 1*), y la aplicación virtual de la operación de éste en la nube (a desarrollarse en la *tesis 3*). Para ello se construyó un modelo de comunicación MQTT de estándar abierto basado en la utilización de bloques de función de la norma IEC 61499.

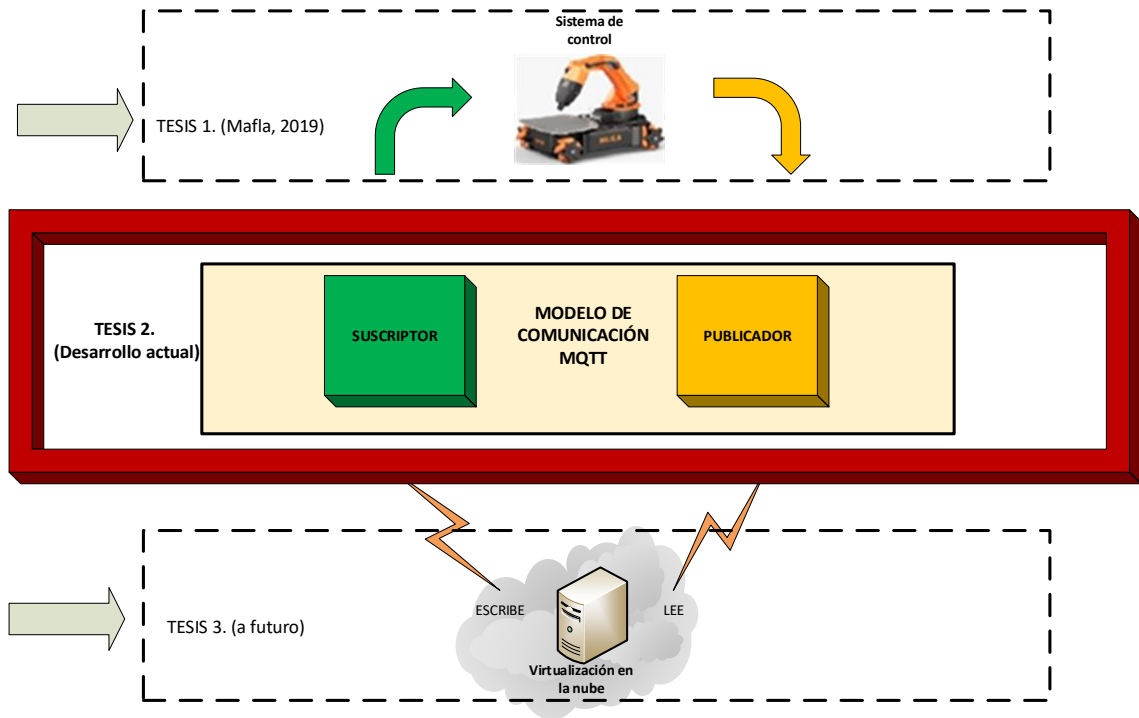


Figura 3-1. Caso de estudio
Fuente: Zambrano, 2019

Las fases de desarrollo que se siguieron para la creación del modelo de comunicación se muestran en la *figura 3-2*:

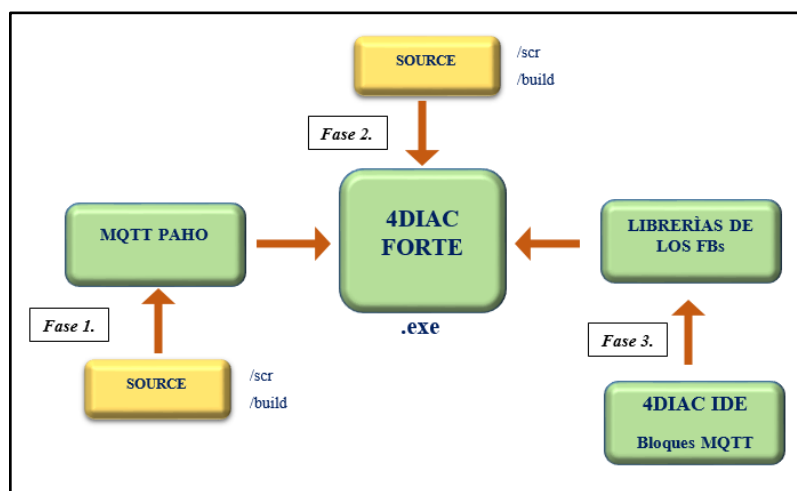


Figura 3-2. Fases de desarrollo del modelo de comunicación MQTT
Fuente: Zambrano, 2019

3.2.1 Fase 1. Creación de librerías del MQTT PAHO

MQTT PAHO es una librería desarrollada por ECLIPSE² que permite la comunicación hacia un servidor MQTT, por lo que fue importante incluirlo dentro del FORTE final de la aplicación - archivo ejecutable del modelo- ya que sin éste no se lograría publicar ni suscribir algún tópico. El procedimiento de generación de código y compilación de archivos se describe en el *Anexo 1*.

3.2.2 Fase 2. Creación de FORTE

En esta fase se creó la arquitectura base del ejecutable de la aplicación del modelo, es decir el FORTE, usando su archivo fuente (*source*) y las librerías creadas en la *fase 1*. Además sobre éste se fueron añadiendo todas las librerías desarrolladas en la siguiente fase. El procedimiento para lograrlo se indica en el *Anexo 2*.

3.2.3 Fase 3. Creación de Bloques de función

Si bien es cierto que la biblioteca de 4DIAC IDE posee los bloques Publicador y Suscriptor necesarios para una comunicación MQTT, también es cierto que éstos no contienen la funcionalidad requerida para este proyecto, por lo que se tuvo que crear FBs propios que incluyan los parámetros necesarios que respondan a los requerimientos de la presente investigación. A continuación, se describe paso a paso lo desarrollado para lograr la creación de los FBs así como también la carga en 4DIAC FORTE de los nuevos bloques.

3.2.3.1 Bloque Suscriptor

1. Se creó un nuevo sistema bajo el nombre de *youbot_mqtt*, donde se guardaron todos los bloques construidos para la aplicación. El primer bloque de función creado fue de tipo compuesto y se lo llamó *youbot_subscribe*, la *figura 3-3* muestra su creación.

² Infraestructura de código abierto para sistemas distribuidos de medición, control y transmisión de procesos industriales, basados en la norma IEC 61499.

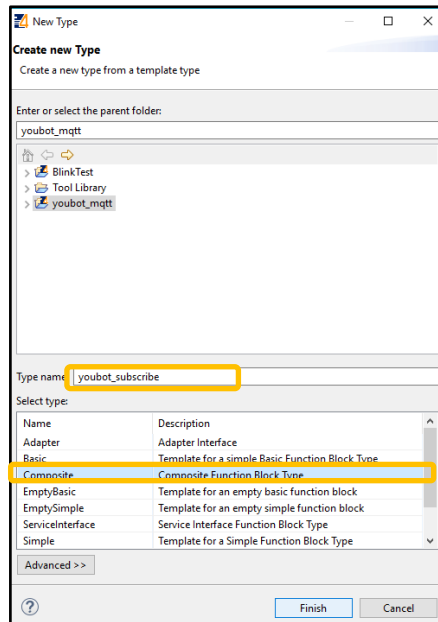


Figura 3-3. Creación del FB youbot_subscribe
Fuente: Zambrano, 2019

2. El FB creado apareció por un lado en la biblioteca de tipos (*Type Library*) del sistema *mqtt_youbot*, y por otro en el área de edición central donde también se encontraban las pestañas de desarrollo correspondientes al nuevo bloque. Así se lo puede ver en la *figura 3-4*.

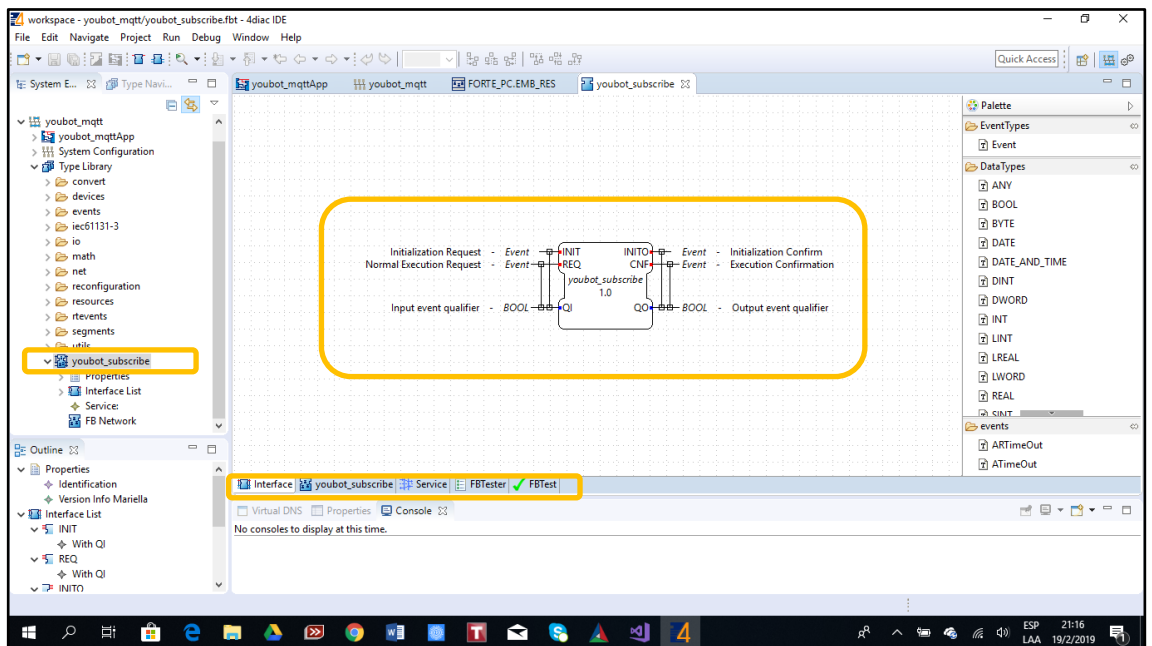


Figura 3-4. Interfaz inicial del youbot_subscribe
Fuente: Zambrano, 2019

3. A continuación se muestra la *figura 3-5* con las nuevas entradas y salidas de datos del bloque *youbot_subscribe* para la presente aplicación.

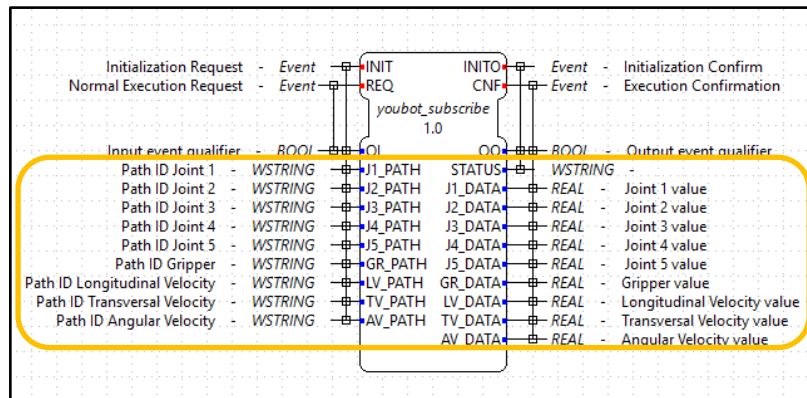


Figura 3-5. Entradas y salidas del *youbot_subscribe*
Fuente: Zambrano, 2019

Las entradas de datos fueron de tipo *WSTRING* y correspondieron a los *PATH ID* referidos al bróker (servidor) MQTT al que la aplicación estaba apuntando en la nube, éstos fueron configurados de acuerdo al siguiente formato: *raw[.].mqtt[tcp://ip:port, cientID, topic]*. Donde:

- *ip*, correspondió al nombre del bróker MQTT con el que se comunicó a la nube, para este caso fue el *iot.eclipse.org*. Cabe mencionar que en el mercado existen muchos brókers libres y privados a los que se puede acceder, pero se eligió el mencionado debido a que es gratuito y se mantiene en línea gran cantidad de tiempo.
- *port*, correspondió al número de puerto del servidor conectado que fue el *1883* asignado según IANA (Internet Assigned Numbers Authority).
- *cientID*, creado para la aplicación y para éste se lo llamó *4Diac*.
- *topic*, correspondió al tema sobre el cual se suscribieron y publicaron los protagonistas de la comunicación. Debido a que existieron nueve (9) parámetros controlados en el Kuka Youbot, según la información obtenida de (Mafla, 2019), se crearon nueve (9) tópicos o canales de comunicación correspondientes a cada uno de ellos, que fueron jerarquerizados, indicando como tópico principal *youbot* y como subtópicos los nueve controles repartidos en la base y brazo del Kuka Youbot, así:

- *youbot/arm/j1 ==>j1, junta 1 del brazo del Youbot*

- *youbot/arm/j2 ==>j2, junta 2 del brazo del Youbot*
- *youbot/arm/j3 ==>j3, junta 3 del brazo del Youbot*
- *youbot/arm/j4 ==>j4, junta 4 del brazo del Youbot*
- *youbot/arm/j5 ==>j5, junta 5 del brazo del Youbot*
- *youbot/arm/GR==>GR, gripper del brazo del Youbot*
- *youbot/base/LV==>LV, desplazamiento longitudinal de la base del Youbot*
- *youbot/base/TV==>TV, desplazamiento transversal de la base del Youbot*
- *youbot/base/AV==>AV, desplazamiento angular de la base del Youbot*

Entonces, las entradas de datos del bloque *youbot_subscribe* quedaron de la siguiente manera:

- *raw[.].mqtt[tcp://iot.eclipse.org:1883, 4Diac, youbot/arm/j1]*
- *raw[.].mqtt[tcp://iot.eclipse.org:1883, 4Diac, youbot/arm/j2]*
- *raw[.].mqtt[tcp://iot.eclipse.org:1883, 4Diac, youbot/arm/j3]*
- *raw[.].mqtt[tcp://iot.eclipse.org:1883, 4Diac, youbot/arm/j4]*
- *raw[.].mqtt[tcp://iot.eclipse.org:1883, 4Diac, youbot/arm/j5]*
- *raw[.].mqtt[tcp://iot.eclipse.org:1883, 4Diac, youbot/arm/GR]*
- *raw[.].mqtt[tcp://iot.eclipse.org:1883, 4Diac, youbot/base/LV]*
- *raw[.].mqtt[tcp://iot.eclipse.org:1883, 4Diac, youbot/base/AV]*
- *raw[.].mqtt[tcp://iot.eclipse.org:1883, 4Diac, youbot/base/AV]*

Por otro lado, las salidas del bloque *youbot_subscribe* fueron de tipo *REAL*, ya que los parámetros de entrada del bloque *youbot_controller* al cual se conectó fueron de ese tipo, es decir solo aceptaba números reales. Se debe mencionar que los datos recibidos por el bloque *youbot_subscribe* correspondieron a los enviados desde el Kuka Youbot y éstos fueron numéricos de hasta tres (3) dígitos, de acuerdo al sistema de control desarrollado por Mafla (2019).

