UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

ESTUDIO E IMPLEMENTACIÓN DE UNA PLATAFORMA DIGITAL PARA EL DESPLIEGUE DE APLICACIONES EN EL MARCO DE LA INDUSTRIA 4.0

GRADO EN
ENGINIERÍA
ELECTRÓNICA
INDUSTRIAL Y
AUTOMÁTICA

TRABAJO DE FINAL DE GRADO

Jan Pérez Reverte
Director **Miguel Delgado**
Co-director **Ángel Fernández**

Junio 2020

Índice

Índice de figuras

Índice de tablas

Agradecimientos

A mis compañeros, por su apoyo todos estos años.

A mi madre, por su cariño incondicional.

A mis amigos, sobre todo por estar. A mis tutores, por guiarme.

Y a Carla. Solo puedo decir GRACIAS. Así, en mayúsculas.

Capítulo I: Introducción

1. Resumen

Nadie puede cuestionar que la transformación digital que ha experimentado la humanidad en las últimas décadas, ha revolucionado la forma en que la sociedad se comunica, organiza o piensa. En un planeta conectado, las tecnologías de la información y la comunicación (TIC) hacen posible entender el mundo como un único ente, en el que sucesos locales de toda índole tienen repercusiones globales.

Los cambios sensibles hoy en día, como consecuencia de la implementación a las TIC en sectores como la educación, el financiero o de entretenimiento han sido el preludio de algo mucho mayor y complejo que pretende transformar el sector industrial a nivel global.

En la actualidad, el mundo se encuentra a las puertas de la cuarta revolución industrial. Una revolución, esencialmente distinta a las anteriores, que pretende sacudir con una masividad sin precedentes el sistema de producción conocido hasta el momento.

Adaptar el sector secundario a nuevos modelos organizativos, así como la implementación de soluciones basadas en la inteligencia de los datos, puede suponer a medio-largo plazo la supervivencia o deceso del tejido industrial. En concreto, **únicamente el 8% de las empresas españolas tiene un grado de digitalización avanzado, muy inferior a la media mundial, situada en torno al 33%, según PwC [1].**

Tradicionalmente, la información recopilada por los sensores en los procesos industriales o la generada por los sistemas de control, posee un carácter meramente informativo y válido para conocer el estado del sistema, en un momento determinado. En este sentido, de acuerdo a la experiencia profesional aportada, la capacidad de comunicar dicha información a las plataformas de gestión empresarial es inexistente por norma general; bien por inutilidad o falta de recursos. Por ende, existe una brecha entre las tecnologías de tipo operativo (OT) respecto a las tecnologías de la información (TI).

En este proyecto, **se realiza una aproximación a los principales conceptos que abarca la cuarta revolución industrial, denominada "Industria 4.0", así como las tecnologías asociadas a su implementación.** La principal motivación es acercar al lector una visión amplia de todo dicho paradigma.

Dentro de estas tecnologías, se enfatiza en el estado arte el concepto de IIoT (Industrial Internet of Things) y de las pasarelas que permiten conectar los dispositivos

de campo con la nube o las plataformas de TI, conocidos comúnmente como "Gateways".

A continuación, se presentan diferentes arquitecturas implementables para conectar sistemas de control industrial clásicos con la nube y las aplicaciones de gestión, en base a los conceptos y tecnologías mencionadas anteriormente.

La finalidad de esta aproximación teórica es ejemplificar todos los aspectos definidos, a través de un caso práctico de transformación digital en el ámbito académico. El objeto de estudio es el laboratorio de automatización 004 ubicado en el edificio TR2 de la Escuela Superior de Ingeniería Industrial, Aeroespacial y Audiovisual de Terrassa (ESEIAAT). En base a las arquitecturas propuestas, se describe de forma exhaustiva la programación de un dispositivo que permita la comunicación de los sistemas de control "obsoletos" con aplicaciones de gestión y el almacenamiento de datos en la nube, mediante una interfaz de programación de aplicaciones (API).

Para finalizar, se proponen **vías complementarias a modo de líneas futuras que dotarían al presente proyecto de una solidez aún mayor**.

2. Abstract

The truth is no one could ever question nowadays that the digital transformation that humanity has gone through in recent decades, has revolutionized the way in which society communicates, organizes or thinks.

The current sensible changes, as a consequence of the implementation of IT in industries such as education, finance or entertainment, have been the prelude to something much grater and more complex that aims to transform the industrial sector globally.

These days, the world is on the verge of the fourth industrial revolution. A revolution, essentially different from the previous ones, that pretends to shake up with an unprecedented massiveness the production system known until now.

Transform the secondary sector to new organizational models, as well as the implementation of solutions based on data intelligence, could mean the survival of the industrial fabric. Specifically, only 8% of Spanish companies have an advanced degree of digitization, which is so below the world average, around 33%, according to PwC [1].

Traditionally, the information collected by sensors in industrial processes or the one generated by control systems, has a merely informative and valid nature to know the state of the system, at a given moment. In this sense, according to the professional experience provided, the ability to communicate such information to business management platforms is generally non-existent; either due to uselessness or lack of resources. Therefore, there is a gap between operational technologies (OT) regarding to information technologies (IT).

In this project, an approach is made to the main concepts covered by the fourth industrial revolution, called "Industry 4.0", as well as the technologies associated with its implementation. The main motivation is to offer the reader a wide view of the entire said paradigm.

Within these technologies, the concept of IIoT (Industrial Internet of Things) and the gateways that allow connecting field devices with the cloud or IT platforms, commonly known as “Gateways”, are emphasized in the state of the art.

Below there are different implementable architectures to connect classic industrial control systems to the cloud and management applications, based on the concepts and technologies mentioned above.

The purpose of this theoretical approach is to exemplify all the defined topics, through a practical case of digital transformation in the academic field. The object of study is the 004 automation laboratory located in the TR2 building of the Escuela Superior de Ingeniería Industrial, Aeroespacial y Audiovisual de Terrassa (ESEIAAT). According to the proposed architectures, this document describes exhaustively the programming of a device that allows communication of the “obsolete” control systems with management applications and data storage in the cloud, through an application-programming interface (API).

Finally, complementary applications are proposed as future lines that would give the present project even greater strength.

4. Antecedentes

De acuerdo con la experiencia laboral del autor, la tendencia general de los últimos 25 años en automatización industrial es la creación de sistemas de control embebidos aislados de las plataformas de gestión. Fruto de la tercera revolución industrial, las fábricas disponen de sistemas automáticos que controlan eficazmente el funcionamiento de los procesos. Es el caso de los controladores lógico programables (PLC, por sus siglas en inglés). Permiten realizar maniobras de control en base al estado de señales digitales/analógicas provenientes de sensores (presostatos, conductímetros, etc.) u otros controladores. Estas señales son procesadas por el PLC que, en función del estado en que se encuentra y su programación, realiza acciones de control para interactuar con el proceso (apertura de electroválvulas, regulación de la consigna en un variador, entre muchas otras). De forma adicional, se dispone de una interfaz hombre-máquina (HMI, por sus siglas en inglés) que interactúa con el controlador permitiendo parametrizar el funcionamiento de la instalación.

Figura 1. Ejemplo de PLCs y HMI. Fuente: Siemens.

Por ende, su funcionamiento es autónomo o con la capacidad de comunicar información de forma limitada, respecto a los demás agentes que controlan la instalación (con carácter general). Además, el valor de la información recopilada o generada por los controladores reside en conocer el estado del proceso en un momento determinado, ya sea para realizar acciones de regulación o de carácter informativo. Este aspecto, responde en gran parte a la problemática más básica y necesaria asociada: el control automático de la instalación.

En la actualidad, **la industria contempla nuevas necesidades en torno a los sistemas de control y su papel dentro de las fábricas**. En un sector en constante evolución, se requieren soluciones que brinden **flexibilidad y modularidad** a las líneas de producción, llevando la automatización al siguiente nivel. Es de comentar la aparición de nuevas tecnologías emergentes con capacidad de revolucionar el funcionamiento de la producción, en consecuencia, la forma en que se organiza la industria. Son claros de aplicación la inteligencia artificial [2] [3], la visión por computador para realizar controles de calidad, o la fabricación aditiva en la producción de piezas .

Más allá de las acciones a nivel de máquina, se requiere integrar estos sistemas dentro de un ecosistema digital. **La información se ha convertido en una fuente de valor**, junto con la calidad, seguridad y velocidad que implica su transmisión. En este sentido, empiezan a surgir necesidades propias de una nueva realidad industrial. La literatura actual es extensa con respecto a proyectos de transformación digital en el ámbito industrial. Se cita como ejemplo de estudio relevante con este proyecto el realizado por Glória, André (2017) [4]. A continuación, se presentan algunos ejemplos de transformación en base a la experiencia del autor:

- Sistemas de telecontrol
- Visualización deslocalizada del estado de la producción
- Centralización de información proveniente de diferentes células en una única plataforma
- Almacenamiento masivo de datos generados por los procesos productivos
- Integración de dispositivos en los sistemas de gestión de recursos empresariales

Estas necesidades presentan nuevos retos de carácter operativo y en términos de gestión, trasladadas principalmente al ámbito de las tecnologías de la información aplicadas a la industria. Por tanto, se precisan nuevas arquitecturas, dispositivos y tecnologías para hacer frente a dichas necesidades de forma eficaz.

5. Objeto del proyecto

Estudiar los principales conceptos y implicaciones en términos de productividad, organización y operación de la industria 4.0, con la finalidad de ofrecer un amplio punto de vista respecto. Teniendo en cuenta que la Industria 4.0 pretende ser un modelo abstracto, más que una implementación directa a nivel técnico, es imprescindible realizar un estado del arte de las principales tecnologías que habilitan su implementación. Por razones expuestas en la presente memoria[1], se pretende enfatizar en aquellas con mayor implicación en la interactuación entre sistemas informáticos y sistemas de control de máquina.

De forma consecutiva, se presentan diferentes estructuras de comunicación basadas en las tecnologías mencionadas. Estas arquitecturas responden a la aplicación del estado del arte actual del *Industrial Internet of Things (IIoT)*, con el objetivo de ejemplificar diferentes métodos de comunicación entre las plataformas de tecnologías de la información y los sistemas de control. Se exponen las ventajas e inconvenientes de cada arquitectura y las características a tener presentes.

Se pretende entonces realizar la implementación de un *Gateway IIoT* a nivel práctico en el ámbito académico, a modo de validación de las estructuras previamente analizadas. Este proceso implica la selección y dimensionado de todos los componentes de *software* y *hardware* necesarios para su aplicación. Se analiza de forma extensa la programación asociada al objeto de estudio, con el objetivo de ejemplificar el desarrollo de dicha implementación en términos operativos.

[1] Para más información, revisar el apartado “Antecedentes”.

6. Objetivos

Este proyecto presenta dos grandes objetivos, uno esencialmente más académico puesto que se realiza un *background* de la industria 4.0, y otro de carácter práctico, con el objeto de implementar una solución en el marco de dicha revolución.

El objetivo general en aspectos teóricos de este proyecto es estudiar el estado del arte de la industria 4.0 y sus tecnologías. Estas deben ser analizadas de forma superficial en caso de no guardar relación con el caso práctico, y de forma más extensa si se ha previsto una implementación.

En relación con el caso práctico, **el objetivo general es la creación y la implementación de una interfaz de programación de aplicaciones (API) que permita a diferentes aplicaciones independientes poder gestionar el estado de la célula industrial descrita**.

Esta API debe estandarizar la entrada/salida de información para que independientemente de la aplicación de supervisión y control utilizada, las acciones de bajo nivel que se realizan a nivel de máquina sean las deseadas.

La comunicación de las aplicaciones respecto al servidor debe ser mediante protocolos fiables y seguros, con una alta calidad en el servicio.

Debe tener funciones de almacenamiento en la nube, para poder acceder a la información histórica del proceso de forma remota.

Unificación de las tecnologías de la información (TI) con las tecnologías de operación (OT) aplicando los conceptos propios de la Industria 4.0 con el objetivo de tener un sistema de gestión integrado de la célula industrial.

7. Alcance del proyecto

El alcance general de este proyecto asume estudiar el estado del arte de la Industria 4.0 y sus tecnologías. Como ejemplo de implementación, se contempla el estudio de un caso práctico en favor de aplicar las tecnologías mencionadas a partir del desarrollo de una interfaz de programación de aplicaciones (API). La finalidad es comunicar de forma eficiente aplicaciones de alto nivel con el control de máquina. De manera que queda exento de su desarrollo el programa PLC, así como la creación de aplicaciones de gestión de alto nivel. A modo de validación, únicamente se contempla el uso de un programa que permita validar el funcionamiento de la API.

A continuación, se describe el alcance de las principales tareas a realizar en el transcurso del proyecto, en función de su naturaleza. Mediante un diagrama de Gantt[2], se expone la duración asignada a cada parte del proyecto.

7.1 Marco teórico

7.1.1 Project Charter

Realizar una primera aproximación al proyecto de forma general. Definir los principales objetivos del proyecto, su alcance y planificación.

Entrega	**Project Charter.**

Tabla 1. Entregables del Project Charter. Fuente: Elaboración propia

7.1.2 Estado del arte de la Industria 4.0

Estudiar a partir de los principales conceptos que definen la industria 4.0, los sistemas ciber-físicos. Hacer alusión y definir las principales características de las tecnologías emergentes en la cuarta revolución industrial.

7.1.3 Redacción de la memoria del proyecto.

Creación de la documentación referente al desarrollo del proyecto y las bases teóricas que sustentan su aplicación.

Entrega	**Memoria de proyecto.**

[2] Diagrama que recoge las fases de desarrollo propuestas en un horizonte temporal definido.

Tabla 2. Entregables de la memoria de proyecto. Fuente: Elaboración propia.

7.1.4 Defensa del proyecto.

Preparación de la defensa del proyecto ante un tribunal, de forma telemática o presencial. Creación del soporte visual necesario.

Entrega	**Defensa del proyecto.** **Soporte visual.**

Tabla 3. Entregables de la defensa del proyecto. Fuente: Elaboración propia.

7.2 Caso práctico

7.2.1 Digitalizar el estado de los retenedores/plataformas

Con la finalidad de poder conocer el estado de los actuadores (en este caso los retenedores y plataformas de la línea), es necesario realizar peticiones Modbus al PLC, tratar la información y, posteriormente, publicar mediante MQTT.

Entrega	**Primera versión del documento API REFERENCE, que expone cómo hacer uso de la API.**

Tabla 4. Entregables de la digitalización del estado de los retenedores y plataformas. Fuente: Elaboración propia.

7.2.2 Gestión de productos y órdenes de producción.

Las aplicaciones pueden introducir pedidos en el sistema y éstas deben ser gestionadas por la API. Se pretende definir las estructuras de datos que darán lugar a la comunicación entre aplicaciones y su integración con el control de máquina.

Entrega	**Ampliación del documento API REFERENCE.**

Tabla 5. Entregables de la gestión de productos. Fuente: Elaboración propia.

7.2.3 Desarrollo de funcionalidades.

Desarrollo de acciones de gestión de la célula, con independencia de la *management* de productos y maniobras de PLC (no son propias de este proyecto de final de grado). Dichas maniobras deben ser de carácter básico y de parametrización de la celda. Se consideran ejemplos:

- Bloqueo de plataformas y retenedores de forma individual

- Bloqueo de retenedores de forma conjunta
- Maniobras de control de máquina de alto nivel
- Volcado de información en el PLC

Entrega	Ampliación del documento API REFERENCE.

Tabla 6. Entregables de la creación de funciones. Fuente: Elaboración propia.

7.2.4 Ampliación de funcionalidades.

Desarrollo de funciones secundarias para el funcionamiento de la instalación, más propiamente relacionadas con el almacenamiento de información.

- Almacenamiento del estado de los retenedores en una base de datos en tiempo real mediante MongoDB
- Almacenamiento de las modificaciones en el estado de los retenedores en la nube mediante MongoDB ATLAS
- Visualización gráfica del estado de los retenedores mediante un *dashboard* implementado en la aplicación de validación

Entrega	Versión definitiva del documento API REFERENCE.

Tabla 7. Entregables de la ampliación de funcionalidades. Fuente: Elaboración propia.

7.2.5 Validación práctica de la solución (Puesta en marcha).

No será posible realizar la validación práctica de la solución a causa de la aplicación del estado de alarma por parte del gobierno (Real Decreto 463/2020) que limita el uso de los espacios de docencia y la movilidad de las personas por la COVID-19 .

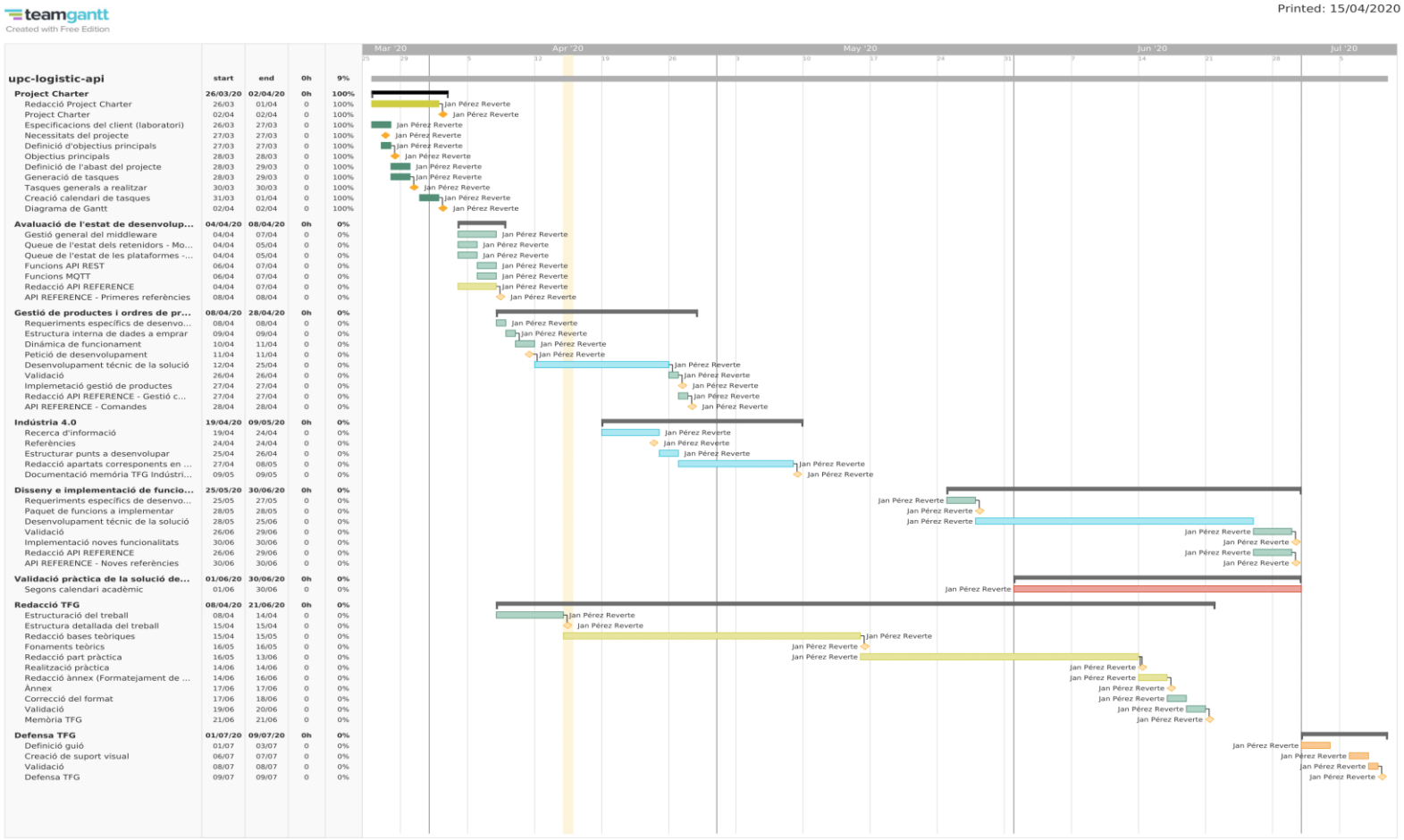

Tabla 8. Diagrama de Gantt del proyecto. Fuente: Elaboración propia.

Capítulo II:

Estado del arte

1. Concepto de Industria 4.0

Según la Real Academia Española (RAE) el término "industria" define:

"Conjunto de operaciones materiales ejecutadas para la obtención, transformación o transporte de uno o varios productos naturales."

Esta definición es acuñada y aceptada ampliamente principalmente porque ha descrito de forma precisa los cerca de 200 años con los que cuenta la industria manufacturera mundial. En la práctica, la industria es mutable en el tiempo y dependiente de la sociedad, economía o política que forma un estado, así como su gran reactividad ante los avances tecnológicos. Estos conceptos, dan a lugar las tres grandes revoluciones industriales que ha conocido la humanidad. Cada revolución industrial tiene asociada una fuente energética que permite su desarrollo y operatividad [5]:

- Primera revolución industrial (1820): Operativa basada en el **vapor**
- Segunda revolución industrial (1870): Operativa basada en la **electricidad**
- Tercera revolución industrial (1969): Operativa basada en **petróleo**

Recordando la definición anterior, es posible observar una clara relación que trasciende entre revoluciones. Todas las fuentes energéticas que permitían la evolución, así como las operaciones asociadas, eran **tangibles** (de forma general y reduccionista).

Es considerado el inicio de la tercera revolución industrial la aparición en el mercado de los controladores lógicos programables, que permitían automatizar procesos industriales y aplicar sistemas de control autónomos en las líneas de producción. En consecuencia, aparecieron nuevos modelos organizativos que permitían la introducción de la computación a nivel industrial.

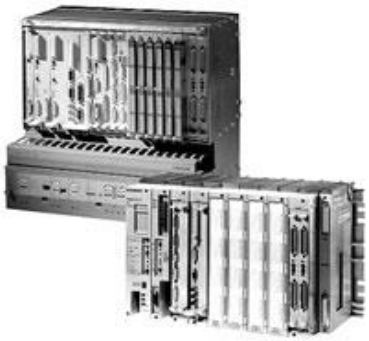

Figura 2. Modicon 084, el primer Programmable Logic Controller (PLC). Fuente: Unicrom.

En la actualidad, tras cerca de 50 años de desarrollo, las tecnologías de la información (TI) han evolucionado de forma exponencial. La irrupción en el mercado de innovadoras soluciones para la interconexión de dispositivos, la recopilación y análisis de datos, o la implementación de modelos de aprendizaje automático (entre otras cuestiones) han llevado a la industria manufacturera global a las puertas de una nueva revolución.

- Cuarta revolución industrial (2016)[3]: Operativa basada en la **información**

En contraposición con las anteriores fuentes energéticas/impulsoras, existe un detalle que desmarca la cuarta revolución industrial respecto a las anteriores: la **intangibilidad** del componente principal que impulsa su operatividad, de acuerdo con Niño-Becerra, Santiago (2017) [5].

Manteniendo un estrecho lazo respecto a la tercera revolución, el principal objetivo de transformar la industria se mantiene: una mejorar sustancial de la **productividad**. Sin embargo, el enfoque y métodos por los cuales esta revolución pretende alcanzar un aumento de la productividad es totalmente distinto.

El clásico concepto asociado de **"con las mismas cantidades de factores productivos y consumibles producir más"** queda obsoleto y reemplazado por **"producir la cantidad necesaria, en el momento conveniente y de la forma precisa"**[4]. Este nuevo paradigma sitúa el análisis de datos (**información**) como motor y fuente impulsora de todo cambio.

[3] El término "Industria 4.0" fue propuesto durante la feria Hannover Messe 2013.
[4] Las anteriores citas textuales corresponden a los argumentos textuales expuestos por el Dr. Santiago Niño-Becerra en el Foro Económico del Diario Montañés (2017).

Figura 3. Evolución de la industria. Fuente: Medium.

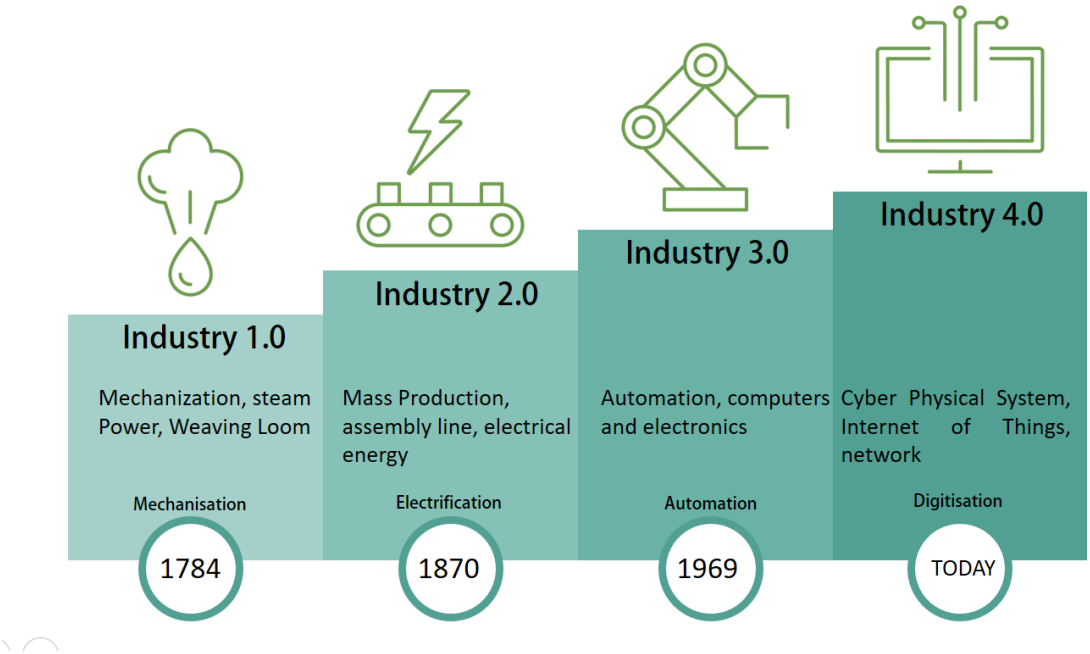

Figura 4. Evolución de la Industria. Fuente: Medium

La denominada “Industria 4.0” plantea una reestructuración transversal que abarca todos los estamentos definidos que componen las factorías, introduciendo nuevas estrategias de organización y comunicación [6].

2. Los sistemas ciber-físicos

Los sistemas ciber-físicos[5] (*CPS* por sus siglas en inglés) presentan una clara y estrecha relación con respecto a la cuarta revolución industrial, principalmente por eliminar la brecha existente entre las tecnologías de operación (OT) y las tecnologías de la información (IT). Acordemente, los CPS integran las operaciones técnicas del proceso de fabricación, como realidad-operatividad del proceso productivo, con los sistemas computarizados y las infraestructuras de comunicación [7].

Los sistemas de control embebidos clásicos, en contraposición con los CPS, se diseñan con el objetivo de operar de forma autónoma sin concebir la existencia de otros procesos dependientes o relacionados. Es decir, toda sección/maquinaria que compone una fábrica limita su funcionamiento a las operaciones propias y exclusivas encomendadas, sin guardar relación ni tener en cuenta el estado de otros procesos.

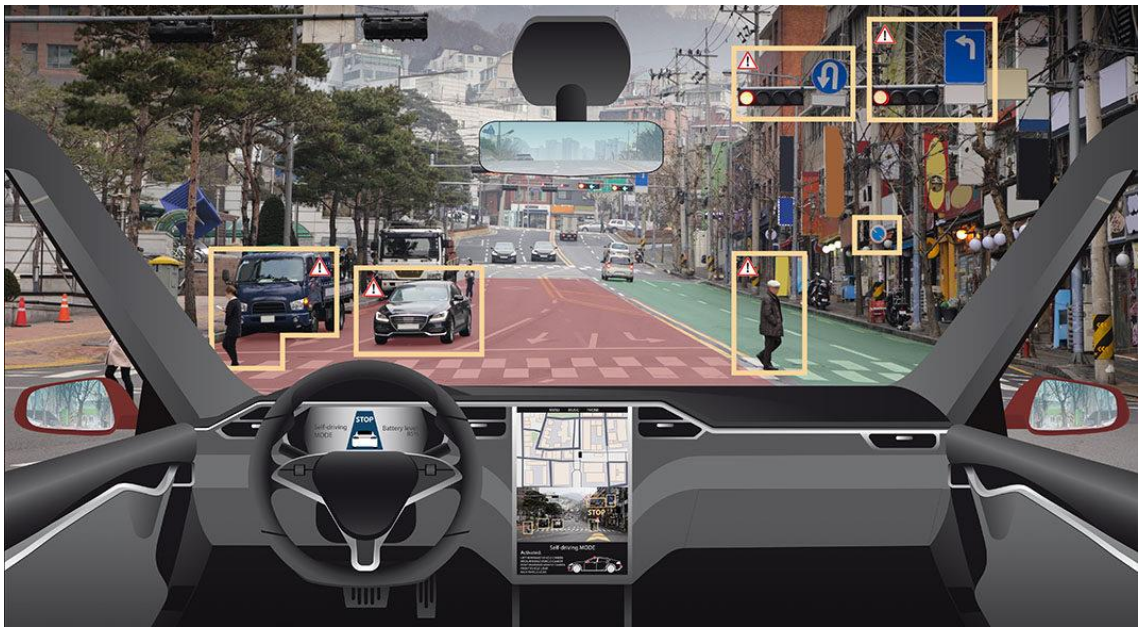

Figura 5. Los coches autónomos/inteligentes representan un claro ejemplo de sistema ciber-físico. Fuente: Fleet People.

Los vehículos autónomos ejemplifican la importancia de los sistemas ciber-físicos. En la Figura 4, es posible observar el reconocimiento que realiza el mismo para analizar

[5] Los sistemas ciber físicos se definen como aquellos dispositivos que integran capacidades de computación, almacenamiento y comunicación con el fin de interactuar con un proceso físico [7].

los obstáculos y elementos que componen la vía. Estos algoritmos de identificación únicamente son posibles de implementar en sistemas basados en computador (ordenador de abordo) y para ejecutarlos requieren imágenes en tiempo real del entorno en el que se encuentra, así como parámetros relativos al estado del vehículo. La instalación de una cámara de visión permite dotar al algoritmo de la información necesaria para analizar y tomar las decisiones pertinentes.

