UNIVERSITAT JAUME I

ESCOLA SUPERIOR DE TECNOLOGIA I CIÈNCIES EXPERIMENTALS

MÀSTER UNIVERSITARI EN ENGINYERIA INDUSTRIAL

VALIDACIÓN DE UNA METODOLOGÍA PARA EL DISEÑO DE APLIACIONES EN EL ESTÁNDAR DE PROGRAMACIÓN IEC 61499 A PARTIR DE MODELOS EN DIAGRAMA DE GRAFCET.

TRABAJO FIN DE MÁSTER

AUTOR/A
Andrés Tendero Vegas
DIRECTOR/A
Julio Ariel Romero Pérez

Castellón, Septiembre de 2022

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

MEMORIA

1. INTRODUCCIÓN Y ANTECEDENTES

1.1. IEC 61131 vs IEC 61499 EN LA ACTUALIDAD

En la actualidad la norma predominante para la programación de PLC's (programable logic controller) es la denominada IEC 61131, la cual se compone de ocho apartados que abarcan conceptos desde información general de los PLC, hasta los lenguajes de programación que se implementan. Todos los proveedores de autómatas basan las programaciones de sus dispositivos en esta norma, aunque se están desarrollando nuevas metodologías para su programación que no están incluidas, como puede ser la programación basada en C/C++ o la norma IEC 61499. [4]

Respecto a la norma IEC 61499, la cual desarrollaremos en este estudio, es una arquitectura que rompe con el estándar establecido por la anterior norma, ya que la IEC 61131 está concebida para la programación de sistemas centralizados, mientas que en la 61499 su objetivo es facilitar el desarrollo de aplicaciones de control con lógica descentralizada o modelos distribuidos.

Este cambio de mentalidad viene precedido por el cambio de paradigma que se está desarrollando en la industria, la cual avanza hacia sistemas distribuidos. Los sistemas con un controlador centralizado están siendo transformados en sistemas distribuidos. En estos sistemas, las pequeñas partes de control tienen su inteligencia propia y pueden comunicarse con las demás, con esto conseguimos que el sistema funcione como una sola entidad. El estándar IEC 61499 define una arquitectura y un modelo para poder desarrollar estos tipos de sistemas industriales, pero no una metodología de programación. Uno de los grandes problemas, para el desarrollo de este tipo de programaciones en el estándar IEC 61131, es que la arquitectura de red en sistemas distribuidos no está definida por la norma, por lo tanto, al realizar la definición del modelo de sistema en la definición de los dispositivos no está gestionado por la norma y tiene que ser desarrollado por el programador. En el caso de la IEC 61499 este modelo se sistema, sí que es definido por la norma, en el que incluye a los dispositivos del sistema y la arquitectura de red por la que se van a comunicar. Esto hace que sea más fácil la integración de distintos equipos que se adapten a la norma porque la arquitectura de red para la comunicación entre ellos viene propuesta por la norma.

Otra de las diferencias respecto al IEC 61131 viene dado por los mecanismos que incorpora el nuevo estándar para poder modificar las aplicaciones sin necesidad de detener la ejecución en el dispositivo. Esto nos permite una gran flexibilidad para la reconfiguración de la aplicación en cuestión, ya que podemos realizar un cambio de modelo sin necesidad de parar la máquina y maximizar su rendimiento. También nos permite modificar la programación de una de las partes diferenciadas del sistema y solo lanzar la modificación al runtime del sistema que hemos modificado.

A continuación, podemos ver algunas de las ventajas que nos ofrece el estándar IEC 61499 [3],[5],[6]:

- Programación orientada a objetos: La estructura de bloques de funciones basada en eventos coincide con la noción de las IT de objetos, métodos y parámetros.
- La capacidad de anidar bloques de funciones para la creación de objetos más complejos permite crear un "caja negra" que el programador puede encapsular y proteger su propiedad intelectual. Este concepto de "caja negra" es un elemento clave para un modelo de negocio basado en la venta de aplicaciones para la automatización.
- El diseño gráfico, en programas compactos dónde no hay muchos bloques de funciones nos ayuda a la compresión e interpretación de este, ya que es más visual que tener que leer líneas de texto como por ejemplo lo tendríamos en un lenguaje con texto estructurado (ST). Aunque en programas complejos dónde la cantidad de bloques y líneas es considerable se convierte en una tarea considerable para llegar a su total comprensión. También podemos destacar las similitudes que tenemos con algunos lenguajes de la norma IEC 61131-3 como pueden ser el Function Block Diagram (FBD) o Sequential Function Chart (SFC), esto supone una mayor facilidad para la inclusión de los programadores al nuevo estándar IEC 61499.
- La flexibilidad arquitectónica es la que nos permite abordar tanto aplicaciones distribuidas como centralizadas, ya que la programación puede ser independiente del hardware. La programación puede ser totalmente centralizada, luego a la hora de la implantación del software el mapeo del programa se distribuirá en función del hardware instalado en cada equipo en función de las variables con las que interactúa físicamente. De este modo, el programador tiene en sus manos el control completo de la aplicación, se puede ejecutar en un controlador potente de forma centralizada o distribuirla en distintos subequipos que tendrán una total interconexión entre ellos sin la necesidad de desarrollar un protocolo de comunicación para ello.

- Por último, la norma se basa en eventos, esto nos permite un control más flexible del sistema, ya que con los eventos podemos tener bajo control el flujo de activación del programa. Esto nos permite tener la seguridad de que no pasaremos al siguiente bloque de funciones hasta que nos llegue el evento de confirmación que en el proceso anterior se ha ejecutado. En la norma IEC 61131, tenemos un ciclo de scan que periódicamente está comprobando todas las condiciones de activación. Esto permite, por ejemplo, que un bloque de función que controle un motor pueda iniciar una orden de mantenimiento en el momento adecuado por conteo de eventos, ya que los eventos y las veces que se ha activado coinciden y se puede desarrollar un mantenimiento predictivo sin necesidad de un software o programa de PLC externo al funcionamiento del sistema para el contador de activaciones del motor.

Por otro lado, la industria está avanzando hacia un marco dónde lo más importante por detrás del funcionamiento de las máquinas/proceso, es la obtención de datos a tiempo real para poder evaluarlos y obtener estadísticas y predicciones para saber cómo está funcionando el sistema y poder corregir las desviaciones lo antes posible para tener un proceso más eficiente. Esto es lo que denominamos Industria 4.0, que se compone de distintos apartados como pueden ser el internet de las cosas (IoT), análisis de datos (Big Data) o la inteligencia artificial (IA) entre otros. Con el estándar IEC 61499, se abre una ventana de oportunidades para poder desarrollar con mayor facilidad todos estos aspectos que necesita la industria 4.0. La reconfiguración de los sistemas, el tratamiento por eventos, que hemos comentado anteriormente y la facilidad de intercambiar información entre los dispositivos, hace que la creación de un dispositivo el cual se comporte de cerebro para el tratamiento de datos y desarrollo de los algoritmos predictivos sea de fácil implantación. [12]

1.2. IEC 61499: CARACTERÍSTICAS Y ARQUITECTURA

A continuación, daremos una breve visión teórica a la norma IEC 61499, cuáles son las características de la arquitectura propuesta y los modelos principales que la componen [2],[5],[10],[11].

La arquitectura del IEC 61499 se basa en tres aspectos claves los cuales son:

- Portabilidad: Las distintas herramientas para la creación del software son capaces de aceptar e interpretar las configuraciones creada por diferentes herramientas de desarrollo.
- Configurabilidad: Cualquier dispositivo, puede ser configurado por las distintas herramientas de desarrollo de diferentes proveedores de dispositivos.
- Interoperabilidad: Los dispositivos pueden trabajar de forma conjunta para realizar las funciones que se pueden desarrollar en aplicaciones distribuidas.

En cuanto a los modelos de la arquitectura, los cuales constituyen el esqueleto de la arquitectura, encontramos:

- Modelo de sistema: El sistema es el elemento que engloba todos los dispositivos y aplicaciones que se ejecutan y las relaciones existentes entre estos dos grupos (*Figura 1*). Las aplicaciones son unitarias, pero pueden estar ejecutándose en varios dispositivos del sistema.

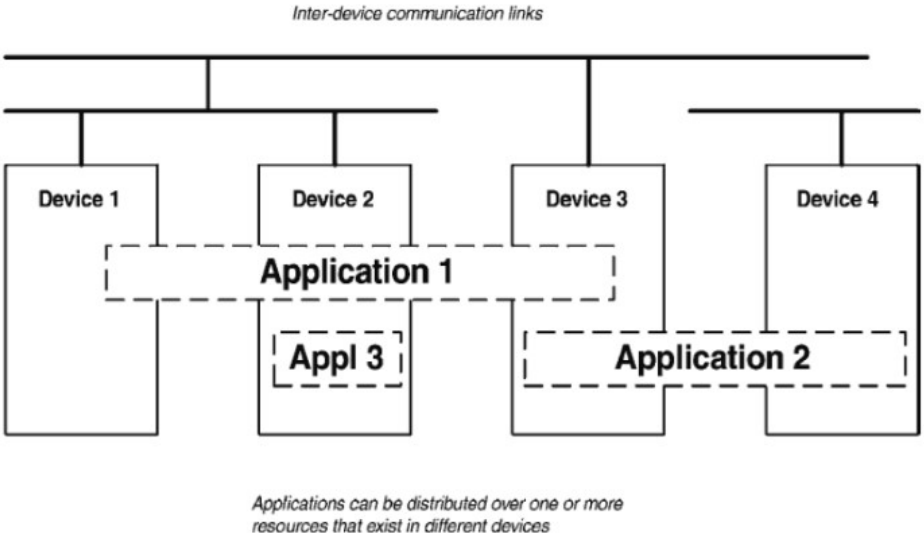

Figura 1: Modelo de sistema [2]

- Modelo de dispositivo: Representa el dispositivo (hardware) que va a desarrollar nuestra aplicación o red de bloques funcionales. Se puede asemejar a un sistema embebido o PC. Este dispositivo es capaz de comunicarse con otros dispositivos o con periféricos conectados a él a través de interfaces de comunicación. También es capaz de comunicar con la red de bloques funcionales mediante una interfaz interna de proceso, la cual nos permite enviar información entre estos bloques (*Figura 2*).

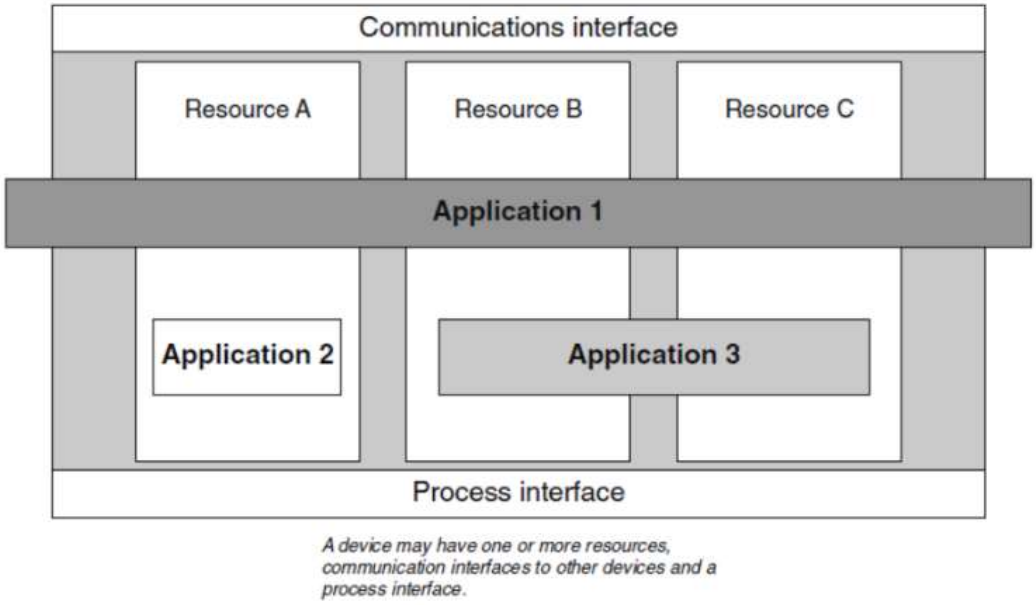

Figura 2: Modelo de dispositivo [2]

- Modelo de recurso: Se puede definir como un conjunto de bloques de funciones del modelo de dispositivo (*Figura 3*). Cada recurso tiene la particularidad de ser independiente al resto, con esto conseguimos que cada recurso pueda ser arrancado, parado o redefinido sin afectar al resto.

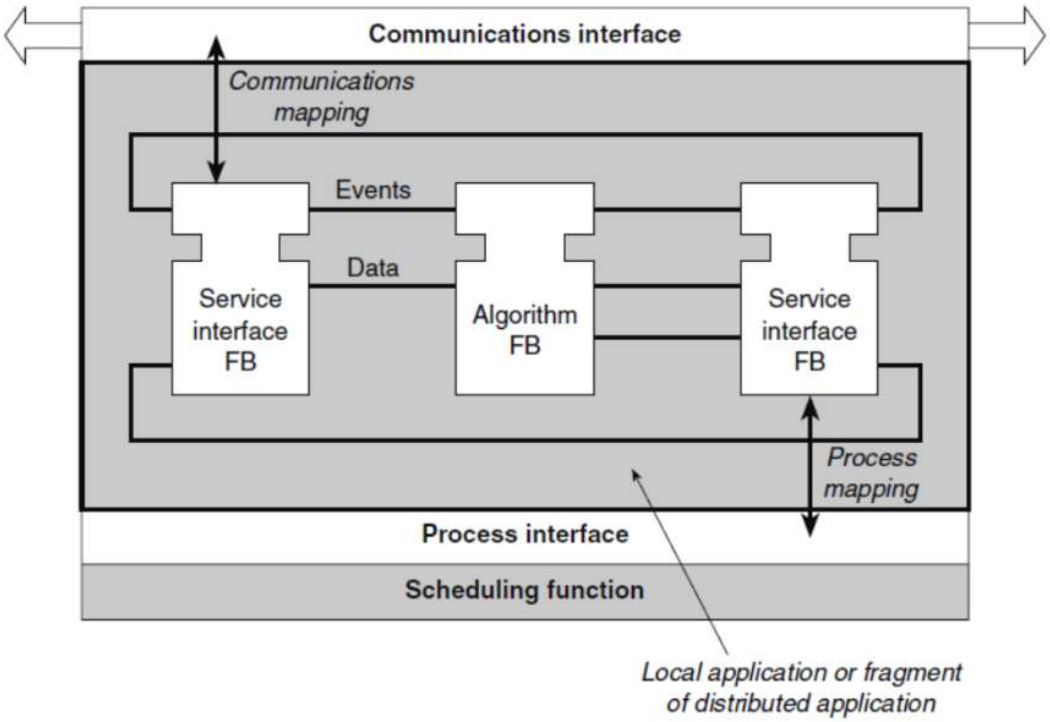

Figura 3: Modelo de recurso [2]

- Modelo de aplicación: Una aplicación es un conjunto de bloques de funciones que se conectan entre sí y se envían entre ellos señales y datos (*Figura 4*). Una aplicación se puede componer de subaplicaciones, estas, compuestas por bloques funcionales, tienen la particularidad de que son reusables, por lo tanto, la diferencia entre aplicación y subaplicación es que la aplicación es única y no se puede instanciar, mientras que las subaplicaciones al igual que los bloques funcionales son instanciables.

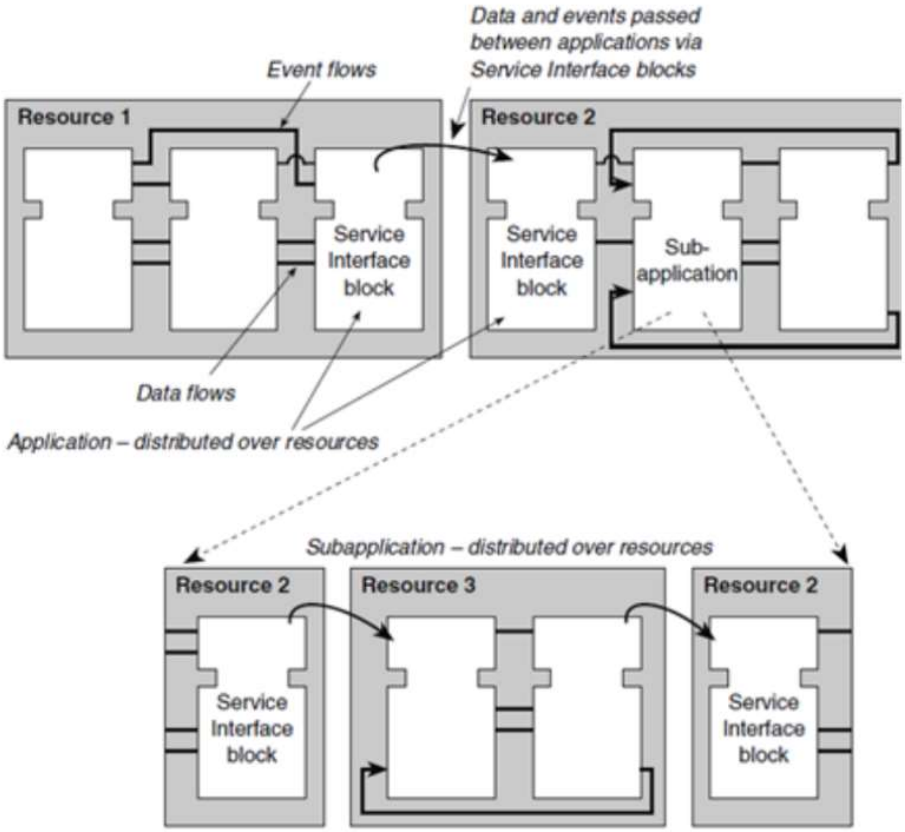

Figura 4: Modelo de aplicación [2]

- Modelo de bloque funcional: Un bloque funcional es un componente computacional en el cual, a través de sus algoritmos predefinidos obtenemos en una variable el dato deseado. En los bloques funcionales, encontramos su interfaz, en la que se representan los datos de entrada y salida, y los eventos que son los que deciden cuando se ejecuta cada bloque funcional. Esto quiere decir que, en nuestro bloque funcional, aunque los datos estén cambiando, si no ejecutamos un evento que nos active el bloque, no conseguiremos que el algoritmo nos cambie el valor del dato de salida. Actualmente se definen tres tipos de bloques funcionales en la norma.
    - Bloque funcional básico (BFB)

      Un bloque de funciones básico se compone de los componentes mencionados anteriormente (datos, eventos y algoritmos), como particularidad de estos bloques tenemos el algoritmo de ejecución que se denomina ECC (execution control chart) (*Figura 5*), el cual es el que interpreta como se debe de comportar el bloque ante una

entrada de evento, y el algoritmo propio del bloque el cual se utiliza para obtener los datos se salida, el cual se programa en diferentes lenguajes de programación como ST, java o C para lograr la portabilidad de modelos entre herramientas. En la *Figura 6* podemos observar la interfaz del bloque.

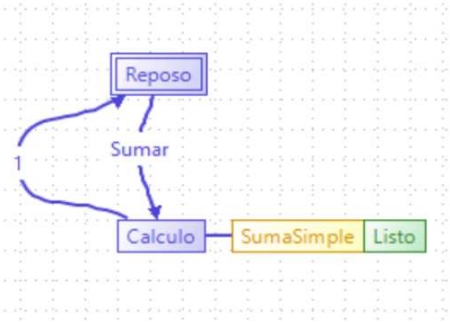

Figura 5: ECC bloque de función [2]

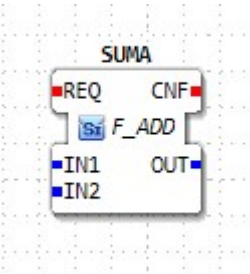

Figura 6: Interfaz bloque de función básico

- Bloque funcional compuesto (CFB)

  Los bloques funcionales compuestos, se componen de BFB, por lo tanto, no tiene ni un ECC propio ni algoritmos propios. La adición de los BFB es la que nos determina como las variables de entrada se transforman a las variables de salida (*Figura 7 y 8)*.