4. La conexión entre eventos y datos de entrada así como entre eventos y datos de salida del bloque *youbot_subscribe* se dio de la siguiente manera:

- Las entradas de datos del FB fueron conectadas al evento *INIT*, el cual habilitó el bloque, con esto se aseguró que las entradas inicialicen la conexión con el servidor remoto.
 - En cuanto al evento *REQ* encargado de ejecutar lo programado en el bloque, fue conectado a otro evento de entrada cuyo índice lo forzó a estar siempre activo, el evento referido fue el *QI* que siempre tomó el valor booleano de 1, indicando que está conectado siempre al servicio.
 - De igual forma, todas las salidas de datos del bloque fueron conectadas al evento de salida *CNF* el cual aseguró que existan siempre nuevos datos disponibles.
 - El evento de salida *QO* encargado de controlar la ejecución de datos del bloque fue conectado al evento de salida *INITO* para establecer una nueva conexión.
5. Como el bloque creado *youbot_subscribe* fue de tipo compuesto, internamente se construyó una arquitectura con la funcionalidad requerida en la aplicación, este FB encapsuló a varios bloques de función simples del tipo *subscribe_1* y *f_string_to_real*, la *figura 3-6* muestra lo mencionado.

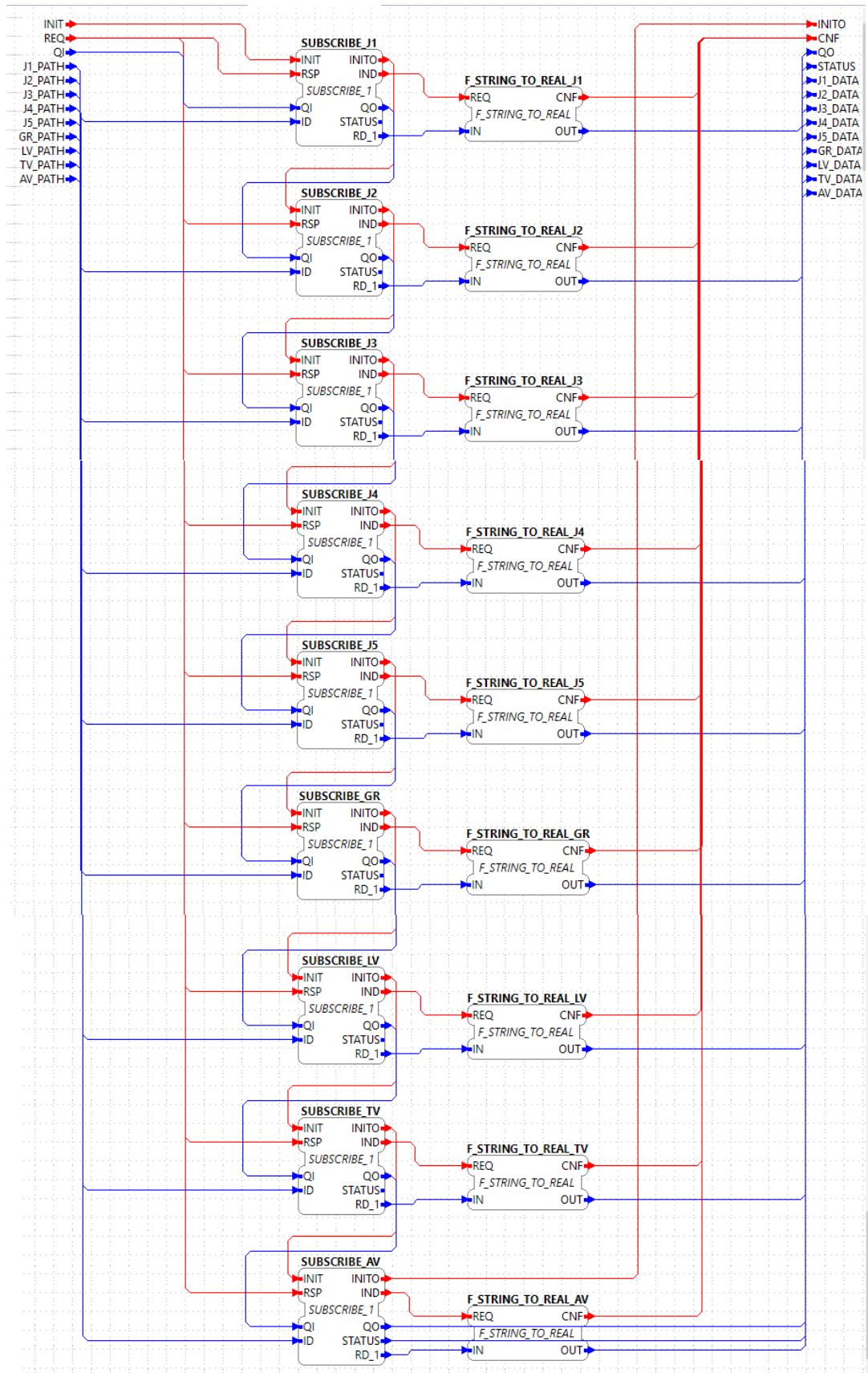


Figura 3-6. Arquitectura interna del yobot_subscribe
Fuente: Zambrano, 2019

Como se puede observar cada entrada del bloque *youbot_subscribe* conectó básicamente dos FBs: uno *suscribe_1* cuyas entradas fueron de tipo *WSTRING* y otro FB que convirtió datos tipo *string* a *real*. Cada par de FBs se conectaron en cascada hacia los siguientes bloques, correspondientes al resto de entradas y salidas del *youbot_subscribe*.

6. Para poder usar el FB desarrollado fue preciso exportarlo y generar así los archivos *.cpp* y *.h* a ser compilados. La *figura 3-7* muestra el wizard de exportación.

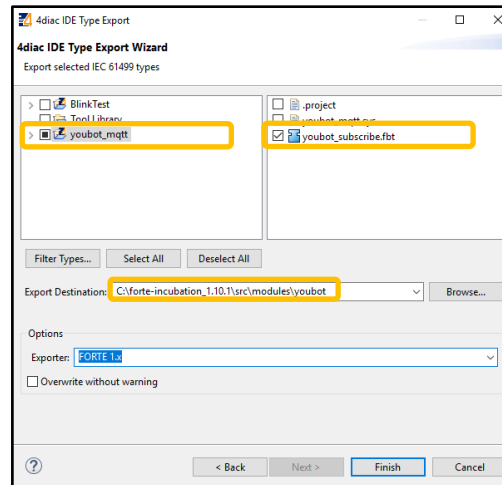


Figura 3-7. Wizard de exportación
Fuente: Zambrano, 2019

7. Con el FB exportado se generaron de manera automática en el directorio contenedor de FORTE *C:\forte-incubation_1.10.1\src\modules\youbot* dos archivos compatibles con el lenguaje de programación C++, el *youbot_subscribe.cpp* y el *youbot_subscribe.h*. En el primero se escribieron los algoritmos de programación para los datos de entrada y salida, además del establecimiento del flujo de eventos para la ejecución de éstos; y en el último se generaron las entradas y salidas del bloque a manera de variables declaradas, así como las librerías a utilizar. La *figura 3-8* indica lo mencionado.

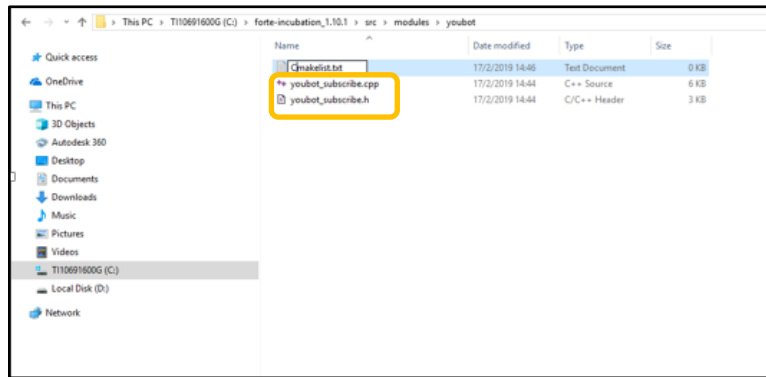


Figura 3-8. Creación de archivos `youbot_subscribe.cpp` y `youbot_subscribe.h`
Fuente: Zambrano, 2019

8. Además de los dos archivos `.cpp` y `.h` se creó un archivo `CMakeList.txt` en donde se agregó las siguientes líneas:

```
forte_add_module(youbot "youbot mqtt subscribe")
forte_add_sourcefile_hcpp(youbot_subscribe)
```

Estas líneas correspondieron a la descripción del módulo desarrollado y todos sus recursos.

3.2.3.2 *Bloque Publicador*

1. Al igual que en el apartado 3.2.3.1, se procedió con la creación de un bloque de función compuesto pero esta vez se trató de un publicador creado bajo el nombre de `youbot_publish` que fue cargado dentro del sistema `mqtt_youbot` como se observa en la figura 3-9.

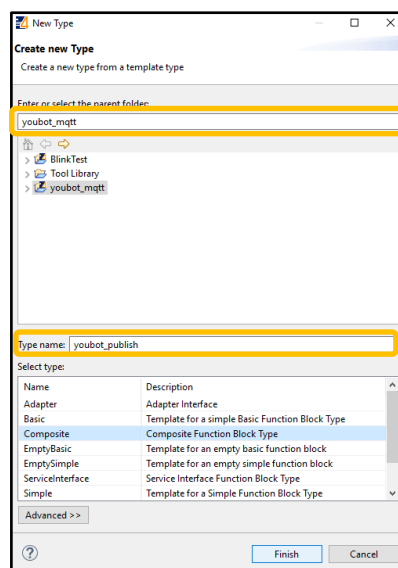


Figura 3-9. Creación del FB `youbot_publish`
Fuente: Zambrano, 2019

2. Con el bloque creado indicado en la *figura 3-10*, se procedió a añadir las entradas y salidas de datos de acuerdo al requerimiento del modelo de comunicación, obteniendo un bloque de función como el mostrado en la *figura 3-11*.

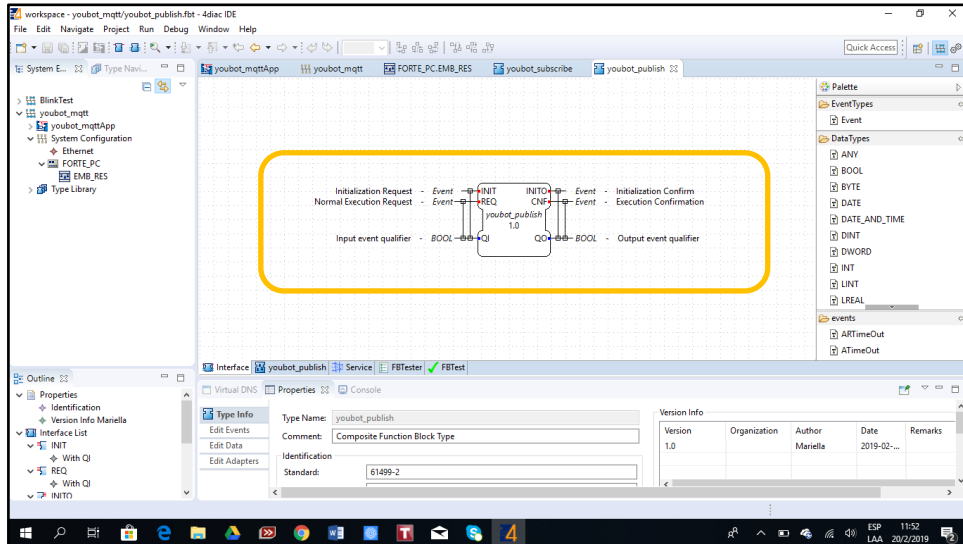


Figura 3-10. Interfaz inicial del youbot_publish

Fuente: Zambrano, 2019

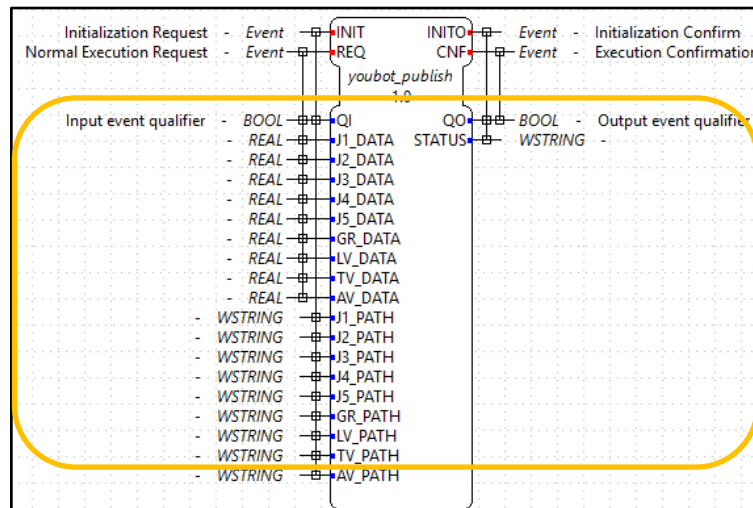


Figura 3-11. Entradas y salidas del youbot_publish

Fuente: Zambrano, 2019

Como se puede observar en la *figura 3-11* los datos de entrada fueron de tipo *REAL* y de tipo *WSTRING*, los primeros fueron creados para publicar números reales referentes a la información generada sobre los movimientos de la base y brazo del kuka Youbot, y los segundos para escribir los *PATHS ID* del servidor remoto al que se deseaba conectar.