Por tanto, **el claro objetivo de los CPS es eliminar la brecha existente entre lo sensores/actuadores (incluyendo los sistemas de control asociados), representantes del estado del proceso en cuestión, con los sistemas computarizados que no tienen acceso directo al medio.**

Un CPS robusto unido a otros campos de la Industria 4.0 puede dar a lugar a soluciones de todo tipo, a destacar como ejemplo la implementación de gemelos digitales (*Digital twin*) en procesos industriales.

En conclusión, para poder tratar la información recopilada por los sensores, procesarla y actuar en consecuencia, es necesario crear capas de comunicación rápidas y estables entre las OT y las IT, generando un sistema ciber-físico.

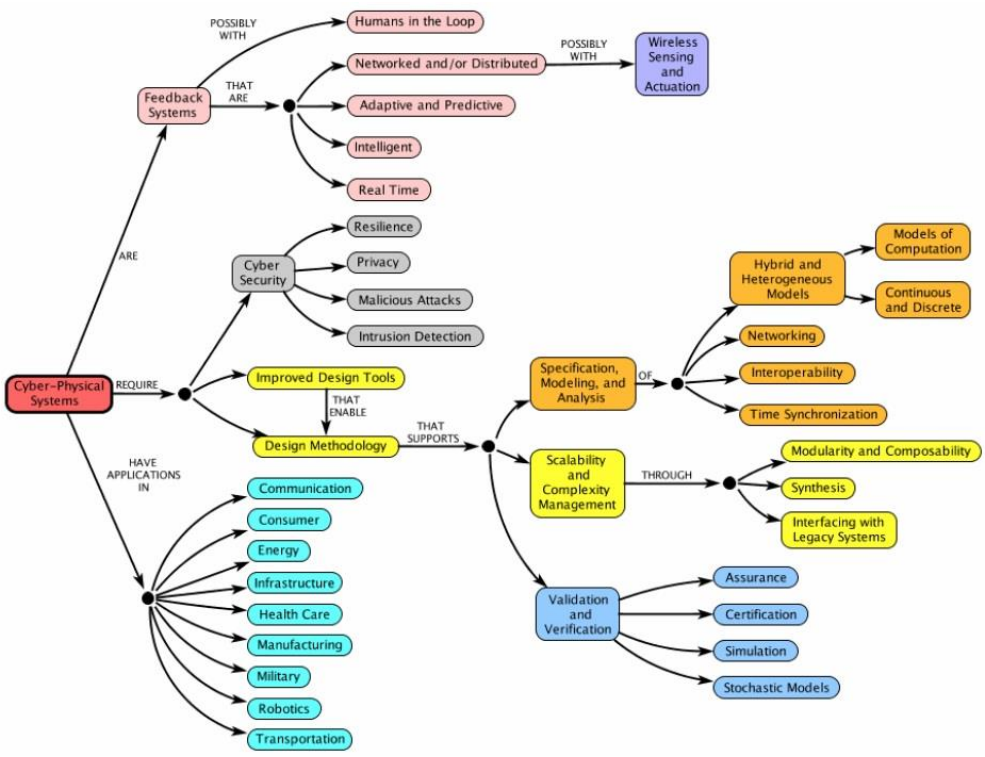

Figura 6. Esquema de factores que componen o guardan relación con los sistemas ciber-físicos. Fuente:

3. Tecnologías relevantes de la Industria 4.0 en este proyecto.

Es necesario entender que la cuarta revolución presenta nuevas formas de organizar las factorías y la lógica de negocio, aunque únicamente presenta conceptos e ideas que necesitan de tecnologías con gran potencial para implementarse.

A modo de clarificación, la primera revolución utilizaba el vapor para generar movimiento, pero era necesario un desarrollo mecánico sin precedentes para poder implementar su uso. De un modo similar, **la industria 4.0 se entiende en forma de concepto y no de implementación en sí misma**. Para ello, son necesarias las tecnologías de actualidad que se exponen a continuación.

Figura 7. Principales tecnologías de la Industria 4.0. Fuente: Pinterest.

Se describen de forma genérica tecnologías que guardan mayor relevancia con este proyecto, pero existen muchas otras de gran implicación en el desarrollo de la cuarta revolución industrial. La fabricación aditiva, la inteligencia artificial o la robótica colaborativa son claros ejemplos [8].

3.1 Integración de sistemas

Una de las premisas clave de la Industria 4.0 es poder integrar los procesos y sectores que componen las fábricas para intercambiar información vital de un modo fiable, rápido y eficiente. Desde una perspectiva tradicional, este traspaso de información suele ser realizado de forma manual o poco automatizada, provocando tiempos de espera. **Es por ende que un descenso significativo de estos tiempo acelera la toma de decisiones, aumentando la productividad y optimizando recursos** [9].

La integración de sistemas dentro del contexto de la Industria 4.0 presenta dos tipos en función de los recursos de la factoría involucrados.

- **Integración vertical**

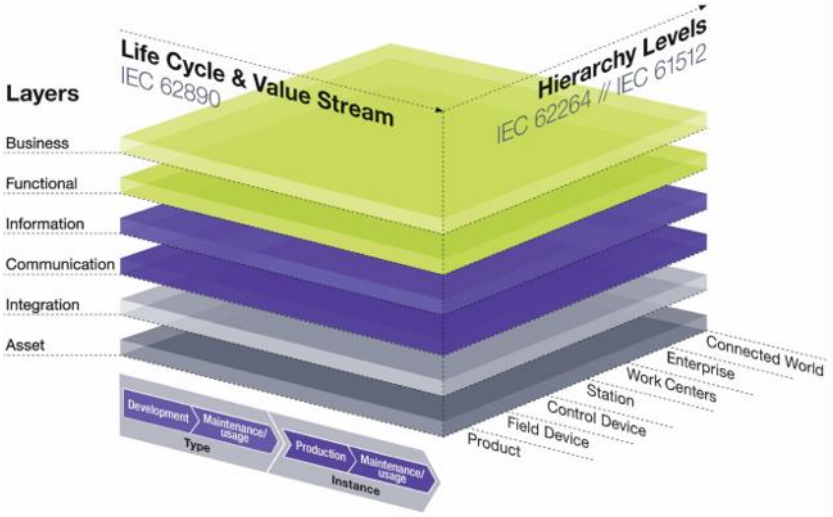

Figura 8. El Reference Architecture for Industrie 4.0 (RAMI 4.0), modelo de implementación hacia la integración total de los sistemas. Fuente: ISA España.

- **Integración horizontal**

3.1.1 Integración vertical

Por un lado, la integración vertical tiene por objetivo **crear un flujo de información continuo entre todos los niveles jerárquicos de la empresa.** Esta integración tiende a comprender desde los sensores implementados a nivel de máquina, hasta el software de gestión de recursos empresariales (*ERP,* por sus siglas en inglés) [9].

El flujo de información varia en tiempos de respuesta y contenido, en función de qué estamentos son conectados. Esto sucede a causa de la existencia de información no relevante para el nivel inmediatamente superior.

3.1.2 Integración horizontal

Por otro lado, este tipo de integración conecta todos los sectores y componentes implicados en la cadena de producción. **"Desde el análisis de mercado, gestión de proveedores, hasta la producción, logística y distribución."** [9].

El principal objetivo es harmonizar el funcionamiento de la instalación, pero no únicamente aquellos sectores propios de la producción. Des de un punto de vista clásico, los módulos de gestión de proveedores, clientes, logística o distribución presentan una clara independencia de la cadena de producción. Mediante la integración horizontal, estos límites desaparecen y pasan a formar de la cadena de forma activa.

3.2 Big Data Analytics

El término macrodatos (*Big Data*) hace referencia a la recopilación, procesado y almacenamiento de grandes conjuntos de datos [10]. La naturaleza de estos grandes conjuntos permite definir cinco características básicas [11].

- **Volumen.** Como consecuencia de la gestión automatiza de la maquinaria y procesos, la cantidad de datos reportados para analizar es masiva.
- **Velocidad.** El flujo constante de información hace que rápidamente los datos queden desfasados, disminuyendo su valor estratégico. Por ello, es necesario analizar de forma rápida y eficiente.
- **Variedad.** Las fuentes de información son dispares y pueden provenir de todo tipo de sistemas conectados.
- **Veracidad.** Esta característica expresa la calidad de la información adquirida del proceso.
- **Valor.** Esta característica refleja el rendimiento que obtiene la empresa de la información tratada.

Esta información puede ser generada en cualquier entorno o proceso. BBVA ha creado un entorno abierto a través del cual las empresas pueden obtener información (anónima) respecto a las transacciones, pagos con tarjeta, o préstamos concedidos a sus clientes [12]. Si bien es cierto que el término *Big Data* es tendencia utilizarlo para emplear este término con la finalidad de acuñar las métricas generadas por el usuario en las TI, su uso puede extenderse a cualquier campo [10].

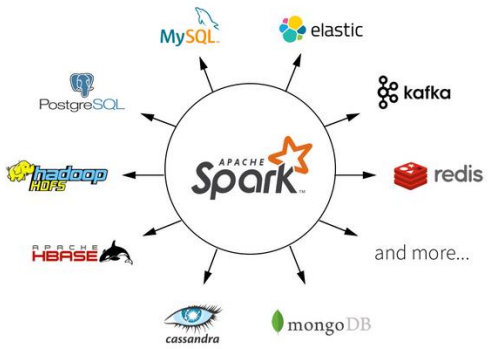

Figura 9. Integración de Apache Spark. Fuente: Apache Foundation.

Los datos generados por los procesos industriales que, tradicionalmente, cumplían el objetivo de conocer el estado del proceso, en la actualidad son relevantes para mejorar la productividad. Según Virgin Athlantic, un Boeing 787 puede generar cerca de medio terabyte de datos por vuelo [13].

Como consecuencia de la masividad de estos conjuntos, es necesario emplear herramientas de procesamiento de datos no tradicionales. Un ejemplo de tecnología al uso es *Apache Spark*.

Las empresas pueden aprovechar esta información generada de forma automática por los procesos, con la finalidad de obtener valor en forma de tendencias o relaciones. En el citado caso del Boeing 787, analizar la información reportada durante el vuelo podría permitir maneras óptimas de reducir el consumo de combustible a partir de los históricos generados.

Actualmente, existen herramientas avanzadas para el análisis de datos, de diferente complejidad y en función del objetivo. Respecto al análisis enforcado a *business inteligence* (históricos de vendas, análisis de beneficios, etc.) existen dos soluciones de gran potencial: Microsoft PowerBI y Tableau.

En un ámbito más enfocado a la ciencia de datos es recomendable el uso de lenguajes de programación. Un claro ejemplo es Python, lenguaje de programación de propósito general pero que se ha convertido en referente para el tratamiento de datos gracias a sus librerías Numpy, Pandas y Matplotlib.

3.3 Cloud Computing

El *Cloud computing* es una nuevo modo de computación dinámica y escalable donde todos **los recursos necesarios (red, servidores, almacenamiento, soporte físico de computación) son virtualizados y ofrecidos como servicio a través de internet** [14].

Éste presenta una arquitectura orientada al servicio y los procesos se ejecutan mediante la computación en paralelo. El proveedor de servicios de *Cloud* (*CSP,* por sus siglas en inglés) es el encargado de gestionar todo el servicio [15].

La infraestructura, mantenimiento y servicios necesarios para su funcionamiento corren a cargo del *CSP*. El cliente únicamente debe hacerse cargo de la implementación de las soluciones que precisa.

En términos generales, esta nueva forma de computación presenta una gran compatibilidad, bajo coste y alta eficiencia que permiten obtener grandes beneficios económicos y empresariales [16].

3.3.1 Características del Cloud Computing

El Cloud Computing presenta una serie de características intrínsecas [15]:

- **Agilidad.** Gran cantidad de servicios soportados en un mismo entorno. A modo de ejemplo, se pueden instanciar máquinas virtuales de diferentes sistemas operativos en cuestión de minutos bajo el mismo entorno de gestión (Proporcionado por el CSP).
- **Fiabilidad.** El mantenimiento de los servidores corre a carga de una entidad especializada externa.
- **Escalabilidad.** Los servicios contratados por el usuario pueden ser incrementados o reducidos en función de las necesidades.
- **Pago por uso.** Únicamente se factura al usuario por los servicios y recursos usados.
- **Servicio bajo demanda.** Los servicios pueden ser contratados o rescindidos en base a las necesidades. Por tanto, presenta modularidad.
- **Monitorización.** Se puede visualizar el estado en tiempo real de los procesos en ejecución mediante servicios web.
- **Seguridad.** Se utilizan técnicas de encriptación para proteger los datos sensibles del cliente.

- **Recursos comunes.** Los procesos de miles de usuarios se ejecutan en paralelo, de forma independiente, bajo la misma infraestructura.

3.3.2 Valoración de las principales características del Cloud Computing

En el apartado anterior, se definen las características propias del *Cloud Computing,* desde una perspectiva neutra. En la siguiente tabla, se realiza una clasificación a modo de valorativo desde un punto de vista empresarial [15].

A favor	Descripción
Menor coste de IT.	Reducción de recursos propios necesarios para mantener las soluciones.
Escalabilidad.	En función de las necesidades, se pueden contratar más o menos recursos en cuestión de minutos.
Agilidad	Posibilidad de configurar entornos complejos en cuestión de minutos.
Continuidad del negocio.	Respaldar datos de valor en la nube como medida de contingencia.
Flexibilidad de acceso.	Permite acceder a los datos de forma descentralizada.
Almacenamiento ilimitado.	Podría definirse como parte de las ventajas de la escalabilidad, aunque es importante destacar la no dependencia de invertir en almacenamiento periódicamente.
Velocidad	El servicio está preparado para servir gran cantidad de datos por segundo.

Tabla 9. Características a favor del Cloud Computing. Fuente: Zhang, Qi (2010) y elaboración propia.

En contra	Descripción
Seguridad	Aún con la existencia de fuertes medidas de contención y seguridad informática, con el objetivo de proteger la integridad de la información, existe el riesgo de sufrir delitos de ciberdelincuencia.
Transferibilidad	Migrar contenido entre *GSP* es complejo y poco desarrollado. Por tanto, puede suponer un problema a largo plazo.
Caída	En caso de caída de los servidores, multitud de procesos pueden verse afectados.
Dificultad	Cada plataforma requiere formación específica para poder llevar a cabo implementaciones.
Privacidad	Todos los activos en información se encuentran a disposición del usuario y, por consiguiente, del GSP.
Control limitado	El usuario no tiene acceso a la gestión de la infraestructura.

Tabla 10. Características en contra del Cloud Computing. Fuente: Zhang, Qi (2010) y elaboración propia.

3.3.3 Modelo de entrega de servicios

De forma genérica, los *CSP* entregan al usuario tres tipos de servicios para su explotación.

- *Software as a Service (SaaS)*
- *Platform as a Service (PaaS)*
- *Infrastructure as a Service (IaaS)*

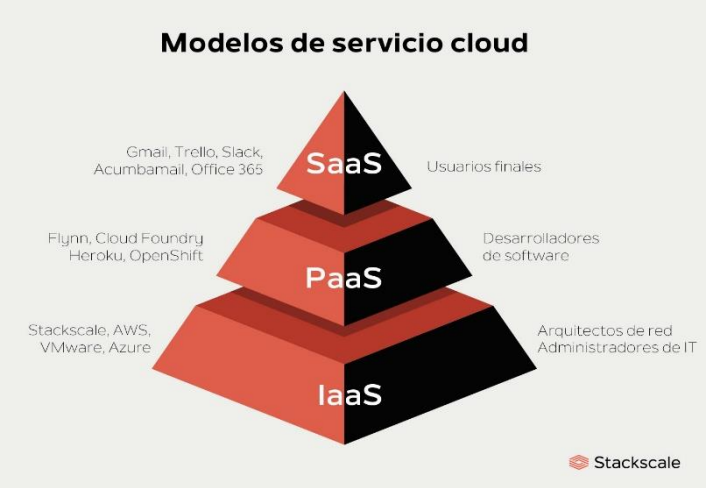

Figura 10. Modelos de servicios del Cloud Computing. Fuente: Stackscale.

Estos servicios pueden ser representados mediante estructura piramidal. Cada modelo de servicio requiere de los recursos inmediatamente inferiores, a efectos prácticos. Por ende, la principal diferencia existente entre ellos es el usuario al que se enfoca el modelo, así como las opciones de configuración de todo el entorno disponibles.

Software as a Service (SaaS)

Este modelo de servicio, también referenciado como "*software* bajo demanda", permite al usuario hacer uso de aplicaciones desarrolladas por el proveedor, bajo la contratación de una licencia comercial. Presenta ventajas heredadas del uso de *Cloud,* como son el uso y el acceso de descentralizado de la aplicación. Un claro ejemplo pueden ser los programas ERP (Enterprise Resource Planning) en sus versiones *Cloud*. Integran todas las funciones necesarias para la gestión empresarial en una única solución modular. Las principales empresas que ofrecen estos servicios son Salesforce, SAP, SAGE, Oracle ERP.

Platform as a Service (PaaS)

Este modelo de servicio ofrece a los desarrolladores el entorno necesario para poder ejecutar sus propias aplicaciones en la nube. Este entorno consta de sistemas

Figura 11. Logo de Google App Engine. Fuente: Google.

operativos, bases de datos, entornos de ejecución de lenguajes de programación y servidores web, entre otros. Las aplicaciones empresariales de propio desarrollo son un claro ejemplo. En este modelo, es importante hacer referencia a que dentro de todo el ecosistema que puede ofrecer cada *CSP,* existen aplicaciones para configurar la ejecución del *software* y monitorización del uso de recursos. Este es el caso de Google, en el cual dentro de *Google Cloud Platform* dispone de *App Engine* para ejecutar las soluciones.

Infrastructure as a Service (IaaS)

Este modelo ofrece al cliente la posibilidad de trabajar sobre la propia infraestructura del proveedor. Esta se compone del almacenamiento, la red, *firewalls* de acceso, entre otras opciones. También permite configurar recursos como los balanceadores de carga de los servidores, direcciones IP estáticas o redes privadas locales. **Se trata del modelo más requerido por las compañías de IT.** Éstas necesitan de la infraestructura para ofrecer un servicio propio a sus clientes. Un claro ejemplo de proveedor de IaaS es Amazon EC2.

3.3.4 MongoDB ATLAS

MongoDB ATLAS es un servicio PaaS de base de datos global en la nube [17]. Es posible gestionar los recursos mediante el servicio web. Utiliza como IaaS los principales proveedores globales de *Cloud* como Microsoft Azure, GCP o AWS.

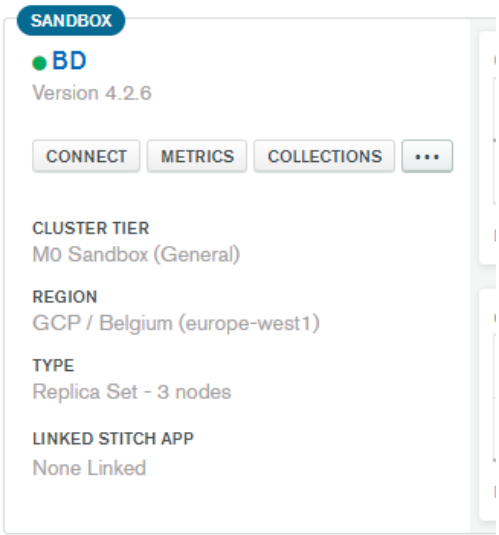

Figura 12. Panel de control de un clúster de MongoDB ATLAS. Fuente: Elaboración propia.

Garantiza una gran disponibilidad, escalabilidad y compromiso respecto a los principales entandares de seguridad de los datos y privacidad. Presenta un amplio ecosistema de *drivers*, integraciones y herramientas que permiten gestionar de forma óptima la base de datos en tiempos mínimos.

MongoDB ATLAS permite al desarrollador crear clústeres[6] de datos. Existe una gran cantidad de librerías[7] de comunicación para que todas las aplicaciones puedan comunicarse al clúster independientemente del lenguaje utilizado.

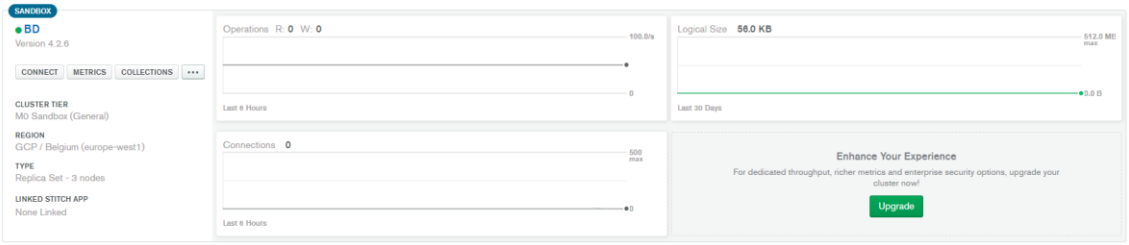

Figura 13. Vista general del tráfico de un clúster e MongoDB ATLAS. Fuente: Elaboración propia.

3.4 Industrial Internet of Things (*IIoT*)

El término Internet of Things (IoT) describe la conexión ubicua de objetos cotidianos a Internet. Como consecuencia de esta característica, estos dispositivos se denominan coloquialmente “objetos conectados” u “objetos inteligentes”.

Desde electrodomésticos (cafeteras, lavadoras, etc.) hasta complementos (camisetas, relojes, etc.), todos estos elementos tienen funciones distintas, aunque comparten una propiedad: **la conectividad**. Esta característica permite adquirir datos del proceso en cuestión y transmitir esta información, para su posterior visualización/análisis.

Un concepto clave respecto al IoT es el despliegue a gran escala (millones) de dispositivos inteligentes con sensibilidad del entorno, con el objetivo poder registrar información relevante en la toma de decisiones u identificación de dinámicas/patrones.

[6] Se define clúster un sistema distribuido de ordenadores unidos entre sí por una red de alta velocidad, con un comportamiento al uso de un único servidor. Los recursos de computación son compartidos entre los usuarios [32].
[7] Una librería responde a un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca. [33]

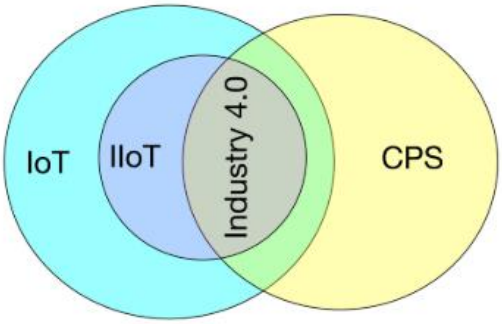

Figura 14. La Industria 4.0 como confluencia de entre IIoT y CPS. Fuente: Sisinni, Emiliano (2018).

Se espera que en el año 2021 existan cerca de 28 billones de objetos conectados en el mundo [18].

Dotar a los sensores industriales de conectividad y aplicar los conceptos mencionados anteriormente se conoce como Industrial Internet of Things (*IIoT*), contemplado como un subgrupo dentro del IoT. El *IIoT* puede producir grandes mejoras en el mantenimiento y seguridad de las factorías, habilitando la interconexión de los sistemas físicos con los digitales, también conocido como sistemas ciber-físicos o *CPS* (29).

TABLE I
COMPARISON BETWEEN CONSUMER IOT AND IIOT

	Consumer IoT	Industrial IoT
Impact	Revolution	Evolution
Service Model	Human-centered	Machine-oriented
Current Status	New devices and standards	Existing devices and standards
Connectivity	Ad-Hoc (infrastructure is not tolerated; nodes can be mobile)	Structured (nodes are fixed; centralized network management)
Criticality	Not stringent (excluding medical applications)	Mission critical (timing, reliability, security, privacy)
Data Volume	Medium to High	High to Very High

Figura 15. Comparación entre el IoT y el IIoT. Fuente: Sisinni, Emiliano (2018).

3.4.1 Arquitectura e implementación

Existen diferentes arquitecturas[8] que permiten implementaciones coherentes del *IIoT*, allanando el camino hacia nuevos métodos organizativos y de servicio [18]. Estas arquitecturas requieren de modelos más específicos, en sintonía con las directrices

[8] Expuestas en “Capítulo III: Arquitecturas”.

principales, pero que permiten una implementación. En referencia al *IIoT*, es ampliamente aceptado el "*three-tier pattern*" (modelo de tres capas).

- **Edge.** Define el dominio de los sensores inteligentes, actuadores y controladores. Respecto estos dispositivos, que interactúan unos con otros, es importante destacar el estándar de comunicación abierto **IO-Link**, que permite comunicaciones punto a punto entre dispositivos, a diferencia de los protocolos clásicos como PROFIBUS que realmente son buses de campo. Forman parte del Edge los *IIoT Gateways,* que habilitan el enlace entre capas.
- **Platform**. Realiza la función de enlace entre la información de los dispositivos en el borde con las aplicaciones de gestión empresarial y análisis de datos. Una premisa importante al respecto, es asegurar la fiabilidad y seguridad de la transmisión. Los entornos de *Cloud* pertenecen a esta capa.
- **Enterprise**. Dominio de aplicación. Comprende el análisis, comprensión y visualización de la información ya filtrada.

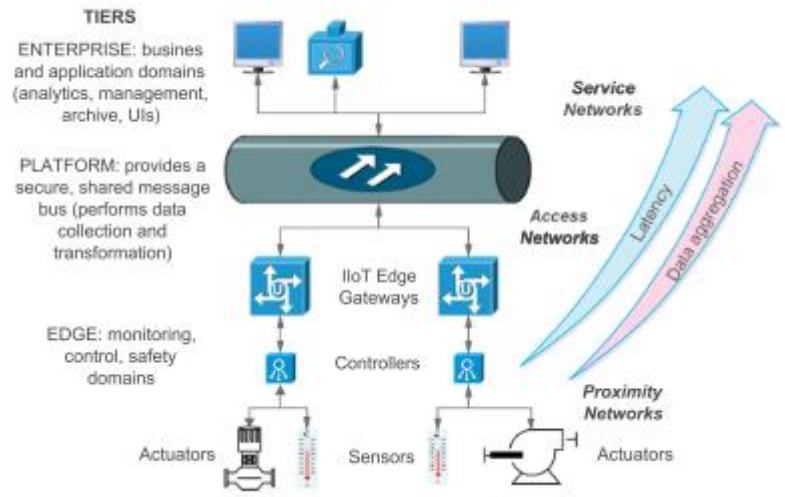

Figura 16. Modelo de arquitectura del IIoT. Fuente: Sisinni, Emiliano (2018).

3.4.2 Conectividad y protocolos

En función de la arquitectura *IIoT* definida y el volumen de dispositivos, existen diferentes opciones de conectividad. También influyen las necesidades y limitaciones operativas a bajo nivel, implicando el uso de comunicaciones inalámbricas o cableadas. Uno de los principales objetivos de la conectividad en *IIoT* es evitar la implementación sistemas aislados, apostando por soluciones que habiliten la comunicación e interoperabilidad entre sistemas.

A modo de clarificar las necesidades en el *IIoT*, se establecen tres macro-capas según los datos que procesan:

- **Networking:** Paquetes y tramas
- **Connectivity:** Mensajes
- **Information:** Estructuras de datos

Cada capa lidia con una problemática distinta y presenta diferentes implementaciones. **Como consecuencia de la variedad de soluciones posibles, no existe una estructura definida e inamovible.** Se engloban los diferentes protocolos que compiten en una misma capa, obteniendo la *"hourglass-shaped IIoT protocol stack"*, un diagrama en forma de reloj de arena. Las capas más amplias implican mayor número de posibles implementaciones, mientras que aquellas de menor superficie implican más rigidez. Asimismo, es posible observar una diversificación dentro de las macro-capas.

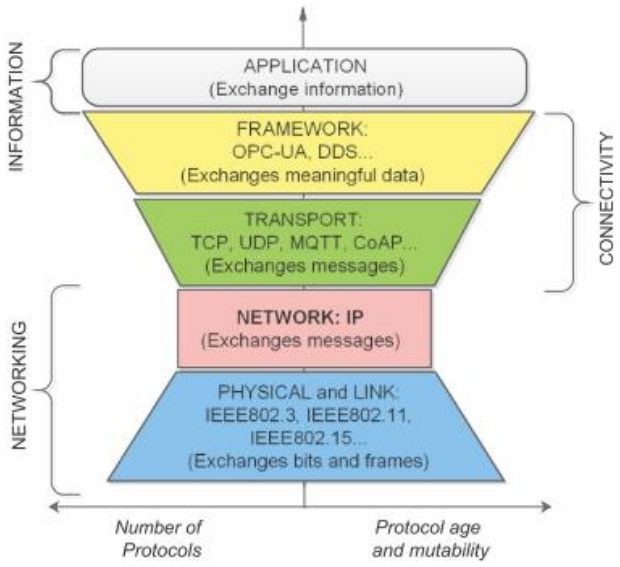

Figura 17. Capas de comunicación en el IIoT. Fuente: Sasinni, Emiliano (2018).

Los protocolos clásicos utilizados en aplicaciones web presentan dificultades intrínsecas para la implementación del *IIoT*, a causa de la propia naturaleza de los datos y la descentralización de los sensores, controladores o *Gateway,* así como todos los otros componentes implicados en el *IIoT* de diferentes capas [18] [19] [20].

El *IIoT* requiere de protocolos de comunicación rápidos, ligeros y con una alta eficiencia, tanto a nivel de recursos como energéticos. Los protocolos clásicos de la "*web stack*" no cumplen estas necesidades, principalmente por el tamaño de los

paquetes, ralentizando la comunicación. En la siguiente figura se puede apreciar la comparativa entre las comunicaciones en IoT y las aplicadas a las aplicaciones web, en base a sus protocolos (bajo el modelo TCP/IP).