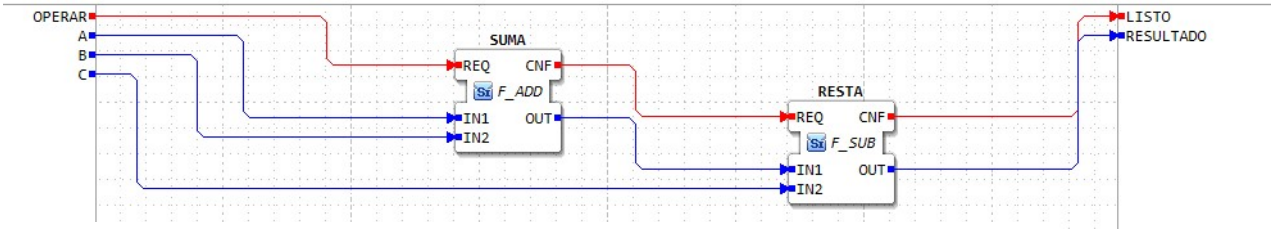

Figura 7: Ejemplo bloque de función compuesto dentro del bloque

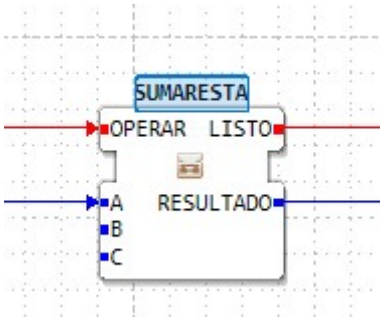

Figura 8: Ejemplo de la representación del bloque de función compuesto en modo compacto

- Bloque funcional de interfaz de servicio (SIFB)
  Por último, encontramos los bloques de interfaz de servicio que son los que nos permiten comunicarnos tanto con los otros dispositivos, como con el mundo exterior como podría ser una base de datos o una nube (Cloud). Estos tienen su propia estructura, distinta a los CFB.

Finalmente podemos encontrar varias herramientas de desarrollo para modelos 61499. Tenemos tanto herramientas de software libre como comerciales. El primer entorno de programación, su función era permitir la visualización de los FB, con el paso del tiempo ha evolucionado hasta poder permitir el testeo de estos bloques. La herramienta se denomina FBDK y actualmente está en desuso, por el hecho de que su interfaz es simple, poco robusto y nada amigable para el usuario.
Por otro lado, tenemos el software 4DIAC, desarrollado en el entorno de programación de Eclipse. La aplicación es muy robusta e intuitiva, permite tanto la representación de los FB como el resto de los modelos descritos anteriormente. A esto, le añadimos una comunidad en la cual podemos encontrar foros y discusiones entre los integrantes, que facilitan la comprensión del software. Por último, destacamos que actualmente se distribuyen dos actualizaciones del software por año.

También podemos destacar que grandes empresas en el sector de la automatización están apostando por el uso de la norma para el futuro, como es el caso de Schneider Electric, el cual se encuentra en pleno desarrollo para incluir un paquete en su software de automatización IoT EcoStruxture Automation Expert.

1.3. MODELOS CENTRALIZADOS Y DISTRIBUIDOS

Previo al desarrollo de los ejemplos desarrollados, vamos a explicar de forma breve las diferencias entre los modelos centralizados y distribuidos.

Se denomina sistema de control centralizado (Figura 9) a aquel que solo tiene un equipo de control. A partir de este, se controlan todos los procesos que tiene lugar en el sistema. Por ello, todos los dispositivos están conectados a este equipo y es el encargado de gestionar la información que se recibe y se envían a los estos dispositivos

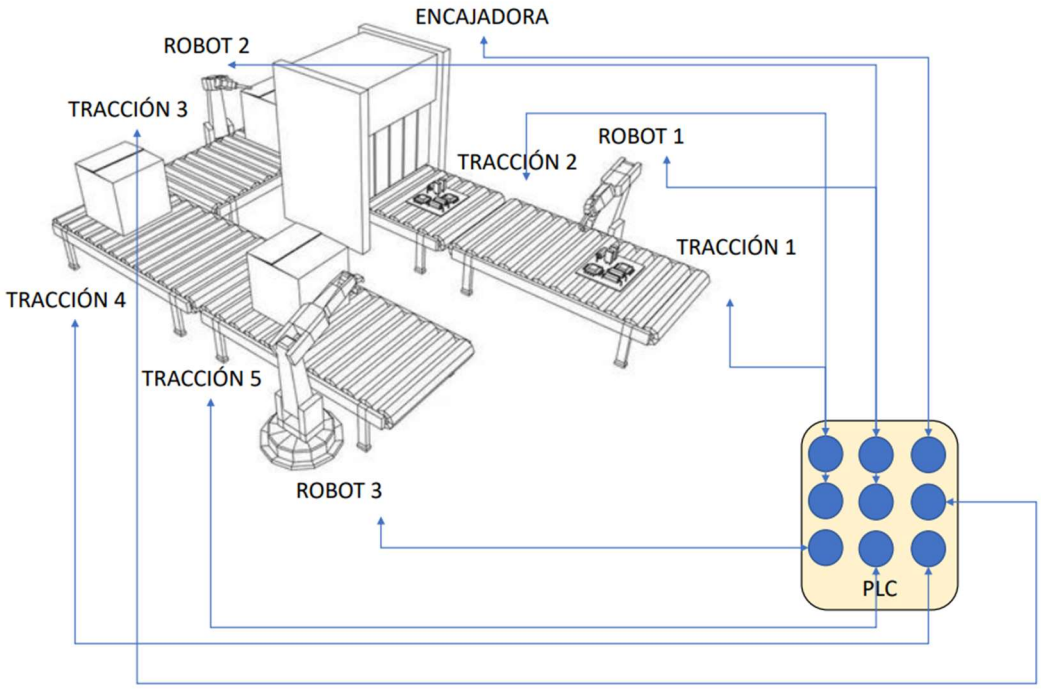

Figura 9: Esquema de un sistema de control centralizado

Por otro lado, tenemos los sistemas de control distribuidos (*Figura 10*). En estos sistemas tenemos varios equipos interconectados entre ellos, cada uno de estos equipos, controla una parte del sistema y están continuamente intercambiando información para que todos ellos puedan saber cómo se está comportando el sistema y actuar en consecuencia.

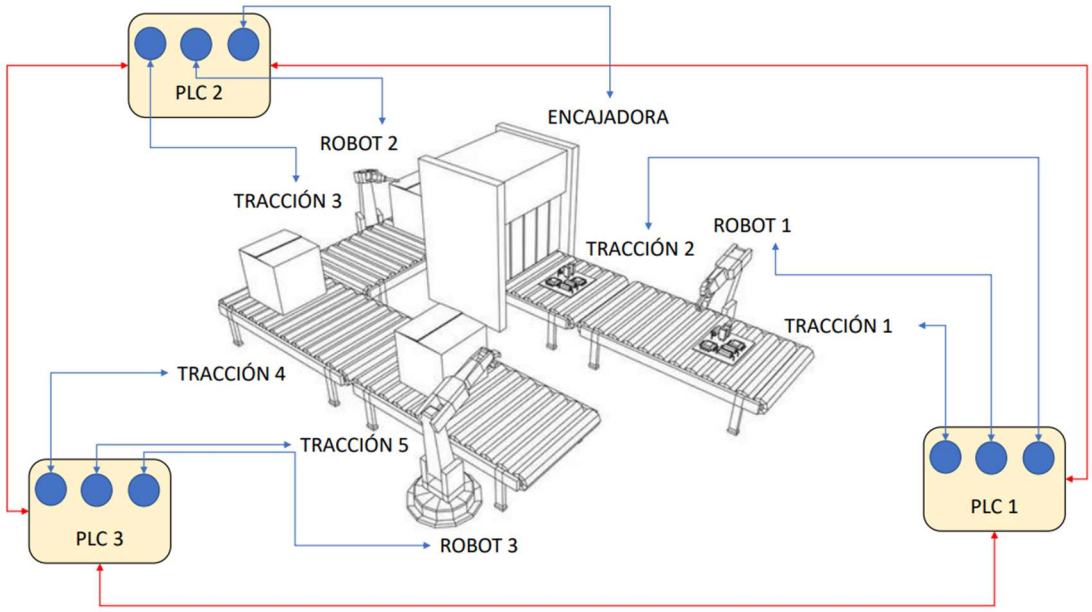

Figura 10: Esquema de un sistema de control distribuido

2. OBJETIVOS DEL PROYECTO

El objetivo principal de este proyecto es la validación de una metodología para el desarrollo de aplicaciones en la norma IEC 61499 para el control de sistemas de eventos discretos a partir de diagramas de Grafcet.

Dado que el estándar es relativamente reciente, existen algunas metodologías, cuya validación se ha limitado a ejemplos muy simples que nada tienen que ver con la complejidad de los sistemas industriales reales. Por ello, este proyecto se va a centrar en el desarrollo tanto de ejemplos sencillos para facilitar la comprensión de la metodología, como algún ejemplo similar a un proceso industrial que nos podemos encontrar en la actualidad.

La hipótesis inicial es poder plasmar en la norma los diagramas de Grafcet que se utilizan para concebir esquemáticamente los procesos industriales automatizados, tanto la parte conceptual como la gráfica (en la medida de lo posible). Por lo tanto, nos centraremos en desarrollar una metodología de trabajo para que la transformación de estos diagramas de Grafcet sean lo más similares a la programación desarrollada con la norma IEC 61499.

Para conseguir este fin, tenemos varios objetivos técnicos específicos que tenemos que abordar, éstos son los siguientes:

OT 1. Adecuación y programación de los diagramas de Grafcet de aplicaciones centralizadas en la norma IEC 61131 a la norma IEC 61499.
OT 2. Utilización de protocolos de comunicación aptos para la industria 4.0 y simulación.
OT 3. Programación de aplicaciones en modo distribuidas adecuadas a la norma IEC 61499.
OT 4. Estudio de la capacidad para adecuar la forma de programar de la norma IEC 61131 a la IEC 61499.

3. PLAN DE TRABAJO

El plan de trabajo del proyecto empieza a partir de la obtención del software 4Diac y de la versión del runtime Forte para la ejecución de los programas, lo cual ha tenido una duración estimada de unas ocho semanas previas al comienzo del cronograma. Esto viene dado porque a pesar de que el runtime es de software libre, la complejidad que tiene a nivel de conceptos computacionales es alta y quedaba fuera del alcance del proyecto, por lo tanto, desde una entidad externa se desarrolló este paso para poder llegar a realizar las programaciones.

En la *Figura 11* podemos ver el plan de trabajo en que se ha desarrollado el proyecto, indicando que el recurso humano principal ha sido Andrés Tendero Vegas, junto a un recurso secundario de apoyo que es Julio Ariel Romero Pérez.

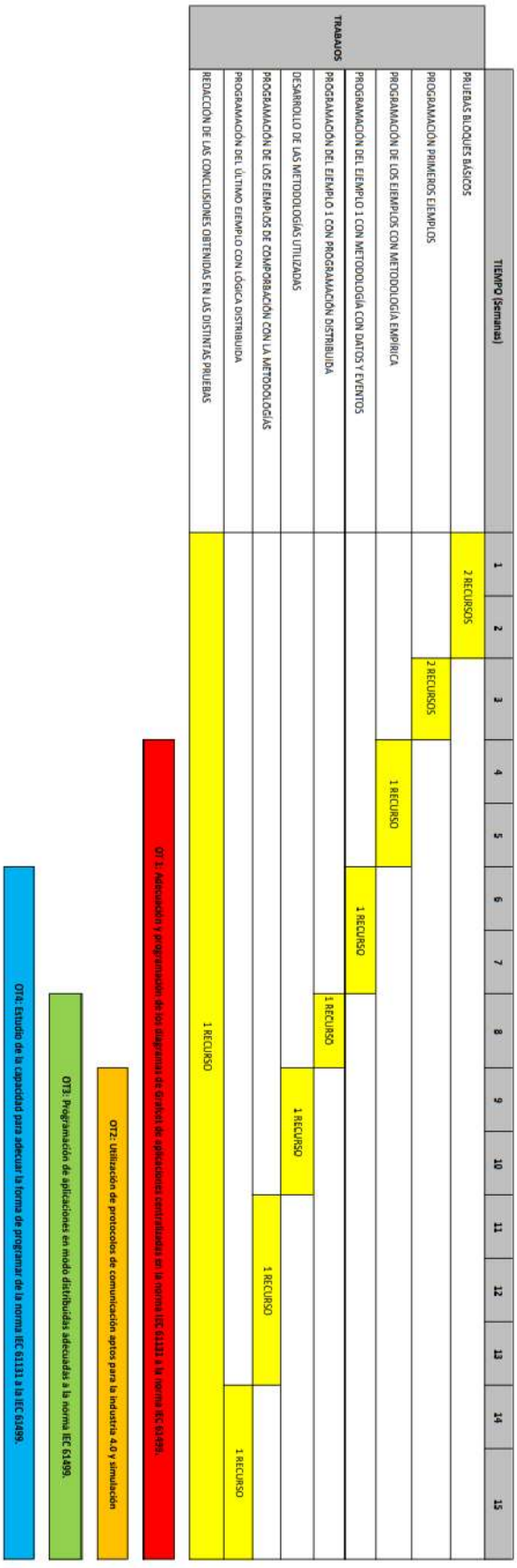

Figura 11: Cronograma y planificación de recursos e hitos

4. MATERIALES Y MÉTODOS

4.1. SOFTWARES UTILIZADOS PARA EL DESARROLLO

En este apartado hablaremos del software utilizado y sus particularidades. Tenemos que diferenciar dos softwares:

El primero sería el entorno de programación y runtime del sistema de control. En este caso hemos utilizado 4DIAC, su versión 2.0.1 con fecha de actualización de diciembre del 2021. En esta versión, nos hemos encontrado con algunos bloques los cuales no estaban en el paquete principal de librerías, por lo tanto, hemos tenido que añadirlos. El FB que ha sido necesario para desarrollar la programación ha sido el CLIENT_1_0, el cual se utiliza en la comunicación OPC-UA para la escritura de variables en el servidor. Como bien hemos comentado, hemos utilizado el paquete de FB que corresponde con el protocolo de comunicación OPC-UA, esto nos ha llevado a utilizar el runtime forte adaptado para OPC-UA.

Por otro lado, el segundo software utilizado, es el destinado para la simulación de las aplicaciones. Este software es utilizado para realizar sistemas virtuales con objetos similares a los reales para poder realizar una simulación de los programas desarrollados. El objetivo final del software es realizar gemelos digitales para tener una copia del sistema real. Con esto conseguimos poder desarrollar sistemas industriales más complejos, con un coste mínimo al desarrollarse en sistemas virtuales, para la validación de nuestras metodologías y no solo basar el desarrollo en conceptos teóricos o ejemplos simples. Este software se denomina Factory io, y se ha utilizado en su versión 2.5.1. Hay que tener en cuenta que en este software también tenemos que configurar que el protocolo de comunicación utilizado es OPC-UA, ya que tiene infinidad de protocolos de comunicación utilizados en la industria como pueden ser MODBUS o incluso propios de proveedores de PLC como Siemens o Omron.

Por último, se ha utilizado también el software UAexpert en su versión 1.2, el cual es un cliente OPC-UA. Con este software se ha conseguido comprobar el funcionamiento de los servidores creados para cada aplicación, la corrección de errores en la creación de variables y la interacción de bloques de funciones básicos entre el servidor y el entorno de programación. No se cuenta como un software principal para el desarrollo de las aplicaciones porque como hemos comentado, sólo ha sido utilizado para realizar pruebas y comprobaciones mínimas, no influye en la simulación ni ejecución del programa de control.

4.2. CONFIGURACIONES EN LOS SOFTWARE DE DESARROLLO

A continuación, se mostrará las configuraciones necesarias en cada uno de los programas utilizados para poder replicar los ejemplos que se describirán posteriormente.

Primero mostraremos la configuración necesaria en el software de simulación Factory io. En este caso, es tan sencillo como elegir el protocolo de comunicación que requerimos en nuestro sistema, el cual será en nuestro caso OPC UA (*Figura 12*). Con el protocolo de comunicación elegido, sólo nos quedara buscar el servidor dónde tenemos las variables del sistema. La dirección que utiliza por defecto 4Diac para la creación del servidor es *opc.tcp://localhost:4840* la cual puede ser configurable por el usuario dependiendo de los dispositivos que tenga en su aplicación. Por último, para tener la configuración completa del servidor en factory io, tenemos que establecer y buscar los nodos (variables) que queremos que se muestren. En este aspecto la aplicación nos ofrece un filtro, en el cual podemos filtrar las variables que están en el servidor. Como se observa en la *Figura 13*, en todas las variables hemos introducido el prefijo *fio* así tenemos las variables localizadas con mayor facilidad.