La forma de conexión fue la siguiente:

- Todos los datos de tipo *REAL* fueron conectados al evento *REQ*, ya que éstos necesitaban activarse y actualizarse cada vez que se ejecutaba un proceso en el bloque referente a los movimientos del robot (Mafla, 2019).
 - Los datos de entrada tipo *WSTRING* en cambio fueron conectados al evento *INIT*, ya que solo necesitaban conectarse una sola vez al servidor remoto y no estar en constante actualización.
3. Internamente el bloque *youbot_publish* utilizó un bloque de conversión *f_real_to_string* y un bloque *publish_1* para cada una de sus entradas de datos, excluyendo las entradas correspondientes a los *PATHS ID*. La forma de conexión se indica en la *figura 3-12*.

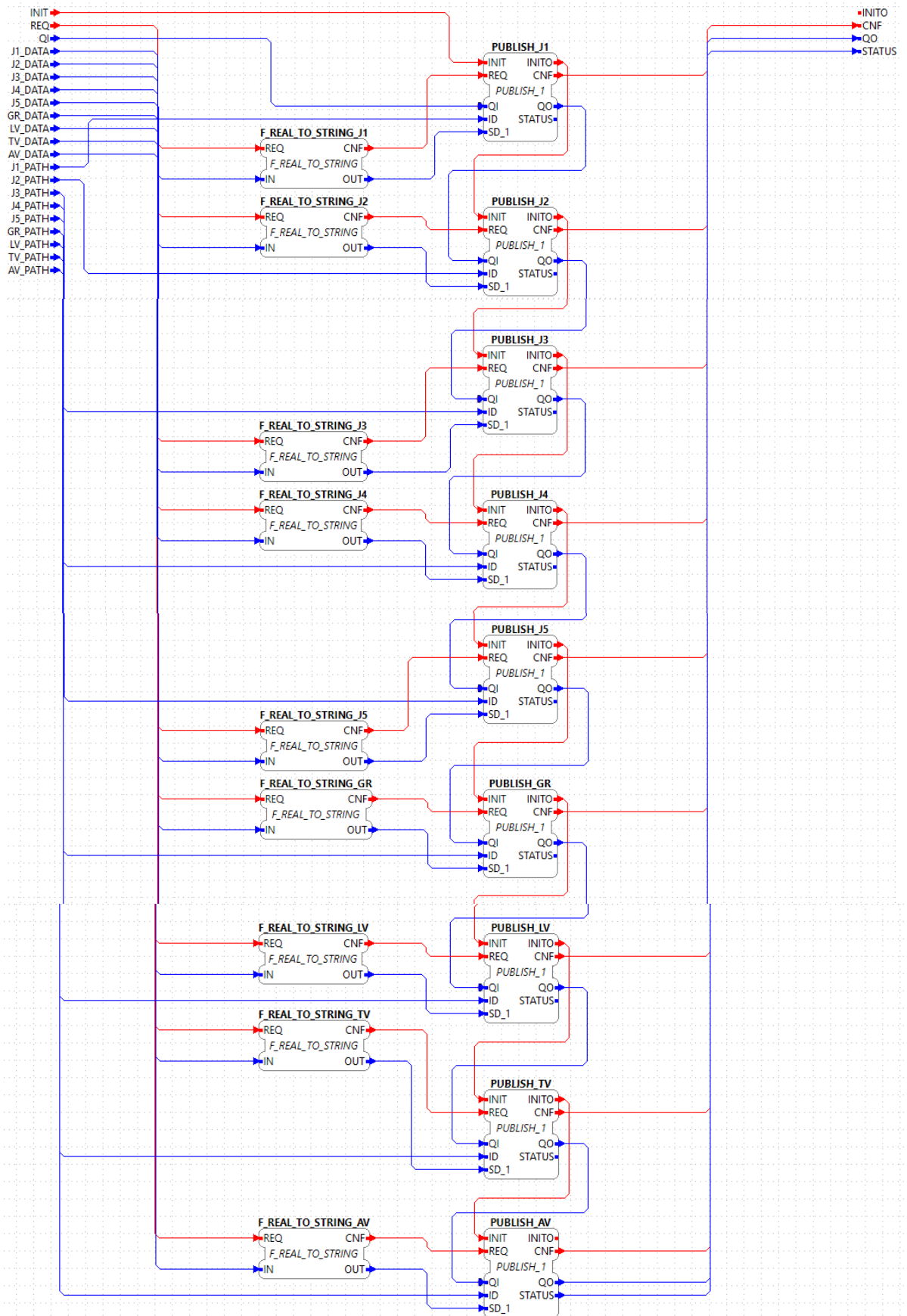


Figura 3-12. Arquitectura interna del youbot_publish

Fuente: Zambrano, 2019

- El bloque creado se exportó, generando en el directorio `C:\forte-incubation_1.10.1\src\modules\youbot` los archivos `youbot_publish.cpp` y `youbot_publish.h` a ser compilados.
- De igual forma dentro del archivo `CMakeList.txt` se agregaron las líneas correspondientes a la descripción del módulo desarrollado y todos sus recursos:

```
forte_add_module(youbot "youbot mqtt publish")
forte_add_sourcefile_hcpp(youbot_publish)
```

3.2.3.3 Bloque Controller

- Finalmente se creó un tercer bloque compuesto llamado `youbot_controller`, el procedimiento para su creación fue similar al descrito en los apartados 3.2.3.1 y 3.2.3.2.
- Las entradas y salidas de datos para este FB se lo muestra en la *figura 3-13*.

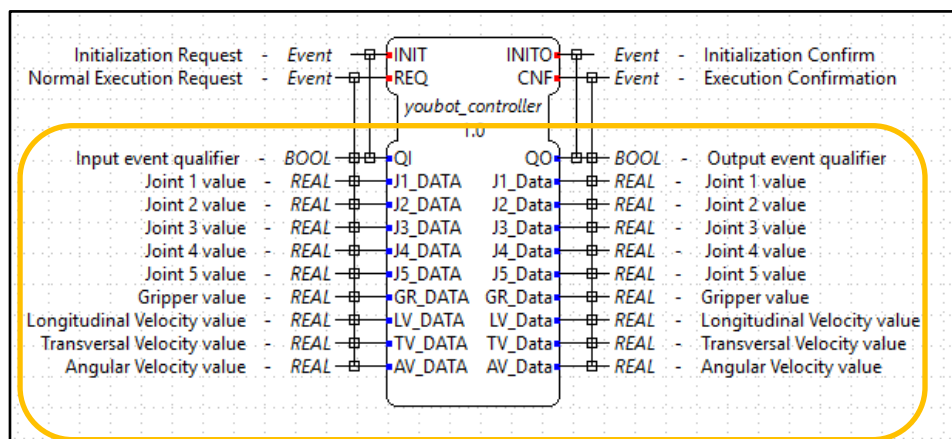


Figura 3-13. Entradas y salidas del FB `youbot_controller`
Fuente: Zambrano, 2019

Como se puede observar los datos de entrada y salida fueron de tipo *REAL* ya que enlazaban datos numéricos entre los bloques suscriptor y publicador. Este bloque fue creado para controlar el flujo de datos enviados, ya que de otra forma los datos entraban en un bucle infinito de envío hasta que un dato nuevo ingrese.

La forma de conexión fue la siguiente:

- Todos los datos de entrada tipo *REAL* fueron conectados al evento *REQ*, para que puedan activarse y actualizarse cada vez que se ejecutara algún proceso.

- Los datos de salida tipo *REAL* fueron conectados al evento *CNF*, para asegurar que existan datos disponibles.

3. Internamente el bloque *youbot_controller* utilizó bloques de conversión *f_real_to_string* y *f_string_to_real*, *f_not*, *f_eq*, *f_permit*, tal como se muestra en la *figura 3-14*.

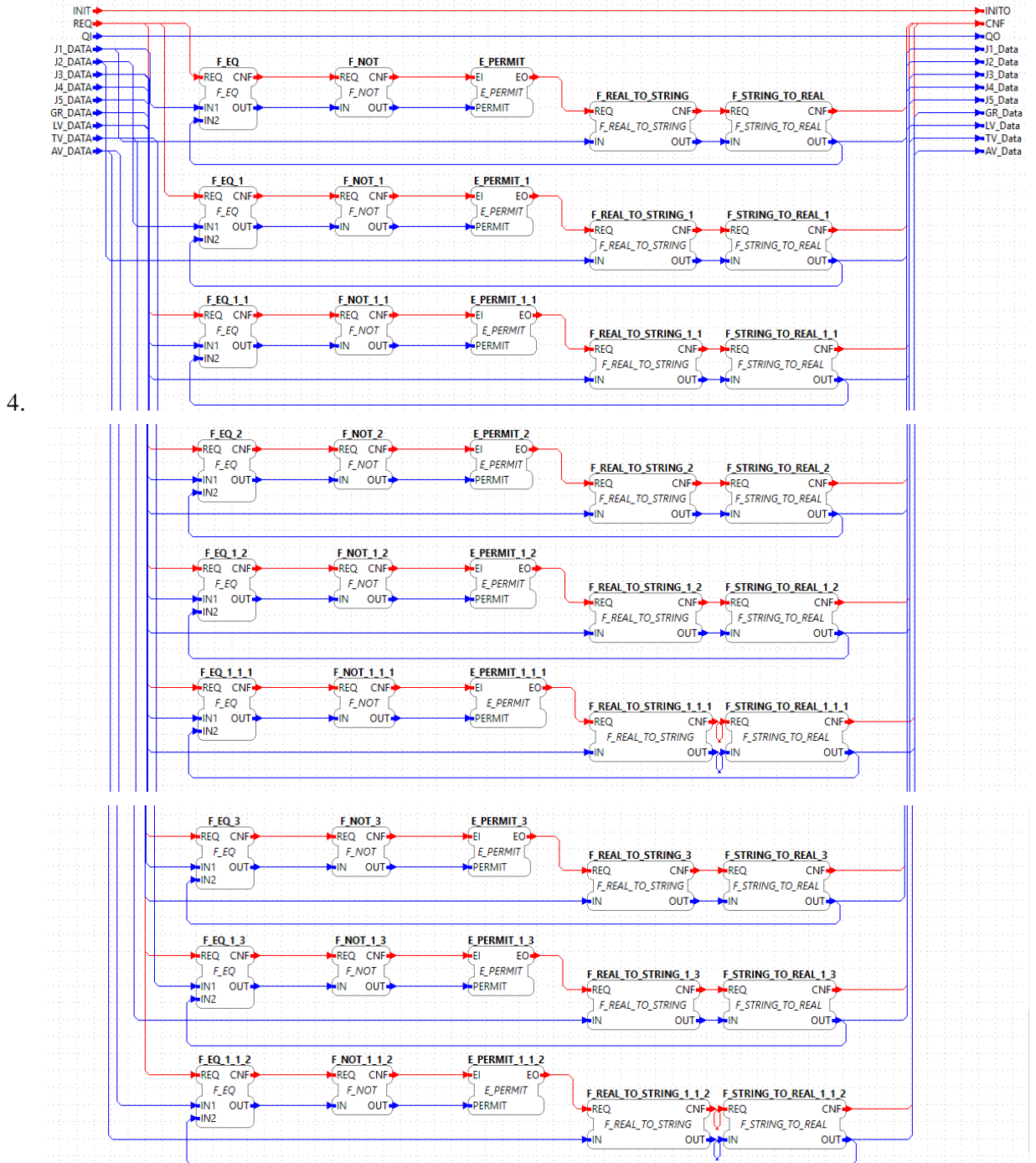


Figura 3-14. Arquitectura interna del *youbot_controller*
Fuente: Zambrano, 2019

- El bloque creado se exportó, generando en el directorio `C:\forte-incubation_1.10.1\src\modules\youbot` los archivos `youbot_controller.cpp` y `youbot_controller.h` a ser compilados.
- De igual forma dentro del archivo `CMakeList.txt` se agregaron las líneas correspondientes a la descripción del módulo desarrollado y todos sus recursos:


```
forte_add_module(youbot "youbot mqtt controller")
forte_add_sourcefile_hcpp(youbot_controller)
```

3.2.3.4 Carga de FBs en 4DIAC FORTE

- Como se cargaron tres nuevos módulos (`youbot_subscribe`, `youbot_publish`, `youbot_controller`) a la carpeta contenedora de `forte` se procedió nuevamente a generar mediante el CMake archivos que puedan ser compilados con Visual Studio, y obtener así el `forte` final de la aplicación. La *figura 3-15* indica el entorno de CMake que genera el código y la *figura 3-16* muestra la plataforma de Visual Studio que compila dicho archivo.