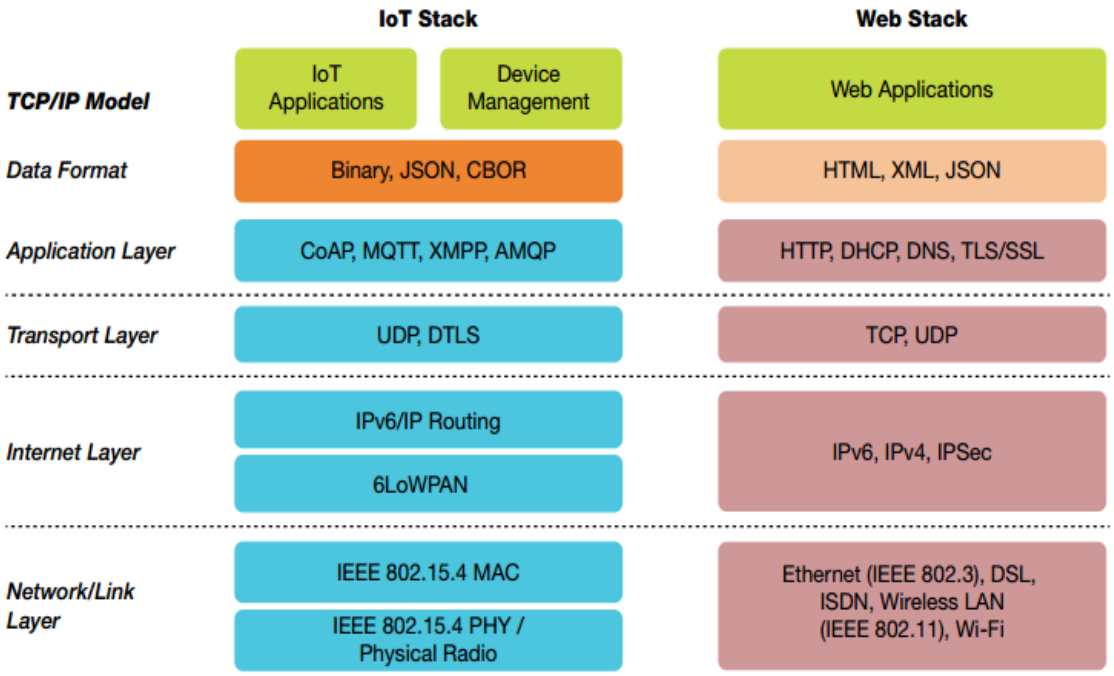

Figura 18. Comparación "*IoT Stack*" versus *"Web Stack"*. Fuente: Aniruddha, Chakrabarti (2015).

Anteriormente se ha mencionado la existencia de protocolos asociados al *IIoT*, permitiendo diferentes formas de implementación para una misma necesidad. Con el objetivo de contextualizar al lector, se presenta un diagrama que engloba los principales protocolos utilizados a nivel industrial y en el ámbito del *IIoT*, en base a la funcionalidad que éstos cubren (esta distribución por capas no es equivalente a ningún modelo, aunque existan ciertas relaciones).

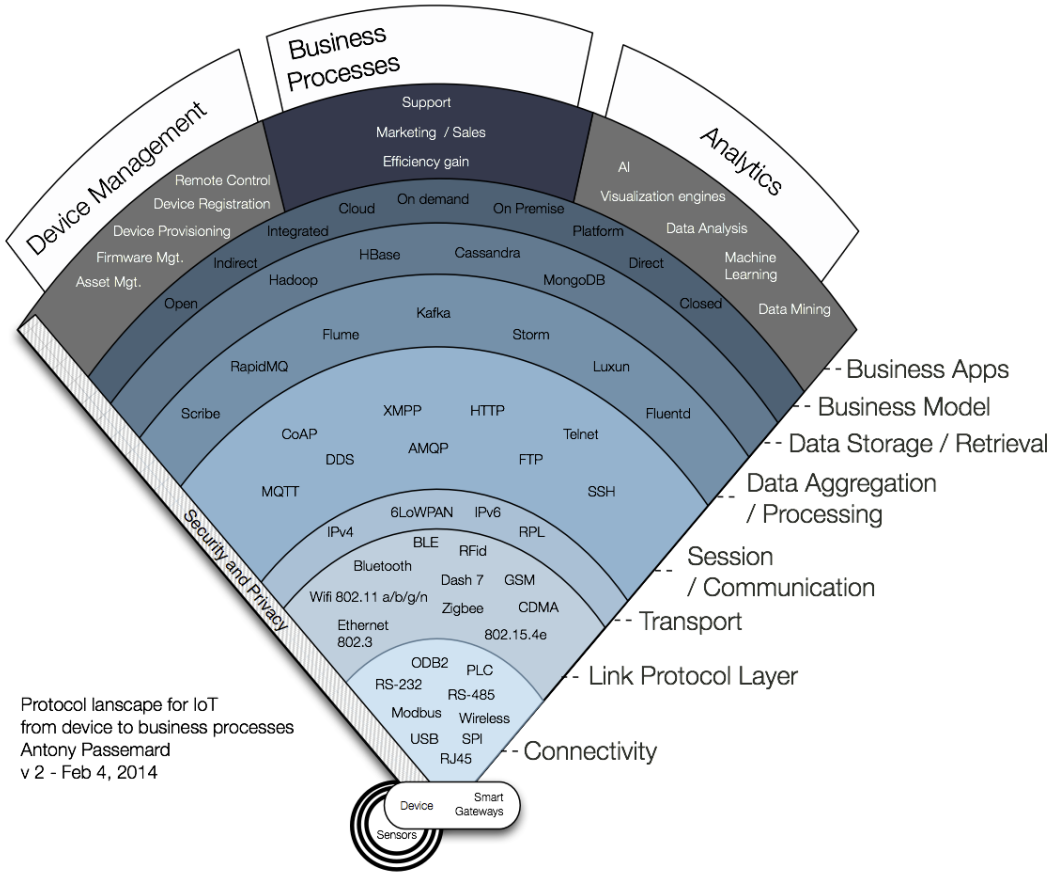

Figura 19. Clasificación de diferentes protocolos según su capa de aplicación. Fuente: Passemard, Antony (2014).

Los niveles más bajos representan las comunicaciones físicas y de enlace entre dispositivos, mientras que las superiores quedan orientadas a funciones de aplicación, gestión de datos y análisis. Las capas intermedias habilitan esta relación (capa de transporte y sesión). A continuación, se detallan brevemente las características de los protocolos relevantes en este proyecto.

Modbus

Modbus es un protocolo de comunicaciones altamente utilizado a nivel industrial situado los niveles 1,2, y 7 del modelo OSI [21]. Presenta dos arquitecturas posibles, maestro/esclavo (RTU, *Remote Terminal Unit*) o cliente/servidor (TCP/IP) [22]. Fue diseñado en 1979 por Modicon para su implementación como protocolo de comunicación con PLCs (*Programable Logic Controller*). De forma que su diseño parte como base para aplicaciones industriales.

Figura 20. Logo de Modbus. Fuente: The Modbus Organization.

Es público y gratuito, fácil de implementar. Permite consultar datos de forma estandarizada independientemente del fabricante o dispositivo en cuestión. Existen diferentes versiones del protocolo con posible implementación:

- **Modbus RTU.** Es la implementación más común del protocolo a nivel industrial. Se utiliza la comunicación serie (RS-232, RS-485) con una trama definida. Se añade una suma de comprobación cíclica (CRC) como mecanismo de comprobación de errores.
- **Modbus ASCII.** Utiliza la comunicación serie, haciendo uso de caracteres ASCII codificando el mensaje. El formato ASCII utiliza un control de redundancia longitudinal (LRC) para la comprobación de errores.
- **Modbus TCP/IP.** Ésta habilita la comunicación Modbus, a través de redes TCP/IP. No necesita de ningún tipo de control de errores, pues las capas que ofrecen el servicio ya realizan esa acción.

Modelo de datos

Modbus requiere de un modelo de datos para acceder a la información, con cuatro tipos de objetos distintos.

Tipo de objeto	Acceso	Tamaño
Discrete Input (Entrada digital)	Lectura	1 bit
Coil (Salida digital)	Lectura/escritura	1 bit
Input register (Registro de entrada)	Lectura	16 bits
Holding register (Registro de retención)	Lectura/escritura	16 bits.

Tabla 11. Objetos de comunicación. Fuente: Modbus Foundation y elaboración propia

Formato de la trama

Las tramas Modbus se conocen como Unidades de Datos de Aplicación (ADU). Las ADU difieren entre versiones del protocolo, aunque mantienen una estructura de datos a encapsular conocida como Unidad de datos del Protocolo (PDU), que contiene los códigos Modbus y la información útil de la comunicación. A continuación, se representa gráficamente cómo se componen las tramas.

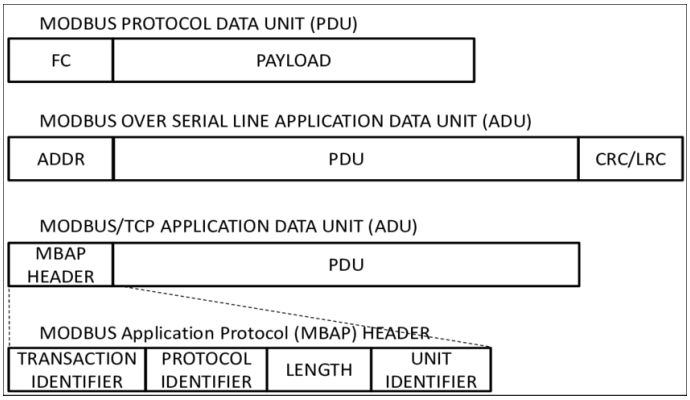

Figura 21. Encapsulado en Modbus. Fuente: The Modbus Organization.

Las PDU se encuentran formadas por un código de función estandarizado entre dispositivos que representa la acción a realizar (por ejemplo, la escritura de un registro), seguidas por los datos que requieren esa acción (valor a escribir).

PDU = Código de función Modbus + Datos.

En el caso de Modbus RTU, la estructura se encapsula por la dirección y el código de redundancia cíclica (CRC), obteniendo la siguiente trama.

Nombre	Longitud (bits)	Función
Inicio	28	Tiempos de silencio
Dirección	8	Dirección de dispositivo
Función	8	Código de función (FC)
Datos	n * 8	Datos para aplicar el FC
CRC	16	Verificación mensaje
Fin	28	Tiempos de silencio

Tabla 12. Trama de Modbus RTU. Fuente: The Modbus Organization.

En el caso de Modbus TCP la estructura de la APU es distinta. Por consiguiente, la trama adquiere la siguiente forma.

Nombre	Longitud (Bytes)	Función
Identificador transacción	2	Sincroniza mensajes cliente/servidor
Identificación protocolo	2	0 en el caso de Modbus TCP
Campo de longitud	2	Número de byte en la trama
Identificación de dispositivo	1	Dirección del esclavo
Función	1	Código de función
Datos	n	Datos de respuesta/comandos

Tabla 13. Trama de Modbus TCP. Fuente: The Modbus Organization.

HTTP

El protocolo de transferencia de hipertexto (*Hypertext Transfer Protocol*, por sus siglas en inglés) es el protocolo de comunicación que permite la transferencia de información en la World Wide Web. Se sitúa en la capa de Aplicación de la comunicación. Fue desarrollado por el World Wide Web Consortium y la Internet Task Force, en una colaboración que culminó en 1999 con la publicación de una serie de estándares RFC, especificando la versión 1.1 del protocolo.

HTTP utiliza la arquitectura de cliente/servidor con acciones de petición/respuesta. Por defecto, los servidores escuchan en el puerto 80. Estas peticiones se realizan mediante métodos de petición (también llamados “verbos”) que permiten realizar acciones sobre los recursos especificados. A continuación, se presentan los más utilizados.

Verbo	Descripción
GET	Solicita una representación del recurso especificado. Obtiene información, en ningún caso tiene ningún otro efecto sobre el servidor.
POST	Envío de datos para que sean procesados por el servidor, especificando el recurso en cuestión.
PUT	Envío de datos para que sean procesados por el servidor, sin especificar

	el recurso en cuestión.
DETELE	Borrado del recurso especificado.

Tabla 14. Verbos de HTTP. Fuente: Elaboración propia.

Es de importancia hacer referencia a la Transferencia de Estado Representacional (REST), protocolo entre sistemas que utilizan directamente HTTP para obtener datos, en cualquier formato (XML, JSON, etc.). Presenta diferentes características fundamentales:

- Es un protocolo cliente/servidor sin estado. Los mensajes HTTP contienen toda la información necesaria para realizar las peticiones
- Opera mediante los métodos de petición claramente definidos de HTTP
- La sintaxis es universal para la identificación de recursos

OPC UA

OLE for Process Control (OPC) es un estándar de comunicación que permite el intercambio de información entre dispositivos de automatización industrial [23]. Esta plataforma posibilita un intercambio de información seguro entre máquinas, con gran compatibilidad entre fabricantes. La OPC Foundation es la responsable del desarrollo y mantenimiento del estándar.

Con la introducción de las arquitecturas orientadas a servicios en el ámbito de la producción, aparecieron nuevos retos en términos de seguridad y modelado de datos. Para ello se desarrolló OPC Unified Architecture (OPC UA), un nuevo estándar que permite ir más allá y conectar los procesos industriales con las aplicaciones de gestión.

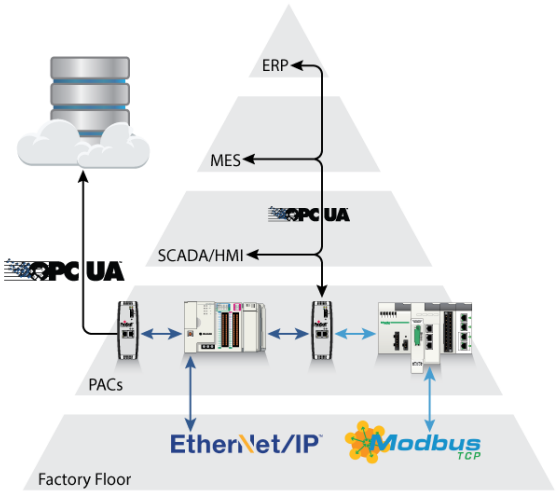

Figura 22. Integración de OPC UA en la arquitectura de comunicaciones. Fuente: ProSoft Technology.

MQTT

Message Queuing Telemetry Transport (MQTT) es un estándar de la capa de aplicación. Es un protocolo ligero y rápido, basado en la arquitectura de publicador/subscriptor, que permite la comunicación entre dispositivos [24]. Hace uso de los protocolos TCP/IP y está diseñado para realizar conexiones remotas en las que se necesita el mínimo uso de información por parte del protocolo, bien por las especificaciones del dispositivo o por disponer de una banda de conexión limitada.

En una comunicación MQTT la información se organiza en *topics* o temas, a los que se accede a modo de rutas. Por ejemplo, si se dispone de un sensor de temperatura respecto a una máquina, esta información (previa configuración del publicador) se encuentra e: “maquina1/temp”.

Por ende, se definen tres entidades que median en la comunicación:

- MQTT Broker. Servidor que gestiona la comunicación. Recibe los mensajes publicados y los redirecciona a los clientes subscritos al tema. Mosquitto es un ejemplo [25]
- Publicador. Agente que publica información en un tema determinado
- Subscriptor. Agente subscrito al tema que recibe la información

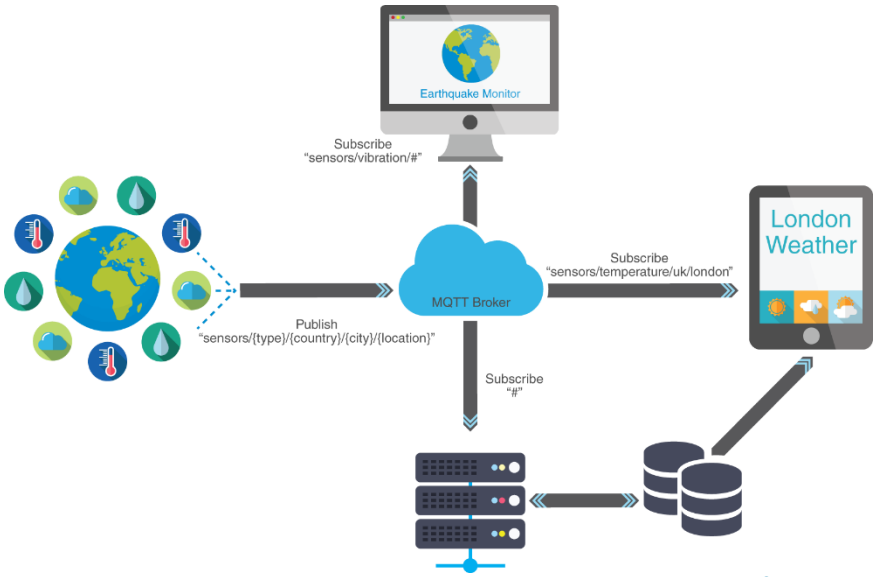

Figura 23. Estructura de comunicación MQTT. Fuente: Stack Overflow.

En MQTT existe la posibilidad de que un mensaje no llegue a su destinatario. Para ello, el protocolo dispone de diferentes niveles de Calidad de Servicio (*QoS*, por sus siglas en inglés). Existen 3 niveles de *QoS*:

- ***QoS* 0 (Como máximo una vez).** Los mensajes no reciben ningún tipo de comprobación, cediendo esa función a las capas inferiores, por tanto, confía en la fiabilidad de TCP. Existe la posibilidad de que algunos paquetes no sean entregados.
- ***QoS* 1 (Como mínimo una vez).** Queda asegurada la llegada de los mensajes, aunque pueden producirse duplicados. El dispositivo receptor confirma la llegada del mensaje al broker, pero si éste último no recibe la confirmación en un tiempo limitado, reenvía el mensaje. Por ende, pueden existir duplicados.
- ***QoS* 2 (Exactamente una vez).** Se asegura que el mensaje llega una única vez al receptor. Se utiliza para información que necesariamente no puede ser duplicada.

Cuanto más alto es el nivel de *QoS*, más compleja se vuelve la comunicación y, en consecuencia, más lenta. Es importante discernir correctamente el nivel de QoS, adecuado para cada tipo de información. Por ejemplo, las métricas de sensores de proceso con una alta tasa de muestreo, la no recepción de un paquete no presenta criticidad. En cambio, perder un mensaje correspondiente a la notificación de una alarma por parte del control de máquina, sí que podría poner en riesgo un sistema. Por tanto, *QoS 2*.

Por defecto, MQTT envía los mensajes de forma plana, sin ningún tipo de cifrado. En base a la sensibilidad de los datos, es prácticamente obligatorio cifrar las conexiones. MQTT dispone de cifrado TLS/SSL (cifrado en la capa de transporte, protocolo TCP) para asegurar la correcta transmisión de información entre nodos.

3.4.3 *IIoT Gateways*

Definición

Un *IIoT Gateway* (o pasarela inteligente) es un **dispositivo físico o programa de *software* que actúa como punto de conexión entre sensores/actuadores, controladores y las plataformas de *Cloud Computing* en el ámbito industrial.** Por tanto, permite la existencia de un flujo de datos bidireccional entre los dispositivos de campo y la gestión empresarial.

En el caso de los sensores industriales, los cuales realizan mediciones de forma prácticamente continua, las pasarelas permiten hacer un primer filtrado, procesado y validación de los datos recogidos. También permite la regulación de información enviada a la nube, con el objetivo de evitar un tráfico de datos innecesario.

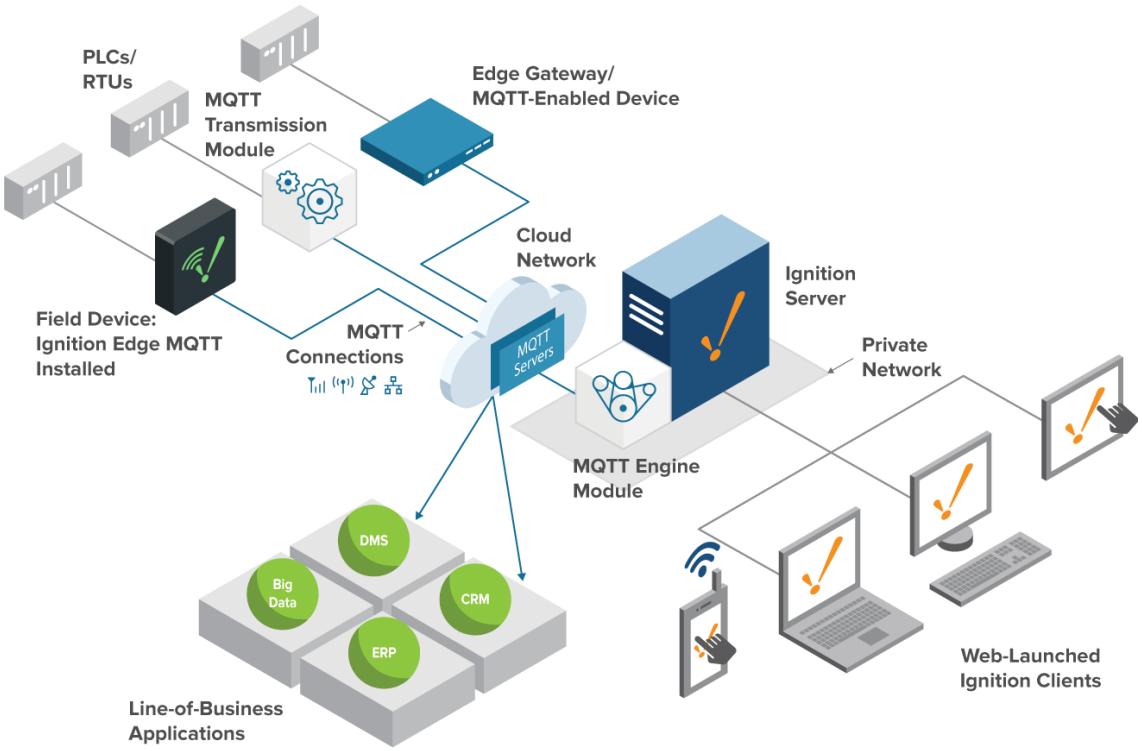

Figura 24. Aquitectura de red con *Gateway*. Fuente: Ignition.

Las pasarelas inteligentes permiten añadir una capa de seguridad, cifrando las comunicaciones entre dispositivos o aislando los controladores de las partes críticas del proceso (motores, resistencias, válvulas) del exterior, controlando el acceso al medio. A efectos prácticos, esta característica funciona como una capa de protección ante la manipulación del proceso, el robo de información o los ataques informáticos.

Otra de las grandes ventajas de utilizar *IIoT Gateways* se basa en la estandarización de las comunicaciones. Tradicionalmente, cada PLC o RTU contaba con un protocolo asociado propio o afiliado al fabricante, dificultando la integración de los dispositivos en la arquitectura. Estas pasarelas disponen de multitud de interfaces y protocolos para conectar con la mayoría de dispositivos, de modo que brindan una capa de abstracción respecto a los dispositivos de campo, estandarizando la comunicación.

También es notable comentar que "*IIoT*" se encuentra, por norma general, asociado a esta interacción con la computación en la nube, aunque los *Gateways* también permiten la comunicación a nivel local entre dispositivos. El término pasarela hace referencia a este hecho, a la capacidad de comunicar información entre protocolos de diferentes ámbitos.

Programación asociada

Sea un dispositivo físico o un módulo en ejecución de servidor compartido, se necesita programar la adquisición de datos, filtrado, junto a otras acciones asociadas. En este sentido, prácticamente todos los lenguajes de programación permiten implementar soluciones acordes a las necesidades. Dentro de todo el paradigma de posibles lenguajes a utilizar, clasificarlos en función de su nivel de popularidad puede conllevar un ahorro en la toma de decisiones, de acuerdo con la experiencia del autor. En una encuesta realizada por la Eclipse Foundation [26], se determinó que lenguaje de programación abarca el mayor número de programadores en el desarrollo de *Gateways*.

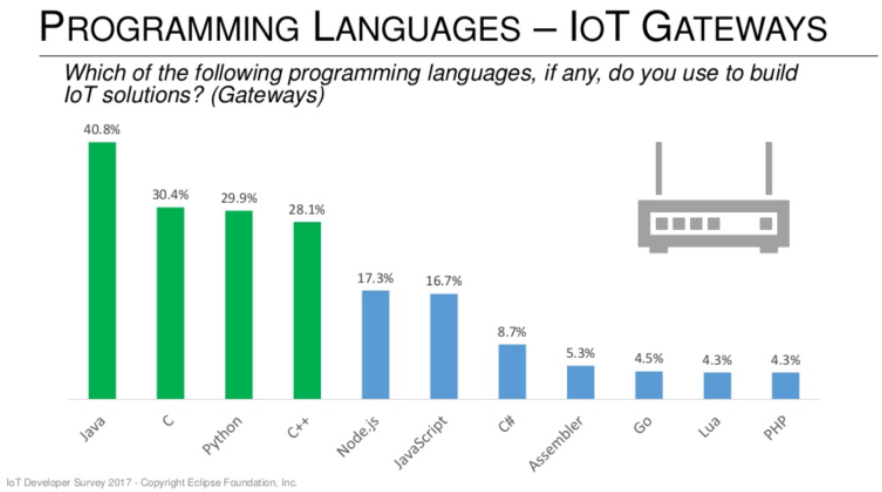

Figura 25. Lenguajes más utilizados en la programación de *IIoT Gateways*. Fuente: Eclipse Foundation.

Java, C, Python y C++ prácticamente superan el 30% de uso, pero es importante destacar como JavaScript y Node.js se sitúan "en una franja media", en torno al 17%. A efectos prácticos, Node.js es un entorno de ejecución de JavaScript fuera del navegador, de forma que pueden ser consideradas por igual. Las tecnologías web están cobrando importancia en el desarrollo de soluciones propias de la Industria 4.0, por contar con características multiplataforma y permitir implementaciones en todos los niveles de organización.

Node-RED

Node.js es un entorno de ejecución de JavaScript fuera del navegador que permite la creación de aplicaciones de red escalables. Es multiplataforma, *open-source*[9], con una gran comunidad de desarrollo. En términos de desarrollo web, permite crear código *backend* (del lado del servidor), en un lenguaje que tradicionalmente se ha empleado para el *frontend.* Es decir, mediante el uso de un único lenguaje, se pueden desarrollar soluciones de toda índole. Node-RED es un ejemplo de herramienta de programación basada en Node.js.

Node-RED [27] es una herramienta de programación visual basada en flujos, desarrollada originalmente por IBM en el año 2013 para conectar dispositivos, APIs y

[9] El software *open source* es un código diseñado de manera que sea accesible al público: todos pueden ver, modificar y distribuir el código de la forma que consideren conveniente [34].

servicios online como parte del IoT. Actualmente, el código fuente y desarrollo es *open-source*, la última versión estable es la 1.0.5.

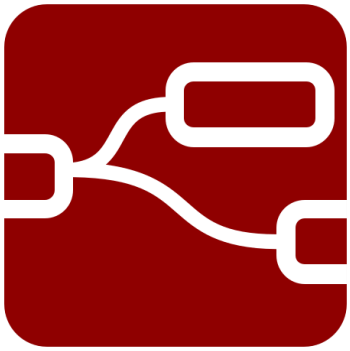

Figura 26. Logotipo de node-RED. Fuente: JS Foundation.

Permite la creación de programas a través de un editor de desarrollo basado en web y ejecutar la solución directamente en el runtime con comodidad. La programación basada en flujos que utiliza node-RED se inventó en la década 1970 por J. Paul Morrison, como una forma de describir el comportamiento de una aplicación mediante cajas negras o "nodos". No es relevante exactamente las que acciones realiza un nodo (más allá de su cometido general), simplemente lo son aquellos datos de entrada/salida que requiere/devuelve.

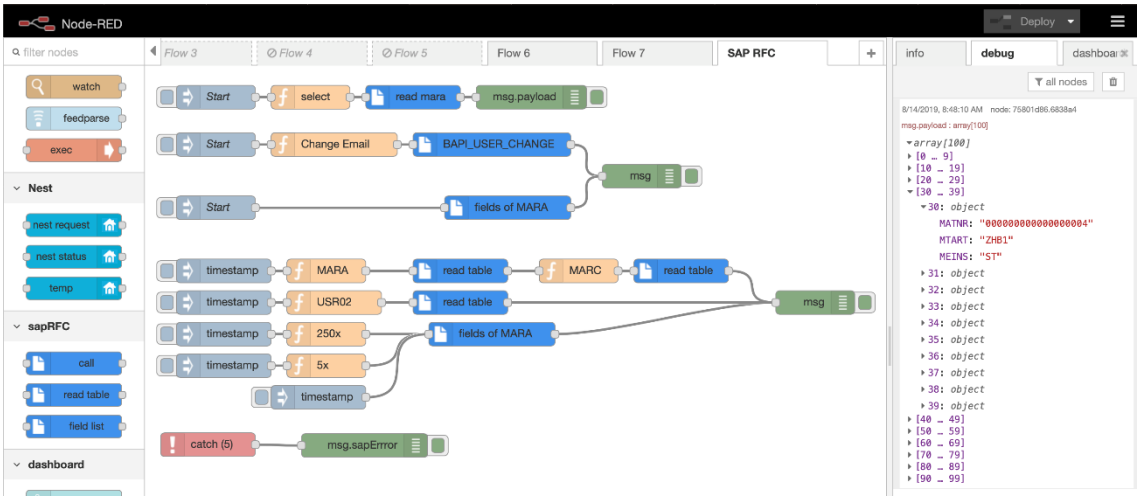

Figura 27. Ejemplo de programa en node-RED. Fuente: sapRFC.

A causa de este paradigma, el programa se organiza en flujos, un conjunto de acciones que se ejecutan de forma independiente y asíncrona. Los "nodos de entrada" inician el flujo dado un acontecimiento (evento, petición, o una ejecución cíclica, entre otros), enviando un mensaje al nodo posterior con la información necesaria para su ejecución.

Este hecho ocurre de forma sucesiva hasta llegar al final del flujo, donde un nodo que no genera mensaje o simplemente no es necesario adjuntar más al proceso.

Node-RED dispone de una comunidad activa de desarrolladores que implementan nuevas librerías de nodos. Mediante ellas, se extiende el uso de Node-RED a aplicaciones de toda índole. También dispone una librería para la creación de *dashboards,* una interfaz gráfica que permite interactuar con el programa, visualizar datos de relevantes o crear formularios de datos, entre otras cosas.

Una de las grandes ventajas de Node-RED es la clara disminución del tiempo necesario para desarrollar una solución (dentro de las limitaciones que comporta), en comparación con otros lenguajes de código.