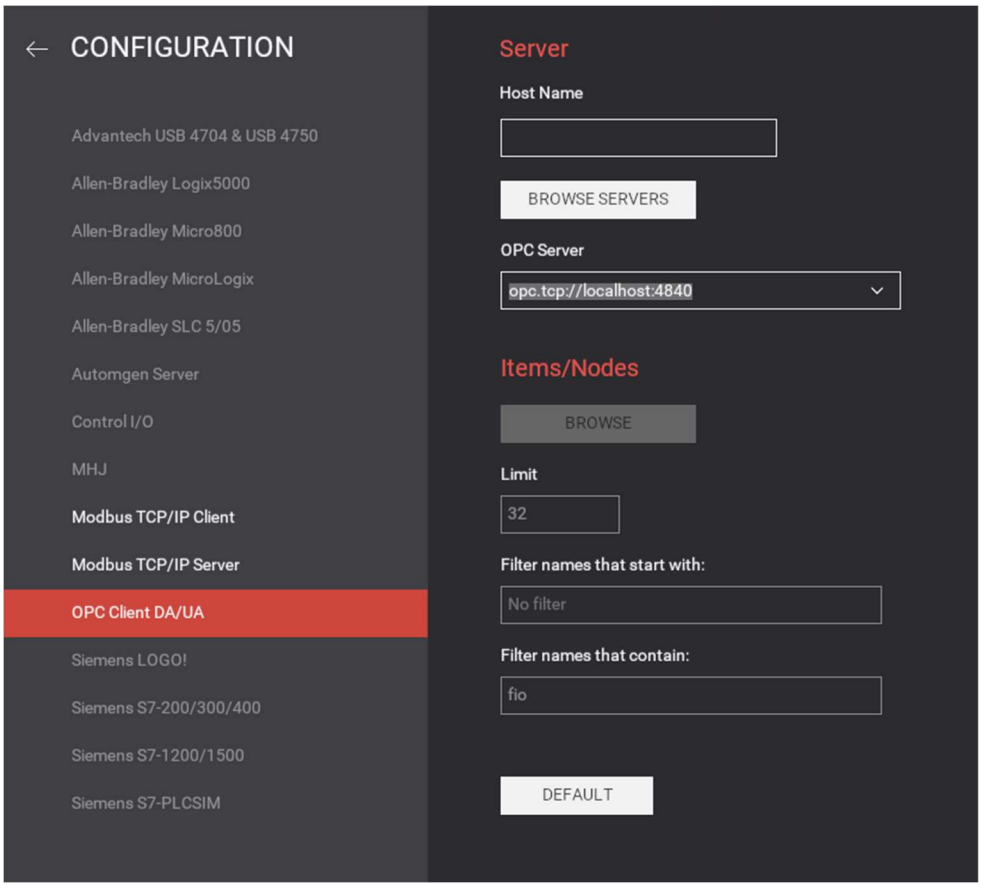

Figura 12: Configuración del servidor en Factory io mediante OPC UA

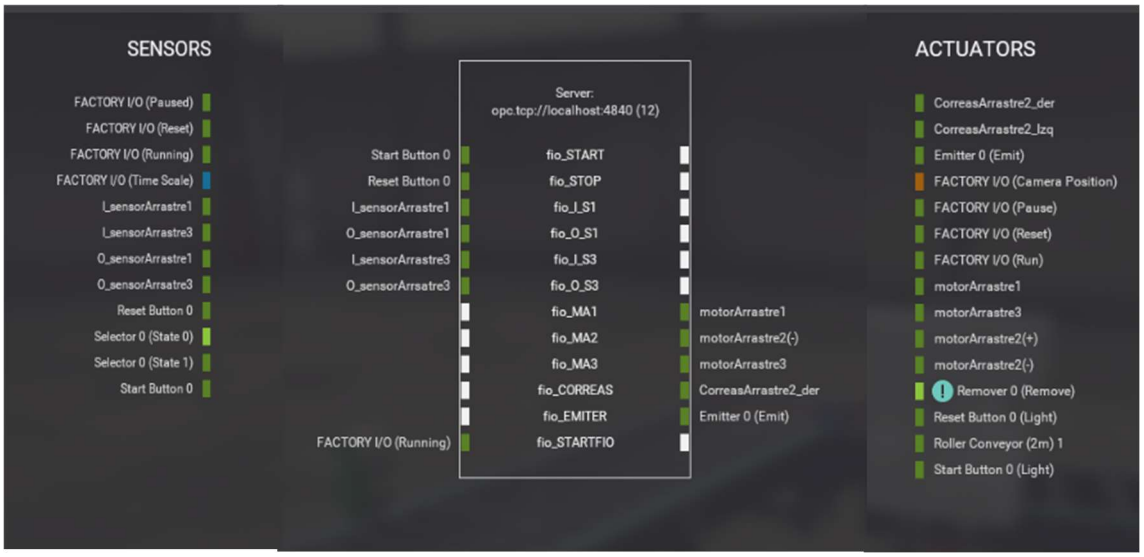

Figura 13: Relación de las variables del servidor con las locales del simulador

A continuación, veremos como configurar Uaexpert, la configuración es muy parecida a la explicada anteriormente para Factory io, ya que se comporta de la misma manera, conectándose como cliente al servidor. En la *Figura 14,* podemos observar cómo realizar la configuración del servidor.

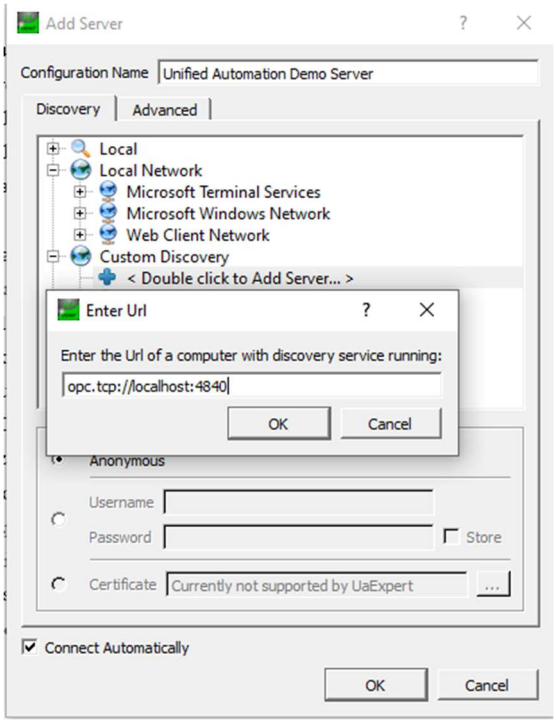

Figura 14: Configuración del servidor en Uaexpert

Por último, explicaremos lo que respecta a 4DIAC, la única configuración del sistema que tenemos que realizar, es a la hora de realizar la configuración de la distribución de los runtime, tenemos que marcar la opción de tipo de variables como DynamicTypeLoad, para que el runtime forte interprete correctamente los bloques de funciones correspondientes a la comunicación OPC-UA. En la *Figura 15,* mostramos como queda la configuración de una de las aplicaciones.

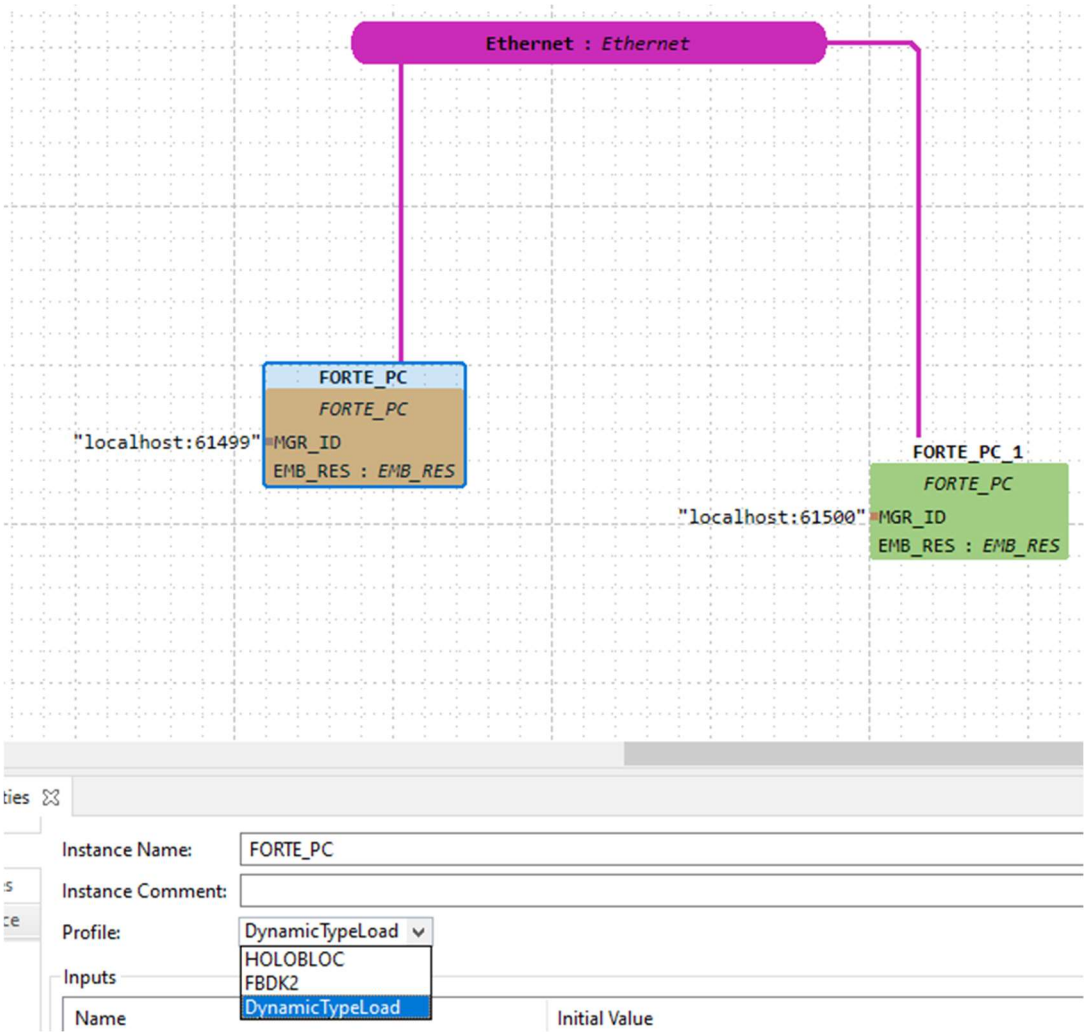

Figura 15: Configuración de la interfaz del sistema de 4DIAC

4.3. 4DIAC (FB) UTILIZADOS EN LA PROGRAMACIÓN

Teniendo claro las configuraciones a realizar en los distintos programas, ahora vamos a realizar una pequeña introducción a los bloques de funciones que se van a utilizar para la programación de las aplicaciones. No pretende ser una guía para la creación de bloques, sino de forma general como trabaja cada uno de los FB para la comprensión de los modelos desarrollados.

4.3.1. BLOQUES DE COMUNICACIÓN

4.3.1.1. SUBSCRIBE

Este bloque nos va a permitir realizar dos acciones en nuestras aplicaciones dependiendo de la cadena de caracteres que le especifiquemos en el parámetro ID.

La primera acción lo que nos va a permitir, es la creación de las variables en el servidor y también la lectura en aplicaciones centralizadas, dónde el servidor y la aplicación ese encuentran en el mismo runtime de ejecución. El parámetro se deberá de configurar de la siguiente forma.

"opc_ua[READ;
/Objects/fio_S1,1:s=fio_S1;
/Objects/fio_S2,1:s=fio_S2]"

El conjunto READ es el que nos indica el funcionamiento que tendrá el bloque, en este caso crea las variables que se denominan fio_S1 y fio_S2.

Por otro lado, cuando tengamos SUBSCRIBE en vez de READ el FB se comporta como un cliente OPC-UA el cual se comunica con el servidor y obtiene los valores de las variables que se le ha indicado. En el proyecto lo utilizaremos para realizar las lecturas de los sensores que se van refrescando en el servidor con la interacción de Factory_io. Aquí se muestra como quedaría la cadena de caracteres en cuestión.

"opc_ua[SUBSCRIBE; opc.tcp://localhost:4840#;
/Objects/fio_S1,1:s=fio_S1;
/Objects/fio_S2,1:s=fio_S2]"

En esta situación como estamos trabajando sobre comunicación OPC UA, tenemos que indicar dónde se encuentra el servidor, que son los caracteres correspondientes a continuación de ***SUBSCRIBE***.

Por otro lado, tenemos los FB que nos permitirán escribir en las variables del servidor. Tenemos dos FB que nos permiten realizarlo.

El primero sólo permite la comunicación siempre y cuando el servidor esté creador en la misma aplicación dónde se esté el FB. Por ahora, este FB no permite conectarse al servidor desde otra aplicación. Por ello, se utiliza en las aplicaciones centralizadas dónde todo se ejecuta en una misma aplicación, tanto servidor como programa.

A raíz de la necesidad de desarrollar aplicaciones distribuidas, hemos tenido que utilizar un segundo FB, el cual nos permite la comunicación con el servidor desde otras aplicaciones.

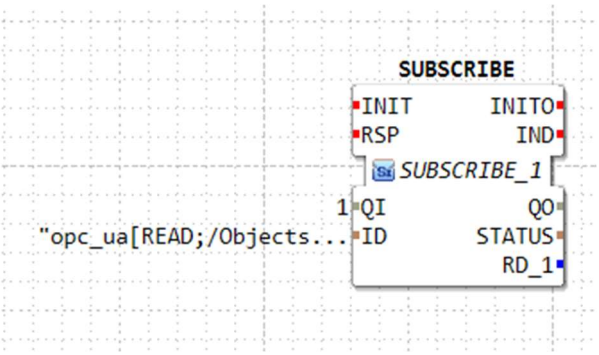

Figura 16: FB Subscribe

4.3.1.2. PUBLISH

La configuración que debemos realizar al bloque de funciones es similar al descrito para el FB SUBSCRIBE. A continuación, se muestra un ejemplo de la configuración del parámetro ID.

"opc_ua[WRITE;
/Objects/fio_A1,1:s=fio_A1;
/Objects/fio_A2,1:s=fio_A2]"

Al indicar ***WRITE*** en la cadena de caracteres, el FB interpreta que su función es la de escribir en las variables del servidor que en este caso de denominan fio_S1 y fio_S2.

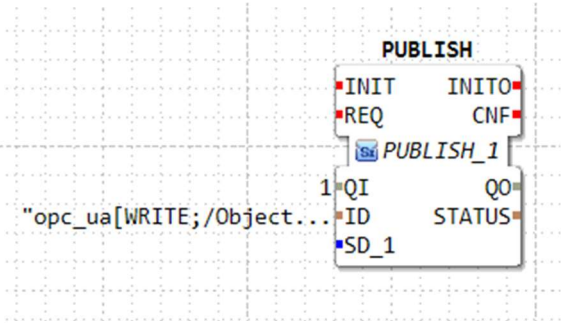

Figura 17: FB PUBLISH

4.3.1.3. CLIENT_1_0

Para el último bloque de funciones de comunicación tenemos la misma configuración que con el resto, la particularidad es que al igual que cuando configurábamos el FB SUBSCRIBE para la comunicación por OPC UA, en este bloque debemos realizar lo mismo, pero con la diferencia que en vez de ser ***SUBSCRIBE*** será ***WRITE***. Aquí podemos ver cómo queda la configuración del bloque.

"opc_ua[WRITE; opc.tcp://localhost:4840#;
/Objects/fio_A1,1:s=fio_A1;
/Objects/fio_A2,1:s=fio_A2]"

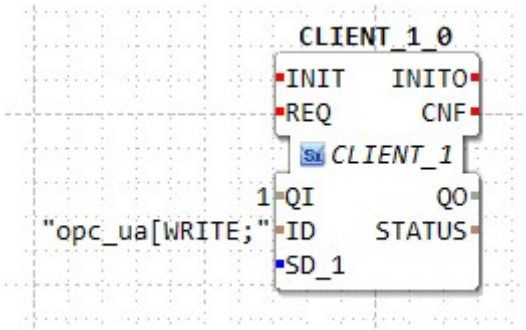

Figura 18: FB CLIENT_1_0

4.3.2. BLOQUES DE PROGRAMACIÓN CON DATOS

En este apartado, comentaremos los bloques de función que utilizaremos en las programaciones referentes a la metodología por datos. La denominación por datos se ha designado para facilitar la comprensión del bloque y así poder diferenciarlos de los bloques que su salida solo es con eventos, los comentaremos posteriormente. Por lo tanto, la particularidad que tienen estos bloques de función es que aparte de tener la interfaz de eventos, tienen la interfaz de datos, la cual es la que lleva la información del proceso hacia los siguientes bloques de funciones.

En este proyecto utilizaremos tres bloques de funciones. Se pueden comparar con algunos bloques o programaciones que se utilizan en el estándar IEC 61131. Ahora lo veremos.

4.3.2.1. FB_SR

Éste primer FB es el utilizado para la activación y desactivación de una variable de tipo BOOL. Realizando una analogía a la norma IEC 61131, funciona como una bobina de SET y RESET del lenguaje Ladder combinado en un mismo interfaz gráfico y funcional.

Con el parámetro S activamos la salida y hasta que no tengamos la condición S=0 y R=1 no desactivaremos la salida. Podemos decir que la prioridad del SET, es mayor que la del RESET.

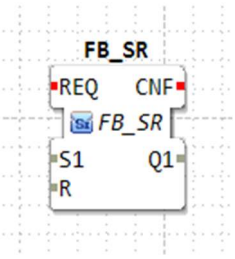

Figura 19: FB_SR

4.3.2.2. FB_R_TRIG / FB_F_TRIG

Los siguientes FB son los de flanco de subida y bajada, los cuales se comportan de forma similar a los descritos en cualquier lenguaje de programación. La particularidad que tienen, como todos los FB utilizados en esta norma es que su activación es a través de la interfaz de eventos.

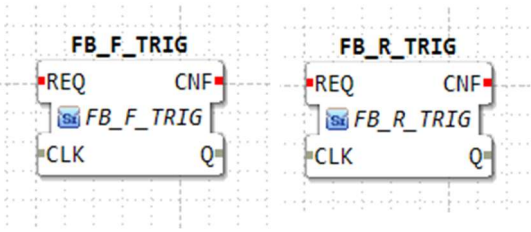

Figura 20: FB R_TRIG y FB F_TRIG

4.3.3. BLOQUES DE PROGRAMACIÓN CON EVENTOS

Otros bloques que tenemos son los que denominaremos bloques de programación con eventos, estos tienen como particularidad que su interacción se basa en la interfaz de eventos, en los bloques también tenemos datos de entrada o salida que utilizaremos para realizar las operaciones internas de los bloques, pero sus principales interacciones se basan en la interfaz de eventos.

4.3.3.1. E_SR

Este bloque se utiliza para generar una salida de datos de tipo BOOL a partir de eventos de entrada. En este caso, los eventos S y R son los que activan o desactivan el dato de salida Q.

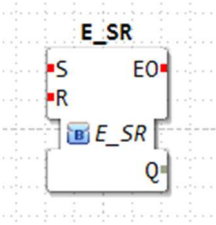

Figura 21: E_SR

4.3.3.2. E_R_TRIG / E_F_TRIG

También utilizaremos, los bloques de flanco de subida y bajada, la funcionalidad es la misma que los utilizados en otros entornos de programación. En este caso, tenemos una entrada de datos junto a su entrada de eventos para hacer su evaluación y la salida del bloque es mediante la interfaz de eventos. Cuando sea cierta la condición se activa el evento de salida.

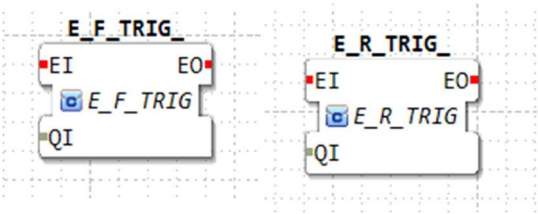

Figura 22: E F_TRIG y E R_TRIG

4.3.4. BLOQUES DE FUNCIONES ESTÁNDAR

Por último, mostraremos los bloques de función que se engloban dentro de la librería denominada como IEC61131-3, por lo tanto, el funcionamiento de los FB es idéntico a los utilizados en el estándar.

4.3.4.1. BOOL2BOOL

Con este bloque lo único que pretendemos es que las salidas de algunos de los bloques se puedan interpretar en los siguientes. En algunas ocasiones la salida del bloque a pesar de ser tipo BOOL, los siguientes bloques no lo interpretan, por lo tanto, al utilizar este bloque corregimos esos errores internos del software.

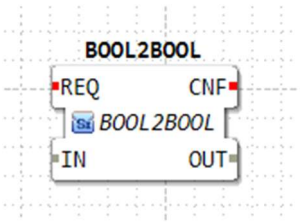

Figura 23: F BOOL2BOOL

4.3.4.2. INT2INT

Al igual que el bloque anterior, este bloque es para la interpretación correcta de las variables a la entrada de algunos bloques. En este caso convertimos un dato de tipo entero en otro de tipo entero.

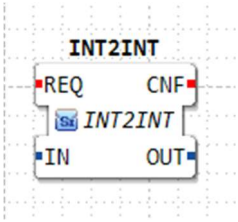

Figura 24: F INT2INT

4.3.4.3. F_AND / F_OR / F_NOT

Estos bloques tienen las características básicas de la lógica booleana con sus tablas de lógica.