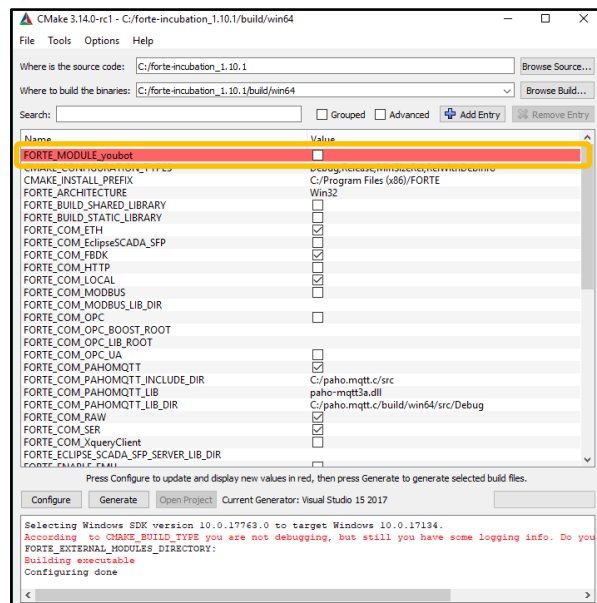


Figura 3-15. Carga de los módulos creados
Fuente: Zambrano, 2019

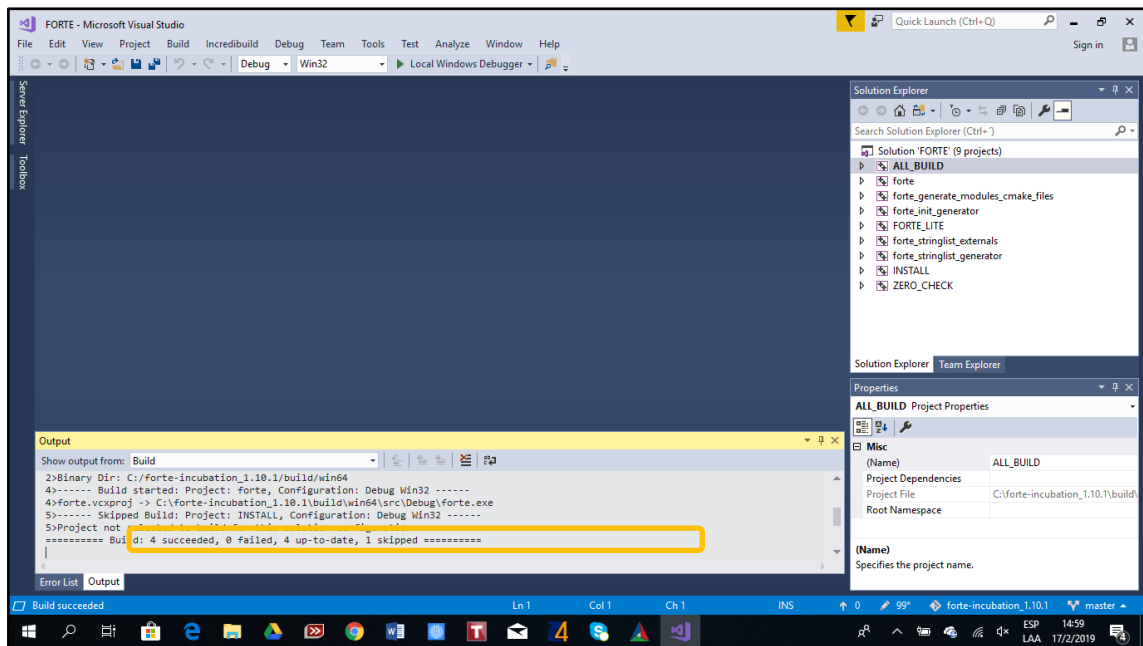


Figura 3-16. Compilación final del forte de la aplicación

Fuente: Zambrano, 2019

3.2.4 Implementación del modelo

El modelo de comunicación MQTT se estableció mediante la creación de tres bloques de función: *youbot_subscribe*, *youbot_publish* y *youbot_controller* que fueron conectados como se indica en la figura 3-17.

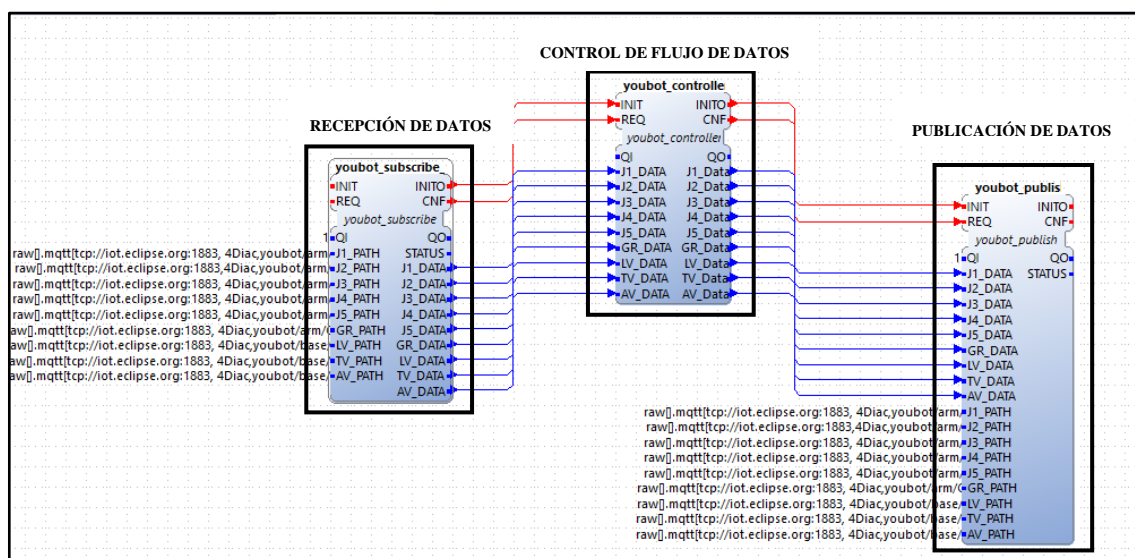


Figura 3-17. Implementación del modelo

Fuente: Zambrano, 2019

El funcionamiento del modelo inicia en el *youbot_subscribe* el cual escucha a través de sus entradas (J1_PATH, J2_PATH,.....AV_PATH) los datos enviados desde el bróker MQTT en la nube, mientras que en sus salidas (J1_DATA, J2_DATA,.....AV_DATA) se reflejan los datos recibidos que deben ejecutarse en el *youbot_controller* de acuerdo al tópic de comunicación utilizado que identifica las partes que se mueven en el robot.

El *youbot_controller* a su vez escribe los datos recibidos a través de las entradas del *youbot_publish* (J1_DATA, J2_DATA,.....AV_DATA) de acuerdo al tópic, y éste último publica los datos nuevamente en el bróker MQTT a través de los pines de comunicación correspondientes (J1_PATH, J2_PATH,.....AV_PATH).

De esta forma el modelo de comunicación funciona de manera cíclica, enviando y recibiendo datos desde y hacia la nube, transmitiendo así la información sobre los movimientos generados en el Kuka Youbot, tal como se indica en la *figura 3-18*.

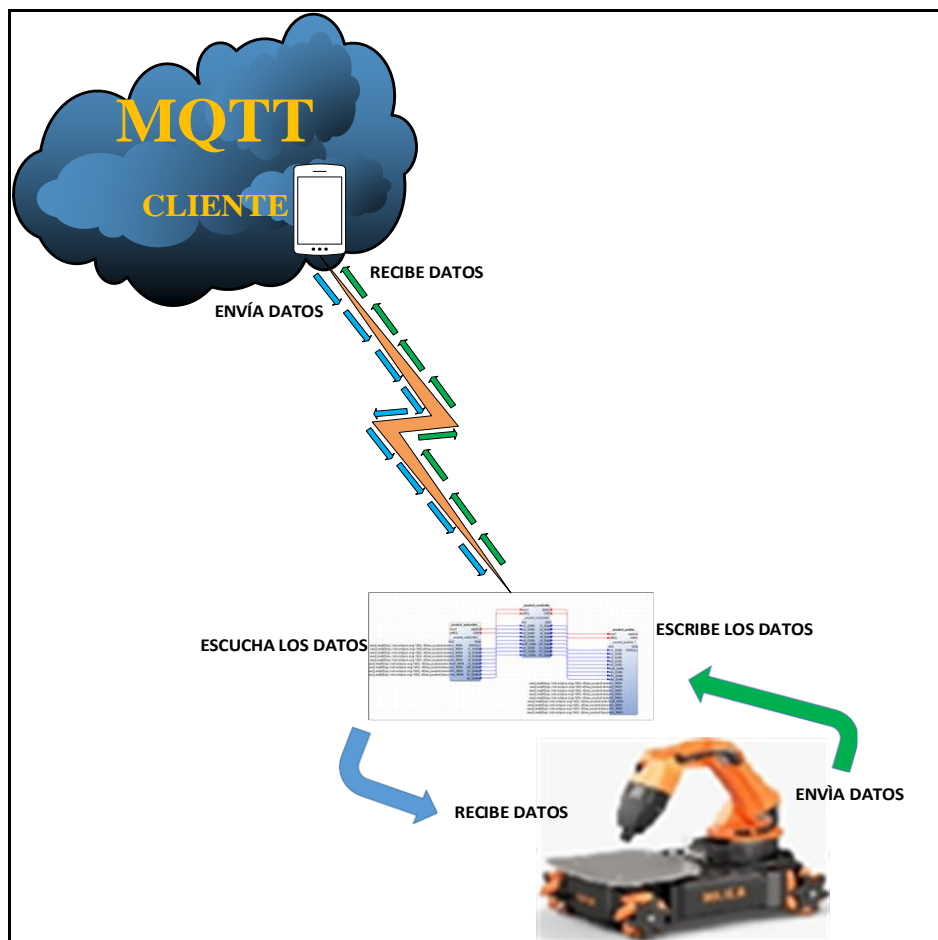


Figura 3-18. Modo de operación del modelo
Fuente: Zambrano, 2019

CAPÍTULO IV

4. RESULTADOS Y DISCUSIONES

En este capítulo se describen y analizan los resultados alcanzados en la investigación, así como las conclusiones a las que se llega una vez que se realizaron diferentes pruebas para comprobar la hipótesis planteada.

4.1 Pruebas de funcionamiento del sistema

Para validar el funcionamiento del sistema de comunicación creado, se probó la transmisión y recepción de datos, enviando datos numéricos de hasta tres dígitos (que son los que genera el sistema de control de movimientos del Kuka Youbot) desde una terminal cloud MQTT hacia el *forte* de la aplicación dispuesta en un ordenador, que simula la presencia del robot. Leídos los datos en el *forte*, se enviaron de regreso hacia la terminal MQTT para comprobar así su recepción.



Figura 4-1. Pruebas de funcionamiento del modelo
Fuente: Zambrano, 2019

La figura 4-1 evidencia el funcionamiento de los FBs, donde el bloque *youbot_subscribe* inicia el proceso recibiendo datos de tipo real de acuerdo al tópic enviado desde la terminal MQTT y a través de sus salidas lo envía hacia el *youbot_controller* el cual controla el flujo de datos y escribe sobre las entradas del *youbot_publish* que se encuentra apuntando al bróker con el que se conecta para realizar el envío de datos.

Además, en la figura 4-1 se puede observar las dos etapas principales que cumple el modelo MQTT, esto es, la *suscripción* y *publicación* de datos. En el ejemplo mostrado en dicha figura, se envía un dato que corresponde al número “5,4” desde la terminal MQTT hacia el modelo de comunicación, dicho dato ingresa al bloque de función *youbot_subscribe* a través de la entrada *raw[.mqtt[tc://iot.eclipse.org:1883,4Diac,youbot/arm/j1]* que corresponde al tópic *youbot/arm/j1* al cual está suscrito el cliente MQTT desde donde se envió el dato. De esa manera se observa cómo el dato es recibido dentro de la aplicación y como guarda una correspondencia bloque a bloque respecto al tópic.

Por otro lado, ya en la etapa de *publicación*, se puede observar cómo el mismo dato “5,4” es escrito en la entrada del bloque *youbot_publish* (J1_DATA) y enviado a través del pin de comunicación correspondiente al tópico *youbot/arm/j1* sobre el cual se desea publicar el dato. Dicho dato se lo puede ver ya publicado en la terminal MQTT, y así se comprueba su recepción.

4.2 Resultados obtenidos en la comunicación de datos

Lo que se valoró de manera cuantitativa fue el número de veces que se enviaron datos desde la terminal MQTT en la nube hacia el *forte* de la aplicación, y el número de veces que se recibieron datos en la terminal MQTT enviados desde la aplicación.

Los datos para realizar las pruebas fueron enviados de acuerdo a los nueve (9) tópicos que se crearon, y ya que se trató de un estudio de carácter experimental, la cantidad de envíos fue aleatoria. Como la distribución de datos obtenidos no fue sesgada ni con observaciones atípicas se consideró trabajar con 30 mediciones por cada canal de comunicación, es decir, por cada tópico.

A partir de esto, se decidió aplicar la prueba *Q de Cochran* trabajando con un nivel de significancia del 5%, para comprobar la hipótesis de investigación que afirma que a través de la programación de bloques de función se puede construir un modelo de comunicación que logre enviar y recibir datos generados desde el kuka Youbot hacia una plataforma cloud utilizando el protocolo MQTT.