Permite ser ejecutado en prácticamente todas las plataformas, más allá de necesitar pocos recursos de computación. Por modo que los dispositivos de computación de bajo coste tienden a ser el principal hardware utilizado para su implementación. A continuación, se exponen algunos ejemplos, con énfasis en aquellos preparados para su uso en entornos industriales.

Modelos comerciales

Siemens SIMATIC IOT2040

SIMATIC IOT2040 es una pasarela inteligente multipropósito creada por Siemens, en el marco de la línea de productos desarrollados con el fin de implantar soluciones basadas en la Industria 4.0. Permite obtener información de diferentes dispositivos de bajo nivel, analizar los datos y su posterior reenvío a los receptores (*Cloud,* bases de datos, aplicaciones de gestión empresarial) [28].

Figura 28. Siemens IoT 2040. Fuente: Siemens.

Utiliza Linux Yocto como sistema operativo, y permite ejecutar aplicaciones de alto nivel desarrolladas en Python, C/C++, Java, incluyendo Node.js y Node-RED.

A nivel de *hardware,* incluye interfaces de comunicación típicas de los dispositivos industriales (RS-232, RS-485), ampliamente utilizadas por controladores y equipos industriales.

Las aplicaciones *IIoT* no tienden a requerir de grandes capacidades de computación, motivo por el cual se ha dimensionado sus especificaciones en base a esta característica. Dispone de una carcasa preparada para entornos industriales y soporte para montaje en carril DIN. Permite añadir tarjetas Mini PCIe de expansión para añadir funcionalidad como conectividad Bluetooth, WiFi, nuevas E/S, entre otras.

Revolution RevPi Core 3+

Revolution Pi es una gama de ordenadores industriales desarrollados por Kunbus, con la peculiaridad de estar basados en la Raspberry Pi. La Raspberry Foundation desarrolló la Compute Module 3+ [29], una versión DDR2 SODIMM (a modo de tarjeta independiente con memoria Flash integrada) de la popular Raspberry Pi 3 Model B+. A partir de este dispositivo, se ha desarrollado toda la gama de ordenadores industriales, completamente adaptados a los principales estándares industriales (EN 61131-2, IEC 61131-2). El modelo presentado a continuación es el RevPi Core 3+.

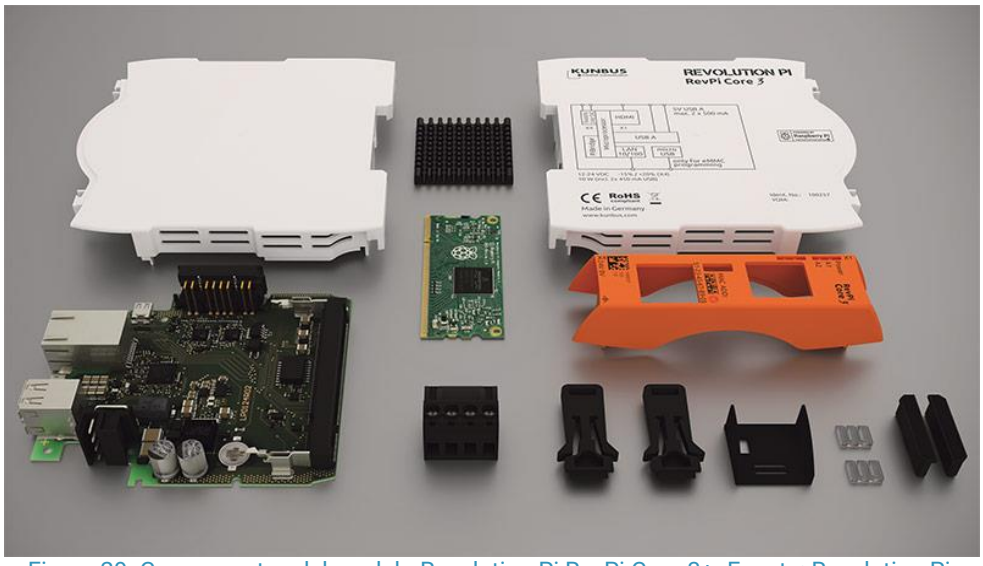

Figura 29. Componentes del modelo Revolution Pi RevPi Core 3+. Fuente: Revolution Pi.

Uno de los mayores puntos fuertes de este dispositivo es la posibilidad de añadir tarjetas de E/S, con la misma facilidad que brinda un PLC mediante unos conectores situados en la parte superior. CODESYS (una plataforma de programación de PLC universal, independientemente del fabricante) permite su programación como cualquier otro dispositivo industrial. Esta funcionalidad aún se encuentra en fase beta. Éste introduce la utilización de recursos de computación a nivel de máquina, por un coste reducido a comparación con los clásicos ordenadores industriales.

Capítulo III: Arquitecturas

1. Arquitectura I: Nueva implementación

Esta arquitectura se presenta para su aplicación en máquinas de nueva fabricación. **Es decir, se requiere la compra de todos los dispositivos de control y medición, aunque no es necesaria la comunicación con controladores o sensores antiguos.**

También puede ser utilizada en aquellas **máquinas que mecánicamente presentan un cierto desgaste, pero se requiere renovar el sistema de control, sensores y actuadores del proceso, por cuestión de obsolescencia o fallos de funcionamiento.**

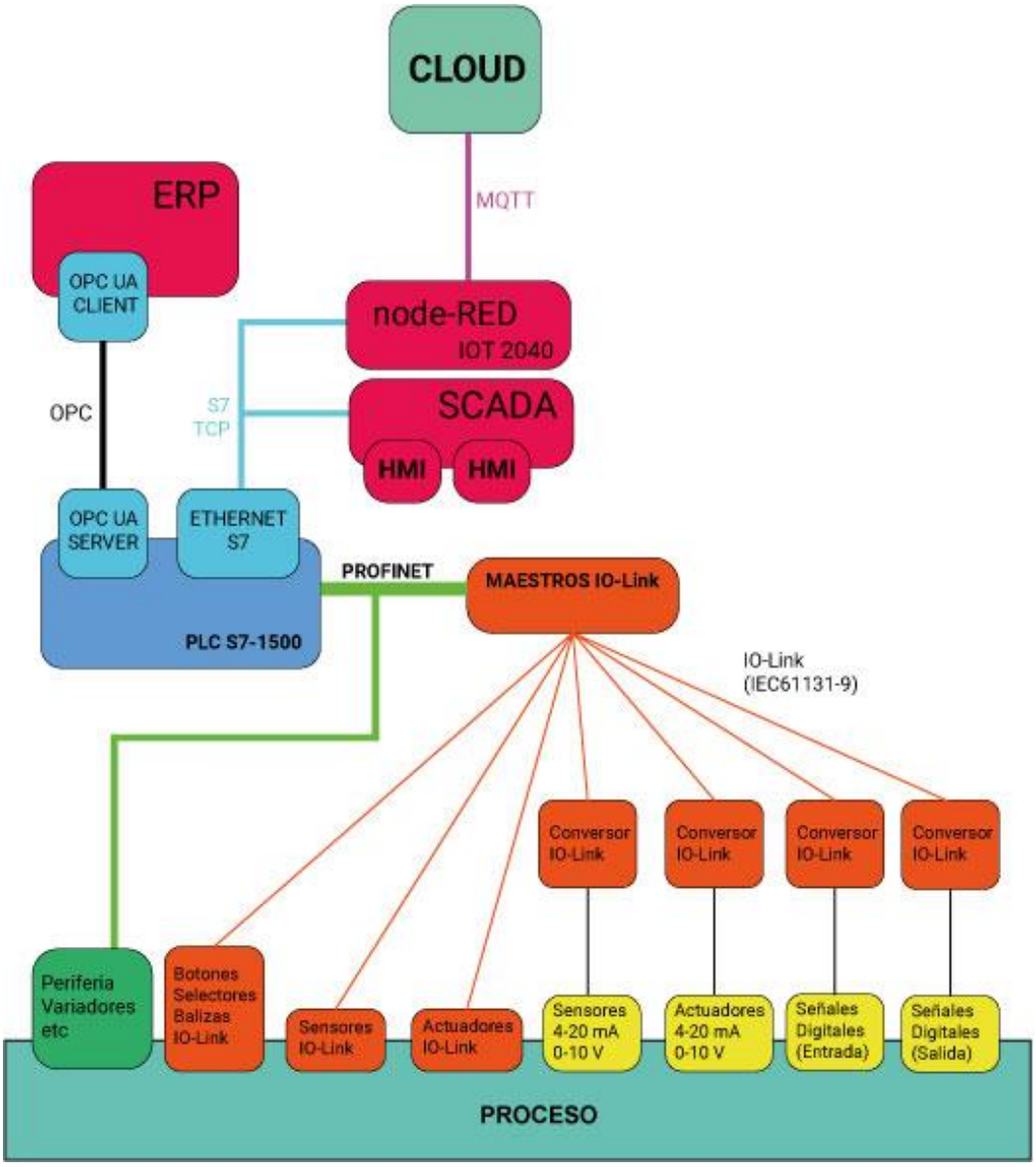

Figura 30. Arquitectura de nueva implementación. Fuente: Elaboración propia.

Referente a la imagen anterior, se pueden distinguir distintos protocolos de a la naturaleza del sensor, aplicación o comunicación, estrechamente ligados dispositivo.

Amarillo	**Sensores y actuadores clásicos de uso industrial, digitales o analógicos.**
Naranja	Sensores, actuadores y dispositivos que utilizan el protocolo IO-Link para intercambiar información.
Verde	Dispositivos que utilizan PROFINET para intercambiar información.
Azul	Uso del protocolo S7 TCP estándar de Siemens para el intercambio de información.
Negro	Comunicación OPC entre dispositivos.
Lila	Intercambio de información mediante MQTT.

Tabla 15. Leyenda de comunicaciones de la arquitectura presentada. Fuente: Elaboración propia.

A nivel de proceso, se observan sensores y actuadores IO-Link (caudalímetros, presostatos, detectores) que transmiten digitalmente la información recopilada del proceso junto con información del estado del sensor, a su respectivo maestro IO-Link. También se incluyen dispositivos de interactuación con los operarios que disponen de comunicación IO-Link (balizas, botoneras, selectores).

A causa del escaso rodaje de IO-Link en comparación con otros protocolos industriales, puede que existan sensores necesarios para el proceso que no dispongan de la capacidad de comunicarse mediante este protocolo. Para ello, se contempla el uso de conversores que transforman las señales digitales/analógicas a IO-Link, con tal de poder centralizar las comunicaciones bajo un mismo protocolo.

Los maestros IO-Link tienen capacidad para albergar 8 sensores/actuadores (la comunicación se realiza de punto a punto, a diferencia de los buses de campo). Su función principal es centralizar el valor de estado de los sensores/actuadores, a modo de pasarela con los sistemas que requieren la información, concretamente, un PLC. Cabe recalcar el uso como plataforma para la gestión de los parámetros de los sensores. Se ha dimensionado un maestro de conectividad PROFINET (se disponen de otros protocolos en función de las necesidades), con el objetivo de que los PLC

puedan acceder a las métricas de los sensores. **Por tanto, se presenta el maestro como un medio para la obtención del estado de los sensores y medio para acceder a los actuadores.**

Referente al sistema de control, se ha dimensionado un PLC Siemens S7-1500 PROFINET. El uso de un protocolo Ethernet Industrial (PROFINET) presenta grandes ventajas respecto al uso de buses de campo clásicos (PROFIBUS). A continuación, se comparan las características más críticas en diseño de la instalación en base a los dos protocolos comentados.

	PROFIBUS	PROFINET
Tipología	Bus de campo	Red
Modelo	Maestro/Esclavo	Cliente/Servidor
Capa física	RS-485	Ethernet
Telegrama	244 bytes	1440 bytes
Fiabilidad	Fallo en un esclavo implica la caída de todo el bus.	Únicamente afecta al dispositivo en fallo, la red funciona correctamente.
Velocidad máxima de TX.	12 Mbit/s.	1 Gbit/s.
Escalabilidad de la red	Máximo de 127 dispositivos por bus.	Dispositivos ilimitados.
Ejes (Motion Control)	30 ejes	>150 ejes

Tabla 16. Comparación PROFIBUS versus PROFINET. Fuente: Elaboración propia.

Como se ha comentado con anterioridad, esta arquitectura asume el uso de sensores/actuadores IO-Link. En caso de requerir comunicación con variadores de velocidad, encoders, o módulos de periferia descentralizada, el uso de PROFINET habilita a que todos estos sistemas puedan formar una parte del mismo control.

El motivo principal por el que se ha dimensionado un PLC S7-1500 es que integra un servidor OPC en el propio controlador, sin requerir el uso de un PC Industrial, con las complicaciones que implica (más dispositivos, coste de los dispositivos, licencia del *software* OPC). El uso de un servidor OPC habilita la comunicación con plataformas de gestión de recursos empresariales (ERP, por sus siglas en inglés), pero también con todo tipo de aplicaciones de propio desarrollo que dispongan del driver cliente OPC.

Por otra parte, los controladores Siemens disponen de un protocolo propio conocido como S7, basado en TCP/IP. Existen aplicaciones SCADA (Progea Movicon NEXT es un ejemplo) que hacen uso de S7 TCP para comunicarse con el controlador. Mediante ellas, se puede parametrizar la instalación, accionamientos e interactuar con el PLC.

De forma complementaria, se ha añadido un dispositivo Siemens IOT2040 a modo de *IIoT Gateway,* permitiendo el volcado de información relevante del proceso a servidores en la nube a través de node-RED. Los proveedores *Cloud* permiten el intercambio de métricas de proceso mediante el protocolo MQTT.

A continuación, se presenta las ventajas/inconvenientes de utilizar la arquitectura descrita.

Ventajas
Uso de protocolos actualmente en plena expansión.
Velocidad en las comunicaciones, seguridad, fiabilidad.
Presenta gran escalabilidad, posibilidad de ampliaciones a largo plazo.
Potencia y capacidad de las CPU Siemens S7-1500.
Digitalización de los valores a nivel de sensor.
Substitución de sensores sin necesidad de parametrizar el nuevo componente.

Tabla 17. Ventajas del uso de la arquitectura expuesta. Fuente: Elaboración propia.

Inconvenientes
Coste material asociado relativamente alto.
Tecnologías en plena expansión, puede dificultar acceder a información al respecto.
Sistema complejo, compuesto por protocolos muy distintos.
Desarrollador PLC con altos conocimientos en tecnologías de la información.

Tabla 18. Inconvenientes del uso de la arquitectura expuesta. Fuente: Elaboración propia.

2. Arquitectura II: Existencia de buses de campo

Verde	**Dispositivos que utilizan PROFINET para intercambiar información.**
Azul	Uso del protocolo S7 TCP estándar de Siemens para el intercambio de información.
Negro	Comunicación mediante librerías y drivers.

Morado	Uso de una línea PROFIBUS.

Esta arquitectura se presenta para aquellos casos en que se requiere **realizar ampliaciones en la planta, manteniendo la periferia y dispositivos de la línea PROFIBUS.**

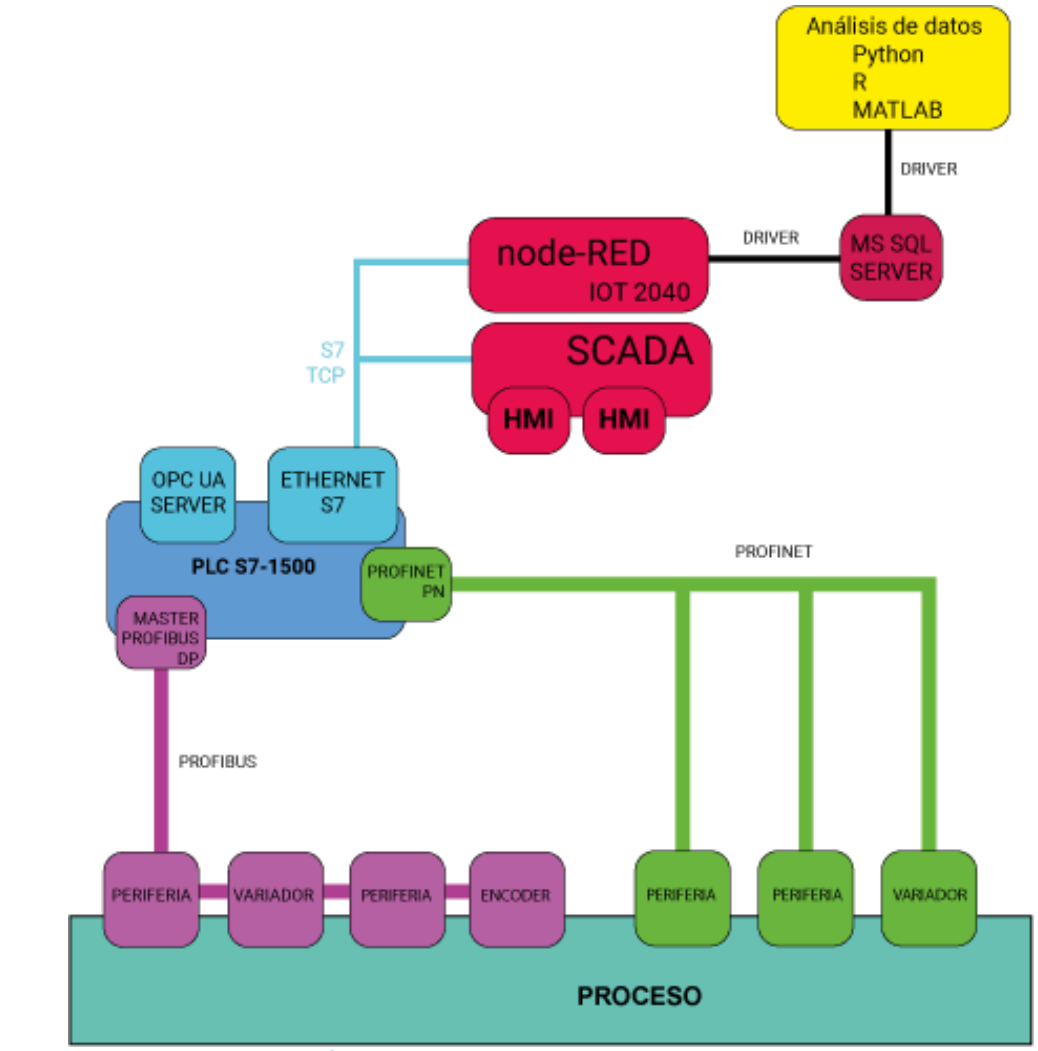

Figura 31. Representación gráfica de la arquitectura con buses de campo. Fuente: Elaboración propia.

Tabla 19. Leyenda de comunicaciones en la arquitectura presentada. Fuente: Elaboración propia.

Cabe destacar como principal característica de esta arquitectura la coexistencia de un bus de campo y una red ethernet industrial, bajo la misma instalación. En la línea PROFIBUS, se mantienen todos los nodos propios del anterior sistema de control, con la diferencia que el maestro PROFIBUS es un S7-1500 con dicha interfaz. Por tanto, se

genera un ahorro sustancial en costes, gracias a la reutilización de todas las cabeceras y dispositivos PROFIBUS. Si bien es cierto, se realiza una inversión en la programación asociada a éstos.

En el caso de la red PROFINET, se añaden las ampliaciones necesarias en la instalación (periferia, variadores, entre otros). Mediante el uso simultáneo de los dos protocolos, se permite una transición gradual de PROFIBUS a PROFINET conforme sea necesario. Por ejemplo, en caso de un fallo en una cabecera PROFIBUS las señales asociadas pueden ser colocadas en una cabecera de periferia PROFINET de forma rápida, puesto que únicamente es necesario declarar la nueva cabecera de la red y cambiar las direcciones de memoria.

Mediante el protocolo S7 TCP es posible comunicar el controlador con diferentes *softwares* SCADA y HMI. De forma paralela, bajo el mismo protocolo S7 TCP, node-RED se comunica con el controlador. El objetivo de utilizar node-RED en esta instalación es realizar inserciones en una base de datos Microsoft SQL Server, con métricas del proceso para su posterior análisis. Para ello, es posible utilizar diferentes lenguajes o programas. MATLAB y Python son ejemplos.

A continuación, se presentan las principales ventajas e inconvenientes de esta arquitectura.

Ventajas
Coste reducido en base a reutilizar la periferia existente.
Transición gradual a PROFINET.
Programación útil, reutilización de código en el cambio a PROFINET.
Potencia y capacidad de las CPU Siemens S7-1500.

Tabla 20. Ventajas del uso de la arquitectura expuesta. Fuente: Elaboración propia.

Inconvenientes
CPU de altas especificaciones, precio muy elevado.
Archivos GSD de algunos dispositivos PROFIBUS podrían quedar obsoletos y no ser programables desde *softwares* actuales.

Tabla 21. Inconvenientes del uso de la arquitectura expuesta. Fuente: Elaboración propia.

3. Arquitectura III: Centralizar la comunicación con diferentes PLC

Esta arquitectura se orienta a instalaciones que disponen de diferentes PLC y se requiere crear una capa para estandarizar la comunicación, entre aplicaciones de gestión y los controladores. No es necesario realizar ningún tipo de ampliación a nivel de control, de forma que requieren diferentes interfaces y protocolos para abarcar el mayor número de controladores posibles.

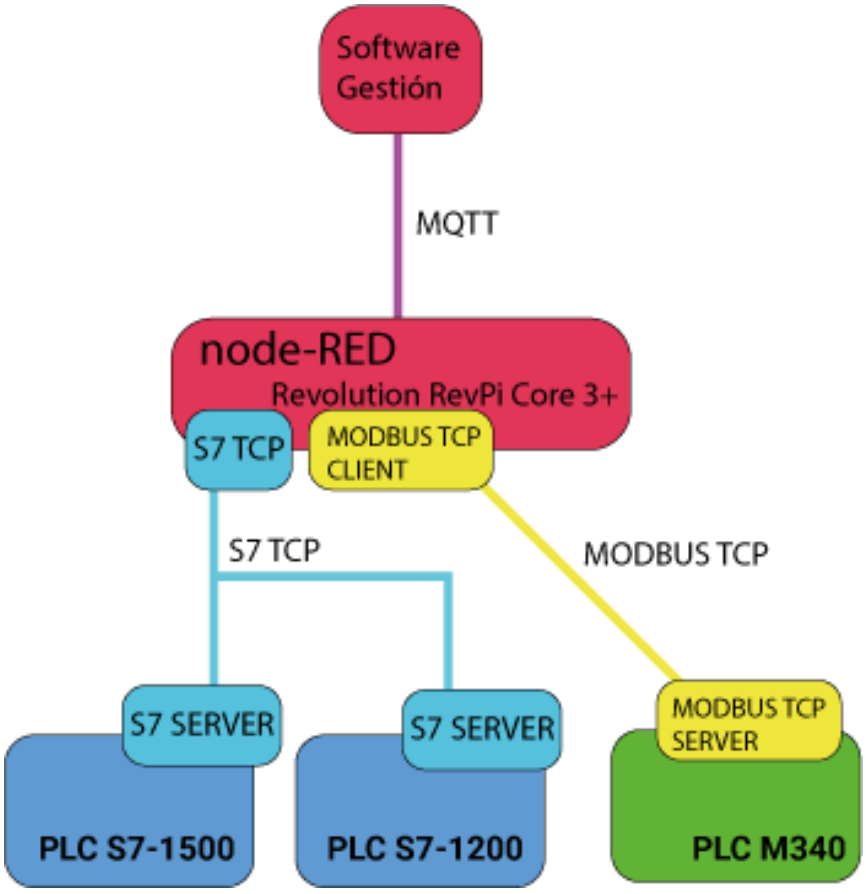

Figura 32. Arquitectura comunicación con diferentes PLC. Fuente: Elaboración propia.

Amarillo	**Dispositivos que utilizan MODBUS TCP para intercambiar información.**
Azul	Uso del protocolo S7 TCP estándar de Siemens para el intercambio de información.
Lila	Intercambio de información mediante MQTT.

Tabla 22. Leyenda de la arquitectura expuesta. Fuente: Elaboración propia.

En esta arquitectura, se contempla la instalación de un PC Industrial (Revolution RevPi Core 3+) para acceder a la memoria de los PLC. Se requiere su uso a modo de interfaz de programación de aplicaciones (API). Se puede observar dos modelos diferentes de PLC, de diferente fabricante. A su vez, el protocolo utilizado para el intercambio de información es distinto.

No se especifica la conexión de los sensores al PLC por no presentar relevancia. La comunicación PLC – PC es independiente de los protocolos y conexiones con los sensores.

Por una parte, los controladores Siemens se comunican mediante el protocolo S7 TCP con el PC industrial. Por otra parte, en el caso del controlador Schneider, esta comunicación se realiza mediante Modbus TCP. Por tanto, se necesitarán las correspondientes librerías de node-RED para poder establecer comunicación entre los dispositivos.

Node-RED obtiene la información necesaria de cada PLC para, posteriormente, filtrar los datos y estandarizarlos con la finalidad de procurar coherencia, independientemente de su origen.

La aplicación de gestión se comunica mediante MQTT con el PC Industrial. La API en constante ejecución en el PC Industrial es la encargada de servir la información requerida o realizar las acciones necesarias cuando la aplicación de gestión lo requiera.

Ventajas
Coste muy reducido, material de bajo coste.
El PC Industrial puede ejecutar diferentes tareas de forma simultánea.
Tiempo de desarrollo necesario muy reducido gracias a node-RED.

Tabla 23. Ventajas del uso de la arquitectura expuesta. Fuente: Elaboración propia.

Inconvenientes
Flexibilidad limitada, total dependencia de las librerías de node-RED.
Programación visual presenta limitaciones intrínsecas en programas complejos.

Tabla 24. Inconvenientes del uso de la estructura expuesta. Fuente: Elaboración propia.

Capítulo III:

Caso práctico

1. La celda industrial

El contenido y fotografías presentadas en este apartado se obtienen del trabajo de final de grado “Estudio de las etapas de automatización de un proceso industrial y sus implicaciones en la gestión de la producción” elaborado por Pablo García Rodríguez [30].

1.1 Introducción

La celda industrial se ubica en el laboratorio 004 de la Escuela Superior de Ingeniería Industrial, Aeroespacial y Audiovisual de Terrassa (ESEIAAT). **Su funcionamiento simula el transporte de materiales y recursos entre estaciones dentro de una planta industrial.**

Figura 33. La celda industrial. Fuente:

La línea se compone de un conjunto de cintas transportadoras accionadas de forma mecánica mediante motores eléctricos. Funciona de forma flexible, permitiendo configurar el transporte de bandejas (a efectos prácticos, simulan el uso de pallets), en función de las necesidades de la producción. Por ende, existen una serie de

retenedores y plataformas que permiten controlar el flujo de bandejas, a través de la instalación.

Todas las señales de la instalación (sensores inductivos, accionamiento de motores, etc.) se recogen por distintos PLC encargados de gobernar el comportamiento de la célula.

Figura 34. Ejemplo de bandeja. Fuente:

En una línea gobernada por tres PLC, comunicar las aplicaciones de gestión con los controladores resulta complejo por la necesidad de realizar peticiones individuales a cada uno de los dispositivos. Por tanto, ese desarrollo práctico se focaliza estandarizar la comunicación entre entidades mediante el uso de una capa intermedia, que funciona a modo de abstracción del bajo nivel que supone la comunicación con PLC. Ésta capa debe ser estándar y facilitar el acceso a la información, por tanto, sencilla de implementar por las aplicaciones de gestión.

1.2 Descripción general de la instalación

La célula flexible está compuesta de 4 tramos, definidos a partir del protocolo utilizado por los PLC para comunicarse con las cabeceras de periferia disponibles, a través del bus. Estas cabeceras contienen los módulos de E/S que conectan los sensores y actuadores con el sistema de control. De este modo, se reduce el número de cables y la complejidad del conexionado, puesto que la información se transmite digitalmente mediante el bus.

El sistema de control está formado por 3 PLC, cada uno con interfaz para un bus de campo distinto

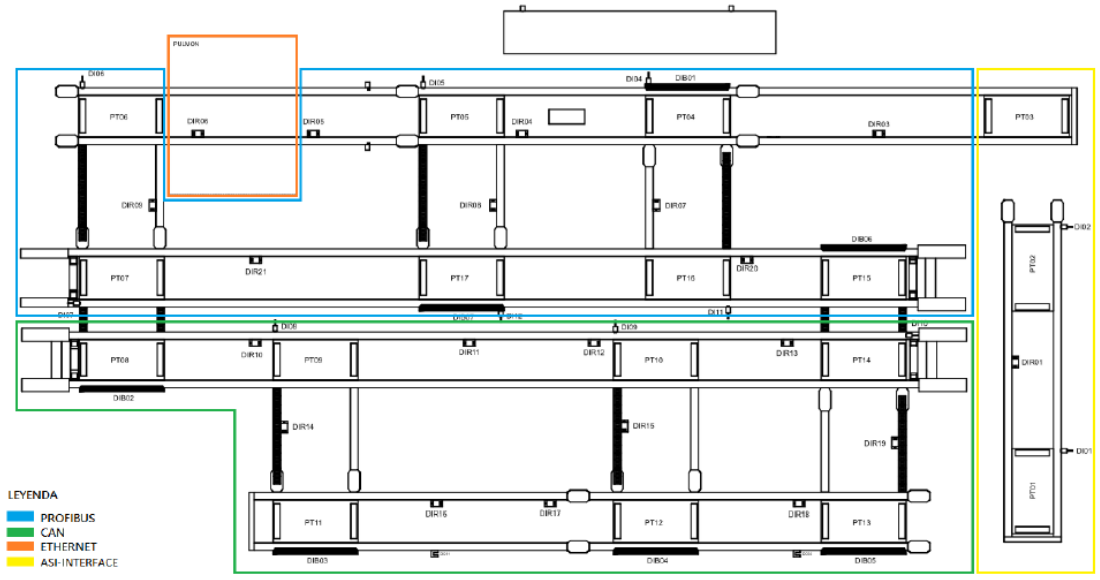

Figura 35. Buses de campo en la instalación. Fuente: García, Pablo.

.