AND			OR			NOT	
0-0	0		0-0	0		1	0
1-0	0		1-0	1			
0-1	0		0-1	1		0	1
1-1	1		1-1	0			

Tabla 1: Operaciones lógicas funciones AND, OR y NOT

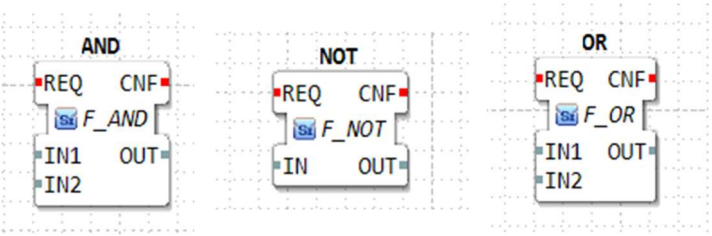

Figura 25: F AND, F OR y F NOT

4.3.4.4. CTUD / TON

También hemos utilizado los bloques de funciones correspondientes a un contador incremental/decremental y un temporizador básico del tipo TON. A continuación, podemos observar sus esquemas de ejecución.

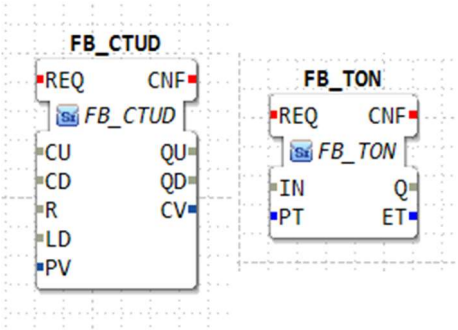

Figura 26: FB CTUD y FB TON

5. RESULTADOS Y DISCUSIÓN

5.1. METODOLOGÍAS DESARROLLADAS

Para poder llevar a cabo el proyecto, se han desarrollado dos metodologías, la primera, propiamente no lo es, ya que se basa en la experiencia del programador a la hora de interpretar los sistemas. Este método se ha utilizado para las fases previas del proyecto, para evaluar si el desarrollo de las aplicaciones iba a ser factible con las librerías desarrolladas actualmente en 4DIAC.

Aunque el método utilizado no se puede considerar como un estándar a seguir, sí que tiene una estructura lógica para poder entender como se ha desarrollado la programación. Dicha estructura de trabajo se compone de los siguientes pasos:

- Creación de los sensores y actuadores en el servidor.
- Creación de los FB_SR de los actuadores.
- Creación de las detecciones de los sensores.
- Creación de forma lógica y por experiencia las activaciones y desactivaciones de los actuadores. Se basa en prueba/error.

Por otro lado, tenemos la metodología aplicada para los ejemplos que denominamos datos o eventos. Esta metodología parte de un previo diagrama de Grafcet desarrollado con sus propias bases.

La estructura que presenta este método es la siguiente:

- Creación de los sensores y actuadores en el servidor o bloque de función pertinente.
- Creación de los bloques FB_SR/E_SR equivalentes a las etapas en los diagramas de Grafcet.
    - En este caso hay que matizar que en la programación gráfica para que el conjunto de bloques sea más entendible, si tenemos que para la activación/desactivación de una etapa puede ser a partir de varias transiciones, entonces lo que realizamos es la programación lógica a través de F_OR, F_AND o los bloques necesarios y luego creamos una subaplicación que será la etapa conjunta.
- Creación de las transiciones equivalentes al diagrama de Grafcet.

- Creación de las detecciones de los sensores (sensor, flanco de subida, flanco de bajada…).
    - Creación de lógica reflejada en el diagrama de Grafcet de las variables que componen la transición.
    - Todos estos elementos se recomienda unirlos en una subaplicación para favorecer la interpretabilidad del programa.
    - Con el software que estamos trabajando es cierto que, de una misma detección de sensor, se puede obtener dos flujos de datos para activar dos transiciones distintas, pero en esta metodología recomendamos para una mejor trazabilidad, duplicar los bloques de detección de sensores y crear dos transiciones distintas, aunque sean iguales en su forma.
- Por último, nos queda la unión de los conectores que llevan datos y los conectores que se encargan del flujo de eventos de control del sistema.
    - En el caso de los eventos hay que tener cuidado de no realizar bucles infinitos de eventos (explicado posteriormente) y no sobre cargar los bloques con los eventos.

Esta última metodología ha sido desarrolla a partir de lo ejemplos descritos posteriormente en el proyecto. Como peculiaridad tenemos la lógica de las transiciones, que explicaremos seguidamente.

En cuanto a los matices que debemos tener en cuenta en la programación de las transiciones del Grafcet al ser volcadas al software de programación encontramos:

- BUCLES INFINITOS DE EVENTOS

  Se puede dar el caso en que tengamos una transición del tipo (***E5 AND S1)*** y la desactivación de E5 sea la transición como se muestra en la *figura 27*, dónde E5 es una etapa del Grafcet y S1 una señal de un sensor. En este caso no podemos poner que la transición se ejecutará por el evento del sensor o de la etapa, ya que en nuestro caso al estar siempre comprobando en el servidor los sensores la etapa de S1 esta activa casi siempre y esto provoca que se ejecute la transición y ejecute también la etapa, que puede cambiar o no su estado de la variable dato, pero independientemente de eso el evento de acabar el FB sí que se ejecutará y volvería a activar la transición, con lo que se formaría el bucle infinito de eventos que colapsa la memoria del runtime.

Por lo tanto, en este caso la transición solo se ejecutaría con el evento del sensor, eliminaríamos el evento de la etapa.

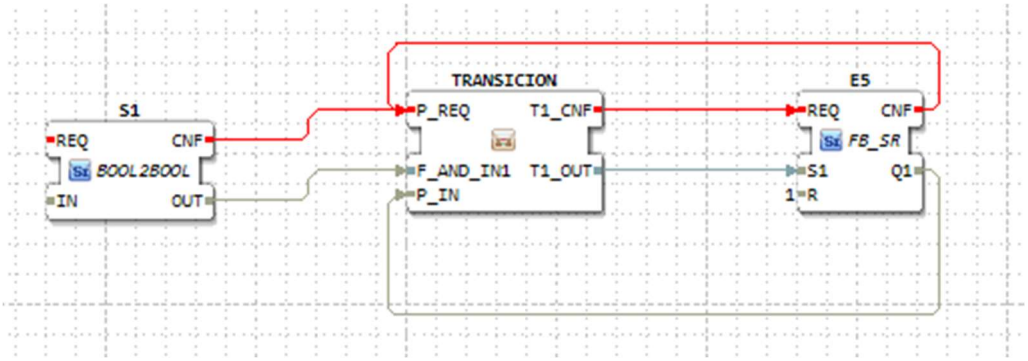

Figura 27: Ejemplo transición bucles infinitos de eventos

- ERROR TIPOS DE DATOS

  Tenemos errores en la interpretación del tipo de dato en la salida de los bloques de funciones de lógica en el software 4DIAC, como pueden ser los bloques F_AND, F_OR o F_NOT. En estos casos para que el programa interprete correctamente el tipo de dato tanto de entrada como de salida, se tiene que utilizar un conversor de tipo de dato del mismo dato al mismo dato. Por ejemplo, BOOL2BOOL. Suele ocurrir este error cuando se anidan varias funciones lógicas OR/AND/NOT. Para seguir un método estructurado, pondremos los bloques de conversión en todas las salidas y nos evitamos este tipo de error, el cual no es de programación. En la *Figura 28* podemos ver un ejemplo de la estructura que se nos queda añadiendo los bloques de conversión del tipo de dato.

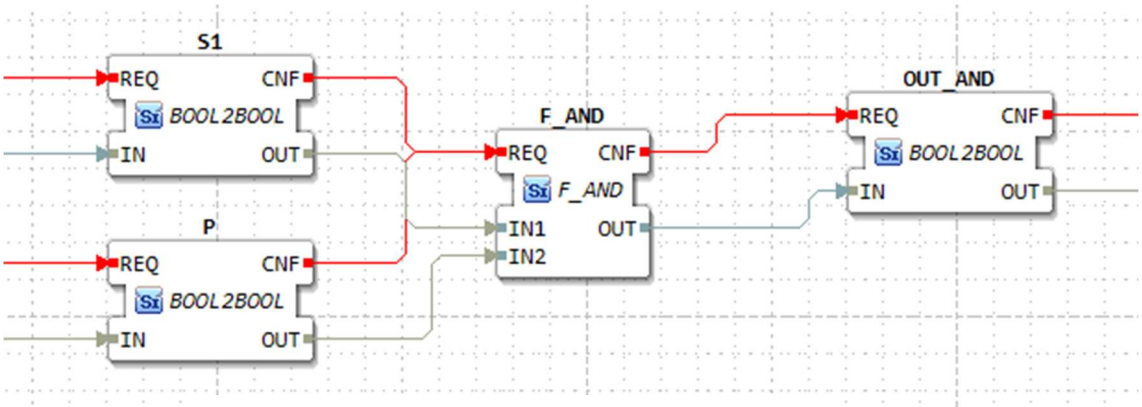

Figura 28: Ejemplo error tipos de datos

- SEÑALES DE ENTRADA A LOS BLOQUES TON Y CTUD

  Respecto a los bloques contadores y temporizadores, hay que tener en cuenta que en el caso del bloque TON (*Figura 30)*, el utilizado en el proyecto, hay que tener activada la entrada de datos hasta que se acabe de ejecutar el bloque. Por otro lado, la entrada de evento solo se tiene que activar una sola vez, ya que si se activa antes de acabar la secuencia interna del bloque se reinicia y volvería a empezar el temporizador.

  Por parte del CTUD (*Figura 29)*, no es tan critica la entrada de eventos, ya que la ejecución del bloque es instantánea, lo que hay que tener en cuenta es que las entradas de CU y CD tienen que venir de un bloque de flanco de subida (R_TRIG), ya que ellas solas no detectan si hay un flanco de subida en la variable. Su entrada es binaria y si tenemos el caso en que se ejecutan cinco eventos mientras CU está activo, el contador incrementará cinco unidades.

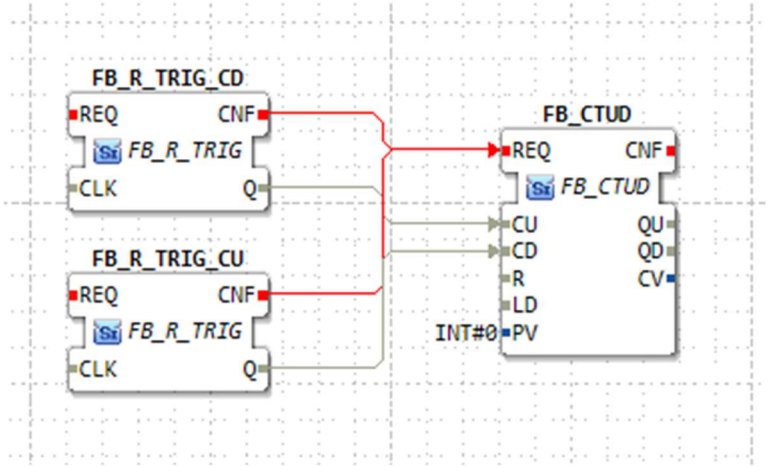

Figura 29: Ejemplo del error de detección de flanco de subido en FB_CTUD

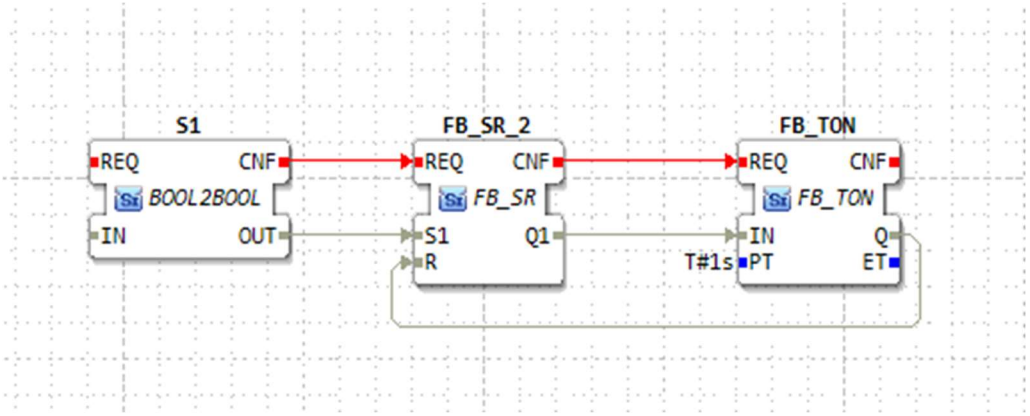

Figura 30:Ejemplo del error de detección de flanco de subido en FB_TON

- DETECCIÓN DE LA ETAPA ANTERIOR

  Aunque en ejemplos muy simples no es necesario que la transición vaya ligada a que la etapa anterior esté activa, cuando estos ejemplos se asemejan a sistemas reales en que los sensores actúan en varios diagramas de Grafcet e incluso la comunicación entre los equipos es sustancial, surge la necesidad de realizar esta detección de la etapa anterior a la transición, por el hecho que un sensor puede estar dando la señal que la transición tiene como consigna. En este caso si no anclamos a la detección de las etapas anteriores podría darse el caso de tener dos etapas del mismo Grafcet activas y dar un error en el sistema.

  En el ejemplo mostrado en la *Figura 31* se muestra un sistema en que, cuando el objeto llega al sensor S se mira si el sensor P no está activo para retirar la caja. En este tipo de sistema si no enclavamos la etapa 2 a la etapa anterior, el sensor P siempre está en FALSE, detectando una caja que se debe retirar, y por lo tanto, siempre estaría mandando la orden de retirar la caja. Tendríamos por ejemplo la E0 y E2 activas o la E0 y E1, cuando no es posible en este sistema el cual es secuencial.

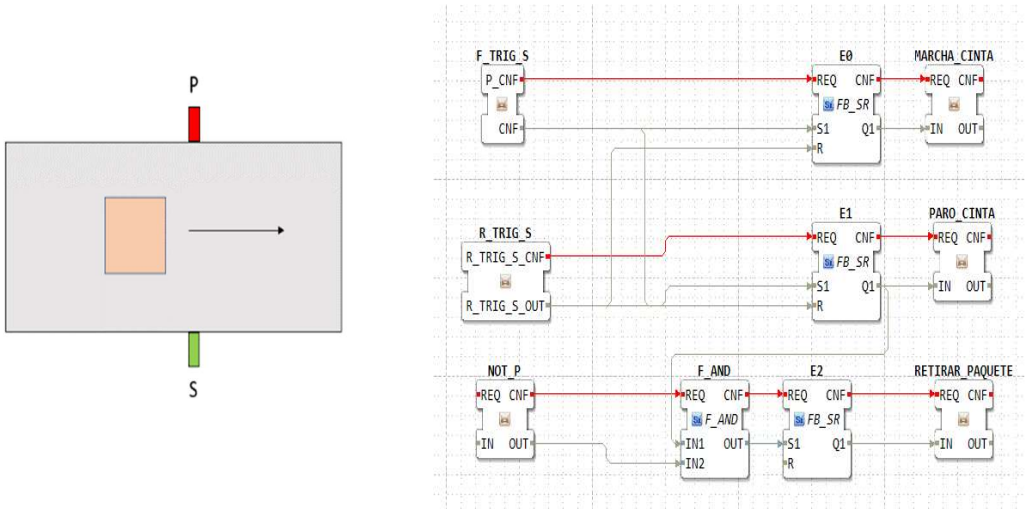

Figura 31: Ejemplo detección etapa anterior

- SOBRECARGA DE EVENTOS

  Un error que puede hacer que el runtime del programa se sature es que a los bloques de función les enviemos una cantidad de eventos que sea muy grande de procesar en un periodo de tiempo corto. Por ejemplo, si tenemos dos bloques que envían eventos a otro, el bloque receptor, por cada evento de uno de los bloques emisores tiene que ejecutarse dos veces, si no tenemos controlada el tiempo de ejecución del bloque, puede que se quede una vez sin ejecutar porque el evento segundo ha llegado primero que acabe el bloque de función y se crea una cola infinita de eventos que no se ejecutan al tiempo que son necesarios.

5.2. EJEMPLOS DESARROLLADOS

5.2.1. EJEMPLO 1 TWO CONVEYORS

El sistema de la *Figura 32* representa dos cintas transportadoras equipadas con un motor y dos sensores cada una (M1, IS1, OS1 y M3, IS3, OS3) y una cinta con correas la cual permite hacer un giro de 90º a los elementos que circulan por ella, equipada con dos motores (M2 y CORREAS). Sobre el sistema hay una caja cada vez. El funcionamiento

es como sigue. Cuando sale una caja y es detectada por el sensor IS1, se activa M1. Al pasar por el sensor OS1, mantenemos M1 y activamos M2 para subir la caja sobre la cinta de giro. Cuando esté en esta cinta, desactivamos M1 y M2 y activamos las CORREAS. Por último, al entrar en la última cinta desactivamos las CORREAS y activamos M3. El ciclo acaba cuando desaparece la caja y aparece una nueva. En la *Figura 33* se muestra un diagrama de Grafcet para el control del sistema.

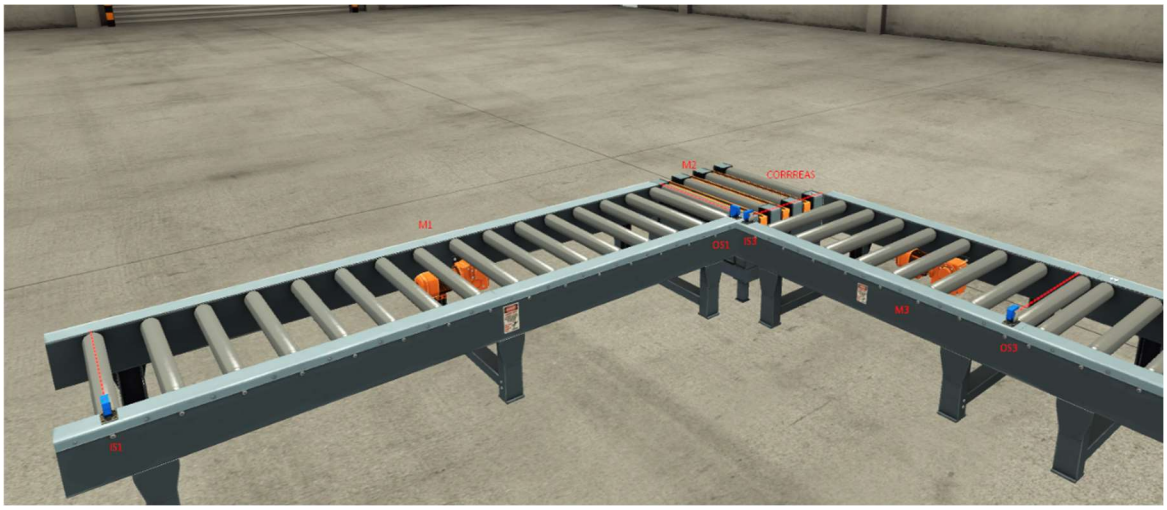

Figura 32: Sistema desarrollado en Factory io del ejemplo Two Conveyors

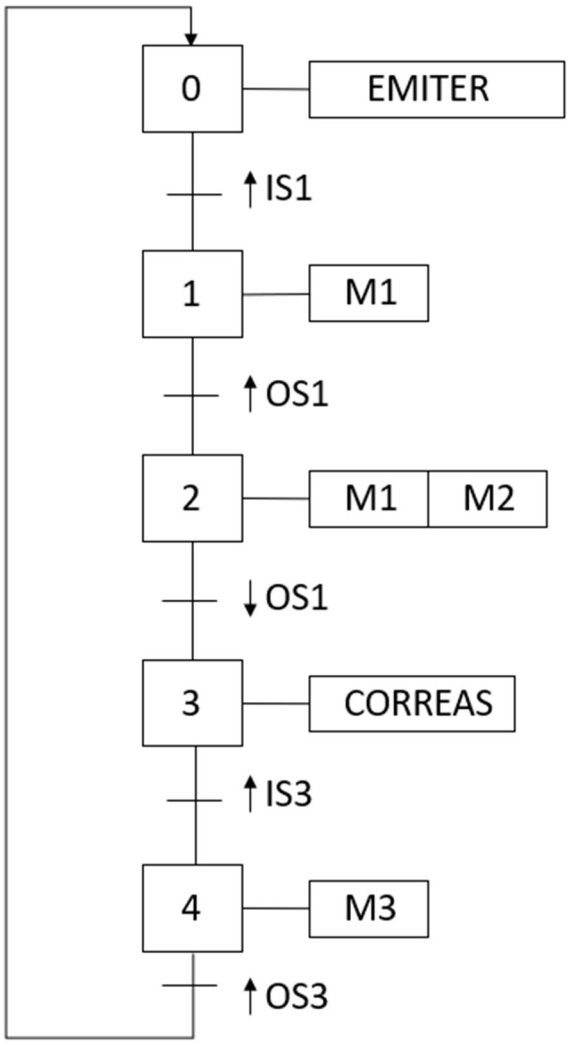

Figura 33: Grafcet ejemplo Two Conveyors centralizado

Para los ejemplos distribuidos, el funcionamiento es idéntico, la diferencia es la adición de un pulsador de MARCHA y PARO para tener un control del sistema y adecuarlo a la realidad, mostrado en la *Figura 34*. Siempre que tenemos activo el pulsador de MARCHA sin que este pulsado el de PARO el sistema debe funcionar. Con el botón de paro, hacemos que se pare el sistema. El modelo de Grafcet para estos ejemplos son los mostrados en las *Figuras 35, Figura 36, Figura 37 y Figura 38*.