Q de Cochran es una prueba estadística no paramétrica que verifica si k tratamientos tienen efectos idénticos (Anderson, 2011), es decir, sirve para comprobar la igualdad de varias muestras relacionadas en una variable dicotómica, esto es, cuando la variable de respuesta puede tomar solo dos resultados posibles, codificados como 0 y 1. Donde, para la presente investigación, el 0 representa el fracaso del envío y/o recepción del dato y el 1 representa el éxito del envío y/o recepción del mismo dato.

A continuación, se presenta la *Tabla 4-1* que muestra la cantidad de envíos que se realizaron desde el cliente MQTT hacia el *forte* de la aplicación, para comprobar la etapa de *suscripción del modelo MQTT*.

Tabla 4- 1: Número de envíos desde el cliente MQTT

	NÚMERO DE ENVÍOS DESDE EL CLIENTE MQTT	
	ÉXITO	FRACASO
Tópico 1 (<i>youbot/arm/j1</i>)	30	0
Tópico 2 (<i>youbot/arm/j2</i>)	30	0
Tópico 3 (<i>youbot/arm/j3</i>)	30	0
Tópico 4 (<i>youbot/arm/j4</i>)	30	0
Tópico 5 (<i>youbot/arm/j5</i>)	29	1
Tópico 6 (<i>youbot/arm/GR</i>)	30	0
Tópico 7 (<i>youbot/base/LV</i>)	30	0
Tópico 8 (<i>youbot/base/TV</i>)	30	0
Tópico 9 (<i>youbot/base/AV</i>)	30	0
TOTAL	269	1

Fuente: Zambrano, 2019

Realizado por: Tatiana Zambrano

La *figura 4-2* describe de manera estadística el número de envíos exitosos y de fracaso que se realizaron desde el cliente MQTT a la aplicación a través de los diferentes tópicos de comunicación. Para lograrlo se trabajó con los datos de la *Tabla 4-1*.

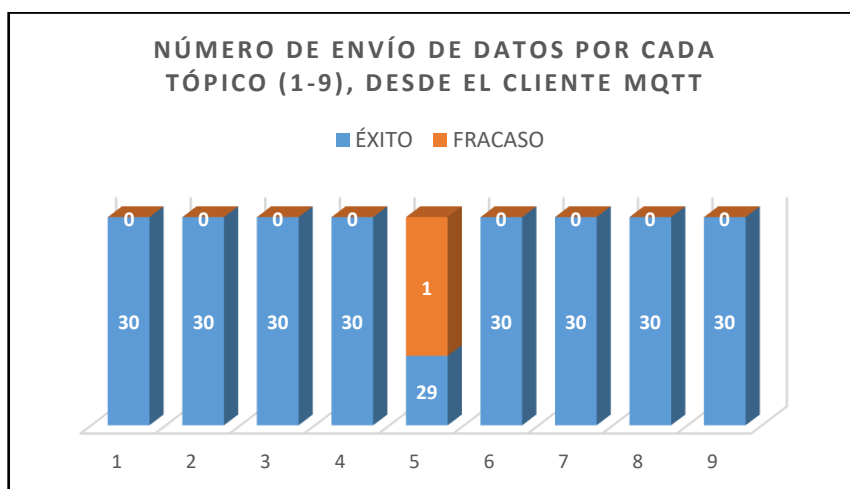


Figura 4-2. Envíos exitosos y de fracaso desde el cliente MQTT

Fuente: Zambrano, 2019

De igual manera, se procedió a comprobar la etapa de *publicación del modelo MQTT*, pero esta vez enviando datos desde el *forte* de la aplicación hacia la terminal MQTT.

La *Tabla 4-2* indica la cantidad de datos enviados desde el *forte* de la aplicación hacia la terminal MQTT. Al igual que en el caso anterior se realizaron treinta (30) envíos.

Tabla 4- 2: Número de envíos desde el forte de la aplicación

	NÚMERO DE ENVÍOS DESDE EL FORTE DE LA APLICACIÓN	
	ÉXITO	FRACASO
Tópico 1 (<i>youbot/arm/j1</i>)	30	0
Tópico 2 (<i>youbot/arm/j2</i>)	30	0
Tópico 3 (<i>youbot/arm/j3</i>)	30	0
Tópico 4 (<i>youbot/arm/j4</i>)	30	0
Tópico 5 (<i>youbot/arm/j5</i>)	30	0
Tópico 6 (<i>youbot/arm/jGR</i>)	30	0
Tópico 7 (<i>youbot/base/LV</i>)	30	0
Tópico 8 (<i>youbot/base/TV</i>)	30	0
Tópico 9 (<i>youbot/base/AV</i>)	29	1
TOTAL	269	1

Fuente: Zambrano, 2019

Realizado por: Tatiana Zambrano

La *figura 4-3* describe de manera estadística el número de envíos exitosos y de fracaso que se realizaron desde la aplicación del modelo de comunicación hacia el cliente MQTT en la nube, a través de los diferentes tópicos de comunicación, los datos para la gráfica fueron tomados de la *Tabla 4-2*.

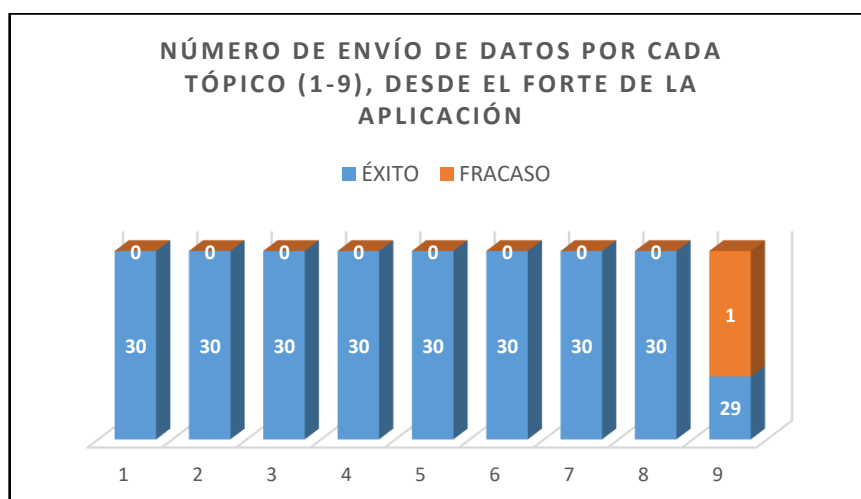


Figura 4-3. Envíos exitosos y de fracaso desde el forte de la aplicación

Fuente: Zambrano, 2019

A continuación se muestra la *Tabla 4-3*. donde se indica de manera consolidada el número total de éxitos y fracasos en el envío y recepción de los 540 datos que fueron utilizados en la prueba.

Tabla 4- 3: Total de éxitos y fracasos en el sistema de comunicación creado.

	Envío	Recepción
Éxitos	269	269
Fracasos	1	1
Total	270	270

Fuente: Software SPSS

Realizado por: Tatiana Zambrano

La *figura 4-4* muestra los porcentajes de efectividad obtenidos en las pruebas de transmisión de datos aplicando el modelo de comunicación creado.

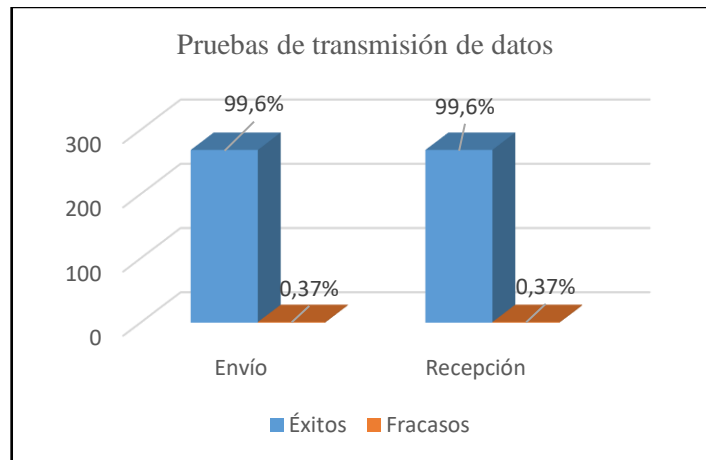


Figura 4-4. Porcentajes obtenidos en las pruebas de Tx de datos

Fuente: Zambrano, 2019

4.3 Comprobación de hipótesis

Con los datos obtenidos, se procedió a aplicar la prueba de hipótesis *Q de Cochran* para comprobar si existe o no igualdad en las nueve muestras relacionadas a la variable de respuesta de éxito o fracaso en el envío y recepción de datos desde el cliente MQTT hacia el *forte* de la aplicación.

Mediante el uso del software estadístico SPSS se obtuvo la *Tabla 4-4* que muestra las frecuencias de los sucesos exitosos y de fracaso en el envío y recepción de datos desde y hacia el bróker MQTT, pudiendo notar que no existen diferencias significativas entre éstas.

Tabla 4- 4: Frecuencias de la variable de estudio

	Valor	
	0	1
topico1	0	30
topico2	0	30
topico3	0	30
topico4	0	30
topico5	1	29
topico6	0	30
topico7	0	30
topico8	0	30
topico9	1	29

Fuente: Software SPSS

Realizado por: Tatiana Zambrano

Además, el SPSS permitió la obtención de la *Tabla 4-5* que indica el resultado de los estadísticos de prueba que sirven para determinar la aceptación o rechazo de la hipótesis. Claramente se observa que el *p-valor* del test *Q de Cochran* 0,630 es superior a 0,05 (*nivel de significancia*), lo que significa que se encuentra en la zona de aceptación, por tanto no se rechaza la hipótesis, es decir, no existe diferencias en cuanto al **éxito** de envío y recepción de los datos de prueba en los nueve tópicos de comunicación.

Tabla 4- 5: Estadísticos de contraste

N	30
Q de Cochran	6,154 ^a
gl	8
Sig. asintót.	,630

a. 1 se trata como un éxito.

Fuente: Software SPSS

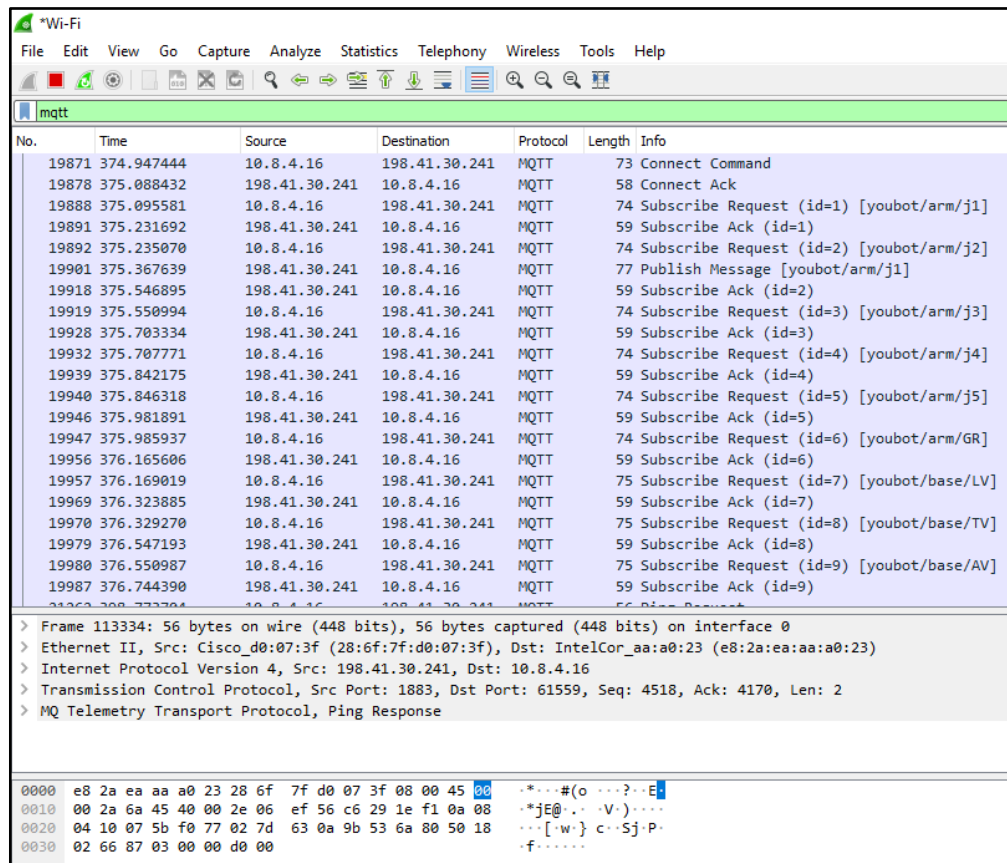
Realizado por: Tatiana Zambrano

Por lo tanto se puede concluir que la hipótesis de investigación planteada se ACEPTA, esto es:

“El protocolo MQTT basado en la norma IEC 61499 permite establecer la comunicación de datos desde el robot Kuka Youbot hacia la nube”

4.4 Medición de tiempos

Además de haber comprobado que a través de la programación de bloques de función se puede establecer un modelo de comunicación MQTT, se realizaron mediciones del tiempo que se demora en transmitir un dato desde la aplicación creada hacia la nube, en las dos etapas: *suscripción* mostrada en la *figura 4-4* y *publicación* mostrada en la *figura 4-5*. Esto se logró insertando un sniffer (*Wireshark*) entre los dos elementos.