Como se ha comentado con anterioridad, el control de flujo de las bandejas se realiza mediante retenedores y plataformas[10]. Se numera cada retenedor con el prefijo DIR y el número asignado. En el caso de las plataformas, éstas se definen con el prefijo anterioridad PT y el correspondiente número. Por tanto, en cada bus[11] se tiene acceso al control de un determinado número de retenedores y plataformas.

Bus de campo	Retenedores	Plataformas
AS-interface	DIR01 – DIR02	PT01 – PT02
CAN Open	DIR03 – DIR09 DIR20 DIR21	PT03 - PT07 PT15 - PT17
PROFIBUS	DIR10 – DIR19	PT08 – PT14

Tabla 25. Retenedores y plataformas asignadas a cada PLC. Fuente: Elaboración propia.

[10] Estos elementos quedan definidos en el apartado "

1.3 Elementos relevantes de la instalación".

[11] Se entiende bus como una línea de transmisión enseriada de información entre dispositivos. PROFIBUS es un bus de campo puesto que conecta los módulos de periferia con el dispositivo maestro, que accede a la información.

1.3 Elementos relevantes de la instalación

1.3.1 Retenedores

Se utiliza a fin de poder controlar el avance de las bandejas por la instalación. Permite bloquear el paso de una bandeja cuando ésta es detectada. Está formado por un sensor inductivo y un cilindro de simple efecto con retorno por muelle.

Cuando el sensor inductivo detecta la presencia de una bandeja (las bandejas disponen de una pieza metálica en la parte inferior), se genera una señal digital. Si el PLC decide bloquear el paso de dicha bandeja, se acciona una electroválvula que activa el cilindro.

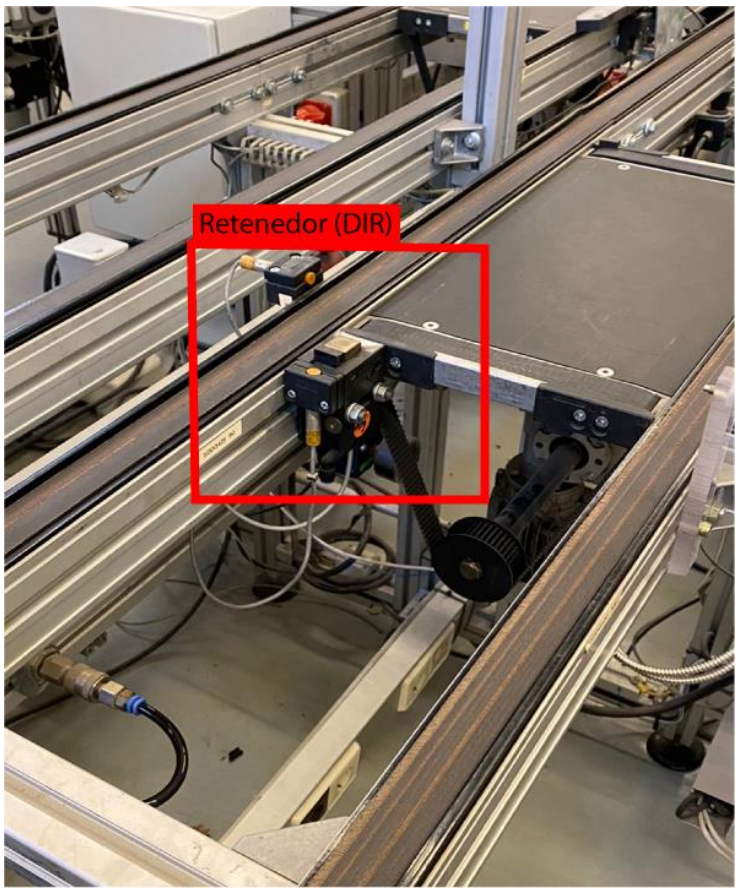

Figura 36. Ejemplo de retenedor. Fuente: Elaboración propia.

1.3.2 Plataformas

Se utiliza con el objetivo de poder dirigir las bandejas entre diferentes caminos, en base a la necesidad de la producción. Se pueden encontrar en las intersecciones y en determinados puntos de una línea. Existen dos tipos de plataformas en la instalación:

- Tipo I. Posición de reposo y subida
- Tipo II. Posición de reposo, subida y bajada

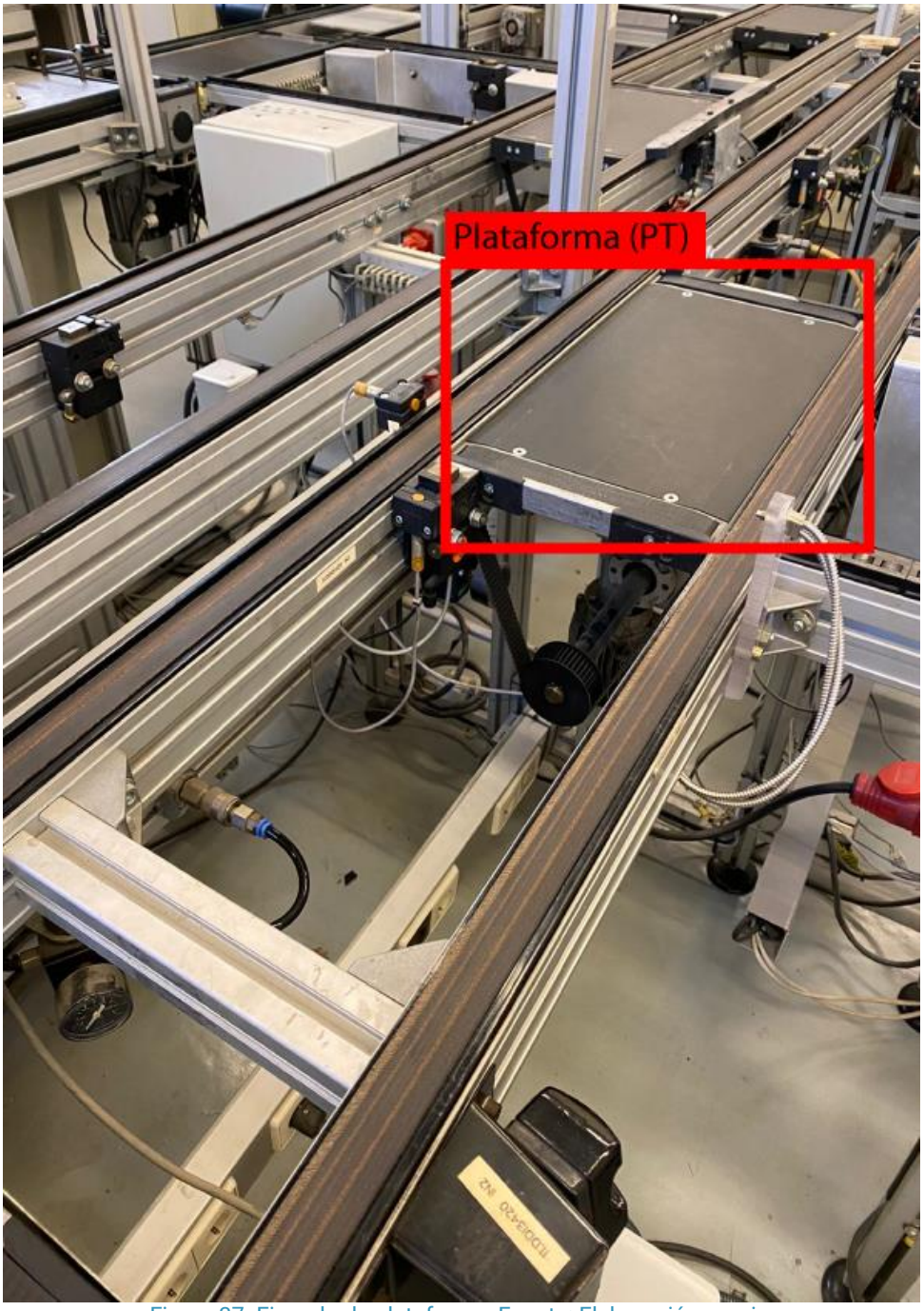

Figura 37. Ejemplo de plataforma. Fuente: Elaboración propia.

2. Diseño y dimensionado de la API

2.1 Limitaciones

El desarrollo técnico de esta solución se ha visto afectado por la aplicación del estado de alarma por parte del gobierno español (Real Decreto 463/2020) que limita la movilidad de las personas y el uso de espacio públicos a causa del COVID-19. Como consecuencia, no ha sido posible el acceso al laboratorio de automatización. Se ha validado la solución mediante la simulación de los PLC, con las limitaciones que ello implica. Por limitaciones del programa Unity Pro (utilizado en la programación de PLC) solo se ha podido simular de forma simultánea un único PLC, que engloba las funciones de toda la instalación y, por tanto, dista en cierto modo de la realidad. Asimismo, en términos de diseño, se han tenido en cuenta la existencia de los otros dos dispositivos.

2.2 Objetivos

En los objetivos de este proyecto[12] se exponen de forma general las principales necesidades que debe cumplir la interfaz. Con carácter general, el principal objetivo es estandarizar la comunicación de diferentes aplicaciones de gestión con la célula industrial.

2.3 Planificación y presupuesto

El timing del desarrollo del caso práctico no ha podido seguir la planificación considerada en el punto de partida, recogida en el diagrama de Gantt, debido a las dificultades de acceso a la universidad y a determinadas herramientas a causa del COVID-19.

Con el objetivo de optimizar el desarrollo al máximo dentro de lo posible, tras los imprevistos surgidos, se ha utilizado la herramienta de gestión de proyectos Trello. Se ha organizado el perfil en listas las tareas a realizar durante la programación de la API, así como la documentación de interés. De forma complementaria, ha permitido llevar a

[12] Recogidos en el apartado “Objetivos”.

cabo un recuento de horas de desarrollo para aproximar la dedicación de cada tarea y el presupuesto a la realidad.

El citado presupuesto de desarrollo de la API se encuentra anexo a esta memoria en el apartado “Anexos”.

2.4 Requerimientos

Es necesario un agente encargado de interpretar las peticiones de las aplicaciones de gestión y traducirlas en acciones a nivel de campo. Por esta razón, este dispositivo debe permitir el acceso a cada uno de los PLC que gobiernan la instalación mediante el protocolo propio de cada controlador y poder realizar lecturas/escrituras de datos. Los controladores están habilitados para comunicar información mediante el protocolo Modbus TCP. En consecuencia, se requiere de una interfaz física RJ-45 y soportar Modbus TCP. Debe permitir la comunicación mediante un protocolo seguro con las aplicaciones de gestión y estandarizar con independencia de la aplicación, las acciones de bajo nivel. Asimismo, debe estar diseñado para trabajar en un entorno industrial bajo la norma EN 61131-2.

2.5 Arquitectura

La célula industrial dispone de 3 controladores PLC y cabeceras de periferia de la marca Schneider Electric.

Línea CAN y Ethernet	Modicon M340
Línea PROFIBUS	Modicon M251
Línea AS-interface	Modicon M340

Tabla 26. Modelos de PLC en la instalación. Fuente: Elaboración propia.

Se ha definido la siguiente arquitectura de red en base a la propuesta realizada en el apartado *“Arquitectura III: Centralizar la comunicación con diferentes PLC”*.

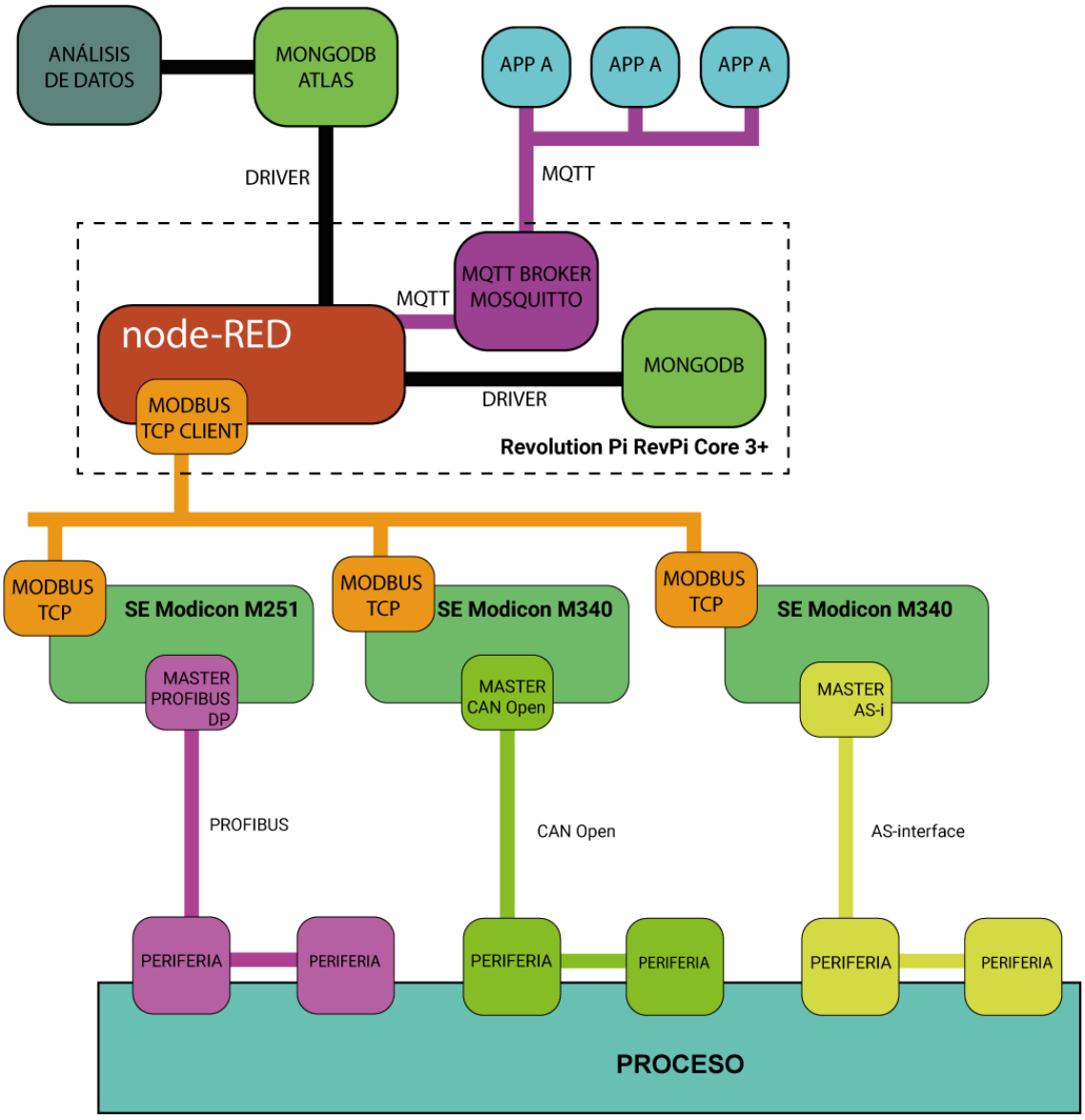

Figura 38. Arquitectura presentada a modo de caso práctico. Fuente: Elaboración propia.

Es posible observar cómo cada PLC contiene sus cabeceras de periferia conectadas a un bus distinto. En este proyecto, el acceso a las señales prácticamente no tiene afectación más allá de definir el nombre del PLC. Sin embargo, se considera un aspecto relevante a exponer puesto que la API accede directamente a las posiciones de memoria del controlador.

Cada controlador actúa como servidor Modbus TCP ante las peticiones de la API, la cual se comunica en calidad de cliente. La API, lee espacios de memoria determinados y escribe registros de los controladores en base al protocolo. Toda información plana adquirida del PLC es filtrada y ordenada para generar estructuras de datos.

Con el objetivo de poder aislar la información recogida y asegurar que se trata de la última disponible, se hace uso de una base de datos local MongoDB. En referencia a la comunicación con las aplicaciones de gestión, se realiza el intercambio de datos mediante MQTT. El propio dispositivo que ejecuta el código de la API contiene de forma paralela un *broker* MQTT "Mosquitto", que habilita esta comunicación.

Tras la variación en el estado de un retenedor o plataforma, los datos se comunican a un clúster de MogoDB ATLAS en la nube, a modo de respaldo de los mismos. En este sentido, diferentes herramientas de análisis de datos pueden conectarse directamente al clúster con tal de obtener la información para generar modelos y cálculos respecto al proceso.

2.6 Selección de componentes

Los componentes se colocarán en un armario industrial de pequeñas dimensiones, montados sobre un carril DIN. Como dispositivo PC Industrial, se ha seleccionado un Revolution RevPi Core 3+, principalmente por cumplir con todos los requisitos[13] necesarios de este proyecto a un coste muy reducido.

Se dimensionan los cables de conexión de red sin apantallamiento y un switch para aplicaciones industriales.

Componente	Descripción	Unidades	Precio (€)
Revolution Pi RevPi Core 3+	PC Industrial	1	290,63
WIWAV WDH-5ET-DC	Switch Industrial	1	469,20
Cable de Cat6 Belden 7812ENH 50 m	Cable de red	1	64,73
Conectores RJ-45	Conector de red	6	17,4

Tabla 27. Componentes dimensionados. Fuente: Elaboración propia.

[13] Expuestos en el apartado “El timing del desarrollo del caso práctico no ha podido seguir la planificación considerada en el punto de partida, recogida en el diagrama de Gantt, debido a las dificultades de acceso a la universidad y a determinadas herramientas a causa del COVID-19.

Con el objetivo de optimizar el desarrollo al máximo dentro de lo posible, tras los imprevistos surgidos, se ha utilizado la herramienta de gestión de proyectos Trello. Se ha organizado el perfil en listas las tareas a realizar durante la programación de la API, así como la documentación de interés. De forma complementaria, ha permitido llevar a cabo un recuento de horas de desarrollo para aproximar la dedicación de cada tarea y el presupuesto a la realidad.

El citado presupuesto de desarrollo de la API se encuentra anexo a esta memoria en el apartado “Anexos”.

2.4 Requerimientos”.

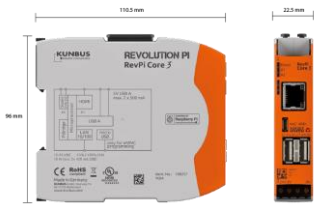

Figura 39. Revolution Pi RevPi Core 3+. Fuente: RS Components.

Figura 40. Switch industrial WIMAV. Fuente: Amazon.

Figura 41. Cable Bremen cat. 6. Fuente: RS Components.

3. Programación de la API con node-RED

En esta sección, se comenta de forma detallada la programación visual asociada a la API.

3.1 Estructuras de datos

Para poder acceder a la información de los PLC de forma organizada, se requiere definir un UDT[14]. Los tipos de datos creados por el usuario se generan a partir de los tipos de datos primitivos (booleanos, enteros, reales). Estas estructuras permiten mantener una cierta coherencia en el programa PLC y, a su vez, facilitar el acceso a la información por parte de la API. Las estructuras vienen determinadas en consenso con los programadores de PLC, para asegurar la correcta programación de la API y la integración de toda la arquitectura.

La estructura PRODUCTO referencia el contenido de una bandeja, mediante un identificador inequívoco, el tipo de producto utilizado (A, B, C) y el número de bandeja en que se encuentra. Cada vez que una bandeja entra en circulación, la API carga los datos referentes en el retenedor inicial (DIR03, no se ha implementado la zona AS-i).

[14] UDT es la abreviación del inglés *user data types*. Hace referencia a los tipos de datos generados por el usuario en el programa PLC.

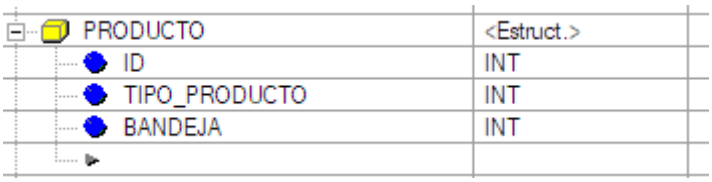

Figura 42. Estructura PRODUCTO. Fuente: Elaboración propia.

En el caso de la estructura RETENEDOR, contiene los diferentes estados en los que se puede encontrar el retenedor (REST, READY, MOVE, BLOQ). También está formado por dos objetos de la estructura anteriormente comentada: PRODUCTO. El nombre utilizado para ellas es PRODUCTO_ACTUAL y PRODUCTO_ANTERIOR. Esto permite tener constancia de la bandeja se encuentra actualmente en el retenedor y la anterior, con tal de poder realizar un *tracking* de las bandejas.

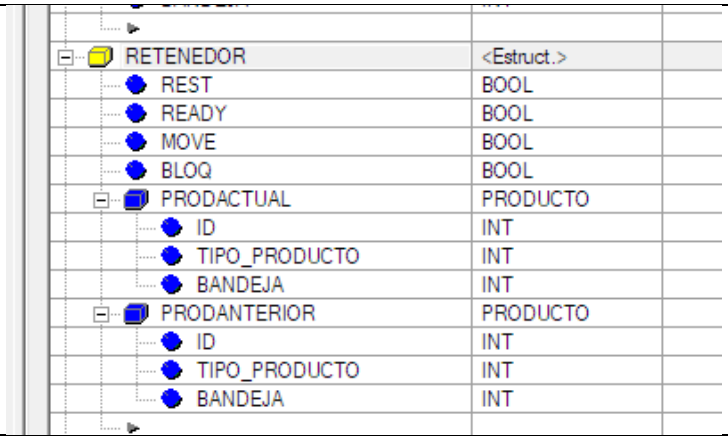

Figura 43. Estructura RETENEDOR. Fuente: Elaboración propia.

Referente a la estructura PLATAFORMA, funciona de forma similar a la estructura RETENEDOR, con la diferencia que presenta más opciones de parametrización. Este aspecto se debe a la existencia de diferentes tipos de plataformas en la instalación. Es probable que no todas las plataformas requieran el uso de determinadas variables internas de la estructura, porqué a nivel operativo la plataforma no dispone de los actuadores necesarios, pero deben ser definidas por aquellos casos en que sí.

En este caso, los estados REST, READY y BLOQ se mantienen, pero se añaden RECTO, DESVIO, BLOQ_RECTO, BLOQ_DESVIO, MOVE_RECTO Y MOVE, DESVIO.

Cabe destacar que se mantienen las subestructuras PRODUCTO dentro de la estructura principal.

Nombre	Tipo
PLATAFORMA	<Estruct.>
REST	BOOL
READY	BOOL
BLOQ	BOOL
RECTO	BOOL
DESVIO	BOOL
BLOQ_RECTO	BOOL
BLOQ_DESVIO	BOOL
MOVE_RECTO	BOOL
MOVE_DESVIO	BOOL
PRODACTUAL	PRODUCTO
ID	INT
TIPO_PRODUCTO	INT
BANDEJA	INT
PRODANTERIOR	PRODUCTO
ID	INT
TIPO_PRODUCTO	INT
BANDEJA	INT

Figura 44. Estructura PLATAFORMA. Fuente: Elaboración propia.

Estas estructuras deben ser declaradas como objetos al uso dentro de la tabla de variables. Se debe definir una dirección de memoria determinada para cada uno de los objetos, con el objetivo de poder localizar la información. **La API solicita la información al PLC en base a la dirección inicial y la cantidad de información requerida.** El retenedor DIR03 (nombre impuesto a la instancia del tipo RETENEDOR) se sitúa en la dirección de memoria %MW400 y tiene una duración de 7 palabras.

DIR03	RETENEDOR	%MW400
REST	BOOL	%MW400
READY	BOOL	%MW400
MOVE	BOOL	%MW401
BLOQ	BOOL	%MW401
PRODACTUAL	PRODUCTO	%MW402
PRODANTERIOR	PRODUCTO	%MW405

Figura 45. Retenedor DIR03 en la dirección %MW400. Fuente: Elaboración propia.

De forma contigua, se declara DIR04 en la dirección de memoria %MW410 para asegurar que no exista una superposición en la información. Todos los demás retenedores se instancian de igual forma dentro de la tabla de variables.

DIR03	RETENEDOR	%MW400	
DIR04	RETENEDOR	%MW410	
DIR05	RETENEDOR	%MW420	
DIR06	RETENEDOR	%MW430	
DIR07	RETENEDOR	%MW440	
DIR08	RETENEDOR	%MW450	
DIR09	RETENEDOR	%MW460	
DIR10	RETENEDOR	%MW100	
DIR11	RETENEDOR	%MW110	
DIR12	RETENEDOR	%MW120	
DIR13	RETENEDOR	%MW130	
DIR14	RETENEDOR	%MW140	
DIR15	RETENEDOR	%MW150	
DIR16	RETENEDOR	%MW160	
DIR17	RETENEDOR	%MW170	
DIR18	RETENEDOR	%MW180	
DIR19	RETENEDOR	%MW190	
DIR20	RETENEDOR	%MW480	
DIR21	RETENEDOR	%MW490	

Figura 46. Instancias de retenedores. Fuente: Elaboración propia.

Es importante comentar los saltos entre direcciones de memoria. Se realizan con tal de marcar que se encuentran en PLC distintos aunque, por las razones comentadas en apartados anteriores[15] "2.1 Limitaciones", todas se encuentran bajo el mismo controlador simulado. En el caso de las plataformas, sucede de la misma forma que con los retenedores. Se les asigna direcciones de memoria para poder acceder a los datos, obteniendo en conjunto:

[15] Expuestas en el apartado "2.1 Limitaciones".

Nombre	Tipo	Dirección
DIR03	RETENEDOR	%MW400
DIR04	RETENEDOR	%MW410
DIR05	RETENEDOR	%MW420
DIR06	RETENEDOR	%MW430
DIR07	RETENEDOR	%MW440
DIR08	RETENEDOR	%MW450
DIR09	RETENEDOR	%MW460
DIR10	RETENEDOR	%MW100
DIR11	RETENEDOR	%MW110
DIR12	RETENEDOR	%MW120
DIR13	RETENEDOR	%MW130
DIR14	RETENEDOR	%MW140
DIR15	RETENEDOR	%MW150
DIR16	RETENEDOR	%MW160
DIR17	RETENEDOR	%MW170
DIR18	RETENEDOR	%MW180
DIR19	RETENEDOR	%MW190
DIR20	RETENEDOR	%MW480
DIR21	RETENEDOR	%MW490
PT03	PLATAFORMA	%MW287
PT04	PLATAFORMA	%MW300
PT05	PLATAFORMA	%MW313
PT06	PLATAFORMA	%MW326
PT07	PLATAFORMA	%MW339
PT08	PLATAFORMA	%MW200
PT09	PLATAFORMA	%MW213
PT10	PLATAFORMA	%MW226
PT11	PLATAFORMA	%MW239
PT12	PLATAFORMA	%MW252
PT13	PLATAFORMA	%MW265
PT14	PLATAFORMA	%MW278
PT15	PLATAFORMA	%MW352
PT16	PLATAFORMA	%MW365
PT17	PLATAFORMA	%MW378

Figura 47. Conjunto de variables en el PLC de simulación. Fuente: Elaboración propia.

3.2 Descripción de los flujos de la API

Anteriormente[16] se ha comentado que la programación visual de node-RED se organiza en *flows*[17]. Cada uno se interpreta como una sección independiente del código, todo y que se puede establecer comunicación mediante un tipo de nodo determinado.

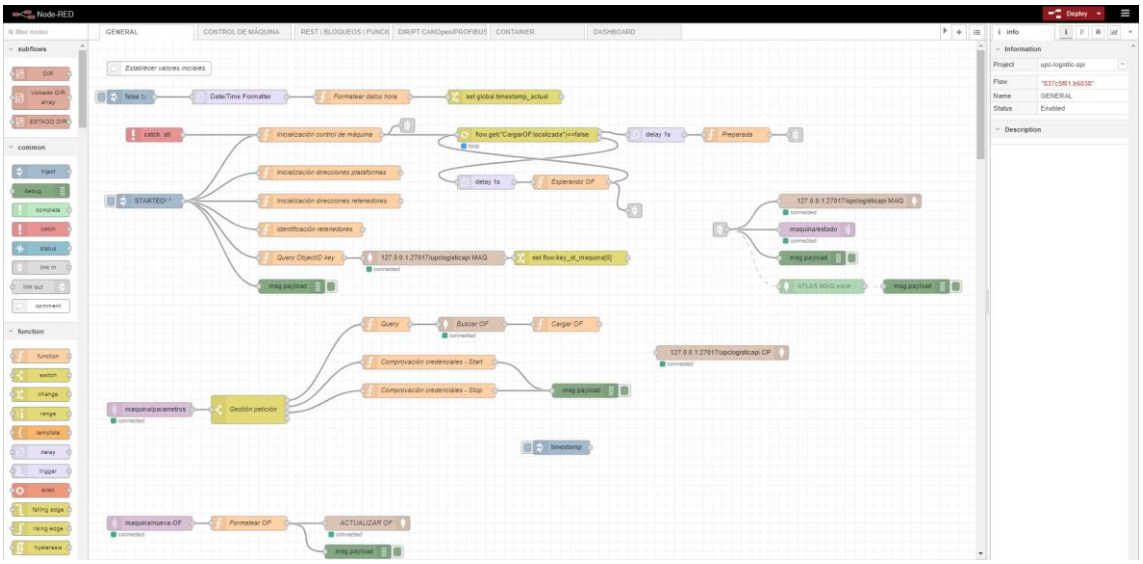

Figura 48. Vista global de la API. Fuente: Elaboración propia.

En este proyecto se han declarado 5 *flows*, cada uno con su respectiva función dentro del objetivo principal de la API:

- **General.** *Flow* encargado de controlar el estado general de la aplicación. Incluye la gestión de productos y órdenes de producción.
- **Control de máquina.** Contiene las primeras acciones de la API relacionadas con el control de máquina. Es decir, el volcado y obtención de las bandejas una vez han realizado todo el proceso.
- **DIR/PT.** *Flow* encargado de digitalizar el estado de los retenedores y plataformas. Es claramente el núcleo de la API por traducir los *arrays* de recogidos del PLC en objetos JSON con todos los parámetros de la línea.

[16] Apartado “Node-RED”.
[17] Se define como *flow* cada sección general dentro del código de node-RED.

- **Funciones.** Contiene soporte para peticiones HTTP REST, la actualización de los bloqueos de los retenedores (individual o total) y pequeñas funciones de interés.
- **Dashboard.** Pequeña GUI que permite visualizar el estado de los retenedores sin necesidad de una aplicación externa.