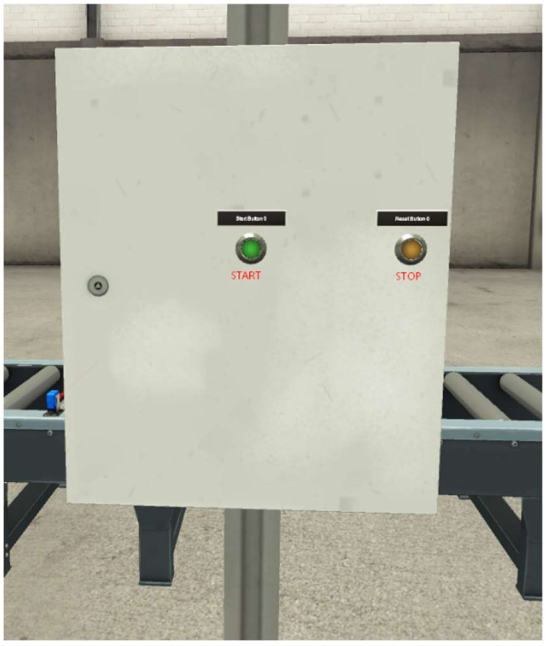

Figura 34: Botonera marcha paro en Factory io para el sistema distribuido del ejemplo Two Conveyors

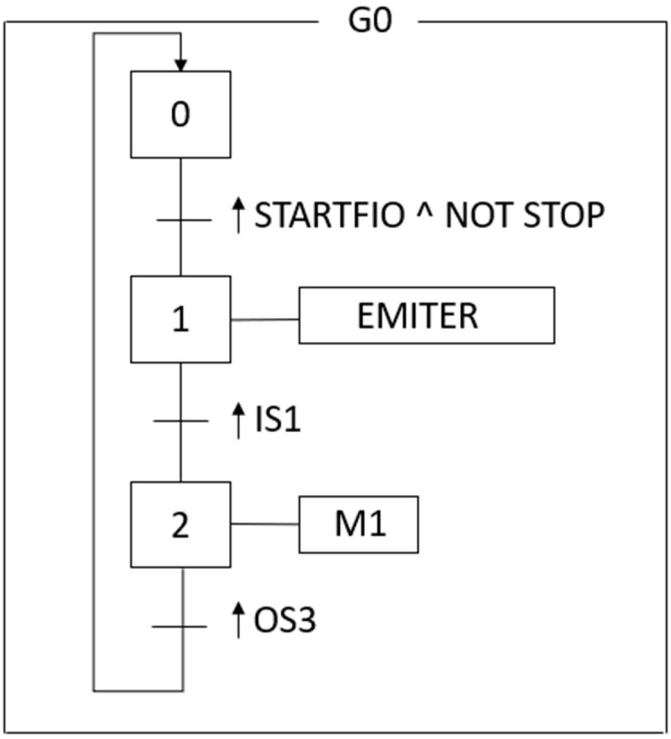

Figura 35: Grafcet G0 Control entrada cajas sistema distribuido

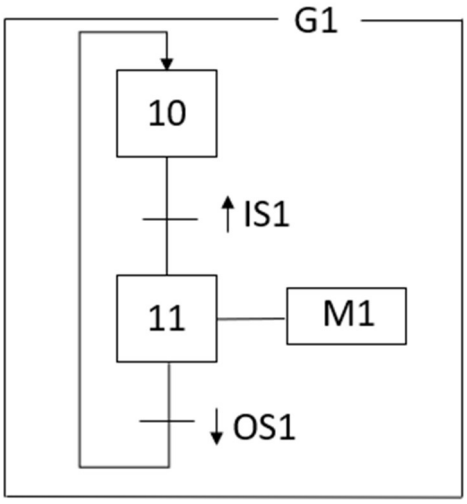

Figura 36: Grafcet G1 control rodillera 1 sistema distribuido

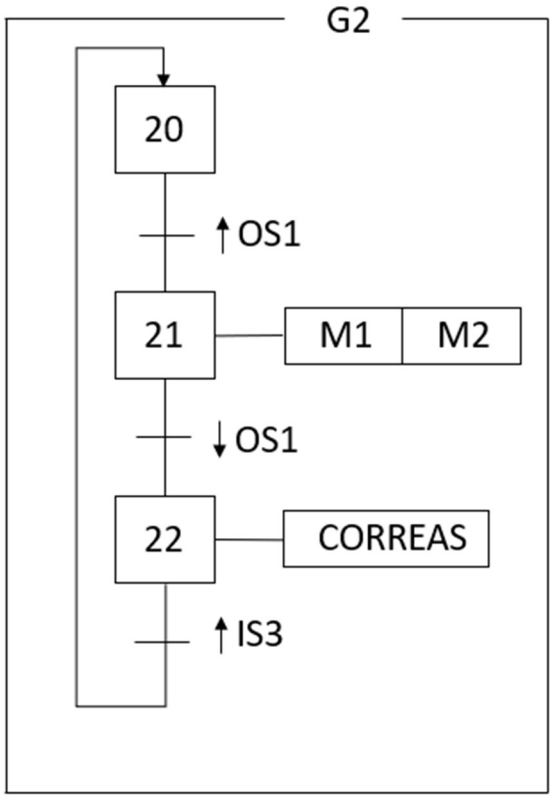

Figura 37: Grafcet G2 control rodillera giro sistema distribuido

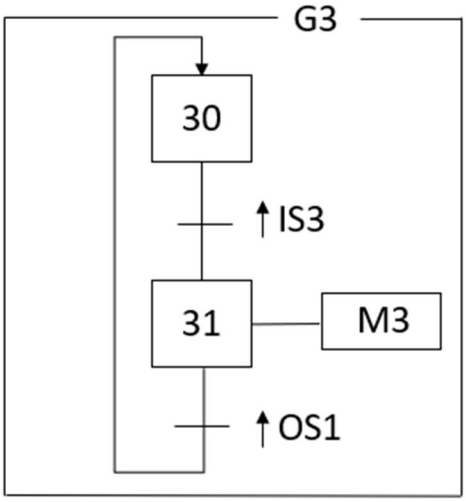

Figura 38: Grafcet G3 control rodillera 3 sistema distribuido

A continuación, se mostrarán las soluciones desarrolladas al ejemplo descrito con las distintas metodologías de trabajo.

Primero tenemos la solución empírica centralizada mostrada en la *Figura 39*. En esta solución los problemas que nos hemos encontrado vienen relacionados con los tipos de datos que se manejan a la salida de los F_NOT y F_AND, los cuales tiene problemas para interpretar los datos BOOL de entrada. No siempre dan error, pero en la mayoría de los casos sí. También podemos observar como la interfaz gráfica de la solución no es muy agradecida para su interpretación al tener cables de datos y de eventos superpuestos.

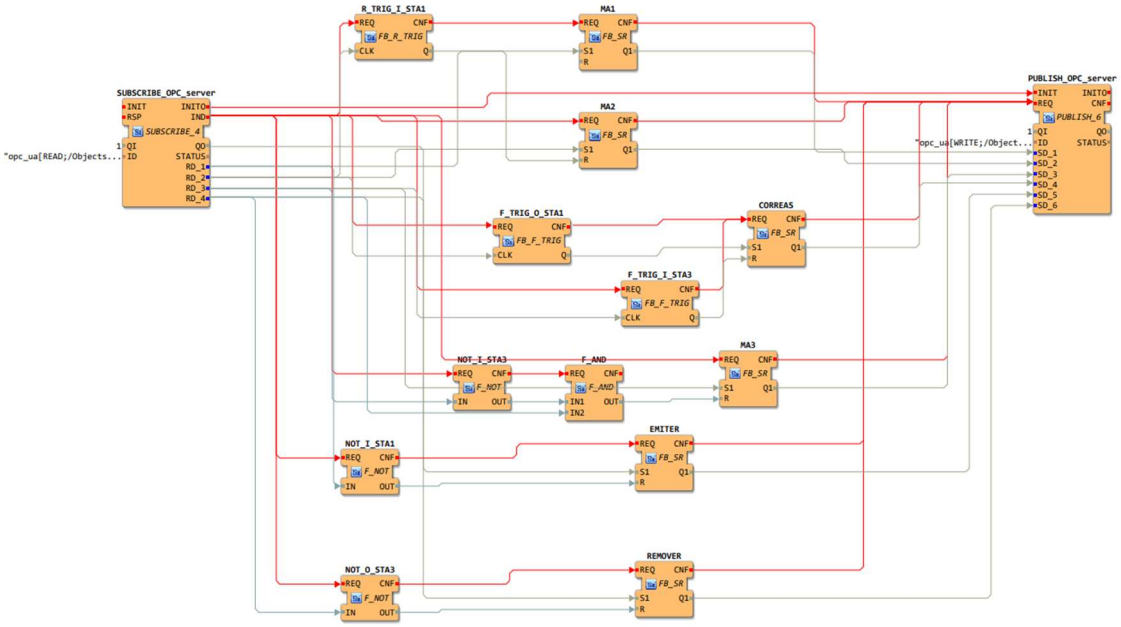

Figura 39: Programación desarrollada para el sistema centralizado mediante el método empírico del ejemplo Two Conveyors

En segundo lugar, tenemos las soluciones de eventos y datos centralizadas mostradas en las *Figura 40 y Figura 41*. Las vamos a comentar juntas por el hecho de que la metodología utilizada es la misma y así a la hora de comparar las soluciones será más fácil.

Como podemos observar en las imágenes, las soluciones son idénticas salvo dos pequeñas diferencias.

La primera diferencia, la tenemos en el tipo de bloque que utilizamos, el cual lo cambiamos de bloque de función de datos a bloque de función de eventos y esto repercute en que las etapas y las salidas de las transiciones se controlan por flujo de eventos. El evento lleva la información tanto de la ejecución del bloque, como la de la variable de tipo dato que necesita el siguiente bloque para ejecutar si función interna. Esto lo podríamos asemejar a una red redundante eléctrica en la que tenemos una línea de fuerza y otra de comunicación y en el caso de programación por eventos en la misma línea tendríamos tanto la fuerza como la comunicación.

La segunda la encontramos en que tenemos una transición menos en el ejemplo desarrollado por la metodología por datos. Esto viene dado por el hecho de que, en los bloques de eventos, no hay desarrollado un bloque el cual permita la función OR con los eventos. Por lo tanto, en la transición inicial (STARTFIO), la cual solo se utiliza una vez al inicializar el programa la podemos juntar utilizando un bloque F_OR con la transición cinco (R_TRIG_OS3) ya que la finalidad de estas dos transiciones en volver a la etapa inicial (E0). Esto nos sirve para simplificar las transiciones y tener el programa más compacto. En el caso por eventos no se puede realizar. También ayuda a descongestionar la cola de eventos que se está ejecutando, ya que los eventos de entrada a estas transiciones son proporcionales a ellas. Por lo tanto, si tenemos cinco transiciones a treinta eventos por minuto, en total tendremos ciento cincuenta eventos, mientras que con cuatro transiciones tenemos ciento veinte eventos.

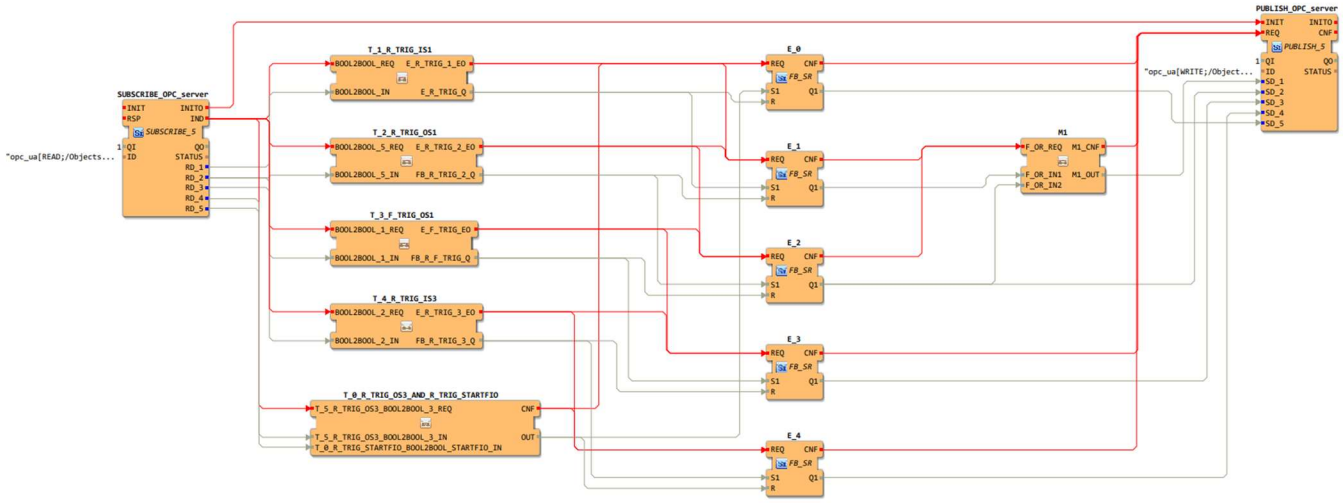

Figura 40: Programación desarrollada para el sistema centralizado mediante el método datos del ejemplo Two conveyors

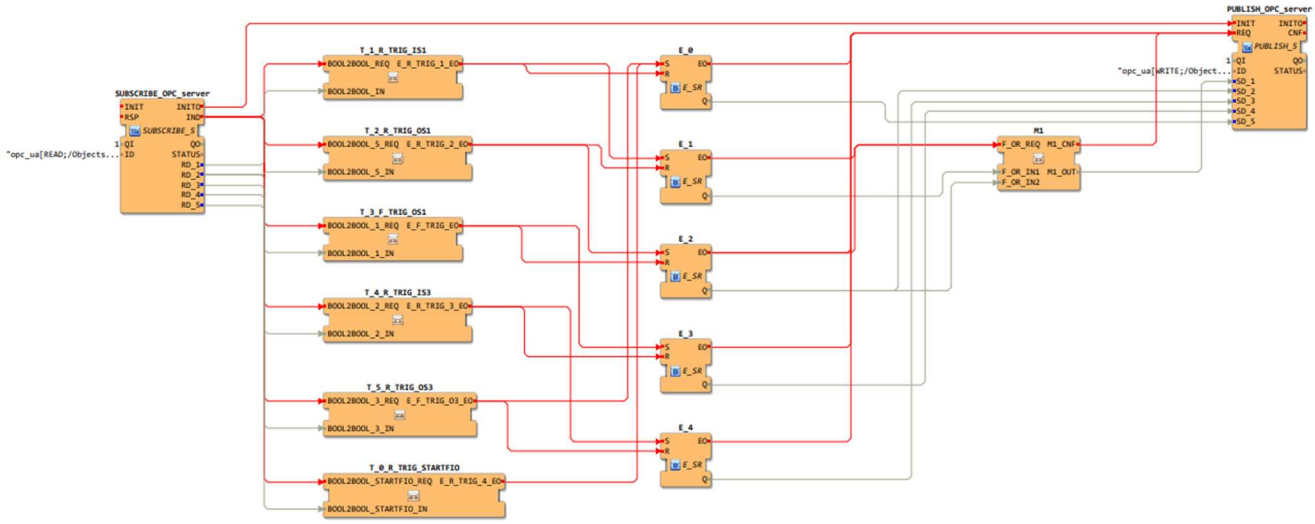

Figura 41: Programación desarrollada para el sistema centralizado mediante el método eventos del ejemplo Two conveyors

Por lo que respecta a las soluciones distribuidas, tenemos la misma casuística explicada en el anterior caso, tenemos una pequeña diferencia en tipos de bloques, pero en este caso no ha sido el principal problema diferencial entre las dos soluciones.

El problema viene dado a la hora de configurar el pulsador de START y STOP, ya que en el caso de eventos necesitamos un bloque llamado E_PERMIT, el cual hace de pasarela de cambio de dato a evento, si el dato es TRUE el evento es TRUE y si el dato es FALSE el evento es FALSE. El problema es que este bloque se tiene que ejecutar permanentemente para comprobar el estado del pulsador y al carecer de un bloque lógico F_AND o F_OR para crear una lógica de control que permita comprobar su estado en un momento adecuado, siempre estaba activando la etapa inicial del grafcet y no dejaba avanzar al programa. Por lo tanto, a día de hoy, con las herramientas que tenemos no es posible desarrollar el ejemplo completo con los bloques de función existentes basándose en la metodología desarrollada en el proyecto. En las figuras siguientes podemos ver las programaciones desarrolladas para el ejemplo.