No.	Time	Source	Destination	Protocol	Length	Info
19871	374.947444	10.8.4.16	198.41.30.241	MQTT	73	Connect Command
19878	375.088432	198.41.30.241	10.8.4.16	MQTT	58	Connect Ack
19888	375.095581	10.8.4.16	198.41.30.241	MQTT	74	Subscribe Request (id=1) [youbot/arm/j1]
19891	375.231692	198.41.30.241	10.8.4.16	MQTT	59	Subscribe Ack (id=1)
19892	375.235070	10.8.4.16	198.41.30.241	MQTT	74	Subscribe Request (id=2) [youbot/arm/j2]
19901	375.367639	198.41.30.241	10.8.4.16	MQTT	77	Publish Message [youbot/arm/j1]
19918	375.546895	198.41.30.241	10.8.4.16	MQTT	59	Subscribe Ack (id=2)
19919	375.550994	10.8.4.16	198.41.30.241	MQTT	74	Subscribe Request (id=3) [youbot/arm/j3]
19928	375.703334	198.41.30.241	10.8.4.16	MQTT	59	Subscribe Ack (id=3)
19932	375.707771	10.8.4.16	198.41.30.241	MQTT	74	Subscribe Request (id=4) [youbot/arm/j4]
19939	375.842175	198.41.30.241	10.8.4.16	MQTT	59	Subscribe Ack (id=4)
19940	375.846318	10.8.4.16	198.41.30.241	MQTT	74	Subscribe Request (id=5) [youbot/arm/j5]
19946	375.981891	198.41.30.241	10.8.4.16	MQTT	59	Subscribe Ack (id=5)
19947	375.985937	10.8.4.16	198.41.30.241	MQTT	74	Subscribe Request (id=6) [youbot/arm/GR]
19956	376.165606	198.41.30.241	10.8.4.16	MQTT	59	Subscribe Ack (id=6)
19957	376.169019	10.8.4.16	198.41.30.241	MQTT	75	Subscribe Request (id=7) [youbot/base/LV]
19969	376.323885	198.41.30.241	10.8.4.16	MQTT	59	Subscribe Ack (id=7)
19970	376.329270	10.8.4.16	198.41.30.241	MQTT	75	Subscribe Request (id=8) [youbot/base/TV]
19979	376.547193	198.41.30.241	10.8.4.16	MQTT	59	Subscribe Ack (id=8)
19980	376.550987	10.8.4.16	198.41.30.241	MQTT	75	Subscribe Request (id=9) [youbot/base/AV]
19987	376.744390	198.41.30.241	10.8.4.16	MQTT	59	Subscribe Ack (id=9)

> Frame 113334: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
> Ethernet II, Src: Cisco_d0:07:3f (28:6f:7f:d0:07:3f), Dst: IntelCor_aa:a0:23 (e8:2a:ea:aa:a0:23)
> Internet Protocol Version 4, Src: 198.41.30.241, Dst: 10.8.4.16
> Transmission Control Protocol, Src Port: 1883, Dst Port: 61559, Seq: 4518, Ack: 4170, Len: 2
> MQ Telemetry Transport Protocol, Ping Response

```
0000 e8 2a ea aa a0 23 28 6f 7f d0 07 3f 08 00 45 00  ..*..#(o ...?..E..
0010 00 2a 6a 45 40 00 2e 06 ef 56 c6 29 1e f1 0a 08  ..*jE@...V)....
0020 04 10 07 5b 0f 77 02 7d 63 0a 9b 53 6a 80 50 18  ...[w}c..SjP..
0030 02 66 87 03 00 00 d0 00  ..f.....
```

Figura 4-5. Medición de tiempos en la etapa de suscripción capturados desde Wireshark
Fuente: Zambrano, 2019

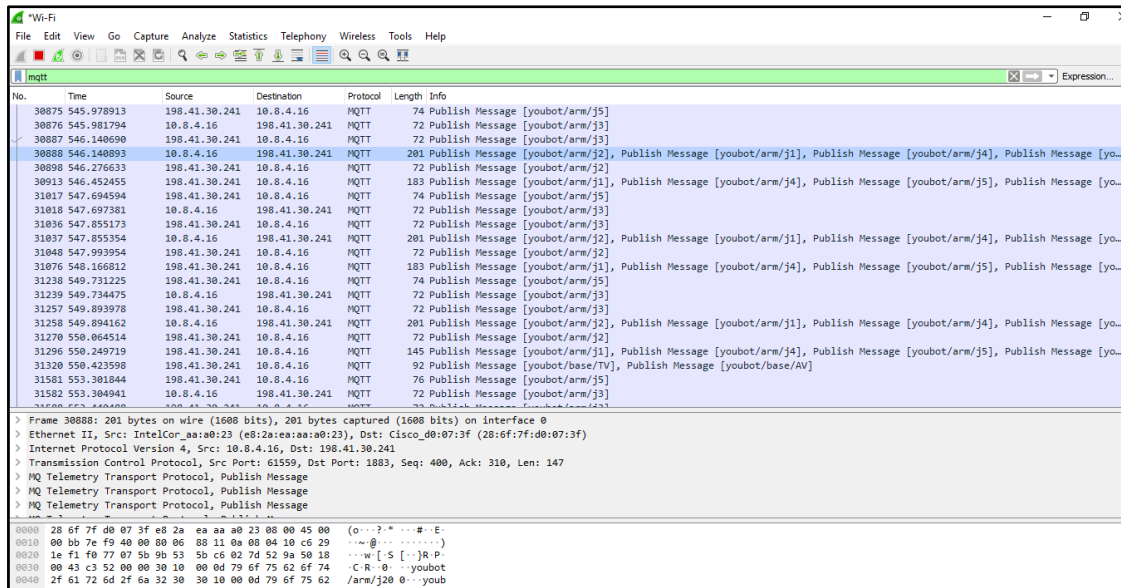


Figura 4-6. Medición de tiempos en la etapa de publicación capturados desde el Wireshark
Fuente: Zambrano. 2019

4.4.1 Resultados obtenidos en los tiempos de transmisión.

La *Tabla 4-6* muestra los tiempos que se obtuvieron en las etapas de publicación y suscripción de datos, al momento de establecer la comunicación entre la terminal MQTT en la nube y el *forte* de la aplicación.

Tabla 4- 6: Tiempos obtenidos en las etapas de publicación y suscripción de datos

TÓPICOS DE COMUNICACIÓN	ETAPA DE PUBLICACIÓN			ETAPA DE SUSCRIPCIÓN		
	TIEMPO MÁXIMO (ms)	TIEMPO MÍNIMO (ms)	TIEMPO PROMEDIO (ms)	TIEMPO MÁXIMO (ms)	TIEMPO MÍNIMO (ms)	TIEMPO PROMEDIO (ms)
Tópico 1	374,00	374,74	374,37	546,88	545,81	546,35
Tópico 2	374,81	374,21	374,51	553,66	553,12	553,39
Tópico 3	374,94	374,30	374,62	547,32	547,21	547,27
Tópico 4	374,55	375,32	374,94	546,98	545,09	546,04
Tópico 5	376,76	374,78	375,77	547,86	546,96	547,41
Tópico 6	375,71	375,27	375,49	550,65	549,65	550,15
Tópico 7	374,75	374,12	374,44	551,33	551,11	551,22
Tópico 8	374,17	374,07	374,12	548,56	546,12	547,34
Tópico 9	375,27	374,36	374,82	546,92	545,29	546,11

Fuente: Zambrano. 2019

Realizado por: Tatiana Zambrano

Como se puede observar, existe diferencia de tiempos en las dos etapas, ya que se trata de una comunicación asíncrona, donde el emisor y consumidor de datos están separados. Esto no significa que la transmisión sea lenta, por el contrario, gracias a su tipología de mensajes que tiene una carga máxima de 256 Mb lo hace ágil y ligero, convirtiéndolo de esa manera en el patrón para las comunicaciones del IoT.

CONCLUSIONES

- Existe un 99,6% de efectividad al transmitir datos utilizando un modelo de comunicación MQTT basado en la programación de bloques de función que establece el envío y recepción de datos generados en un robot Kuka Yubot hacia un bróker MQTT en la nube.
- El utilizar brókers públicos para establecer una comunicación MQTT representa un 0,37% en pérdida de conexión, ya que no todo el tiempo se encuentran en línea.
- Los tiempos de transmisión obtenidos en las etapas de publicación y suscripción son diferentes, demostrando así que se trata de un protocolo asíncrono que separa al emisor y receptor tanto en tiempo como en espacio.
- Las plataformas de desarrollo y ejecución de los bloques de función bajo la norma IEC 61499, permiten una versatilidad al momento de la programación ya que utilizan lenguajes open source ampliamente manejados.

RECOMENDACIONES

- Se recomienda crear nuevos modelos de comunicación basados en la arquitectura desarrollada en esta investigación, pero adaptándolos a diferentes protocolos IoT para poder establecer diferencias en cuanto a características de QoS.
- De igual forma, se recomienda incorporar un código de programación de control de errores que mantenga un monitoreo constante del estado de la red y permita establecer en caso de falla un enlace redundante.

BIBLIOGRAFÍA

- Andrew Banks and Rahul. (2017). V 3.1.1, MQTT,OASIS.
- B Kehoe S Patil, P. A. K. G. (2014). A survey of research on cloud robotics and automation, 1–12.
- Bani Yassein, M., Shatnawi, M. Q., Aljwarneh, S., & Al-Hatmi, R. (2017). Internet of Things: Survey and open issues of MQTT Protocol.
- Boschetti, M., Baglio, V., Ruiu, P., & Terzo, O. (2015). A Cloud Automation Platform for Flexibility in Applications and Resources Provisioning. *Proceedings - 2015 9th International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2015*, 204–208. <https://doi.org/10.1109/CISIS.2015.29>
- Chenaru, O., Stanciu, A., Popescu, D., Sima, V., Florea, G., & Dobrescu, R. (2015). Open cloud solution for integrating advanced process control in plant operation. *2015 23rd Mediterranean Conference on Control and Automation, MED 2015 - Conference Proceedings*, 973–978. <https://doi.org/10.1109/MED.2015.7158884>
- Colombo, A. (2017). Industrial Cyberphysical Systems, (march), 6–16.
- Cuevas, E., López, M., & García, M. (2012). *Ultrasonic Techniques and Industrial Robots: Natural Evolution of Inspection Systems*. Retrieved from <https://pdfs.semanticscholar.org/3066/5d4a1d7149a702460f37a141da07d1be6cbd.pdf>
- Dai, W., & Vyatkin, V. (2010). Redesign distributed IEC 61131-3 PLC system in IEC 61499 function blocks. *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010*. <https://doi.org/10.1109/ETFA.2010.5641239>
- De Sousa, M. (2010). Analyzing the compatibility between ISA 88 and IEC 61499. *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010*. <https://doi.org/10.1109/ETFA.2010.5641308>
- Dhar, P., & Gupta, P. (2017). Intelligent parking Cloud services based on IoT using MQTT protocol. *International Conference on Automatic Control and Dynamic Optimization Techniques, ICACDOT 2016*, 30–34. <https://doi.org/10.1109/ICACDOT.2016.7877546>
- Ding Yi, Fan Binwen, Kong Xiaoming, & Ma Qianqian. (2016). Design and implementation of mobile health monitoring system based on MQTT protocol. *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 1679–1682. <https://doi.org/10.1109/IMCEC.2016.7867503>
- Dumitrache, I. (2016). Factories of Future towards next industrial revolution. *Ieee*, 3–4.
- Faber, J., Diaz, A., Eduardo, L., Rojas, B., & Morales, J. V. (2011). Transformaciones lineales de

- dimensión finita , aplicadas al desarrollo del modelo cinemático directo para el robot KUKA KR 60 JET R en cursos de álgebra lineal y dibujo de máquinas, *XIX(2)*, 33–47.
- Garc, C. A., & Castellanos, E. X. (2018). Cpps bajo iec 61499 e isa 88 en dispositivos de bajo costo para procesos por lotes.
- García Soria, A. (2017). Plataforma domótica basada en Raspberry Pi y el protocolo MQTT.
- Guellouz, S., Benzina, A., Khalgui, M., & Frey, G. (2016). Reconfigurable Function Blocks : Extension to the Standard IEC 61499.
- Hermann, M., Pentek, T., & Otto, B. (2016). Design principles for industrie 4.0 scenarios. *Proceedings of the Annual Hawaii International Conference on System Sciences, 2016–March*, 3928–3937. <https://doi.org/10.1109/HICSS.2016.488>
- Houimli, M., Kahloul, L., & Benaoun, S. (2017). Formal Specification , Verification and Evaluation of the MQTT Protocol in the Internet of Things. *2017 International Conference on Mathematics and Information Technology*, 214–221.
- Insaurrealde, C. C. (2016). Modeling standard for distributed control systems: IEC 61499 from industrial automation to aerospace. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)* (pp. 1–8). IEEE. <https://doi.org/10.1109/DASC.2016.7777980>
- Jayaram, A. (2016). Lean Six Sigma Approach for Global Supply Chain Management using Industry 4 . 0 and IIoT, 89–94.
- Jose Carlos Mercé Godoy. (2015). Autor: josé carlos mercé godoy tutor: julio ariel romero perez convocatoria noviembre de 2015.
- Kang, D. H., Park, M. S., Kim, H. S., Kim, D. Y., Kim, S. H., Son, H. J., & Lee, S. G. (2017). Room Temperature Control and Fire Alarm/Suppression IoT Service Using MQTT on AWS. *2017 International Conference on Platform Technology and Service, PlatCon 2017 - Proceedings*. <https://doi.org/10.1109/PlatCon.2017.7883724>
- Lindgren, P., Eriksson, J., Lindner, M., Lindner, A., & Tec, C. I. (2016). End-to-End Response Time of IEC 61499 Distributed Applications over Switched Ethernet, *3203(c)*. <https://doi.org/10.1109/TII.2016.2626463>
- Locomotec. (2011). KUKA youBot User Manual, 1–43.
- Lozada, E. C. (2018). Transparencia Flexible para un Sistema de Teleoperación Bilateral para Robots Móviles en la industria del Gas y Petróleo, *00*, 1–9.
- Mektoubi, A., Hassani, H. L., Belhadaoui, H., Rifi, M., & Science, C. E. D. (2016). New approach for securing communication over MQTT protocol A comparaisn between RSA and Elliptic Curve, *0*.
- Mirelez-Delgado, F., Morales-Díaz, A., Ríos-Cabrera, R., & Pérez-Villeda, H. (2015). Control Servovisual de un Kuka youBot para la manipulación y traslado de objetos. *Congreso Nacional de Control Automático, AMCA 2015*, (2012), 239–244.