3.2.1 Flow I: General

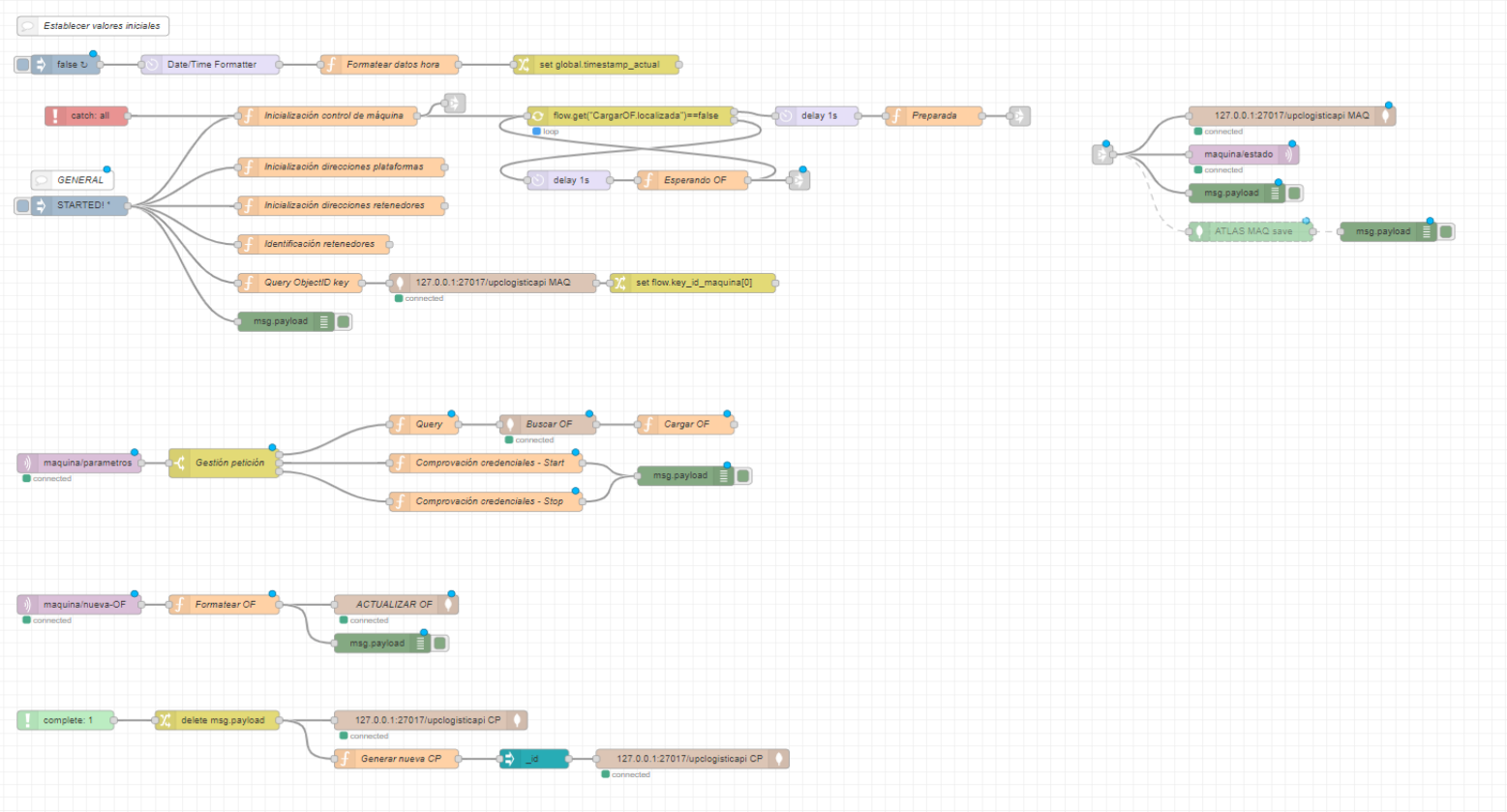

Figura 49. Flow I: General. Fuente: Elaboración propia.

Fecha actual

El primer conjunto de nodos genera un *timestamp* como variable global. Este tipo de variables son accesibles desde cualquier flujo, por definición. Para ello, se realiza una petición cada segundo al nodo *Date/Time Formatter*. Se obtiene la hora UTC, por lo que es necesario convertir al fuso horario español y presentar los valores en formato AAAA/MM/DD HH:SS.

Figura 50. Flujo para la obtención de un *timestamp*. Fuente: Elaboración propia.

Inicialización de la API

El flujo iniciado por “STARTED!” controla la inicialización de la API y las direcciones de los retenedores/plataformas. En el momento en que entra en ejecución la API, el nodo “STARTED!” inicia el flujo, activando el nodo de función “Inicialización control de máquina”.

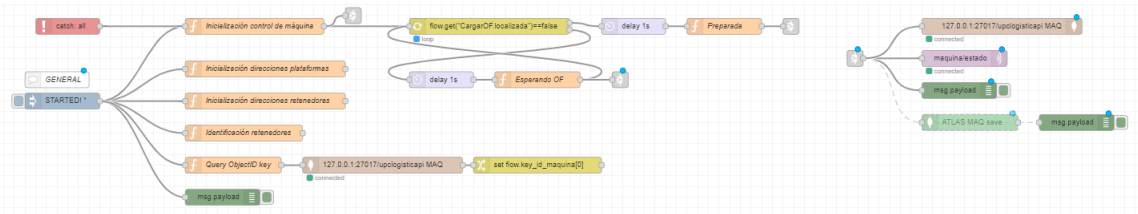

Figura 51. Flujo de inicialización de la API. Fuente: Elaboración propia.

El objeto JSON “Estado_Maquina” es de acceso global y describe el estado de la API (actualmente en “Inicio”), la fecha de la última actualización de estado y la orden de fabricación, aún no determinada. “CargarOF” determina si se ha escogido la orden de producción.

El parámetro _id contiene el ObjectID que, posteriormente, emplea MongoDB para actualizar el documento que corresponde al estado de la máquina con los nuevos valores. Finalmente, se envía como mensaje al siguiente nodo (mediante el objeto *msg*) el valor de estado de la máquina. Este sería un ejemplo de ejecución de un nodo de función.

```
//inicializacación array
global.set("Estado_Maquina",
{
    "Objeto": "API",
    "Estado": "Inicio",
    "Fecha_Estado": global.get("timestamp_actual"),
    "OF_Actual": {
        "ref": "",
        "prod":[]
    },

});

flow.set("CargarOF",
{
    "localizada": false
});
msg._id=flow.get("key_id_maquina[0]"),
msg.payload = global.get("Estado_Maquina")
return msg;
```

Figura 52. Nodo "Inicialización control de máquina". Fuente: Elaboración propia.

A continuación, el flujo[18] entra en un bucle mientras no sea cargada por parte del usuario una orden de producción. Cada segundo se comprueba el estado de "CargarOF.localizada". En caso de no cumplir la condición, se ejecuta el nodo "Esperando OF". Su función es actualizar el *timestamp* de estado de la máquina.

Una vez se ha cargado correctamente la orden de fabricación (posteriormente se describe este proceso), la máquina pasa a estado de "Preparada" a la espera de la orden de marcha por parte del usuario.

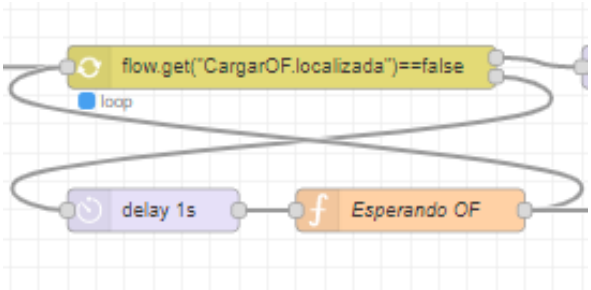

Figura 53. Bucle de control. Fuente: Elaboración propia.

Los nodos de función "Inicialización control de máquina", "Esperando OF" y "Preparada" están conectados con un grupo de nodos encargados de retransmitir el valor de estado del objeto "Estado_Maquina" por MQTT y actualización en MongoDB. Podría ser volcado su valor a MongoDB ATLAS de forma opcional. También se muestra su valor en la consola *debug* de node-RED.

[18] Se hace referencia a flujo cuando es una porción de flujo extraída de un *flow*.

Figura 54. Estados de máquina como mensajes en la consola de *debug*. Fuente: Elaboración propia.

En caso de detectar algún error en la ejecución, la API sería reiniciada por el nodo "catch" que conecta de forma alternativa con "Inicialización control de máquina".

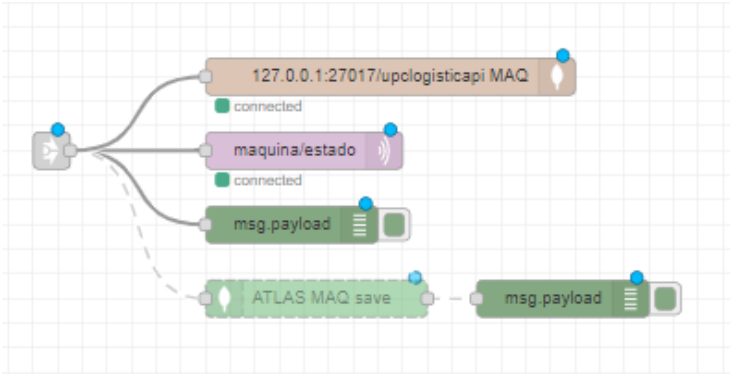

Figura 55. Nodos publicadores de información. Fuente: Elaboración propia.

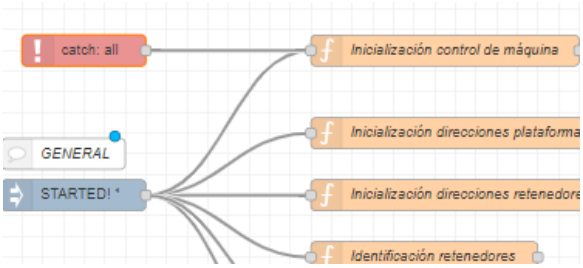

Figura 56. Nodo "catch". Fuente: Elaboración propia.

Control y parametrización de la API

Este flujo permite la modificación del estado de la API mediante el *topic* MQTT "máquina/parámetros". Se puede ejecutar tres comandos en base a su programación.

- Carga de una orden de fabricación (OF)
- *Start* de la API
- *Stop* de la API

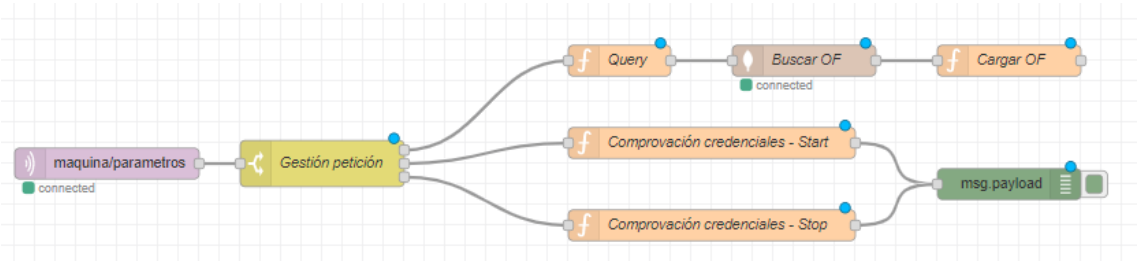

Figura 57. Flujo para parametrizar el estado de la API. Fuente: Elaboración propia.

Antes de poder llevar a cabo el arranque de la máquina, es necesario la selección de una orden de producción previamente definida (el proceso de inserción se describe posteriormente). Para seleccionar la orden de producción, se requiere el un identificador.

El nodo MQTT "maquina/parametros" conecta con un nodo *switch* que, en base al valor del estado de la máquina, decide qué hacer con el objeto *msg* recibido.

Figura 58. Nodo "Switch". Fuente: Elaboración propia.

En caso de estar a la espera de una orden de fabricación, se ejecuta el nodo "Query", encargado de establecer el usuario que realiza la petición y la coloca la referencia en *msg.payload* para ser utilizada como filtro de búsqueda en la base de datos MongoDB.

```
global.set("Estado_Maquina.user", msg.payload.user)
msg.payload = {
    ref: msg.payload.ref
}
return msg;
```

Figura 59. Nodo "Query". Fuente: Elaboración propia.

En caso de ser localizada, se ejecuta el nodo "Cargar OF". Se modifica el objeto descrito "CargarOF", anteriormente descrito con los valores de producción. El bucle inicial finaliza y la API pasa a modo "Preparada".

El arranque de la API únicamente es posible realizarlo si se encuentra en estado “Preparada”. Entra en ejecución el nodo “Comprobación de credenciales - Start”

```
flow.set("CargarOF",
{
        "localizada": true,
        "ref": msg.payload[0].ref,
        "prod":msg.payload[0].prod
});

return;
```

Figura 60. Nodo "Comprobación de credenciales - Start". Fuente: Elaboración propia.

Se comprueba que el objeto *msg* es enviado por el usuario que actualmente está utilizando la máquina. En caso afirmativo, se evalúa el valor de la cadena contenida en el atributo *msg.payload.action*. Este parámetro indica qué comando se desea realizar (iniciar el proceso o pararlo, por ejemplo). De contener un “Start” modifica el estado de maquina a “Producción”. También se actualiza el *timestamp* de estado para dejar constancia en el histórico. Llegado este punto, la maquina se encuentra está activa e insertando bandejas en la línea.

Para poder parar el proceso es necesario estar en modo producción “Producción”. La recepción del mensaje ejecuta el nodo “Comprobación de credenciales - Stop”, con un funcionamiento similar al anteriormente descrito.

```
if(msg.payload.user==global.get("Estado_Maquina.user")){
    if(msg.payload.action=="Stop"){
        global.set("Estado_Maquina.Estado","Paro");
        global.set("Estado_Maquina.Fecha_Estado",global.get("timestamp_actual"))
    }
}else{
    node.error("Usuario no válido",msg)
}
msg.payload = global.get("Estado_Maquina")
return msg;
```

Figura 61. Nodo "Comprobación de credenciales - Stop". Fuente: Elaboración propia.

Gestión de productos

La gestión de productos comprende la inserción de órdenes de fabricación y su procesado para ser utilizado por los flujos del control de máquina.

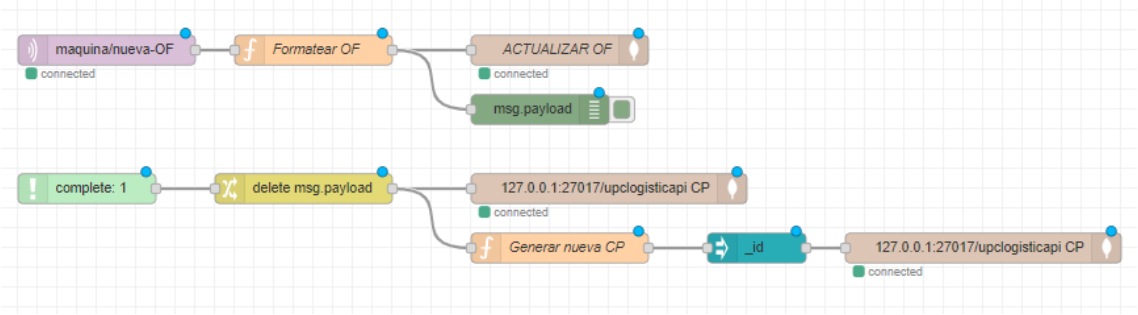

Figura 62. Flujo que gestiona la adición y gestión de productos. Fuente: Elaboración propia.

En caso de querer introducir una nueva orden de fabricación, debe ser agregada mediante el *topic* "maquina/nueva-OF". Se introduce en el *topic* un objeto JSON de estructura determinada, se inserta dentro de la base de datos MongoDB. Esto permite la reutilización de órdenes de producción.


```
_id: ObjectId("5ef73542418ef955fc741a6d")
ref: 1
prod: Array
  0: Object
     tipo: "A"
     qty: 3
  1: Object
     tipo: "B"
     qty: 5
```


Figura 63. Ejemplo de orden de fabricación. Fuente: Elaboración propia.

Para transformar la orden de fabricación (OF) en la cola de producción (CP), se requiere un flujo independiente. Este se inicia en el nodo "complete: 1", excitado por la ejecución del nodo "Preparada", comentado anteriormente[19]. Significa que en el momento en que la API entra en modo "Preparada", se debe generar la nueva cola de producción (CP), para que cuando se reciba orden de marcha, la cola ya esté lista para su uso.

[19] Apartado "Inicialización de la API".

Primeramente, se elimina la cola de producción anterior (acción *remove* de todos los documentos de la colección CP) y se ejecuta el nodo "Generar nueva CP". Principalmente, este nodo "expande" un único objeto ("OF_Actual") que contiene la referencia de la OF, el tipo de producto y la cantidad a producir en una inserción en la colección CP de la base de datos para cada elemento a producir. En esa inserción, se codifica una referencia inequívoca para cada producto (en base a su referencia de OF, tipo y cantidad).

La programación de este nodo es relativamente más compleja que en los anteriores apartados, puesto que se usa la función *forEach,* la cual permite el tratamiento individual de los objetos de un array. Esta función tiene *callback*[20].

También se traduce los tipos de producto "A", "B" y "C" a "1", "2", "3", respectivamente. Esto es debido a que en el programa PLC las estructuras se han definido con valores enteros para simplificar la programación. Posteriormente, se asigna una _id a cada objeto y se inserta en la base de datos.

[20] Las funciones *callback* es una función que pasa otra función como argumento.

```
var OF = global.get("Estado_Maquina.OF_Actual")
let objetos = []
msg._id = null
let tipo

OF.prod.forEach((el,i) => {
    switch (el.tipo) {
        case "A":
            tipo = 1;
        break;
        case "B":
            tipo = 2;
        break;
        case "C":
            tipo = 3;
        break;
    }
    for (let y = 0; y < el.qty; y++) {
        let ob = {
            "id": `${OF.ref}${tipo}${y}`,
            "tipo": tipo,
        };
        objetos.push(ob);
    }
});

objetos.forEach((el,i)=> {
    msg.payload = el;
    node.send(msg);
})
return;
```

Figura 64. Generación de la cola de producción. Fuente: Elaboración propia.

Figura 65. Cola de producción. Fuente: Elaboración propia.

3.2.2 Flow II: Control de máquina

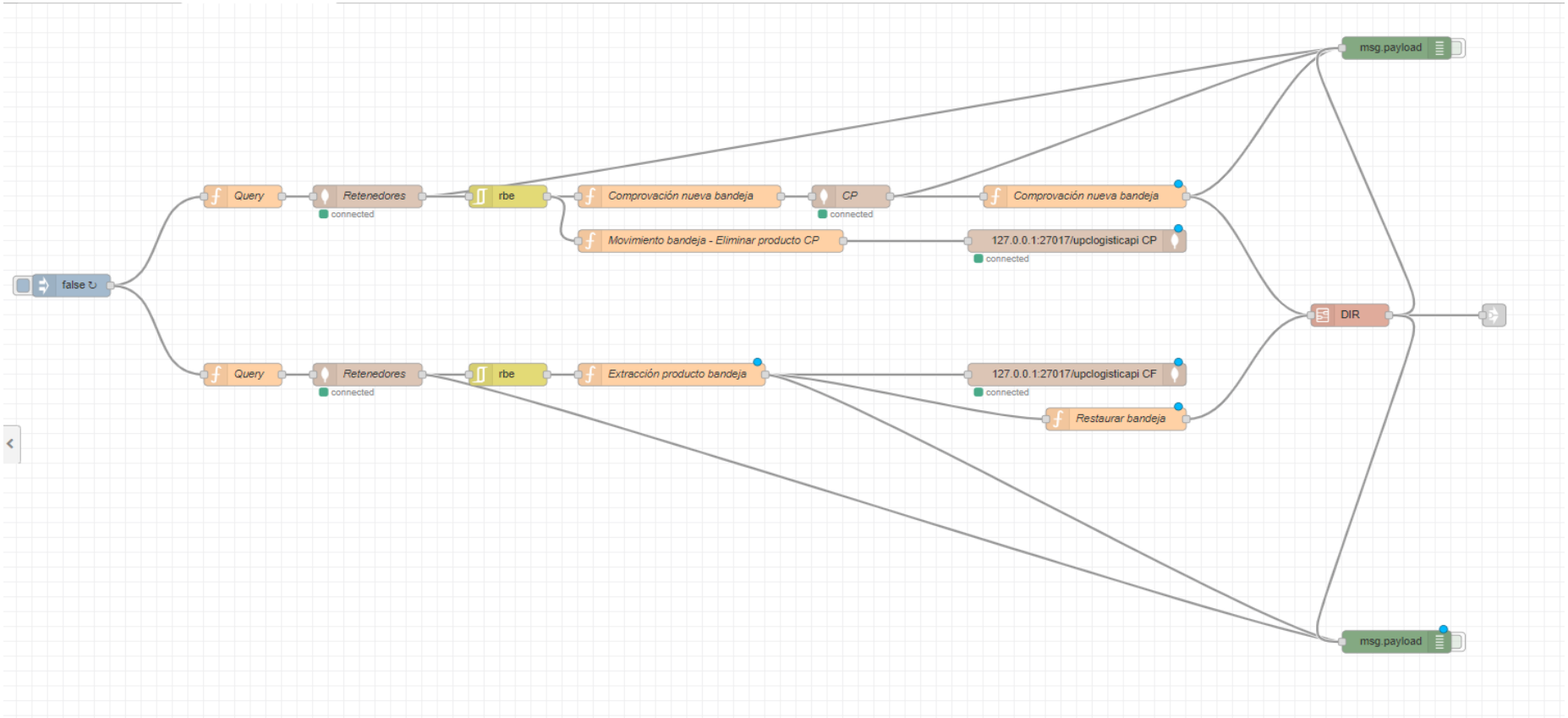

Figura 66. Flow II: Control de máquina. Fuente: Elaboración propia.

Principalmente, este *flow* implementa las acciones más básicas necesarias para el control automático de la instalación. Se han programado las funciones de volcado de la información de una bandeja en el PLC y recogida de los datos una vez acabado el proceso. Para ello, se ha denotado como retenedor principal DIR03 y como retenedor de fin de línea DIR19.

Inserción de bandejas en la instalación

Se realiza la comprobación del estado del retenedor inicial cada 2 segundos. Para poder realizar el volcado de una bandeja, es necesario que la máquina se encuentre en estado de "Producción". **En caso afirmativo, se prepara una petición a la base de datos para obtener la información del retenedor DIR03.**

Cumplir las condiciones anteriormente expuestas implica la obtención cíclica del estado de DIR03, dificultando la programación. Para ello solventar este problema e iniciar un único flujo, se utiliza el nodo "rbe", encargado de filtrar el objeto *msg*. En caso de que la información contenida en *msg* sea exactamente igual a la del objeto msg inmediatamente anterior, se filtra el *msg* y no se procesa.

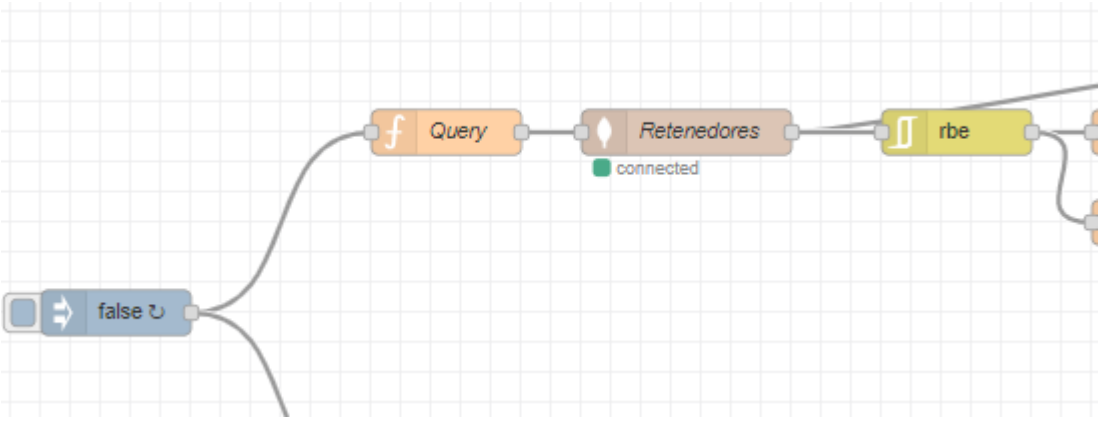

Figura 67. Peticiones cíclicas a la base de datos. Fuente: Elaboración propia.

Si el objeto *msg* implica un cambio respecto a su valor anterior, se comprueba en qué estado se encuentra el retenedor. Si se encuentra READY, es decir, con una bandeja lista para su envío, se procede a volcar la información de un producto. De forma paralela, se guardan los valores del objeto DIR03 en *msg.direntrada*. Es necesario para el volcado de los datos actualizados con la información del producto, puesto que no es necesario modificar más estados de los estrictamente necesarios. Esta maniobra se implementa en el nodo "Comprobación nueva bandeja".

Posteriormente, se realiza una petición a la colección CP para obtener la lista de productos por entrar en la línea. Se selecciona el primer valor del array devuelto. Su ObjectID se almacena para que, en el momento que el retenedor pase a estado MOVE (enviando la bandeja), se elimine de la cola de producción (CP) el objeto ya volcado.

Para completar el proceso, se actualiza el estado del retenedor (DIR03) obtenido como *msg.direntrada*. Se introducen los valores del producto en la estructura y se envían al nodo "DIR".

El nodo "DIR" es un subflujo, un conjunto de nodos dentro de un nodo; posteriormente se define con detalle su funcionamiento[21]. Su objetivo es traducir la estructura DIR03 a un array con los parametros necesarios para su volcado en el PLC, mediante el nodo "Modbus Flex Write", implementado en el *flow "FUNCIONES"*.

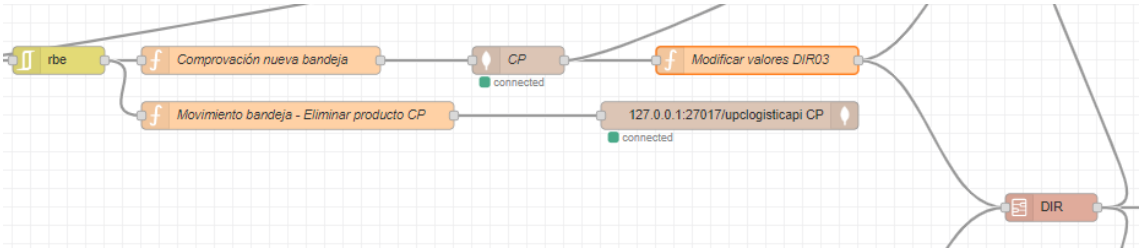

Figura 68. Volcado de una bandeja. Fuente: Elaboración propia.

Extracción de bandejas

El proceso de extracción es muy similar al de inserción, pero mediante el retenedor DIR19 que se ha escogido como fin de proceso. Se realizan peticiones a la base de datos para conocer el estado de DIR19. En el momento que existe un cambio en su estado, el nodo "rbe" permite su paso y estudian sus estados. En caso de encontrarse en estado "READY", se extrae la información contenida de PRODUCTOACTUAL. A continuación, se estandariza la información del producto para cumplir con la misma estructura de la cola de producción (CP). Finalmente, se inserta el valor en la cola de finalización (CF).

De forma paralela, se elimina el valor del producto de la estructura de DIR19 y se modifica su estado a REST. Para poder volcar los datos, se estandariza la estructura a

[21] Apartado "DIR".

un array de enteros[22] con los parametros necesarios para su escritura en el PLC mediante “Modbus Flex Write”. Esto se realiza mediante el subflujo “DIR”.

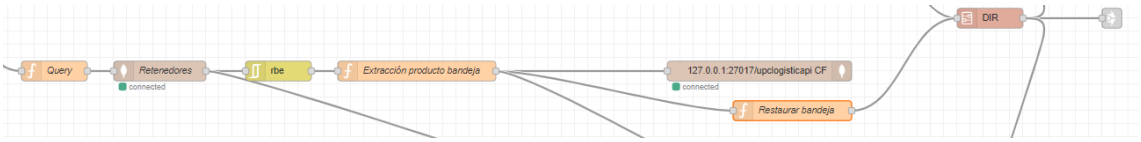

Figura 69. Extracción de una bandeja. Fuente: Elaboración propia.

DIR

Existen determinados flujos que se utilizan de forma repetitiva en diversas partes de la API. Mediante un subflujo, el código queda encapsulado de forma que puede ser instanciado en distintas maniobras, a modo de función. Concretamente, el subflujo “DIR” realiza la conversión de la estructura retenedor a un objeto que contiene los parámetros necesarios para su volcado en el PLC (dirección de memoria inicial, duración, tipo de escritura), junto con un array de enteros que codifican la información.

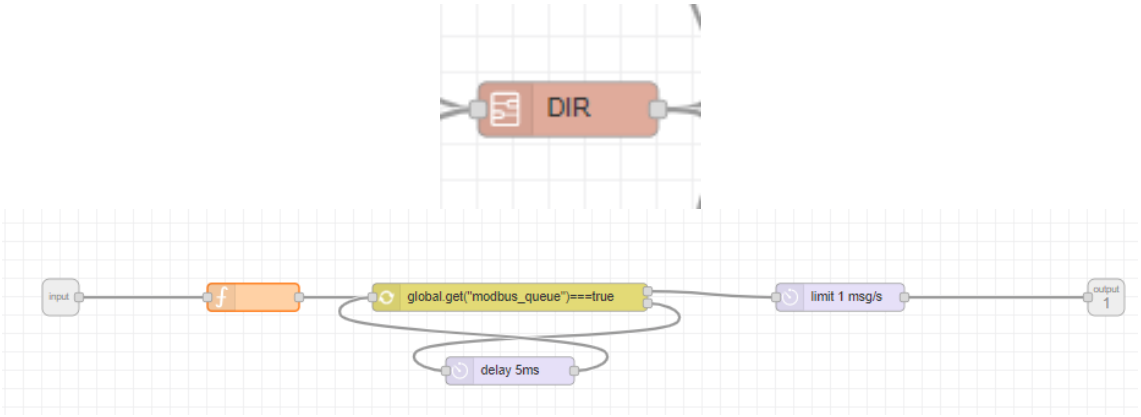

Figura 70. Subflujo "DIR". Fuente: Elaboración propia.