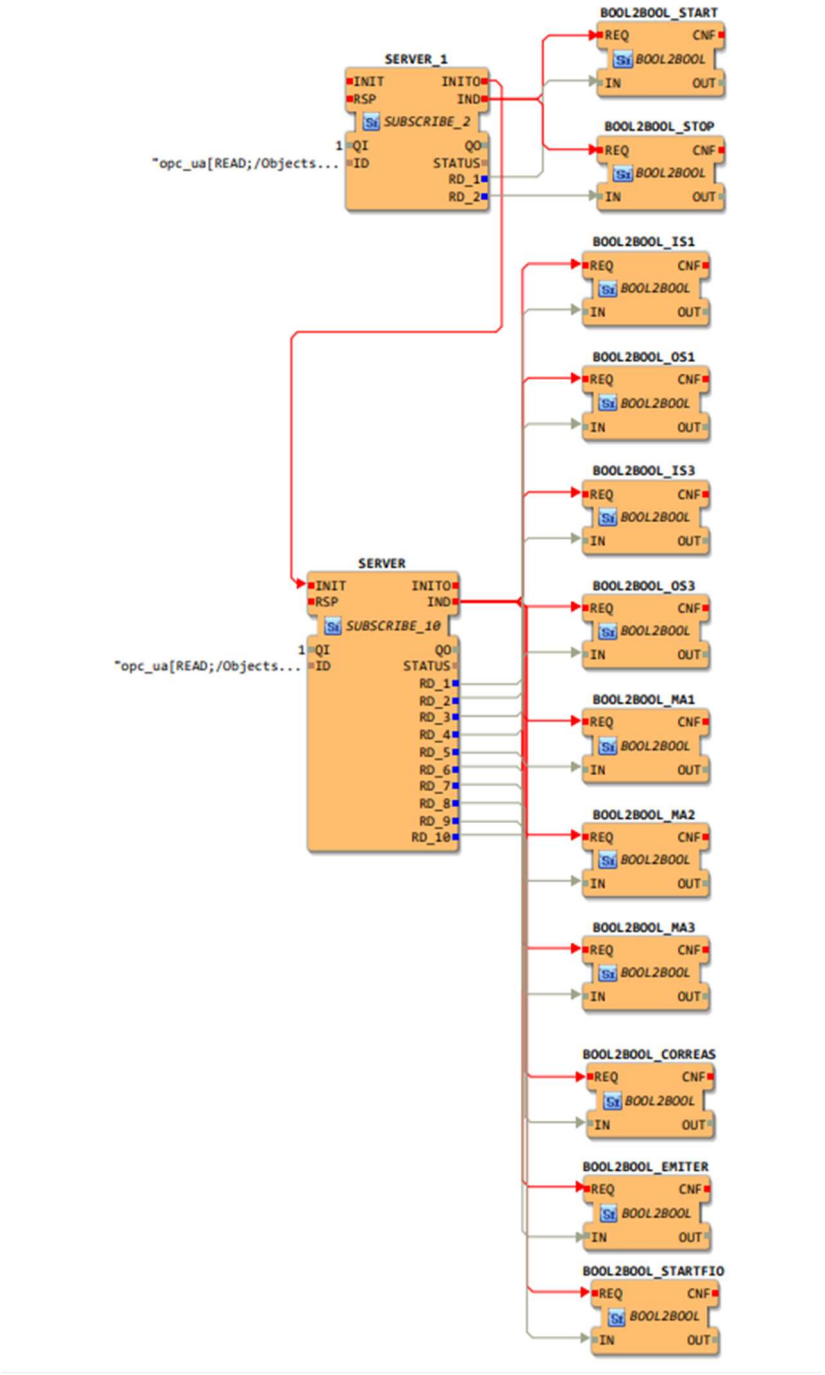

Figura 42: Programación desarrollada para la creación del servidor del sistema distribuido del ejemplo two conveyors

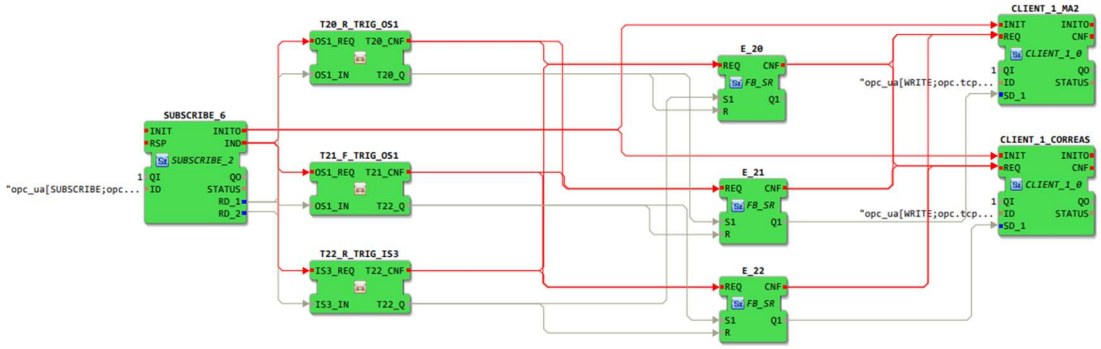

Figura 45: Programación desarrollada para el Grafcet G2 del sistema distribuido con el método datos

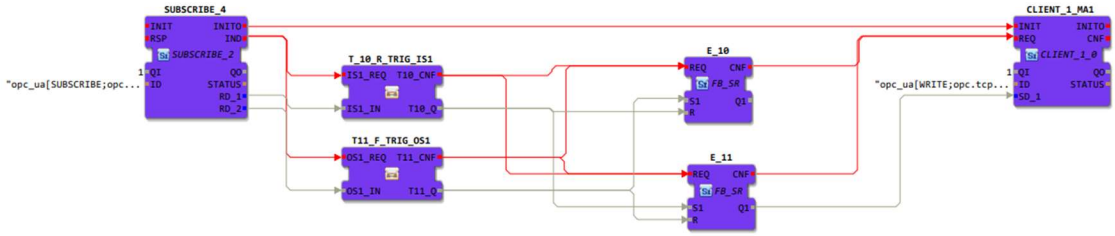

Figura 44:Programación desarrollada para el Grafcet G1 del sistema distribuido con el método datos

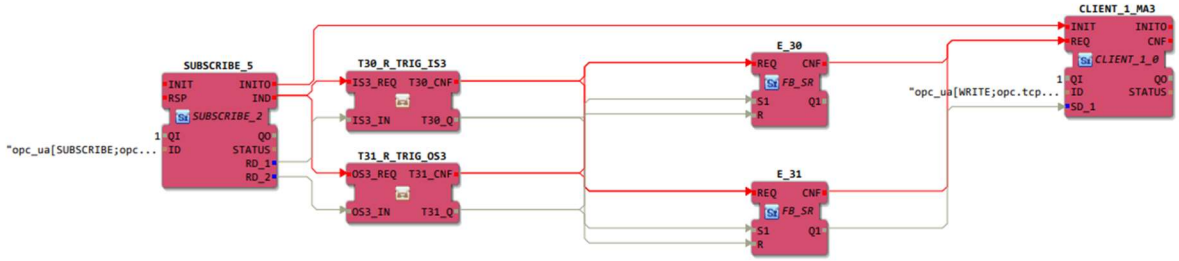

Figura 43: Programación desarrollada para el Grafcet G3 del sistema distribuido con el método datos

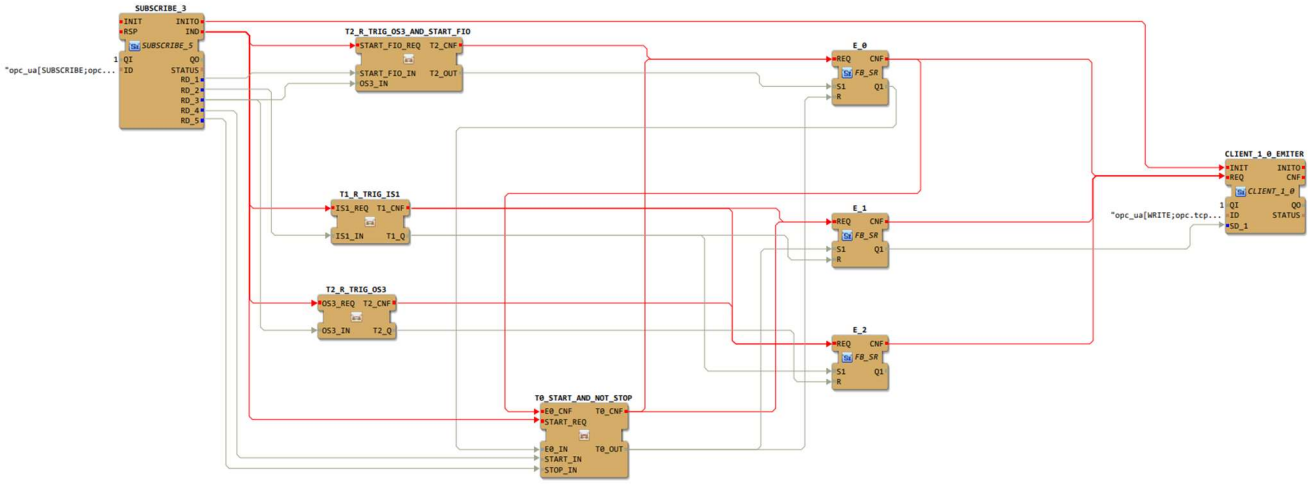

Figura 46:Programación desarrollada para el Grafcet G0 del sistema distribuido con el método datos

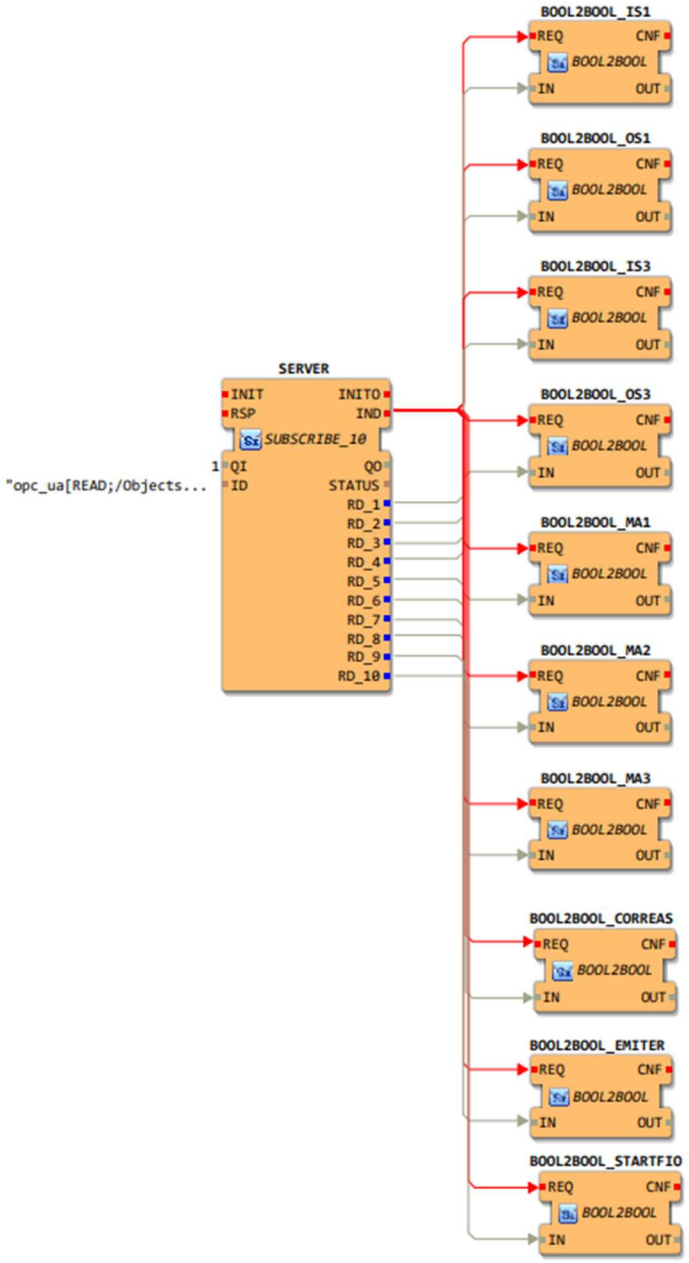

Figura 47: Programación desarrollada para la creación del servidor del sistema distribuido

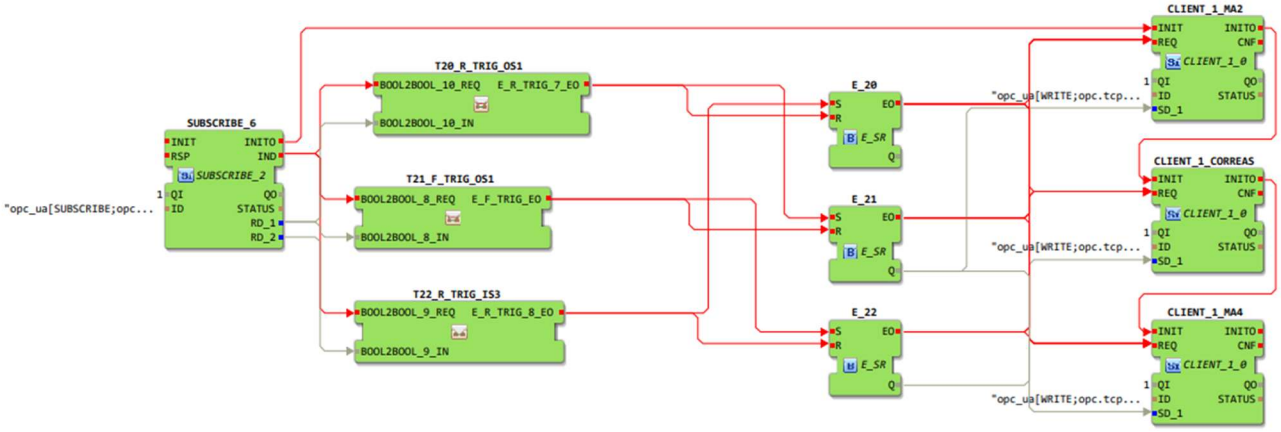

Figura 49: Programación desarrollada para el Grafcet G2 del sistema distribuido con el método eventos

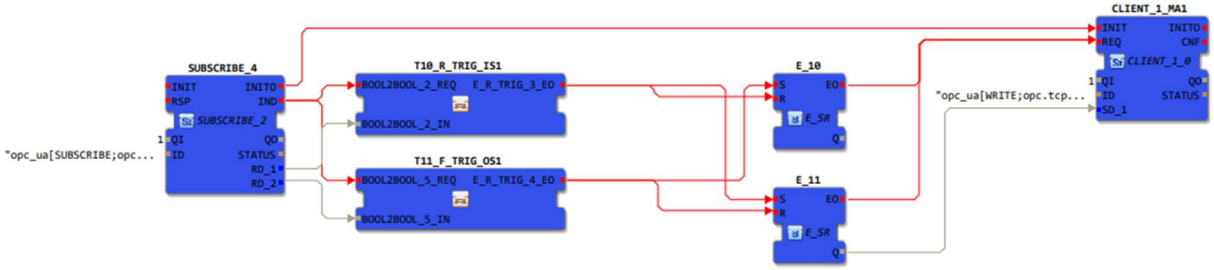

Figura 48: Programación desarrollada para el Grafcet G1 del sistema distribuido con el método eventos

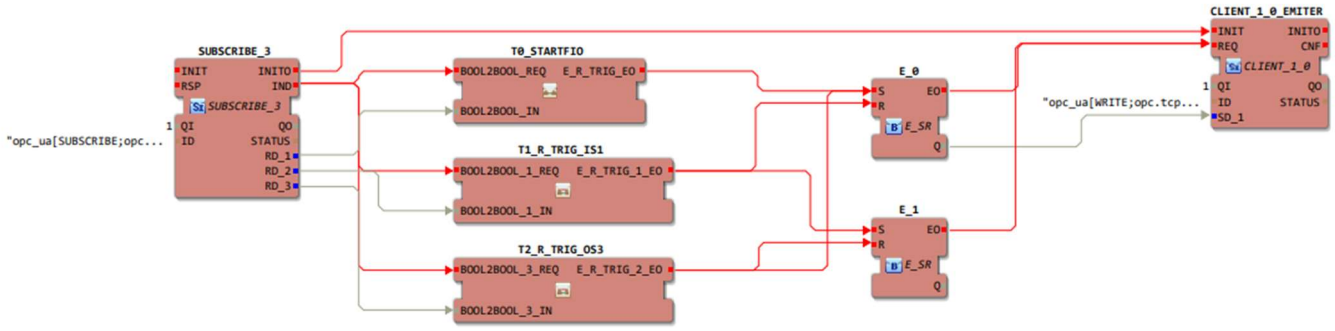

Figura 50: Programación desarrollada para el Grafcet G0 del sistema distribuido con el método eventos

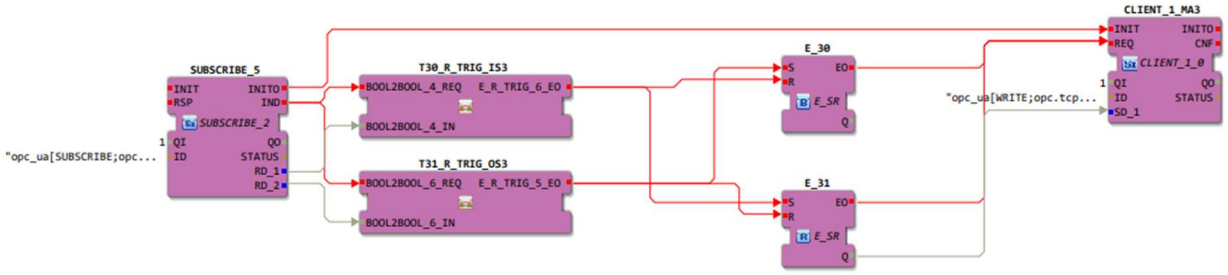

Figura 51: Programación desarrollada para el Grafcet G3 del sistema distribuido con el método eventos

Los ejemplos que mostraremos a continuación han sido extraídos del Libro de Automatización Industrial de la Universitat Jaume I [1].

5.2.2. EJEMPLO 2 CONTADOR LÁMPARA

El sistema de la *Figura 52* representa una cinta transportadora con motor M, dos detectores (C y D), dos pulsadores con enclavamiento (P e I) y un indicador Q. Sobre la cinta solo hay una caja cada vez. El funcionamiento es como sigue. Cuando se activa P la cinta comenzará a moverse. Cuando la caja es detectada por D, se debe incrementar el contador de cajas y activar el indicador Q durante un segundo. Si el pulsador I está activo cuando la caja es detectada por D entonces la cinta se detiene hasta que se desactive I. La desactivación de P implicará la parada de la cinta cuando una caja alcance la posición del detector C, activándose cuando se vuelva a pulsar P.

Figura 52: Sistema desarrollado en Factory io del ejemplo contador lámpara

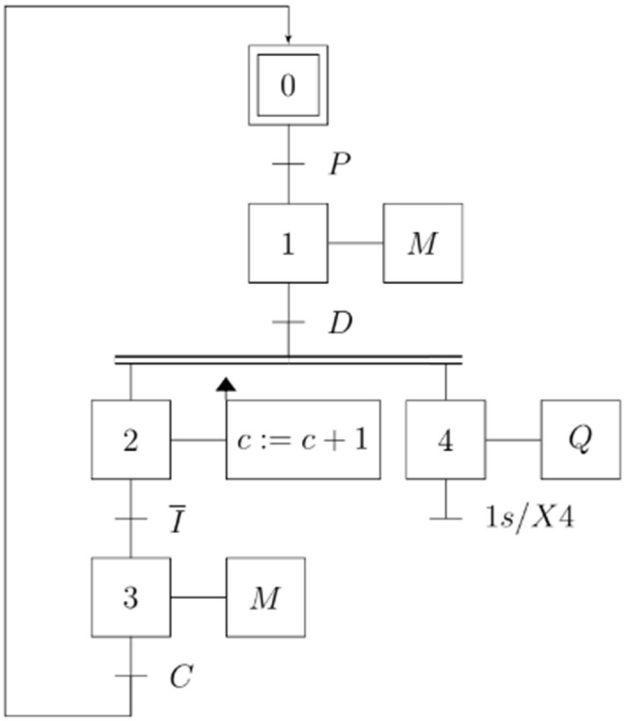

Figura 53: Grafcet ejemplo contador lámpara centralizado

En este ejemplo, al tener únicamente el Grafcet desarrollado para la solución centralizada (*Figura 53)*, las soluciones desarrolladas son centralizadas. Tenemos la solución empírica, la solución con datos y la de eventos.

La primera de ellas, la empírica (*Figura 54)*, como en el caso anterior fue desarrollada para validar el software de programación con un ejemplo más complicado que el mostrado anteriormente, ya que aquí tenemos pulsadores de marcha y paro e incluso un contador, que eran bloques desconocidos para nuestro conocimiento del software. Se puede observar como la programación se hace más difícil de seguir e interpretar que en el caso anterior ya que entran en juego muchos más elementos. Y las condiciones que activan o desactivan los actuadores no son directas, con lo que encontramos muchos bloques F_AND, F_OR y F_NOT para crear la lógica e incluso bloques FB_SR para crear biestables para mantener las señales en el estado deseado.