- Osathanunkul, K., Hantrakul, K., Pramokchon, P., Khoenkaw, P., & Tantitharanukul, N. (2017). Configurable Automatic Smart Urinal Flusher based on MQTT Protocol.
- Semle, P. A. (2016). Protocolos IIoT para considerar, 32–35.
- Singh, M., Ma, R., VI, S., & Balamuralidhar, P. (2015). Secure MQTT for Internet of Things (IoT). <https://doi.org/10.1109/CSNT.2015.16>
- Thramboulidis, K. (2009). IEC 61499 function block model: Facts and fallacies. *IEEE Industrial Electronics Magazine*, 3(4), 7–23. <https://doi.org/10.1109/MIE.2009.934788>
- Vlad, V., Popa, C. D., Pentiu, R. D., & Buzduga, C. (2014). Control architecture for power distribution systems based on IEC 61850, IEC 61499 and holonic concepts. *EPE 2014 - Proceedings of the 2014 International Conference and Exposition on Electrical and Power Engineering*, (Epe), 132–136. <https://doi.org/10.1109/ICEPE.2014.6969883>
- Vyatkin, V. (2009). The IEC 61499 standard and its semantics - Bridging the Gap Between PLC Programming Languages and Distributed Systems. *Industrial Electronics Magazine, IEEE*, 3(4), 40–48. <https://doi.org/10.1109/MIE.2009.934796>
- Vyatkin, V. (2011). IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics*, 7(4), 768–781. <https://doi.org/10.1109/TII.2011.2166785>
- Wan, J., Tang, S., Shu, Z., Li, D., Wang, S., Imran, M., & Vasilakos, A. V. (2016). Software-Defined Industrial Internet of Things in the Context of Industry 4.0. *IEEE Sensors Journal*, 16(20), 7373–7380. <https://doi.org/10.1109/JSEN.2016.2565621>
- Wenger, M., & Zoitl, A. (2012). Re-use of IEC 61131-3 structured text for IEC 61499. *2012 IEEE International Conference on Industrial Technology, ICIT 2012, Proceedings*, 78–83. <https://doi.org/10.1109/ICIT.2012.6209917>
- Yokotani, T., & Sasaki, Y. (2017). Comparison with HTTP and MQTT on required network resources for IoT. *ICCEREC 2016 - International Conference on Control, Electronics, Renewable Energy, and Communications 2016, Conference Proceedings*, 1–6. <https://doi.org/10.1109/ICCEREC.2016.7814989>
- Zoitl, A., & Vyatkin, V. (2009). IEC 61499 architecture for distributed automation: The “glass half full” view. *IEEE Industrial Electronics Magazine*, 3(4), 7–22. <https://doi.org/10.1109/MIE.2009.934789>

ANEXOS

ANEXO A.

Creación de librerías del MQTT PAHO

1. Se instaló *Cmake* como una herramienta multiplataforma para generación de código (scripts de compilación) y *Visual Studio* como compilador de C++ para Windows.
2. Se descargó el *source* (*archivo fuente*) de MQTT PAHO, para lograr que la aplicación se comunique bajo el protocolo MQTT.
3. A partir del *source* de MQTT PAHO y haciendo uso de CMake se generaron los archivos a ser compilados.
4. Dentro de CMake se seleccionó el directorio de origen *C:/paho.mqtt.c* de donde se extrajo el código fuente como se observa en la *figura 1A*, y se creó el directorio *C:/paho.mqtt.c/build/win64* como destino del archivo que se va a construir, mostrado en la *figura 2A*.

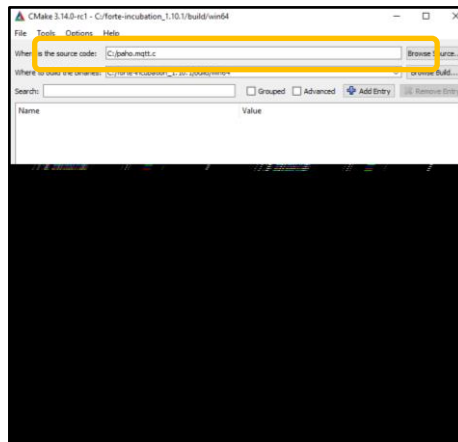


Figura 1A. Directorio origen de pahoo.mqtt

Fuente: Zambrano, 2019

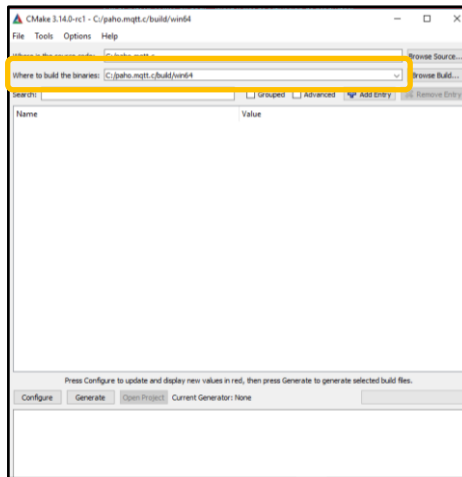


Figura 2A. Directorio destino del archivo a construir a partir de paho.mqtt
Fuente: Zambrano, 2019

- Se eligió Visual Studio como compilador del proyecto y posteriormente se lo configuró seleccionando los parámetros y librerías necesarias para la compilación del *MQTT PAHO* como se indica en la *figura 3A*.

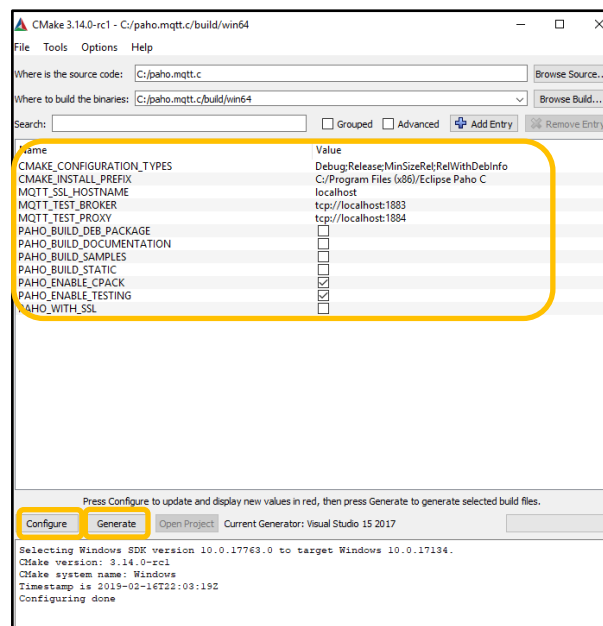


Figura 3A. Parámetros y librerías para compilación de MQTT PAHO
Fuente: Zambrano, 2019

- Una vez realizada la configuración se confirmó que los archivos fueron creados correctamente para ello se verificó su existencia en el directorio *C:\paho.mqtt.c\build\win64* mostrado en la *figura 4A*, y se continuó con el proceso de compilación del archivo generado llamado *Eclipse Paho C.sln* haciendo uso de Visual Studio.

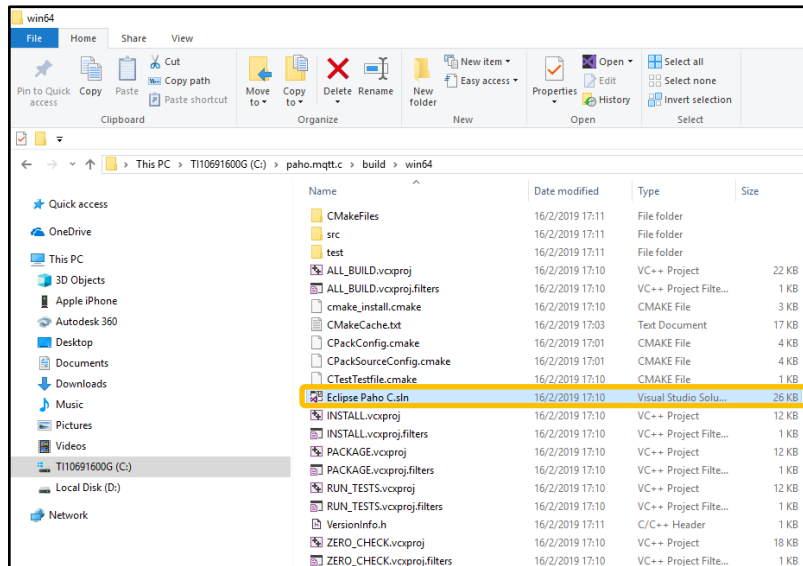


Figura 4A. Creación de archivo Eclipse PahoC.sln
Fuente: Zambrano, 2019

7. En la plataforma de Visual Studio se procedió con la construcción de los archivos correspondientes al *Eclipse Paho C.sln*. Las figuras 5A y 6A muestran lo mencionado.

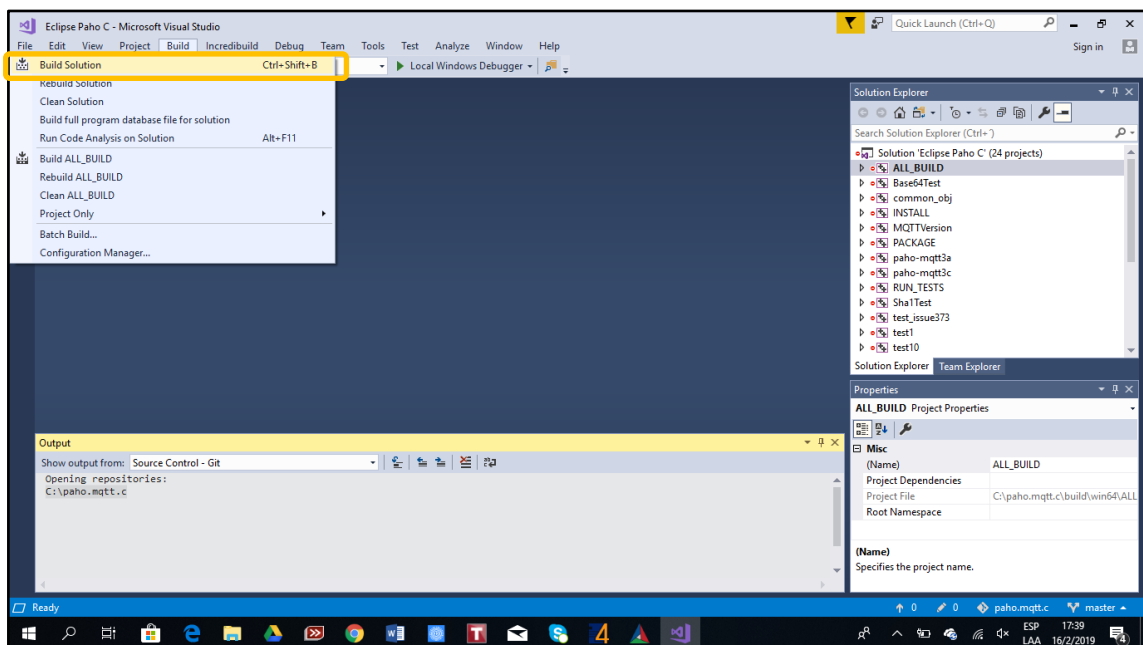


Figura 5A. Construcción de archivo
Fuente: Zambrano, 2019

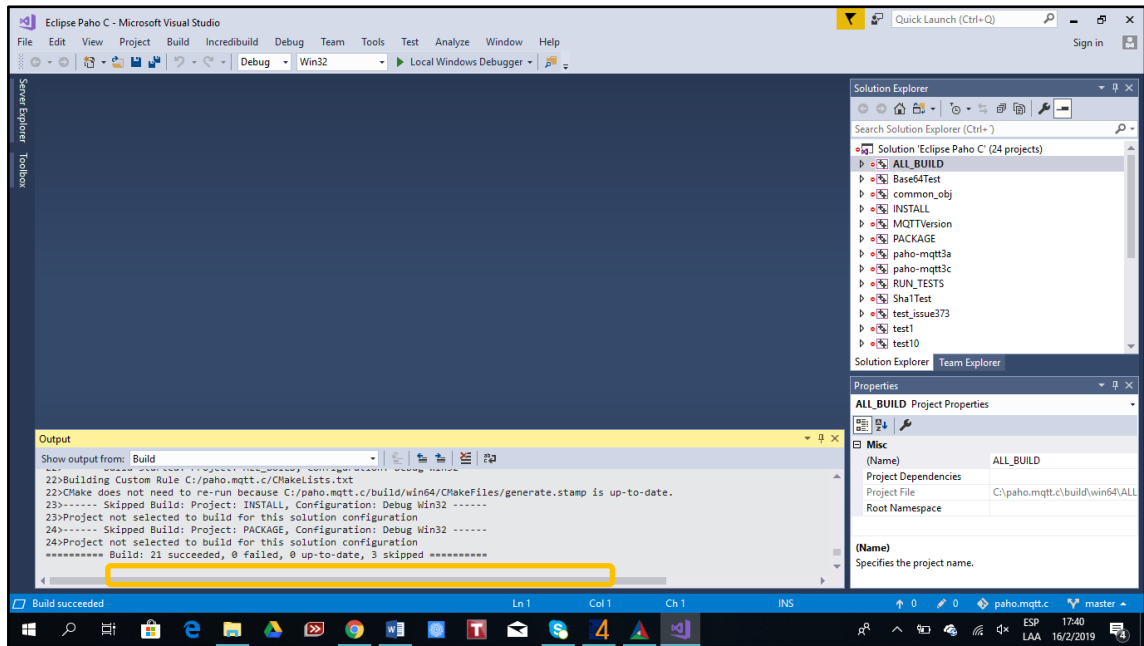


Figura 6A. Archivos construidos satisfactoriamente
Fuente: Zambrano, 2019

ANEXO B.