[22] Un array de enteros es un vector que contiene de forma consecutiva y organizada números enteros.

Está formado por un nodo de función que realiza la conversión y, para asegurar el correcto envío del objeto al nodo "Modbus Flex Write", un bucle de control que comprueba si la librería Modbus está en uso. En caso de estar en uso, espera 5 ms antes de volver a comprobarlo. Si por caso contrario está disponible, limita el envío a 1 msg/s para evitar la saturación el búfer.

El nodo función codifica dos booleanos en un único word de formato entero (un BOOL equivale a 6 posiciones de memoria en el PLC). Posteriormente, se organizan los enteros para que coincidan con la posición de memoria correspondiente.

Finalmente, se añade los parámetros para la comunicación Modbus TCP en *msg.payload* junto con el array de datos.

```
var array_plc = []
var ret = msg.payload
var dirmem = global.get("direcciones_retenedores["+msg.payload.DIR+"]")
if(ret.REST){
    if(ret.READY){
        array_plc[0] = 257;
    }else{
        array_plc[0] = 1;
    }
}else{
    if(ret.READY){
        array_plc[0] = 256;
    }else{
        array_plc[0] = 0;
    }
}


if(ret.MOVE){
    if(ret.BLOQ){
        array_plc[1] = 257;
    }else{
        array_plc[1] = 1;
    }
}else{
    if(ret.BLOQ){
        array_plc[1] = 256;
    }else{
        array_plc[1] = 0;
    }
}

array_plc[2] = ret.PRODACTUAL.ID;
array_plc[3] = ret.PRODACTUAL.TIPOPROD;
array_plc[4] = ret.PRODACTUAL.BANDEJA;
array_plc[5] = ret.PRODANTERIOR.ID;
array_plc[6] = ret.PRODANTERIOR.TIPOPROD;
array_plc[7] = ret.PRODANTERIOR.BANDEJA;
msg.retdestino = msg.payload.DIR;


    msg.payload = {
    'value': array_plc,
    'fc': 16,
    'unitid': 10,
    'address': dirmem,
    'quantity': 8

    }
```

Figura 71. Función del nodo "DIR". Fuente: Elaboración propia.

3.2.3 Flow III: Funciones

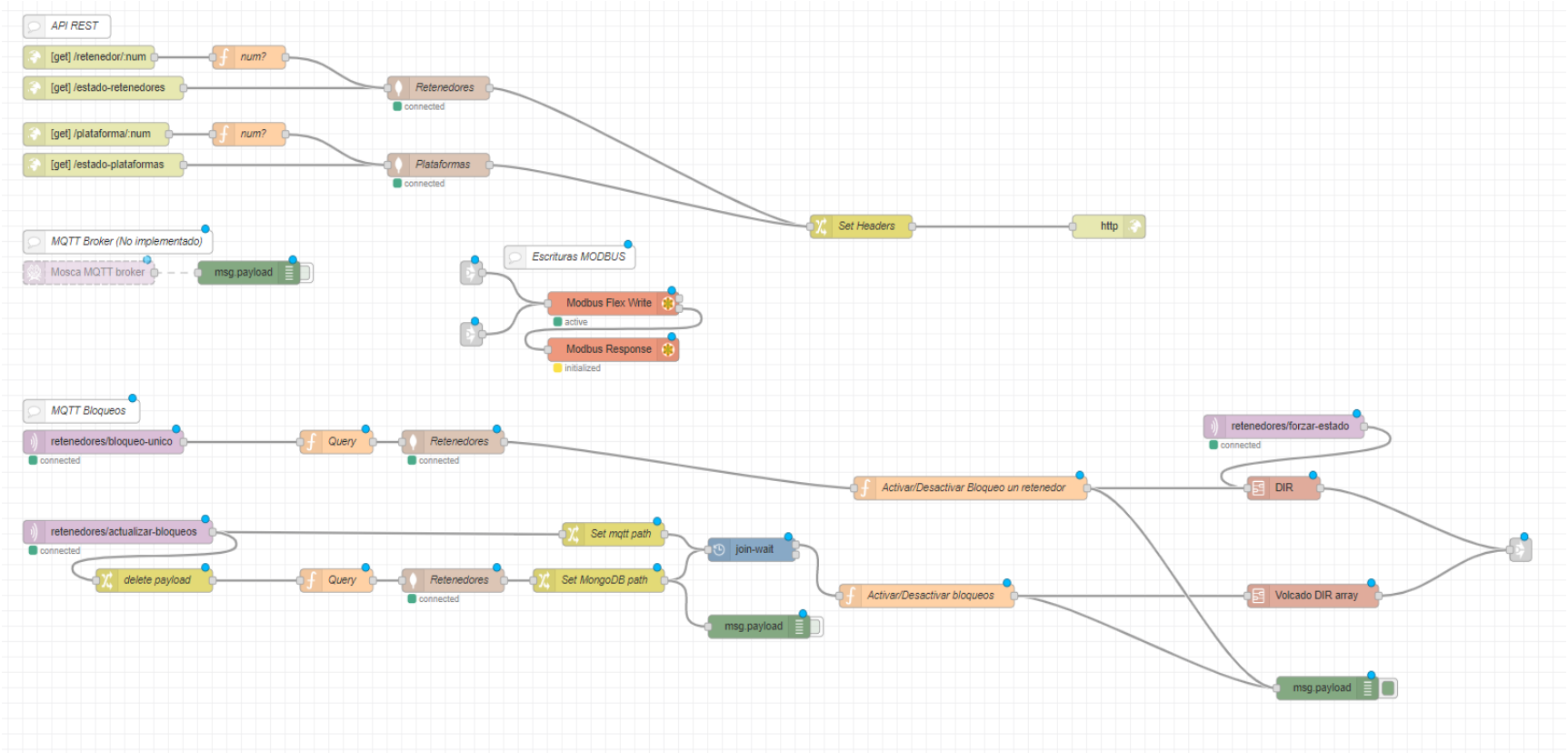

Figura 72. Flow III: Funciones. Fuente: Elaboración propia.

API REST

Este *flow* comprende las principales funciones que requiere una aplicación para modificar el estado de la máquina. En la parte superior izquierda se contemplan 4 funciones HTTP API REST, todas ellas mediante el verbo GET (petición de información).

- **/retenedor/num.** Devuelve un objeto JSON del estado de un retenedor concreto.
- **/estado-retenedores/.** Devuelve un array de objetos JSON con los estados de todos los retenedores.
- **/plataforma/num.** Devuelve un objeto JSON del estado de una plataforma concreta.
- **/estado-plataformas/.** Devuelve un array de objetos JSON con los estados de todas las plataformas.

Todas ellas llaman al nodo MongoDB "Retenedores" o "Plataformas" para la obtención de los objetos, con la diferencia que si se requiere un único retenedor/plataforma se utiliza como parámetro de filtrado en la base de datos. Finalmente, se devuelve el objeto como respuesta a la petición.

```
1 msg.payload = {
2     _id: ""+msg.req.params.num+""
3 }
4 return msg;
```

Figura 73. Uso del parámetro de obtenido de la petición GET. Fuente: Elaboración propia.

Bloqueo de un retenedor

El *topic MQTT* "retenedores/bloqueo-unico" interpreta la entrada de un objeto que contiene el retenedor objetivo y el valor booleano de bloqueo (activado, desactivado). Primero, se realiza una petición a la colección Retenedores para obtener el estado del retenedor en cuestión. En el nodo función "Activar/Desactivar bloqueo un retenedor", se asigna directamente al parámetro BLOQ del objeto retenedor el obtenido mediante MQTT. Finalmente, se traduce la estructura mediante el nodo "DIR" y se envía al PLC.

```
var ret = msg.payload[0]
ret.BLOQ = msg.actualizarbloqueo
msg.payload = ret
return msg;
```

Figura 74. Actualización de bloqueos. Fuente: Elaboración propia.

Bloqueo de todos los retenedores

El *topic* "retenedores/actualizar-bloqueos" espera la entrada de un array de booleanos [0..21] que corresponde al valor de bloqueo de cada retenedor. Para ello, se disocia el flujo en dos mensajes. El primer flujo contiene el valor del array comentado, mientras que el segundo realiza la petición del estado de todos los retenedores del sistema. Los dos flujos confluyen en el nodo "join-wait" que espera la llegada de los dos mensajes para continuar.

A continuación, se utiliza la función *forEach* para asignar a cada término del array de objetos el valor correspondiente dentro del array de booleanos.

```
function enviarbloqueos(ret, index) {
    ret.BLOQ = bloqueos_dir[counter]
    counter++
}


var counter = 3;
var bloqueos_dir = msg.paths.mqtt
var estado_retenedores = msg.paths.MongoDB

estado_retenedores.forEach(enviarbloqueos);
msg.payload = estado_retenedores


return msg;

```

Figura 75. Bloqueo de todos los retenedores. Fuente: Elaboración propia.

Finalmente, el array que contiene los estados de todos los retenedores, actualizados se traduce en un objeto que permite al "Modbus Flex Write" el volcado de los valores en el PLC. Esta conversión se realiza mediante el subflujo "Volcado DIR array". Funciona de forma similar al subflujo "DIR", mencionado anteriormente, con la diferencia que se convierte un volumen de datos equivalente a todos los retenedores.

Forzar el estado de un retenedor

Mediante el *topic* MQTT se puede realizar el forzado de una estructura retenedor íntegramente, siempre y cuando sea estándar y acorde con el formato establecido en "3.1 Estructuras de datos". El nodo "DIR" traduce la estructura al formato admitido por el nodo "Modbus Flex Write".

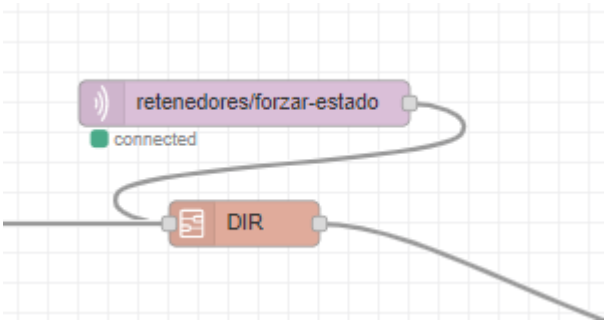

Figura 76. Forzado del estado de un retenedor. Fuente: Elaboración propia.

3.2.4 Flow IV: DIR/PT

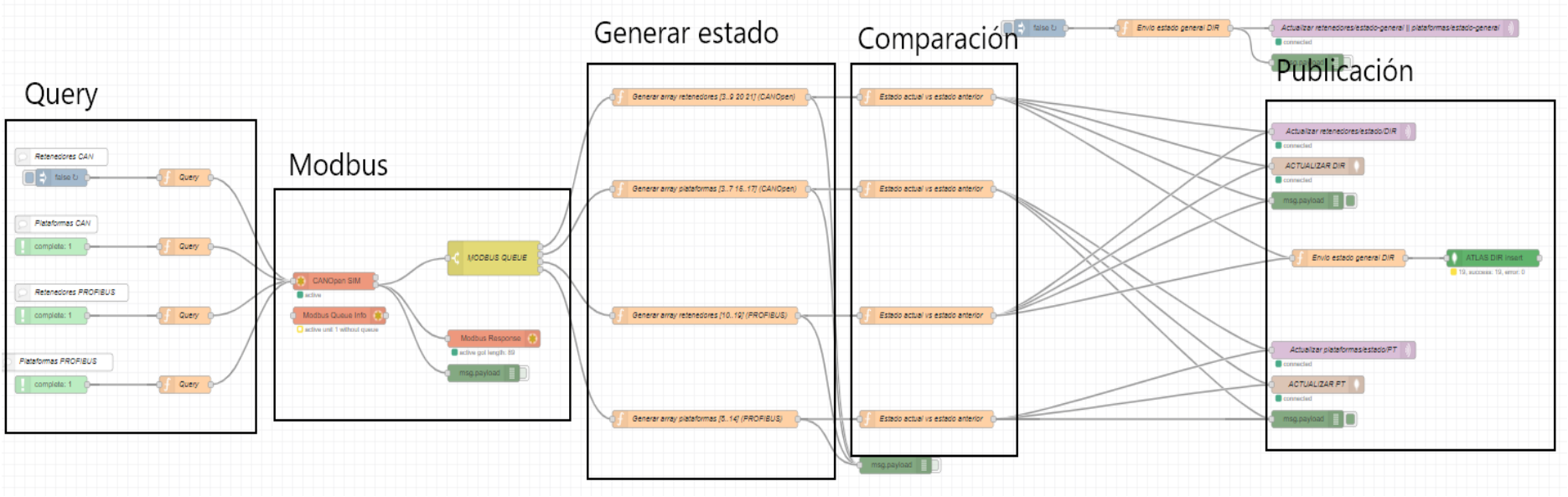

Figura 77. Flow IV: DIR/PT. Fuente: Elaboración propia.

Este *flow* tiene como principal objetivo la obtención del estado de los retenedores y plataformas. **Se considera el *flow* más importante, puesto que permite conocer el estado de la instalación, convirtiéndose en la base para prácticamente todas las demás funciones de interés.**

De forma general, el funcionamiento pasa por realizar peticiones de paquetes de información al PLC de forma cíclica. **En total, se realizan 4 peticiones por ciclo al PLC y de forma secuencial.** La información contenida en las 4 respuestas del PLC agrupa todos los datos respecto al estado de los retenedores y plataformas, los cuales se filtran y organizan hasta obtener objetos JSON.

Con el objetivo de facilitar al lector la comprensión del funcionamiento del actual *flow*, se disgregan los nodos en base a la necesidad a la que responden (apreciable en la anterior figura). **Tanto para retenedores como plataformas, el proceso de obtención del estado es exacto en cada subdivisión, con la única diferencia que las estructuras son intrínsecamente distintas.** A continuación, se definen las subdivisiones anteriormente mencionadas.

- ***Query***[23]. Preparar la petición Modbus a realizar al PLC. Especificación de la dirección de memoria inicial, duración, tipo de lectura a realizar. Se asigna un valor *String* en *msg.topic* con el objetivo de identificar a qué retenedores/plataformas corresponde la información contenida en el mensaje.
- ***Modbus.*** Obtención de la respuesta a la petición. El mensaje en cuestión contiene un array de enteros con la información referente al estado de los retenedores o plataformas en *msg.payload*. Se enruta el mensaje en base al *String* de *msg.topic* para permitir el correcto tratamiento de la información.
- **Generar estado.** La información contenida en *msg.payload* en formato de *array* de enteros, se traduce a un objeto JSON coherente con la estructura utilizada por el programa PLC[24].
- **Comparación.** Se compara el valor de los estados del objeto actual con el obtenido en la petición inmediatamente anterior. En caso de presentar una

[23] Término inglés ampliamente utilizado para hacer referencia a generar una petición a un sistema. Su traducción literal es “consulta”.
[24] La definición de las estructuras está disponible en la sección “3.1 Estructuras de datos”.

variación, se actualiza el valor del objeto correspondiente en los *arrays* globales "array_retenedores" o "array_plataformas". Si el objeto es idéntico al anterior en cuanto a sus valores, el mensaje es desechado. De esta forma, se obtiene un filtro: únicamente es comunicado el objeto si es estrictamente necesario porqué se ha producido una modificación. La reducción del tráfico de datos innecesarios es sensible como consecuencia de esta funcionalidad.

- **Publicación.** Si el estado del retenedor ha sufrido una modificación, se publica el objeto en el *topic* MQTT correspondiente y se actualiza su valor en la base de datos local MongoDB. De forma paralela, se inserta un nuevo documento con el valor del objeto en el clúster MongoDB ATLAS, con la finalidad de registrar tal modificación.

Query

El nodo "false" dispara la ejecución del flujo de forma cíclica cada 2 segundos. **Técnicamente se introduce un único mensaje en cada ejecución**, por ende, el objeto *msg* se reutiliza en todo el proceso de petición de valores[25]. En otras palabras, **el mismo objeto *msg* recorre todos los nodos de la instalación.**

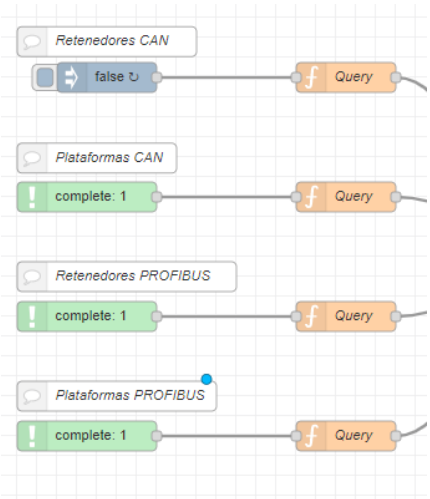

Figura 78. Nodos "Query". Fuente: Elaboración propia.

Esto sucede porque, una vez ejecutado el nodo "Generar array"[26], se dispara el nodo "complete" inmediatamente posterior; con el valor de *msg* obtenido de la ejecución de

[25] Anteriormente se ha comentado que se producen 4 peticiones al PLC para obtener el estado de todos los elementos de la instalación.
[26] Independientemente de si se trata de la información de un retenedor o plataforma.

"Generar array". El *msg.payload* de este último objeto obtenido será sobrescrito con la petición Modbus del siguiente tramo de información a obtener y así sucesivamente, hasta completar todo el flujo.

Se analiza el código del nodo "Query" correspondiente a los retenedores, los cuales deberían ser controlados por el PLC CAN Open para afianzar los conceptos anteriormente descritos.

```
1  global.set("modbus_queue",true)
2  var query = {
3      value: msg.payload,
4      'fc': 4,
5      'unitid': 2,
6      'address': 400,
7      'quantity': 98
8  }
9  msg.topic = "dircan"
10 msg.payload = query
11 return msg;
```

Figura 79. Programación de un nodo "Query". Fuente: Elaboración propia.

Primeramente, se establece como *true* el valor de la variable global "modbus_queue". Esta acción se implementa para evitar colisiones con los procesos de escritura[27], los cuales pueden estar requiriendo el uso de la librería paralelamente. El atributo *msg.payload* contiene la información necesaria para realizar la petición. Es relevante comentar que el PLC de simulación responde a peticiones iguales o inferiores a 157 posiciones de memoria consecutivas, es decir, realizar diferentes peticiones consecutivas se convierte en una necesidad.

Se establece en el atributo *msg.topic* el String "dircan" para su posterior identificación y tratamiento de datos, tal como se ha comentado con anterioridad. **Este String es único para cada petición, por tanto, existen 4 tipos de mensajes distintos en base a este filtro.**

Modbus

La respuesta obtenida del PLC a la petición realizada requiere ser tratada, puesto que un *array* de enteros no presenta valor por sí mismo. Para ello, **en base al valor del atributo *msg.topic*, se encamina el mensaje.**

[27] En "DIR" se explica la estrategia utilizada en la escritura para evitar las colisiones.

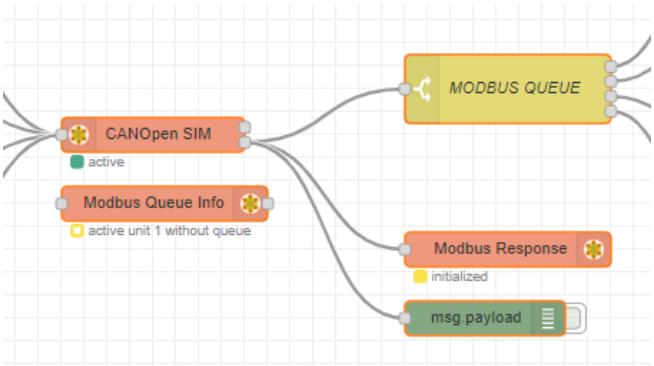

Figura 80. Nodos de petición al PLC de simulación. Fuente: Elaboración propia.

El nodo "MODBUS QUEUE"[28] tiene la funcionalidad de disgregar los mensajes a partir del identificador. En caso de ser un mensaje con valor del atributo *msg.topic* "ptcan", se enrutará por la segunda salida del nodo. Esto ocurre de la misma forma para los demás tipos de identificadores existentes en la imagen contigua.

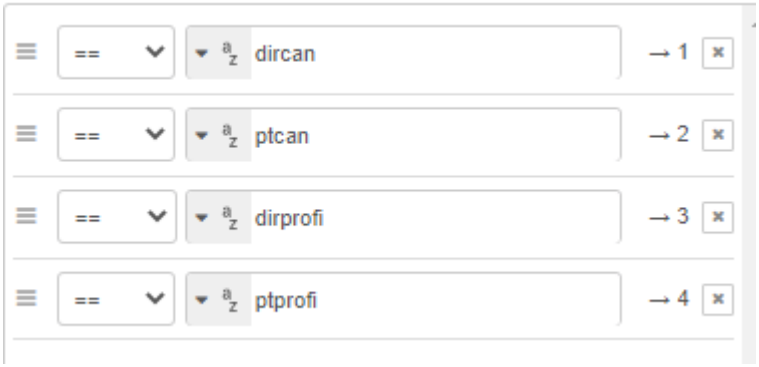

Figura 81. Programación del nodo "MODBUS QUEUE". Fuente: Elaboración propia.

[28] Este nombre se añade para facilitar el entendimiento del flujo, pero la realidad es que este nodo se identifica en la paleta de node-RED como "switch".

Generar estado

La traducción del array de valores a objetos JSON, respetando las estructuras previamente definidas, se realiza mediante un bucle. A continuación, se analiza el código correspondiente a la generación de los objetos del PLC CAN Open.

```
var array_local_retenedores = [];
var a = 0;
var b = 0;
for(var i=3;i<=21;i++){
if(i<=9 || i===20 || i===21){
array_local_retenedores[i]= {
        "DIR": i,
        "REST": msg.payload.buffer[a+1],
        "READY": msg.payload.buffer[a],
        "MOVE": msg.payload.buffer[a+3],
        "BLOQ": msg.payload.buffer[a+2],
        "PRODACTUAL":{
            "ID": msg.payload.data[b+2],
            "TIPOPROD": msg.payload.data[b+3],
            "BANDEJA": msg.payload.data[b+4]
        },
        "PRODANTERIOR":{
            "ID": msg.payload.data[b+5],
            "TIPOPROD": msg.payload.data[b+6],
            "BANDEJA": msg.payload.data[b+7]
        }
    }

a = a + 20;
b = b + 10;
}
}

msg.payload = array_local_retenedores;
return msg;
```

Figura 82. Programación de un nodo para la generación de una estructura RETENEDOR. Fuente: Elaboración propia.

El número de ejecuciones del bucle corresponde al total de retenedores en la instalación. En cada ejecución, se realiza un filtrado en base al número ejecución actual, si coincide con el valor de un retenedor de CAN Open[29] se produce a la extracción de la información.

Gracias al hecho que las estructuras poseen un tamaño determinado[30], es posible recorrer el array de enteros en base a dos punteros (variables a y b) que incrementan

[29] Retenedores DIR03 – DIR09, DIR20 y DIR21.
[30] En el caso de los retenedores, 8 words (cada word equivale a 2 bytes), con 2 words de separación con el inmediatamente consecutivo.

su valor en el equivalente al tamaño de un retenedor (inicialmente tienen valor 0 y aumentan en 10 y 20 respectivamente) tras cada ejecución. De esta manera, a y b identifican el primer entero de cada retenedor dentro del array, y mediante la suma de constantes, se accede al valor necesario en la declaración de la estructura. Se devuelve el array de objetos JSON en el atributo *msg.payload*.

Comparación

Para poder discernir si ha variado algún estado del retenedor o plataforma, **se requiere de la creación de un nodo de función que permita filtrar todos aquellos mensajes idénticos en valor al recogido actualmente, por la base de dato MongoDB**. Esta implementación separa las repuestas sin valor (correspondiente al estado actual, no ha existido ninguna variación en el estado) de aquellas que si contienen una modificación. Podría ser posible realizar esta funcionalidad con el nodo "rbe", pero **en el momento de la programación de esta sección de código, no se tenía constancia de su existencia**.

```
var actual= [];
var array_retenedores_local = []
actual = msg.payload;
var ultimoestado
array_retenedores_local = global.get("array_retenedores")

for(var i=3;i<=21;i++){
if( i<=9 || i===20 || i===21){
    if(actual[i].DIR != global.get("array_retenedores["+i+"].DIR") ||
       actual[i].REST != global.get("array_retenedores["+i+"].REST") ||
       actual[i].READY != global.get("array_retenedores["+i+"].READY") ||
       actual[i].MOVE != global.get("array_retenedores["+i+"].MOVE") ||
       actual[i].BLOQ != global.get("array_retenedores["+i+"].BLOQ") ||
       actual[i].PRODACTUAL.ID != global.get("array_retenedores["+i+"].PRODACTUAL.ID")
       ){
        global.set("array_retenedores["+i+"]",actual[i])
        msg.payload = actual[i]
        msg._id = ""+actual[i].DIR+""
        msg.topic = "retenedores/estado/"+actual[i].DIR+""
      node.send(msg);
    }else{
        global.set("array_retenedores["+i+"]",actual[i])
        }
    }
}

return;
```

Figura 83. Comparación del estado actual del retenedor con el anterior registrado.
Fuente: Elaboración propia.

Se realiza el recorrido del *array* de estados[31] y, si el número de ejecución coincide con el referente a un retenedor, **se ejecuta una segunda comparación para determinar si se ha modificado su estado versus el inmediatamente anterior**.

[31] Se analiza el código referente a la generación de los objetos del PLC CAN Open.

En caso de resultar afirmativo, se ajusta el atributo *msg.topic* para contener la ruta del *topic* MQTT correspondiente. Finalmente, se actualiza su estado en el *array* utilizado para dicha comprobación y el objeto *msg* es enviado. Por lo contrario, en caso de resultar negativo, únicamente se actualiza en valor en el *array* comentado.

Cabe señalar que en la segunda comparación es parametrizable la cantidad de estados que se requieren comprobar, para poder graduar la permisividad del filtro.

Publicación

La publicación de los mensajes hace uso del contenido en *msg.payload,* como objeto a publicar en las diferentes plataformas contempladas. En MQTT se genera un *topic* propio para cada retenedor o plataforma, con actualización a cada variación del estado. De forma paralela, también se ha creado un *topic* MQTT que publica el array de estados cada 2 segundos independientemente de sus modificaciones. **Se ha contemplado esta solución dada la comodidad que supone hacer uso de la misma en determinadas aplicaciones**[32].

En cada modificación, el objeto es insertado en el clúster MongoDB ATLAS para identificar una variación en el estado de un retenedor o plataforma.

[32]Hacer uso de esta función implica escuchar un único *topic* MQTT, mientras que en el caso individual se realiza la escucha de 17 *topics* independientes.

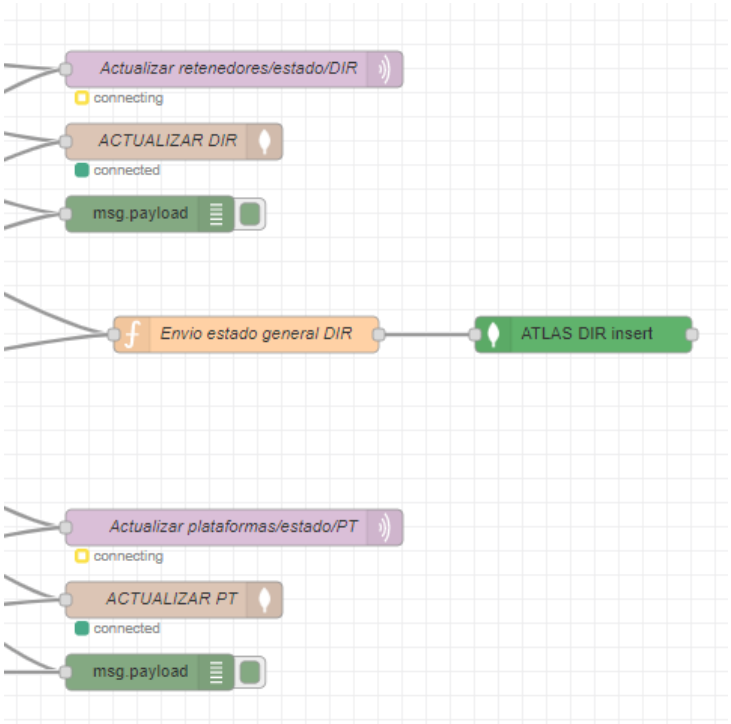

Figura 84. Nodos encargados de publicar los mensajes. Fuente: Elaboración propia.

4. Validación

Con el objetivo de validar de forma experimental[33] la API, se hace uso de un cliente MQTT que permite el envío de mensajes de forma manual. Se ha decidido utilizar MQTT Explorer , principalmente por permitir la reutilización de mensajes. A continuación, se recrea todo el proceso de interactuación para validar su funcionamiento.

[33] Se entiende como experimental una validación simulada del proceso, diferente a la validación práctica que requeriría el uso de la línea física.

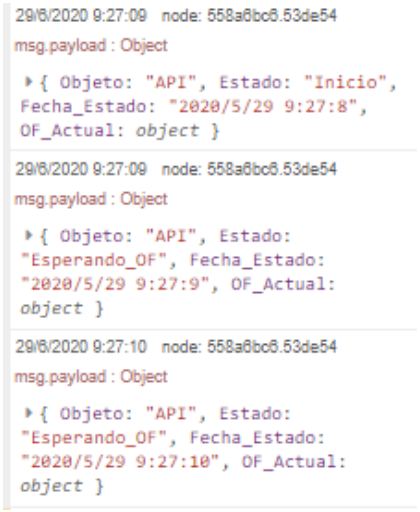

Figura 85. Mensajes de estado en la consola *debug.* Fuente: Elaboración propia.

Tras el arranque del programa en node-RED, de forma automática la API se coloca en estado de "Esperando_OF", de acuerdo con las capturas realizadas a la consola de *debug.* Para poder utilizar una orden de fabricación, previamente es necesario realizar su inserción en la base de datos. Esta inserción se realiza mediante el *topic* MQTT "maquina/nueva-OF", la cual requiere un objeto JSON con los parámetros necesarios[34].