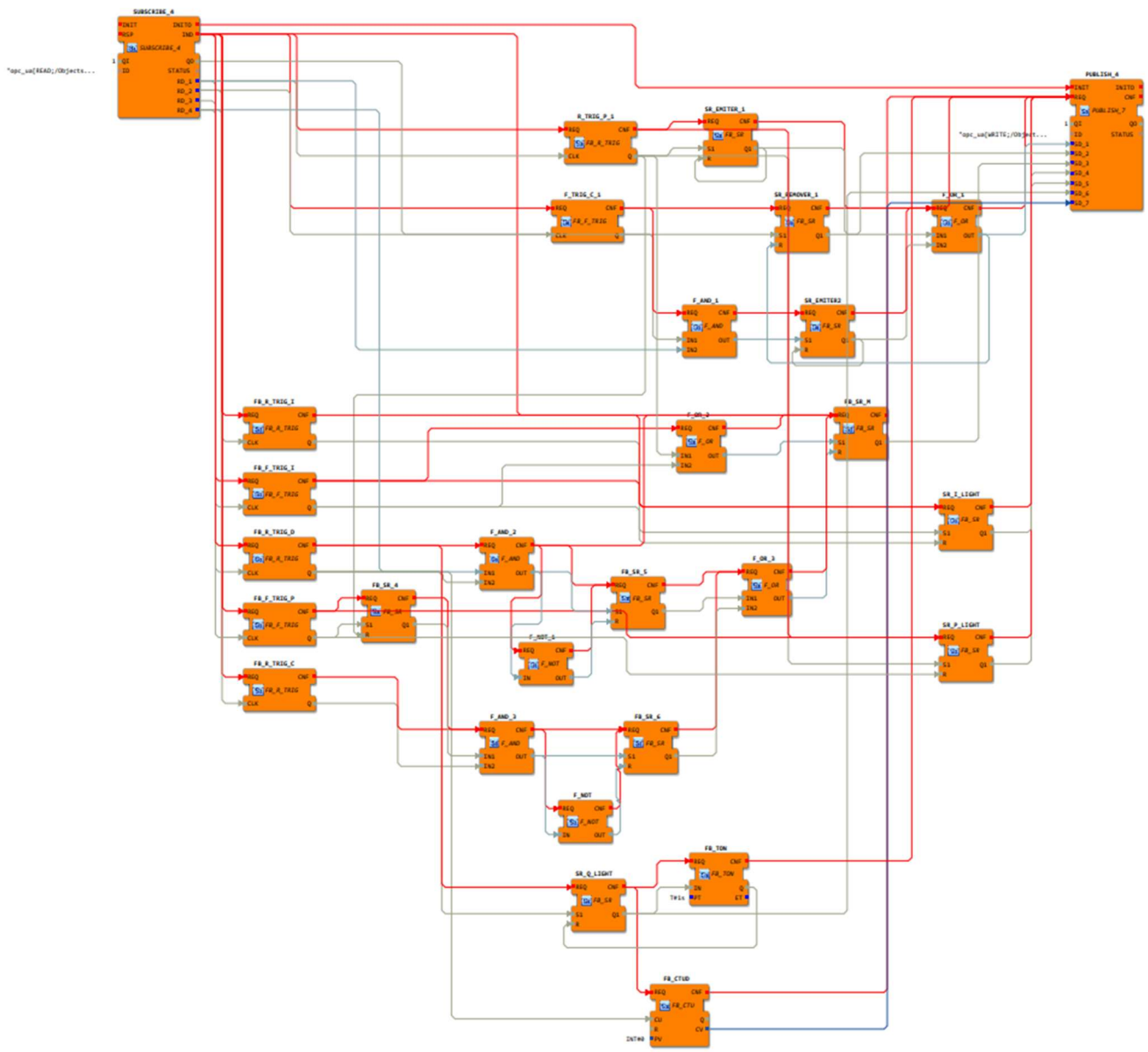

Figura 54: Programación desarrollada para el sistema centralizado mediante el método empírico del ejemplo contador lámpara

En segundo lugar, se intentó desarrollar la solución con eventos, ya que la intención era desarrollar la metodología basándose en ellos, porque como hemos comentado antes si la comunicación va por un mismo canal de comunicación, se asegura que cuando llega la información de ejecución llega la información de proceso y así no estamos enviando información de proceso sin sentido a sabiendas que la información de ejecución no va a llegar o viceversa. El problema es que a la hora de hacer las transiciones de los diagramas de Grafcet nos encontramos dificultades. En este ejemplo nos hemos encontrado las siguientes:

- Pulsadores marcha y paro. No hay forma de realizarlo con las especificaciones que tenemos, por el hecho de que el único bloque que nos permite realizarlo es el E_PERMIT y como hemos comentado antes da más problemas de los que soluciona. Por lo que en el ejemplo estas transiciones se han desarrollado en control por datos
- Cuando una transición necesita comprobar que una etapa anterior esta activa, no lo podemos realizar por falta del bloque AND o OR para eventos. Por lo tanto, también se ha tenido que desarrollar con bloques de datos.
- Por último, en el bloque CTUD al tener un bloque E_PERMIT para transformar los datos en eventos, teníamos el problema que el contador incrementaba más de lo que debía, porque al bloque E_PERMIT había llegado más eventos de los que tocaban por peculiaridades del sistema. Ya que si en la E2 le enviamos eventos, cuando acaba su proceso los transmite aguas abajo y eso no lo podemos controlar por el hecho de que el servidor está enviando eventos con un ciclo determinado.

  En la *Figura 55* se muestra cómo queda el ejemplo utilizando los bloques de eventos junto con los bloques de datos necesarios para su correcto funcionamiento.

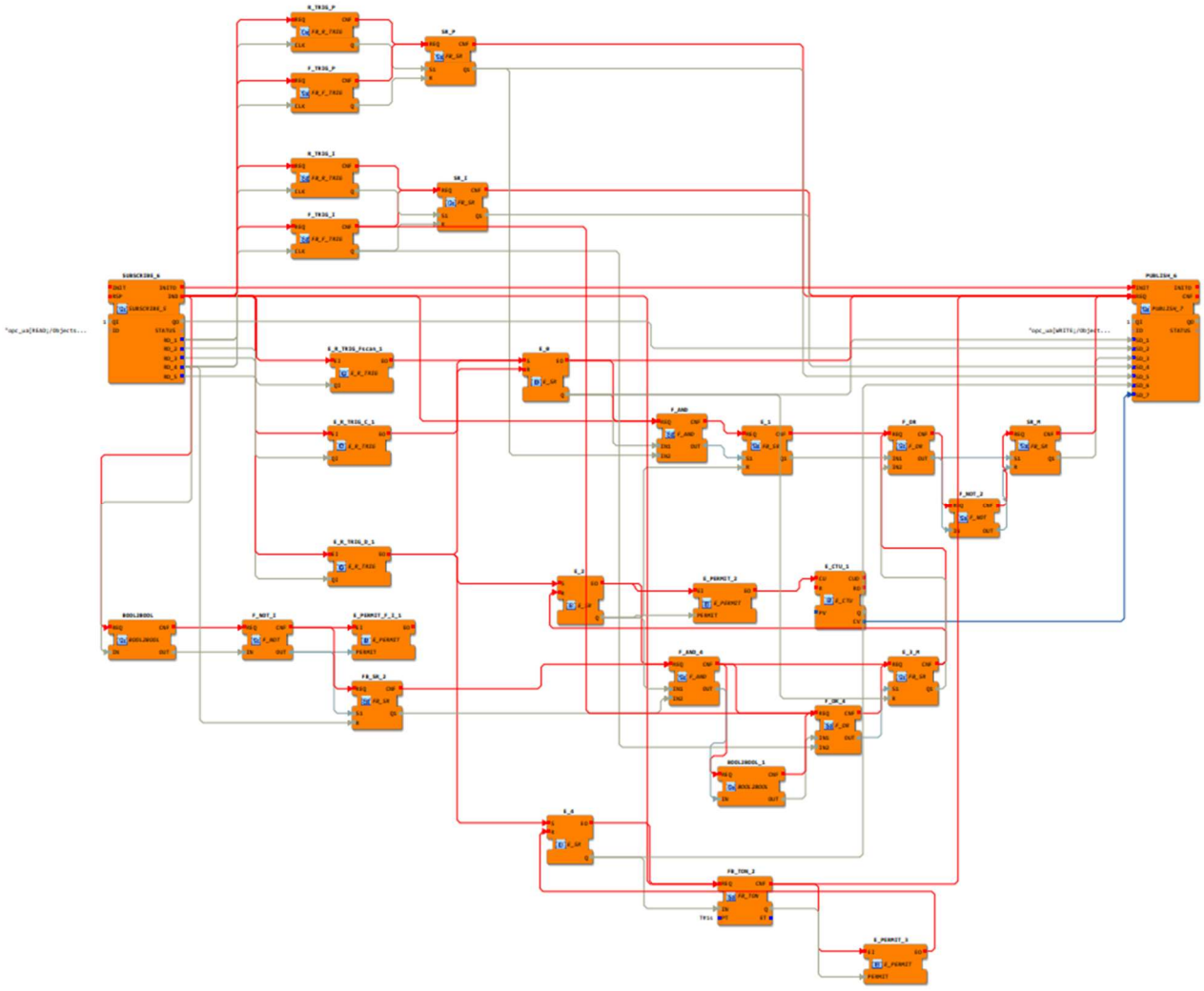

Figura 55: Programación desarrollada para el sistema centralizado mediante el método eventos del ejemplo contador lámpara con bloques de datos

Por último, tenemos el ejemplo desarrollado con la metodología del proyecto, con datos. Aquí no hemos encontrado ninguna dificultad para desarrollarlo. Además, se queda un programa muy limpio y estructurado que facilita su lectura y comprensión al igual que la búsqueda de errores. En la *Figura 56* podemos ver la solución que plasma el Grafcet desarrollado.

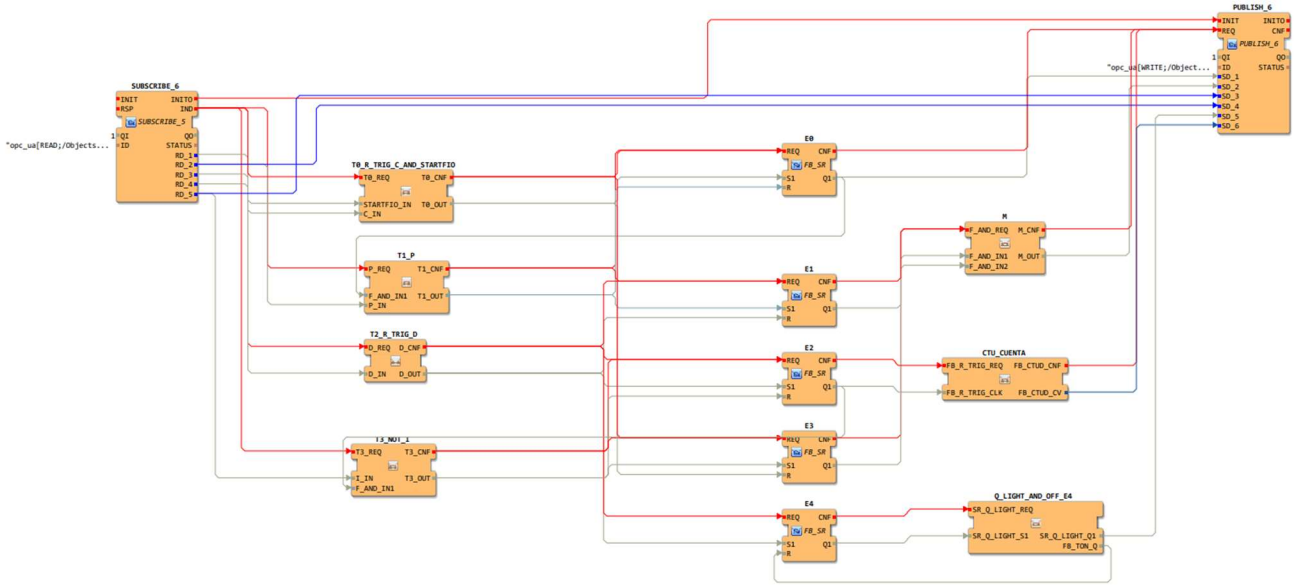

Figura 56: Programación desarrollada para el sistema centralizado mediante el método datos del ejemplo contador lámpara

5.2.3. EJEMPLO 3 OPERACIONES AB

Considérese el sistema de la *Figura 57*.

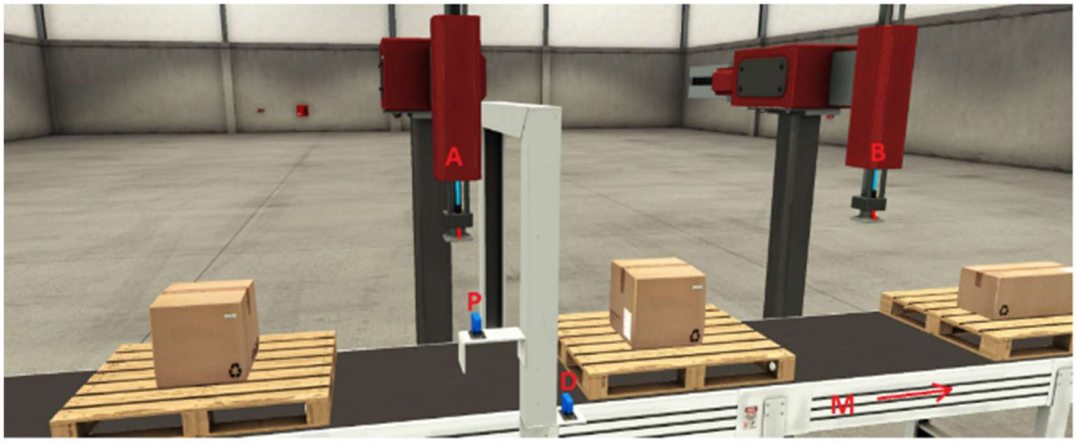

Figura 57: Sistema desarrollado en Factory io del ejemplo operaciones AB

Por una cinta accionada por el motor M circulan palets espaciados regularmente, de forma que cuando hay un palet en A, hay otro en B. Sobre estos palets puede haber dos tipos de cajas; altas y bajas. En las cajas altas hay que realizar las operaciones A y B (podrían ser marcado, etiquetado, etc). El detector inductivo D detecta el paso de los palets, mientras que la fotocélula P detecta la presencia de caja alta sobre palet. Las operaciones A y B duran 0.2 y 0.5 segundos respectivamente.

En este último ejemplo de sistemas centralizado, sólo se han desarrollado con la metodología empírica y con datos, ya que como se vio en el ejemplo anterior, no tenemos las herramientas suficientes para poder seguir utilizando la metodología por eventos. El diagrama de grafcet lo podemos ver en la *Figura 58*.

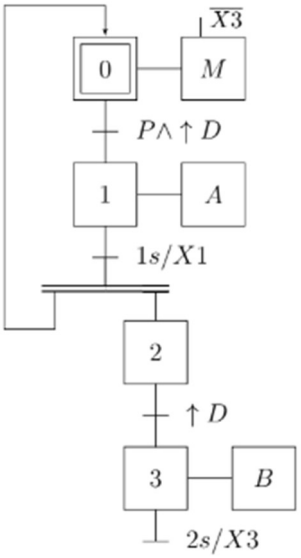

Figura 58: Grafcet ejemplo contador operaciones AB

En cuanto a la metodología empírica, tenemos una programación muy fácil, ya que el funcionamiento del sistema es simple, al tener pocos sensores y actuadores. Al no tener funciones anidadas como en el ejemplo anterior ni pulsadores de marcha y paro, el ejemplo se resuelve de forma muy sencilla. En la *Figura 59* podemos ver el desarrollo de la programación con este método.

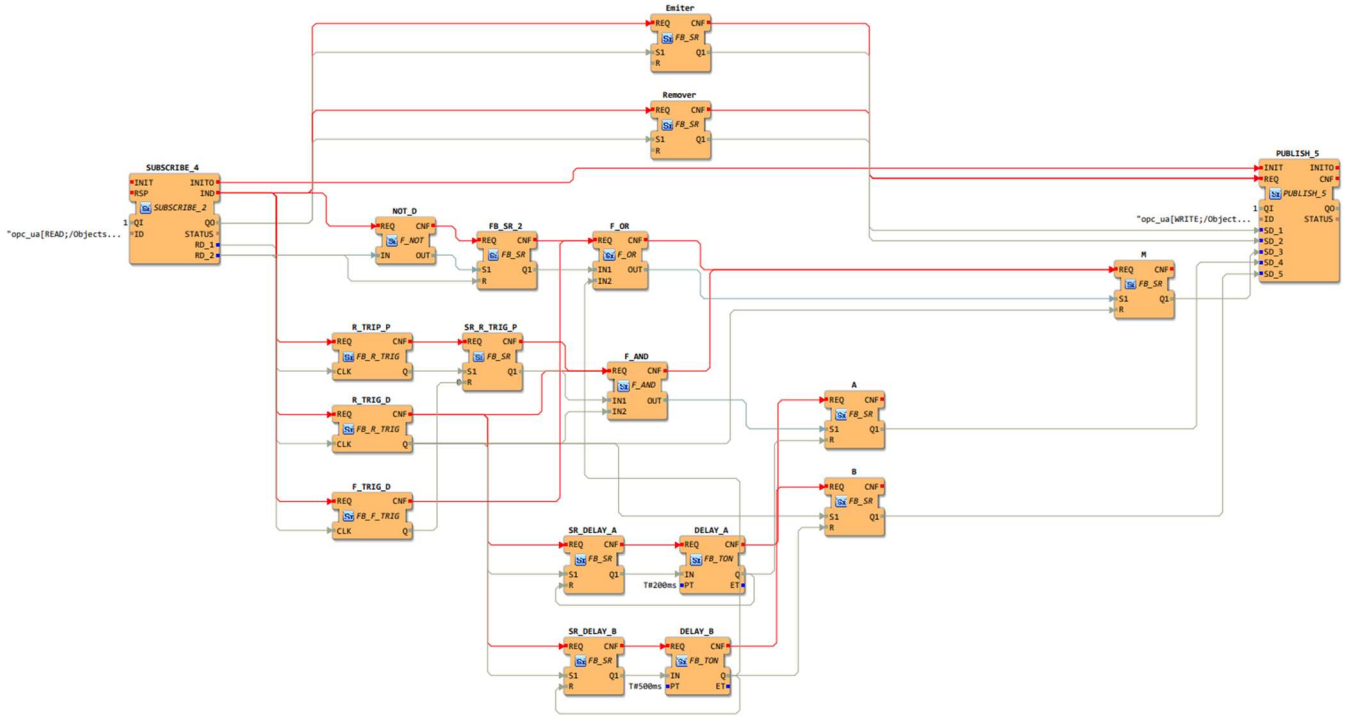

Figura 59: Programación desarrollada para el sistema centralizado mediante el método empírico del ejemplo operaciones AB

Con respecto a la metodología con datos (*Figura 60)* podemos destacar que, para la activación de la E0, se utiliza el reset de la E1 en vez de la salida del bloque DELAY_X1, esto se realiza así porque se da el caso en que el delay al resetear la E1, resetea esta etapa antes que da la señal a la T0 de que pongo en set la E0, por lo tanto, el programa se queda en stand-by y no llega a funcionar correctamente. Esto se puede deber porque estamos ejecutando tanto el servidor como el programa de 4DIAC y el programa de simulación en local y si el procesador del equipo no es lo suficientemente potente para ejecutar con rapidez todas las órdenes de trabajo, al tener activaciones y desactivaciones de una forma anidada, necesitamos una alta ejecución de procesos. En principio en equipos industriales dónde la carga computacional no sea tan alta, debería funcionar sin problema, pero en este caso no es viable realizar la comprobación por la falta de hardware.