Creación de FORTE

1. Se compiló 4DIAC FORTE a partir de su código fuente. Con el archivo descargado llamado *forte-incubation_1.10.1* se utilizó el CMake indicando la ubicación de donde va a ser extraído el *source* y la ubicación en donde se construirán los nuevos archivos, posteriormente se procedió con la configuración de éstos tal como se indica en la *figura 1B*.

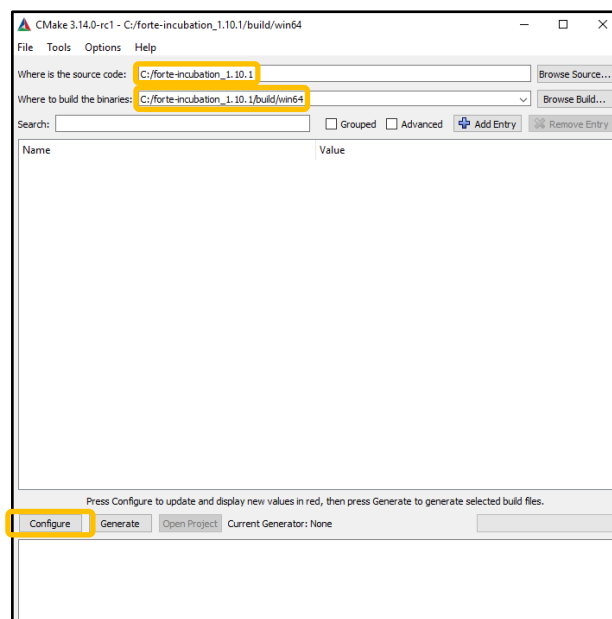


Figura 1B. Selección de directorio fuente y destino de archivo a compilar
Fuente: Zambrano, 2019

2. Para evitar errores al configurar con CMake, se añadieron los siguientes parámetros y librerías mostrados en la *figura 2B*:
 - *FORTE_ARCHITECTURE Win32*
 - *FORTE_MODULE_CONVERT*
 - *FORTE_MODULE_IEC61131, FORTE_MODULE_UTILS*

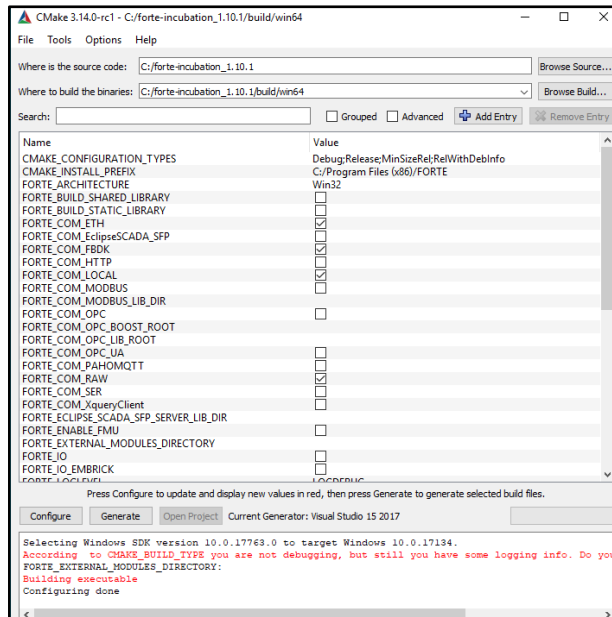


Figura 2B. Librerías añadidas para proceso de compilación
Fuente: Zambrano, 2019

3. Cargadas las librerías y generados los archivos se comprobó su creación en el directorio `C:/forte-incubation_1.10.1/build/win64`. Y, previo a la compilación final de `forte.sln`, se regresó nuevamente al CMake para añadir la librería correspondiente a `MQTT PAHO` que fue la `FORTE_COM_PAHOMQTT` creada en la *fase 1* del desarrollo, la cual se muestra en la *figura 3B*.

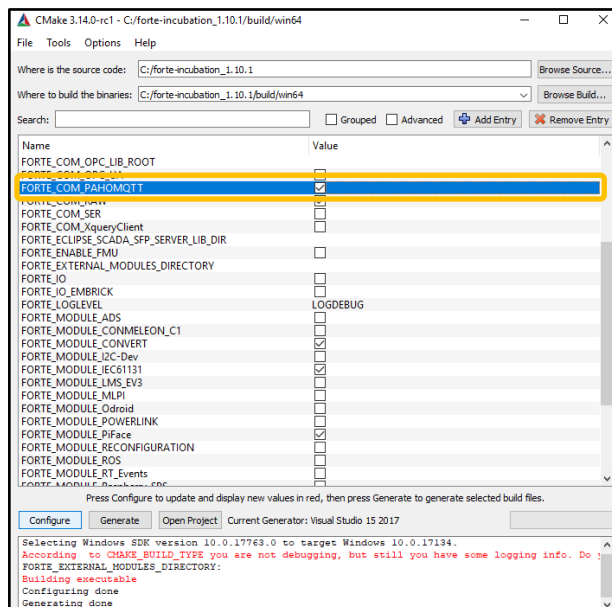


Figura 3B. Librería MQTT PAHO añadida para compilación de forte
Fuente: Zambrano, 2019

4. Al configurarla y generarla aparecieron errores en cuanto a la librería *FORTE_COM_PAHOMQTT_LIB*, cuyo archivo *paho-mqtt3a.dll* aún no estaba especificado ni tampoco su ubicación por tanto se procedió con su corrección mostrada en la *figura 4B*. Cabe mencionar que para evitar errores el archivo *paho-mqtt3a.dll* se copió al directorio principal de forte *C:\forte-incubation_1.10.1\build\win64\src\Debug* ya que es allí donde se creó el forte principal, todo esto se lo indica en la *figura 5B*.

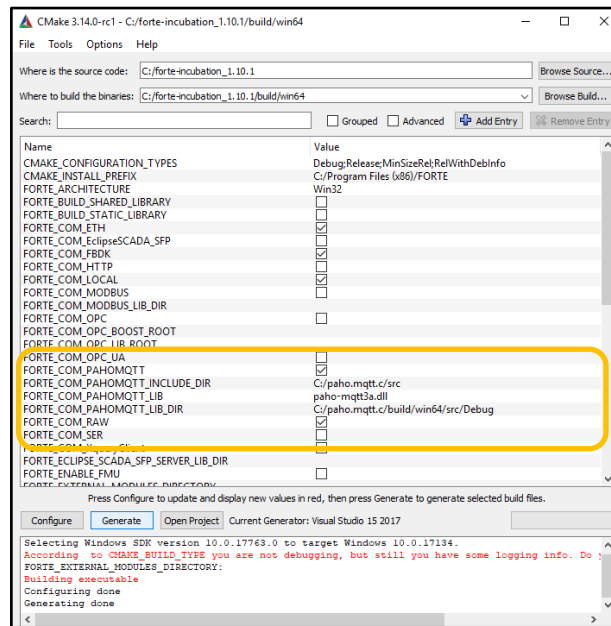


Figura 4B. Ubicación paho-mqtt3a.dll dentro del directorio contenedor de forte
Fuente: Zambrano, 2019

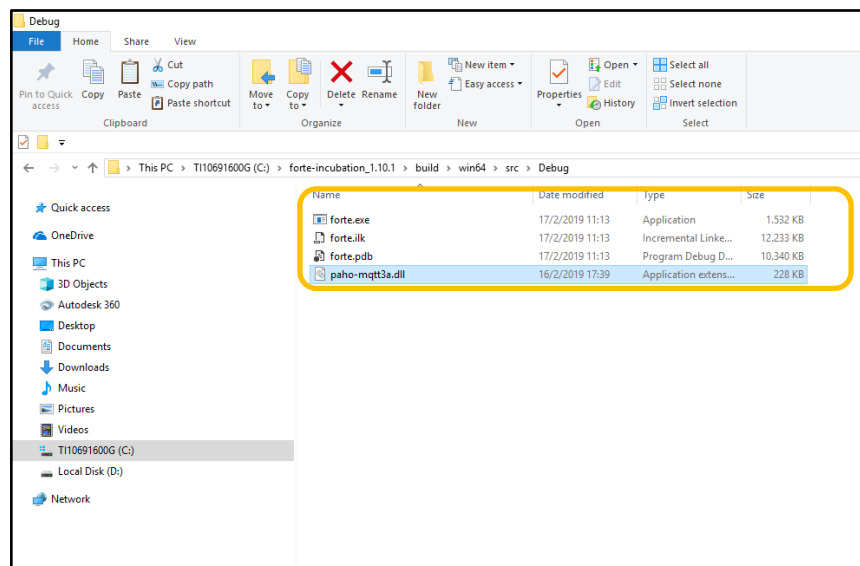


Figura 5B. Archivo forte creado
Fuente: Zambrano, 2019

5. En la *figura 6B*, se indica la inclusión de los parámetros y librerías: FORTE_ARCHITECTURE Win32, FORTE_MODULE_CONVERT, FORTE_MODULE_IEC61131, FORTE_MODULE_UTILS, necesarios para la correcta compilación.

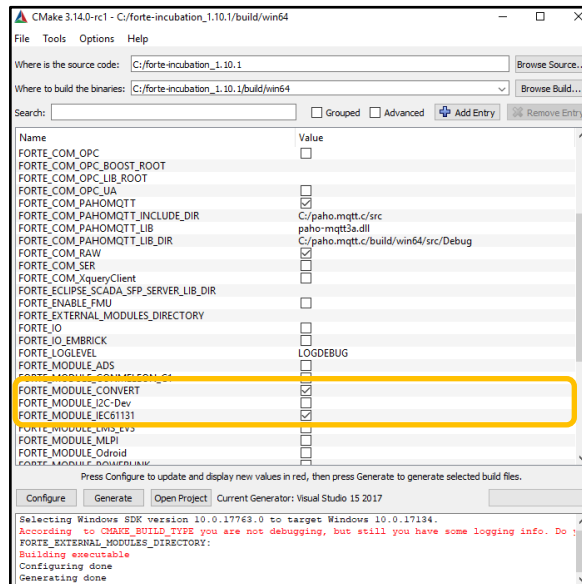


Figura 6B. Selección de parámetros y librerías para compilación final
Fuente: Zambrano, 2019

6. Ya sin errores se continuó con la compilación del archivo *forte.sln* a través de Visual Studio, obteniendo un forte final depurado y listo para cargar los nuevos bloques de función. La *figura 7B*. muestra lo mencionado.

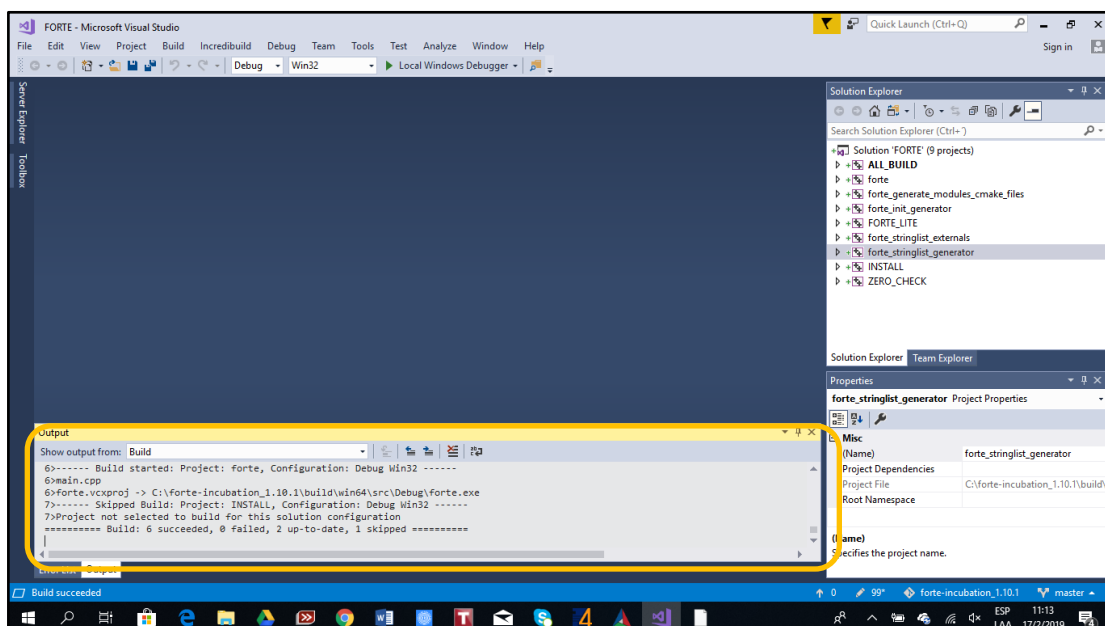


Figura 7B. Archivo forte compilado
Fuente: Zambrano, 2019