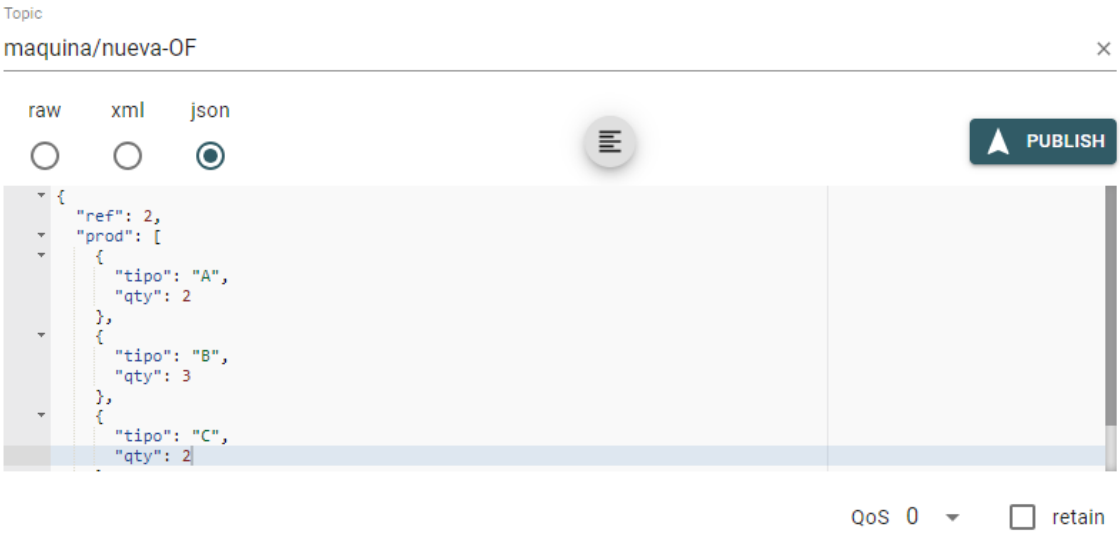

Figura 86. Inserción de una nueva orden de fabricación. Fuente: Elaboración propia.

[34] Descritos en el Anexo "API REFERENCE".

Una vez publicado el mensaje, se crea un documento en la colección OF de la base de datos MongoDB. La API actualmente se encuentra en estado “Esperando-OF”. Para poder seleccionar la orden de fabricación recientemente insertada, es necesario publicar un mensaje en “maquina/parametros” con un identificador de usuario y la referencia de la orden requerida.

Figura 87. Orden de fabricación en la base de datos. Fuente: Elaboración propia.

De forma automática, la API recoge los datos de la orden de fabricación y pasa a modo “Preparada”, acorde con el mensaje mostrado en la consola de *debug*. Paralelamente, se ha generado la cola de producción (CP) para su volcado en el PLC en el momento pertinente.

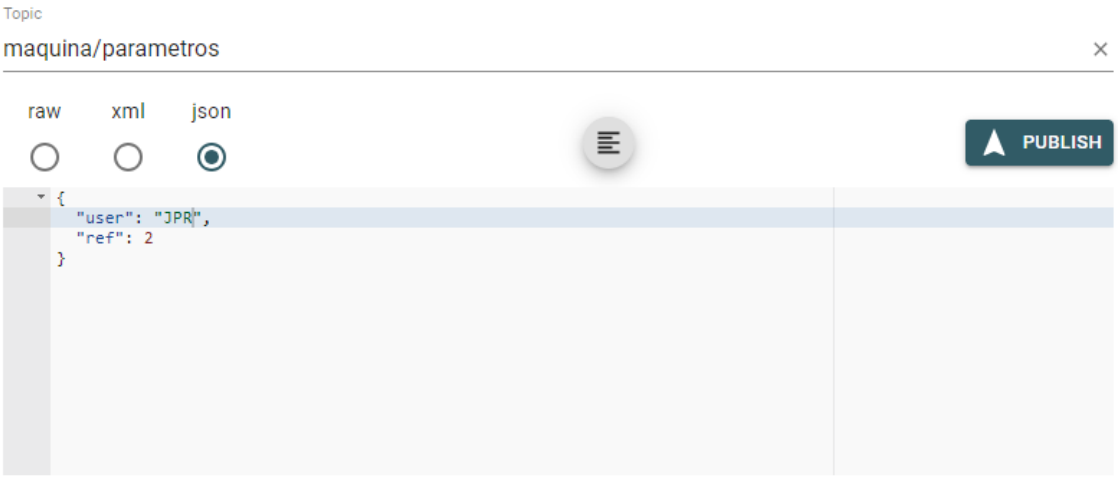

Figura 88. Selección de una orden de fabricación. Fuente: Elaboración propia.

Figura 89. Mensajes de estado de la API en la consola *debug*. Fuente: Elaboración propia.

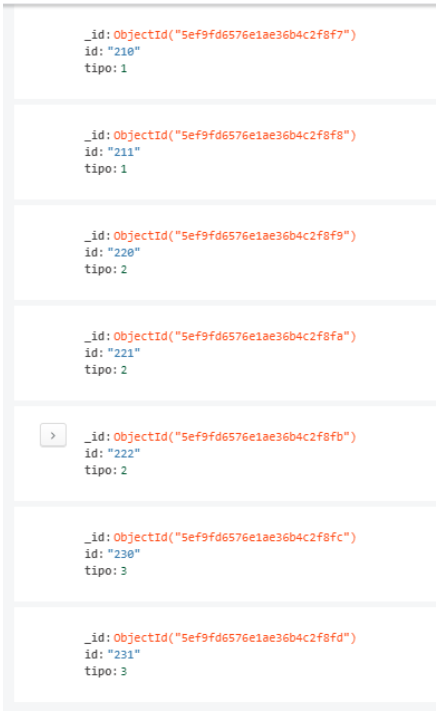

Figura 90. Cola de producción. Fuente: Elaboración propia.

Para poner en marcha la instalación, se publica en el *topic* “maquina/parametros” un objeto con el identificador de usuario y la acción a realizar. En este caso, “Start”. Si no coincide el identificador de usuario del mensaje con el ya establecido en la API, se reinicializaría la API puesto que sería necesario empezar de nuevo. Tras la publicación del mensaje, la máquina pasa a modo “Producción”.

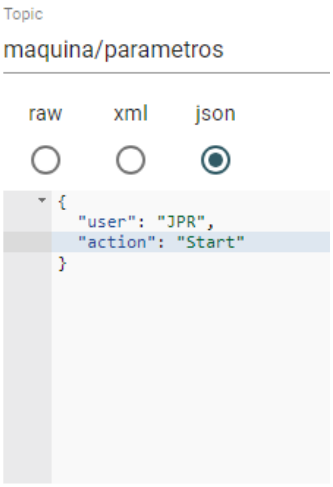

Figura 91. Encendido de la API. Fuente: Elaboración propia.

Figura 92. Mensaje de estado de "Producción" en la consola *debug*. *Fuente: Elaboración propia.*

Desde la inicialización de la API, se analiza constantemente el estado de los retenedores y plataformas. La primera ejecución publica en la consola *debug* un mensaje por cada retenedor o plataforma. Por tanto, la evaluación de estado es continua, independientemente del estado de la API.

En el momento que la API pasa a modo “Producción”, se comprueba de forma periódica el estado del DIR13 para permitir el volcado de información de las bandejas. Si el retenedor se encuentra en estado de READY, automáticamente se realiza el volcado de los datos. En este caso, se trata del producto con referencia “210”, del tipo “B”.

Figura 93. Mensajes de estado de los retenedores en la consola *debug*. Fuente: Elaboración propia.

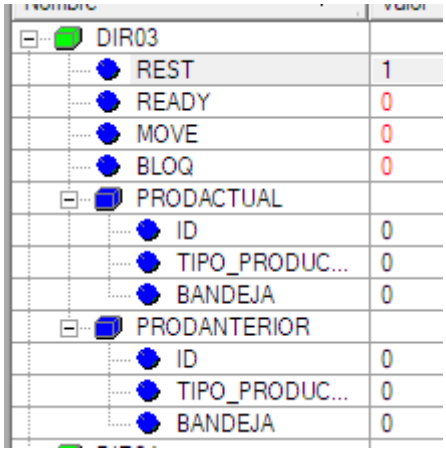

Figura 94. Retenedor DIR03 en modo "REST". Fuente: Elaboración propia.

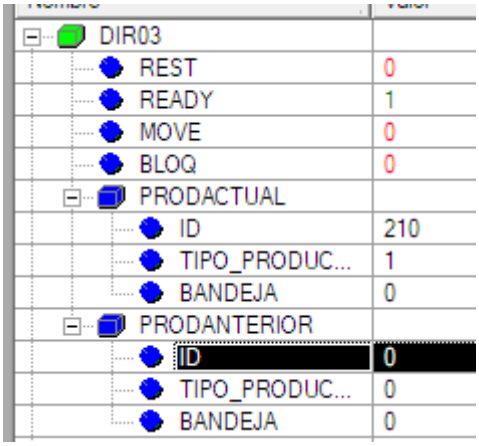

Figura 95. Retenedor DIR03 en modo "READY" con la información de la bandeja disponible. Fuente: Elaboración propia.

Una vez la información volcada en el PLC, la bandeja ya está lista para recorrer toda la instalación. **En este proyecto, no se han implementado funciones más allá del volcado y recogida de bandejas. Por tanto, el bloqueo de las plataformas que permitiría encaminar la bandeja por la instalación, no está disponible**[35].

En el momento en que la bandeja que contiene el producto “210” de tipo “B” llega al retenedor final (DIR19), éste se encuentra en estado “READY”. Esta variación es detectada por la API, que, automáticamente, recoge los datos del producto y pone en estado de “REST” el retenedor; puesto que la bandeja ha sido extraída manualmente. A continuación, se elimina de la cola de producción (CP) el producto finalizado y se insertan los valores en la cola de finalización (CF).

[35] No se ha querido dar un sentido práctico a la línea, puesto que se trata de un proyecto técnico en el ámbito académico y no de ejecución real en planta.

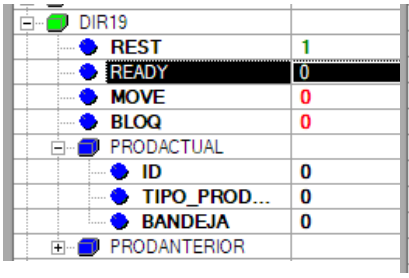

Figura 96. DIR19 en modo "REST" sin bandeja. Fuente: Elaboración propia.

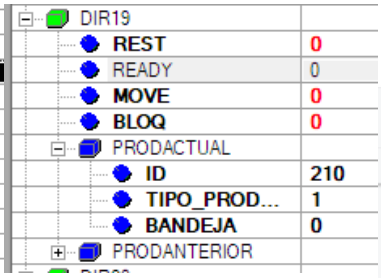

Figura 97. DIR19 en modo "READY" con la información de la bandeja disponible. Fuente: Elaboración propia.

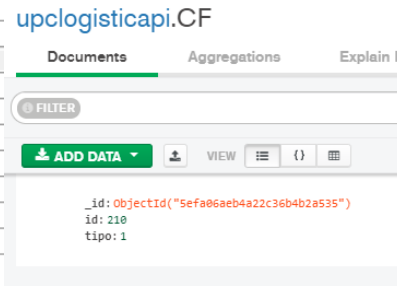

Figura 98. Cola de finalización. Fuente: Elaboración propia.

Independientemente del proceso de gestión de productos y arranque de la API, se puede bloquear el funcionamiento de los retenedores mediante los topics “retenedores/bloqueo-unico” y “retenedores/actualizar-bloqueos”. El primer *topic* requiere un objeto con el identificador del retenedor a bloquear/desbloquear y el nuevo valor deseado.

Por otro lado, en caso de querer parametrizar el bloqueo de todos los retenedores a la vez, se requiere la publicación de un array de booleanos con el valor a actualizar para cada estado de bloqueo. Esta implementación es de gran utilidad cuando se precisa establecer un funcionamiento general para la instalación36.

[36] En caso de crear una estación y necesitar parar el flujo de bandejas porque se está trabajando con ellas, se pueden establecer los bloqueos mediante la publicación de un único mensaje.

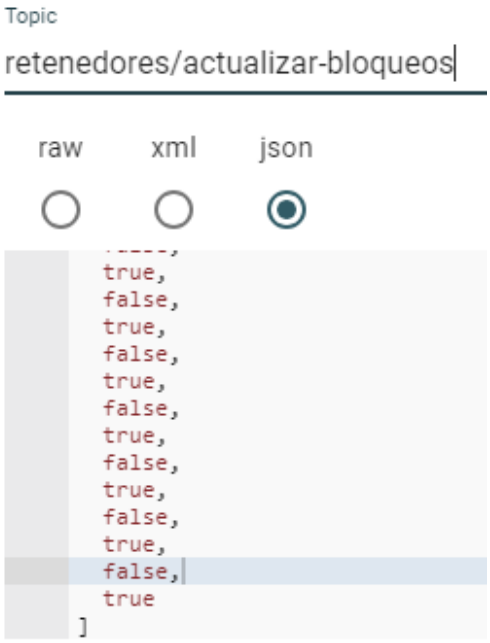

Figura 99. Array de bloqueo. Fuente: Elaboración propia.

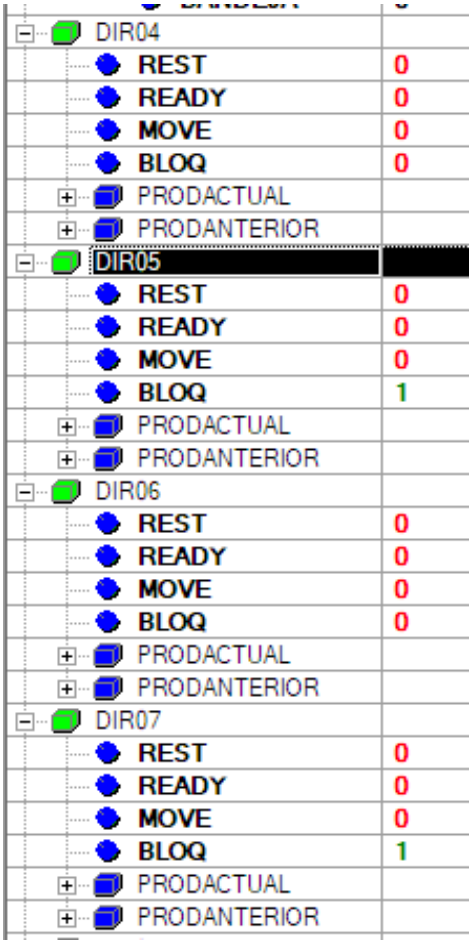

Figura 100. Estado de los bloqueos de diferentes retenedores.
Fuente: Elaboración propia.

Se precisa validar el forzado de un retenedor al completo mediante la publicación del objeto JSON en el *topic* "retenedores/forzar-estado". Para ello, se establece el objeto como mensaje en la aplicación MQTT Explorer. **El resultado obtenido en el PLC corresponde término a término con el objeto definido.**

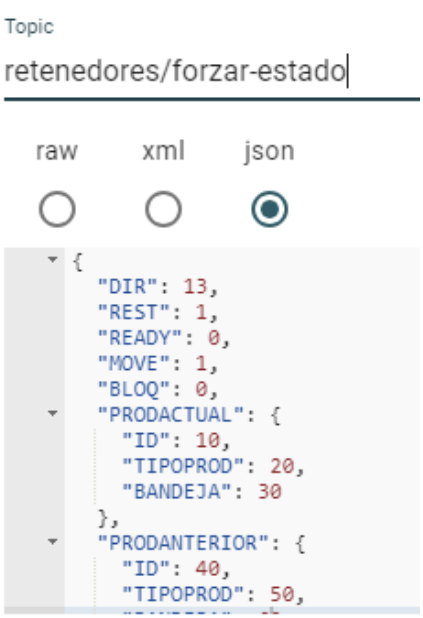

Figura 101. Forzado de un retenedor. Fuente: Elaboración propia.

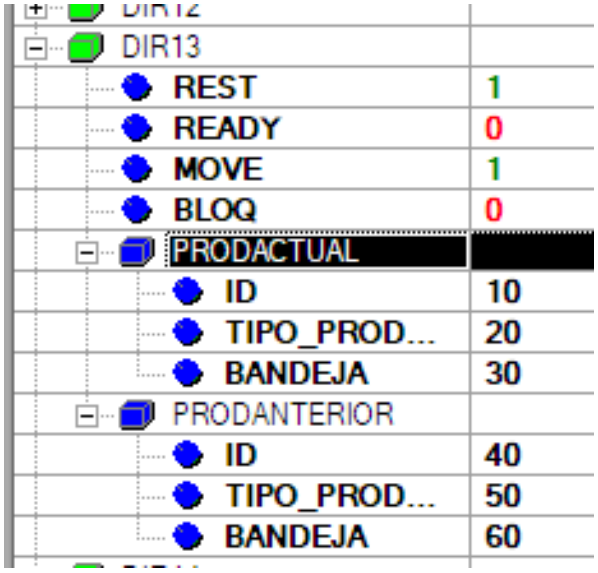

Figura 102. Estado del retenedor forzado. Fuente: Elaboración propia.

Anteriormente, se ha mencionado que para obtener el estado de los retenedores y plataformas es posible utilizar de forma alternativa a MQTT a partir de peticiones HTTP API REST. En el caso de uso de las peticiones HTTP al API REST, se utiliza el cliente Postman[37] para comprobar que el funcionamiento es acorde al requerido[38] [31]. Para realizar la petición, únicamente se requiere la IP, el *endpoint*[39], y el verbo que se pretende utilizar. A continuación, se presentan las peticiones y respuestas obtenidas.

[37] Postman es una aplicación que realiza peticiones a APIs para obtener los valores sin la necesidad de desarrollar un *frontend* específico.
[38] Se precisa obtener el estado de los retenedores y plataformas de forma individual o colectiva.
[39] Ruta del servidor a la que se pretende acceder.

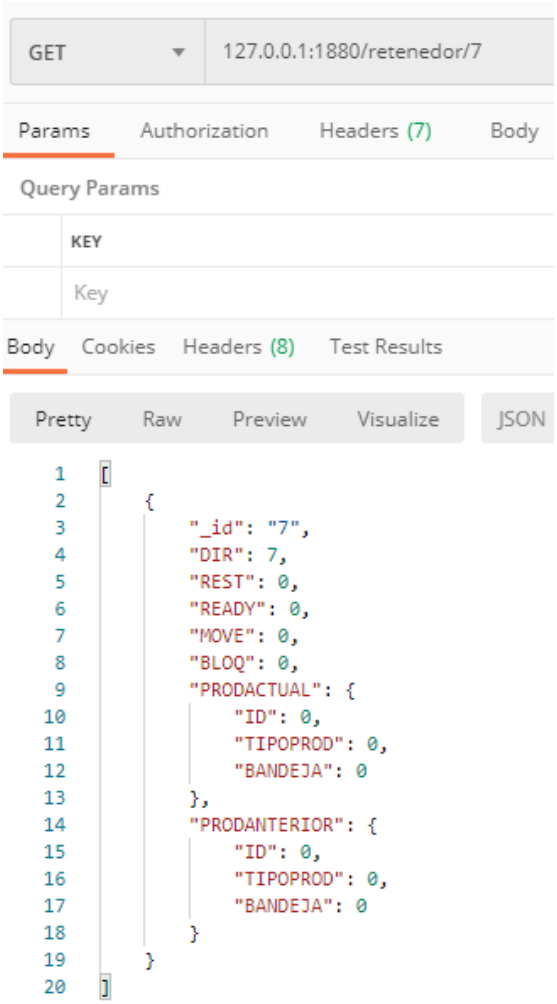

Figura 103. Respuesta a la petición del estado de un retenedor concreto. Fuente: Elaboración propia.

Figura 104. Respuesta a la petición de estado de todos los retenedores. Fuente: Elaboración propia.

Figura 105. Respuesta a la petición del estado de una plataforma concreta. Fuente: Elaboración propia.

Figura 106. Respuesta a la petición de estado de todas las plataformas. Fuente: Elaboración propia.

Capítulo IV: Conclusiones

1. Conclusiones generales

En el marco teórico de este proyecto se ha realizado un paso por los conceptos más importantes asociados a la cuarta revolución industrial, desde una perspectiva académica. Entender que la Industria 4.0 se ramifica hasta el aspecto más básico dentro de la organización industrial, sin mostrarse como una implementación por si misma, es clave para entender cómo la intangibilidad de este concepto lo hace tan potencialmente revolucionario.

Existe una pregunta que va estrechamente asociada a la cuarta revolución industrial: ¿Por qué ahora? La respuesta es simple: la sociedad y las tecnologías de la información están más unidas que nunca. Es ahora cuando se han dado todos los ingredientes sociales y tecnológicos necesarios para que la Industria 4.0 se pueda llevar a cabo.

Cabe destacar que la información ha relevado su papel de adjunto en la toma de decisiones y se ha convertido en el objeto de valor más importante en el ámbito empresarial. Y, en consecuencia, en el industrial.

La mayor problemática asociada a la Industria 4.0 es la dificultad de implementar los conceptos comentados en el marco teórico dentro de la realidad industrial. Este proyecto ha demostrado que es posible, en base a los avances en tecnologías de la información aplicadas a la industria, conectar la realidad operativa de una instalación con la cara más intangible y virtual de un negocio.

De entre todas las tecnologías actuales y, en cuanto a la aplicación práctica de este proyecto, se constata que el *Industrial Internet of Things* presenta un potencial de transformar la organización de las factorías prácticamente inimaginable. De modo que, más allá de quedar resuelta la brecha entre las tecnologías de la información y las de operación, existen solucione totalmente gratuitas que permite realizar implementaciones de coste reducido; de acuerdo con la experiencia profesional aportada por el autor, la presente memoria, y el presupuesto adjunto al proyecto.

Las arquitecturas expuestas ejemplifican cómo trasladar a la realidad las tecnologías tratadas en el Estado del arte, desde un punto de vista práctico. Concretamente, el poder recurrir al uso de *Gateways* industriales facilita el trabajo de los ingenieros de desarrollo. De hecho, en el caso concreto del presente proyecto, la innovadora tecnología node-RED ha demostrado que se puede realizar una implementación

operativa, gracias a la programación visual, en tiempos de desarrollo menores en comparación con otras tecnologías y otros lenguajes de programación.

Este proyecto, a priori, se ideó para ser implementado a nivel operativo en el laboratorio de automatización. No obstante, el COVID-19 supuso un giro drástico en torno a la previsión de desarrollo del proyecto. Como consecuencia, el proyecto quedó limitado a un modelo *beta* validado de forma simulada, a falta de trasladarlo al entorno real del laboratorio.

En el desarrollo de este proyecto, el autor ha extraído aprendizajes de valor tanto a nivel transversal como técnico. Por ello, se considera vital seguir investigando de forma académica la aplicación de las tecnologías de la información en el ámbito industrial, con la finalidad de llevarlo a un escenario práctico y contribuir en la mejora continua de la industria.

2. Líneas futuras

De acuerdo con las conclusiones generales expuestas, este proyecto no ha podido tener el alcance deseado a causa de la pandemia mundial que ha asolado el país (COVID-19). La crisis vinculada a la misma ha conllevado un cambio de paradigma en términos sociales y económicos, así como en el ámbito del desarrollo de proyectos.

Concretamente, la validación experimental de la instalación no ha sido posible como consecuencia del cierre de la universidad. De forma análoga, el enfoque de prácticamente todo el proyecto ha tenido que ser modificado y llevado a cabo a modo de simulación. Por ende y en la línea de las conclusiones extraídas en el presente documento, esta API todavía no está dispuesta para su implementación en el laboratorio.

De esta forma, podemos considerar entonces que la primera línea de desarrollo futuro es la modificación de la propia API para que ésta pueda ser implementada desde un punto de vista práctico. En este sentido, se plantean nuevos retos asociados como es el caso del comportamiento de la API en sí o, más concretamente, la interacción con los controladores; cuando se precise el acceso a tres PLC distintos, con altos tiempos de respuesta.

Anteriormente, se ha mencionado la implementación de un sistema de tracking de bandejas en base al recorrido teórico que realizan. Esta función no recibe ningún feedback físico de la localización real del objeto y, por tanto, las bandejas pueden ser extraídas durante el proceso sin dejar constancia de ello. Para solventar esta problemática, se presenta una segunda línea de actuación basada en la integración de la API con la tecnología RFID, disponible en el laboratorio, con el objetivo de realizar un tracking real de las bandejas (actualmente implementado). De esta manera, se aceleraría el programa de los controladores (menores tiempos del ciclo de SCAN) y se conocería la posición de las bandejas en tiempo real, mejorando sustancialmente la fiabilidad del sistema.

En términos de planificación, se recomendaría realizar la implementación de la primera línea futura propuesta (aplicación de la API en el laboratorio) y, posteriormente, la segunda (tracking automático de las bandejas). Ambas supondrían un avance considerable en el proyecto base, propuesto en el presente documento.

Referencias

[1] R. Geissbauer, L. Evelyn, S. Schrauf, and S. Pillsbury, "PwC Global Digital Operations 2018 Survey," 2018.

[2] J. Lee, H. Davari, J. Singh, and V. Pandhare, "Industrial Artificial Intelligence for industry 4.0-based manufacturing systems," 2018.

[3] P. Larrañaga, D. Atienza, J. Diaz-Rozo, A. Ogbechie, C. Puerto-Santana, and C. Bielza, *Industrial Applications of Machine Learning*. 2018.

[4] A. Glória, F. Cercas, and N. Souto, "Design and implementation of an IoT *Gateway* to create smart environments," *Procedia Comput. Sci.*, vol. 109, pp. 568–575, 2017.

[5] D. S. Niño-Becerra, "La Industria 4.0," in *Foro Económico Diario Montañés*, 2017.

[6] "The Industrial Internet of Things Volume G1: Reference Architecture," 2019.

[7] N. Jazdi, "Cyber physical systems in the context of Industry 4.0," in *Proceedings of 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, AQTR 2014*, 2014.

[8] A. G. Frank, L. S. Dalenogare, and N. F. Ayala, "Industry 4.0 technologies: Implementation patterns in manufacturing companies," *Int. J. Prod. Econ.*, vol. 210, pp. 15–26, Apr. 2019.

[9] "Integración de sistemas en la Industria 4.0 – Signals IoT." [Online]. Available: https://signalsiot.com/integracion-de-sistemas-en-la-industria-4-0/. [Accessed: 06-May-2020].

[10] "Macrodatos." [Online]. Available: https://es.wikipedia.org/wiki/Macrodatos. [Accessed: 12-May-2020].

[11] C. Juan, "¿Cuáles son las 5 V's del Big Data?," *Think. Innov.*, Nov. 2016.

[12] "BBVA API Market." [Online]. Available: https://www.bbvaapimarket.com/documentation/bbva/paystats-download. [Accessed: 12-May-2020].

[13] "Boeing 787s to create half a terabyte of data per flight, says Virgin Atlantic | Computerworld."

[14] G. Xiong, T. Ji, X. Zhang, F. Zhu, and W. Liu, "Cloud operating system for industrial application," in *10th IEEE Int. Conf. on Service Operations and Logistics, and Informatics, SOLI 2015 - In conjunction with ICT4ALL 2015*, 2015, pp. 43–48.

[15] S. Namasudra, P. Roy, and B. Balusamy, "Cloud computing: Fundamentals and research issues," in *Proceedings - 2017 2nd International Conference on Recent Trends and Challenges in Computational Models, ICRTCCM 2017*, 2017, pp. 7–12.

[16] K. B. Nayar and V. Kumar, "Cost benefit analysis of cloud computing in education," *Int. J. Bus. Inf. Syst.*, vol. 27, no. 2, p. 205, 2018.

[17] MongoDB, "MongoDB Atlas," p. https://www.mongodb.com/cloud/atlas, 2019.

[18] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informatics*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.

[19] Aniruddha Chakrabarti, "Emerging Open and Standard Protocol Stack for IoT," vol. 1, no. 1, pp. 2–6, 2015.

[20] A. Gilchrist, *Industry 4.0: The Industrial Internet of Things*. 2016.

[21] "Modbus." [Online]. Available: https://es.wikipedia.org/wiki/Modbus. [Accessed: 02-Jun-2020].

[22] F. A. Candelas Herías, "Comunicación con RS-485 y MODBUS," *Univ. Alicant.*, no. 37800, pp. 1–28, 2011.

[23] "What is OPC? - OPC Foundation." [Online]. Available: https://opcfoundation.org/about/what-is-opc/. [Accessed: 04-Jun-2020].

[24] "MQTT Version 5.0." [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. [Accessed: 08-Jun-2020].

[25] "Eclipse Mosquitto." [Online]. Available: https://mosquitto.org/. [Accessed: 12-Jun-2020].

[26] "*Gateways* IoT | Aprendiendo Arduino." [Online]. Available: https://aprendiendoarduino.wordpress.com/2018/11/21/*Gateways*-iot/. [Accessed: 10-Jun-2020].

[27] "Node-RED." [Online]. Available: https://nodered.org/. [Accessed: 10-Jun-2020].

[28] "SIMATIC IOT2000 | SIMATIC IOT *Gateways* | Siemens Spain." [Online]. Available: https://new.siemens.com/es/es/productos/automatizacion/sistemas/simatic/pcs-industriales/iot-*Gateways*/iot-2000.html. [Accessed: 11-Jun-2020].

[29] "Compute Module 3+ – Raspberry Pi." [Online]. Available: https://www.raspberrypi.org/products/compute-module-3-plus/. [Accessed: 12-

Jun-2020].

[30] P. García Rodríguez, “Estudio de las etapas de automatización de un proceso industrial y sus implicaciones en la gestión de la producción.,” 2020.

[31] “Postman | The Collaboration Platform for API Development.” [Online]. Available: https://www.postman.com/. [Accessed: 29-Jun-2020].

[32] “Clúster de computadoras.” [Online]. Available: https://es.wikipedia.org/wiki/Clúster_de_computadoras. [Accessed: 29-Jun-2020].

[33] “Biblioteca (informática).” [Online]. Available: https://es.wikipedia.org/wiki/Biblioteca_(informática). [Accessed: 29-Jun-2020].

[34] “¿Qué es el open source? Red Hat Foundation.” [Online]. Available: https://www.redhat.com/es/topics/open-source/what-is-open-source. [Accessed: 29-Jun-2020].