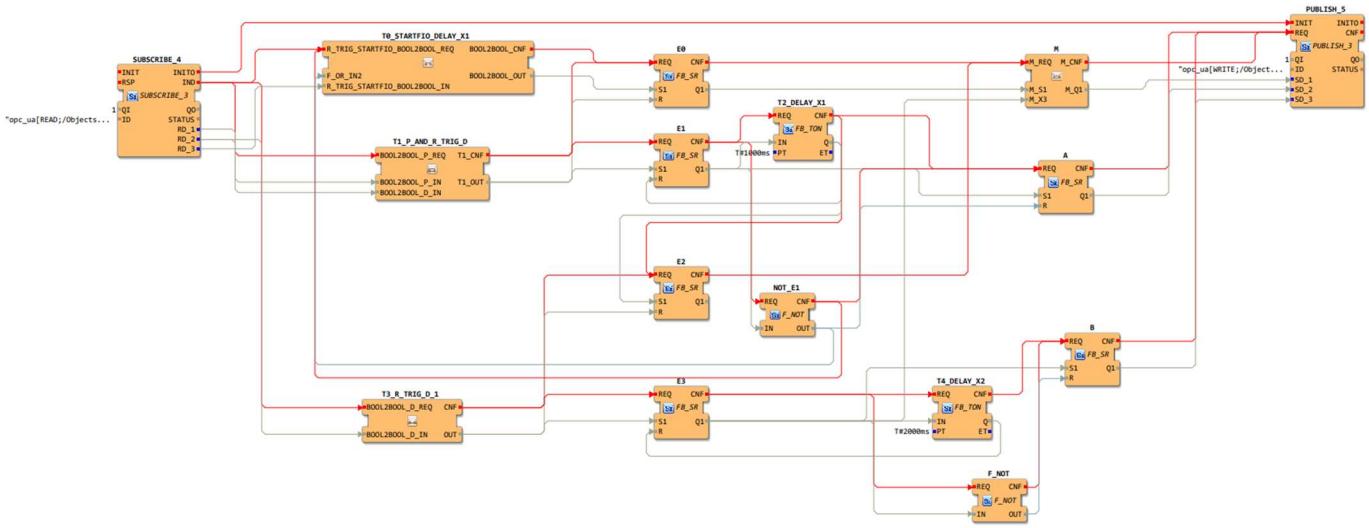

Figura 60: Programación desarrollada para el sistema centralizado mediante el método datos del ejemplo operaciones AB

5.2.4. EJEMPLO 4 SORTING BY HEIGHT

En la *figura 62* se muestra el esquema de un sistema para la clasificación de cajas que llegan por la cinta de entrada EntryConveyor a la mesa rotatoria TurnTable. Las cajas altas deben moverse hacia la cinta transportadora derecha RightConveyor y las cajas más bajas hacia la cinta de la izquierda LeftConveyor. La cinta de entrada tiene capacidad para varias cajas que llegan aleatoriamente por la cinta alimentadora FeederConveyor. Las cintas RightConveyor y LeftConveyor tienen capacidad para una sola caja, y deben estar activas solo si están en moviendo una caja en la dirección correspondiente. Los sensores y actuadores divididos por dispositivos del sistema son los siguientes:

Feeder Conveyor:

- FeederConveyor (Motor cinta)

Entry Conveyor:

- EntryConveyor (Motor cinta)
- AtEntry (Sensor entrada)
- HighBox (Sensor detección cajas)
- AtTurnTable (Sensor salida)

Turn Table:

- Load (Motor rodillos)
- Unload (Motor rodillos)
- Turn (Motor giro)
- AtFront (Sensor detección de cajas)
- AtBack (Sensor detección de cajas)
- LoadPosition (Sensor detección posición de carga de la mesa)
- UnloadPosition (Sensor detección posición de descarga de la mesa)

Right Conveyor

- RightConveyor (Motor cinta)
- AtRightEntry (Sensor entrada)
- AtRightExit (Sensor salida)

Left Conveyor

- LeftConveyor (Motor cinta)
- AtLeftEntry (Sensor entrada)
- AtLeftExit (Sensor salida)

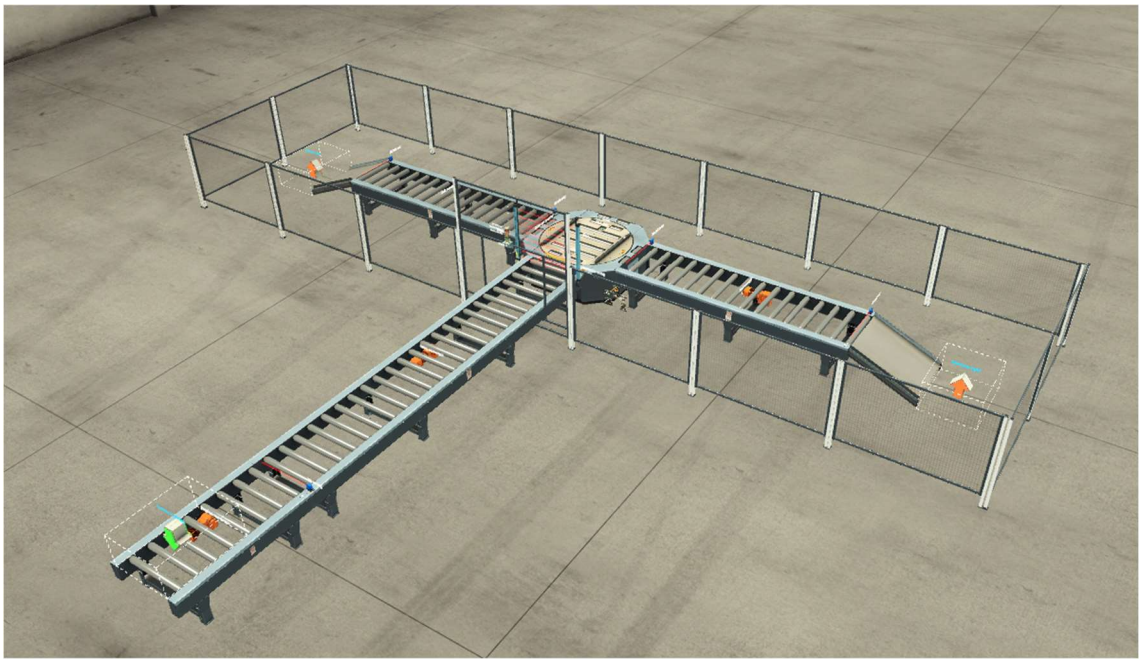

Figura 61: Imagen del sistema en Factory Io

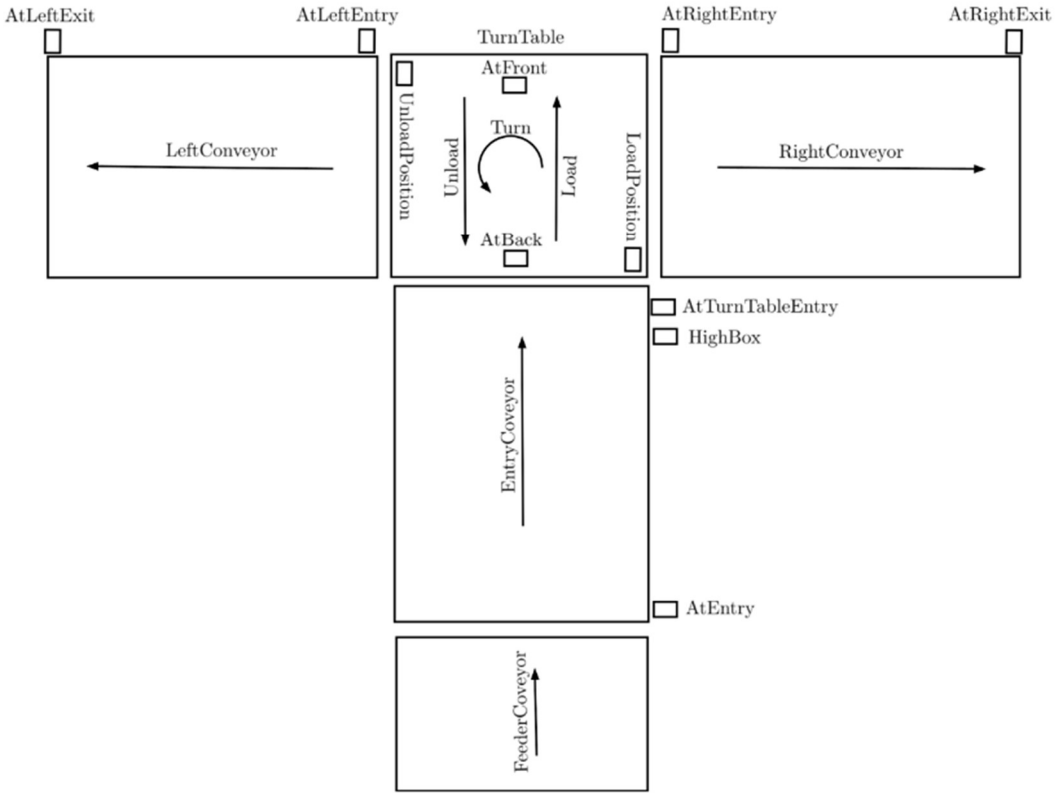

Figura 62: Esquema del sistema Sorting by height

Los diagramas de Grafcet desarrollados para el sistema distribuido se muestran en las figuras siguientes.

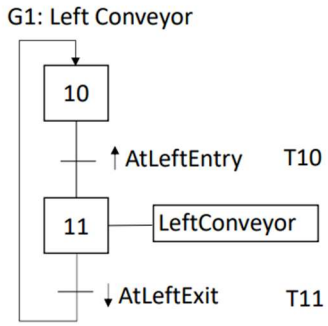

Figura 63:Grafcet G1Left Conveyor

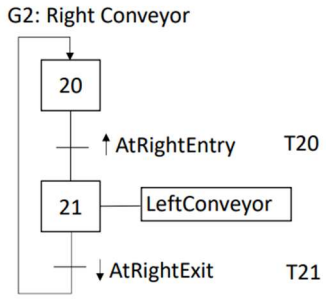

Figura 64: Grafcet G2 Right Conveyor

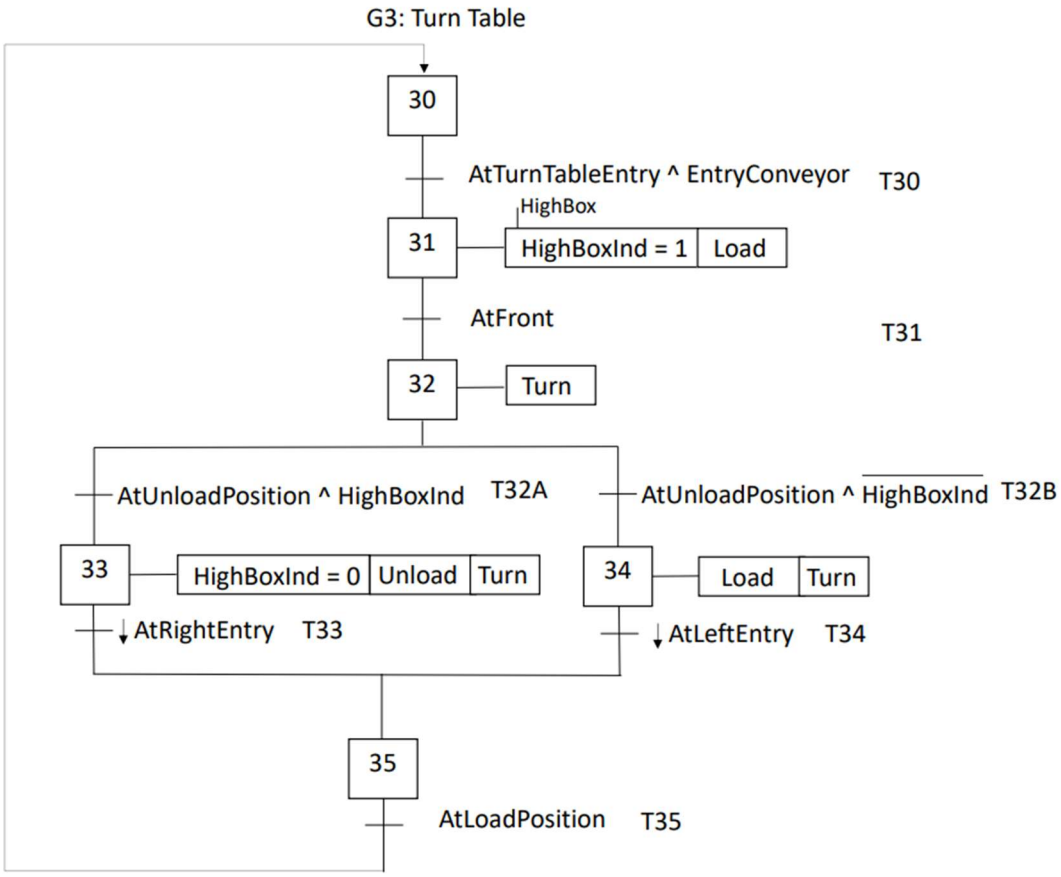

Figura 66: Grafcet G3 Turn Table

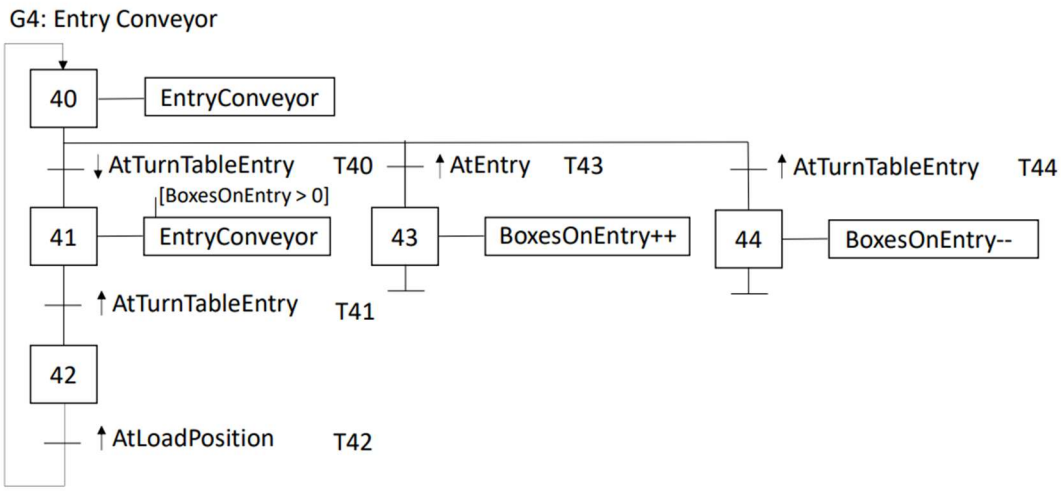

Figura 65: Grafcet G4 Entry Conveyor

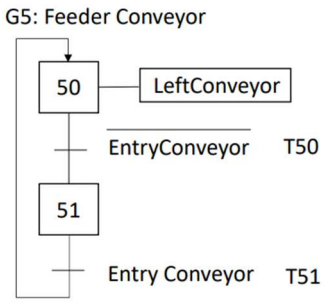

Figura 67: Grafcet G5 Feeder Conveyor

Este último ejemplo se ha desarrollado con la metodología de datos, porque como bien hemos comentado en los ejemplos anteriores, los elementos de las otras metodologías no están lo suficientemente desarrolladas como para poder realizar este tipo de ejemplos, los cuales presentan una mayor complejidad.

Como peculiaridad en este sistema, el cual tiene una interconexión de los sistemas amplia, se ha tenido que implementar en las transiciones un dato que va en función de la etapa anterior a la transición. Esto ayuda a que una transición no se active hasta que esté en la etapa anterior a su ejecución, ya que como puede haber varios elementos coexistiendo en el sistema, la activación de algunos sensores se puede dar en varios momentos del ciclo productivo de la máquina. Esta activación de los sensores no tiene que interferir con el funcionamiento del sistema y por ello es necesario el dato de activación de la etapa anterior, para no activar etapas en momentos que no son los adecuados.

Estos datos de las etapas no son necesarias en todas las transiciones, aunque se podrían implementar sin mayor complejidad y realizar un sistema más robusto. Pero con esto cargamos computacionalmente al sistema y ralentiza mucho la adquisición de datos del servidor y da fallos el sistema por no tener una velocidad de intercambio de datos adecuada a la velocidad del programa de simulación. Por lo tanto, solo se ha utilizado este recurso en las transiciones críticas que hacían que el sistema funcionara de manera incorrecta.

A continuación, mostramos las programaciones desarrolladas en 4DIAC para el sistema descrito anteriormente.

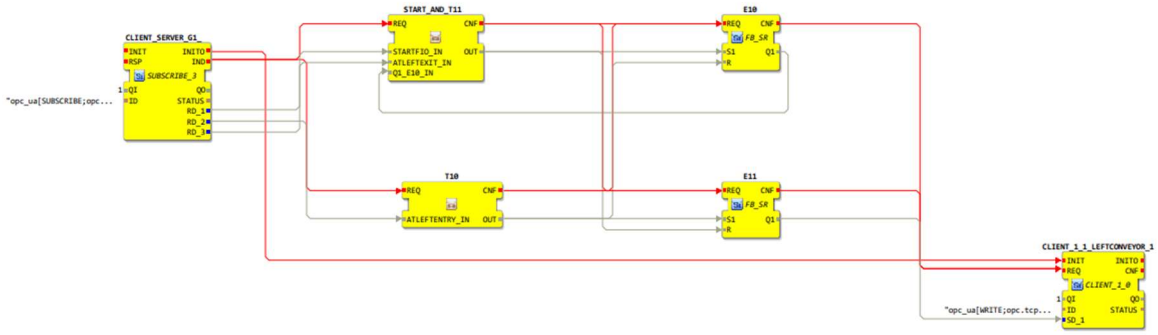

Figura 68: Programación desarrollada para el Grafcet G1 Left Conveyor

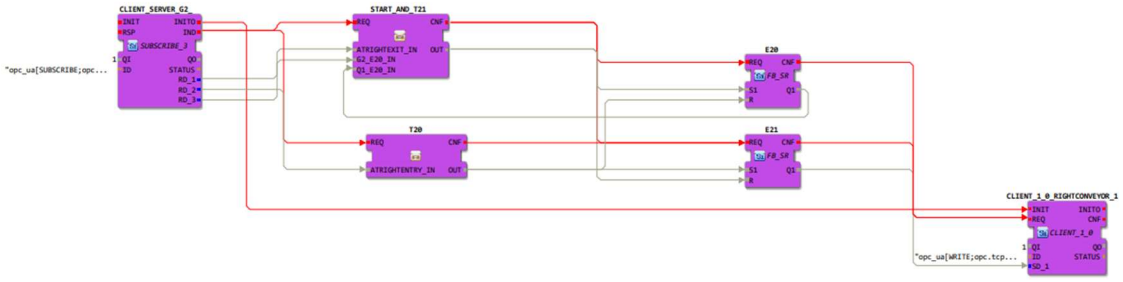

Figura 69: Programación desarrollada para el Grafcet G2 Right Conveyor

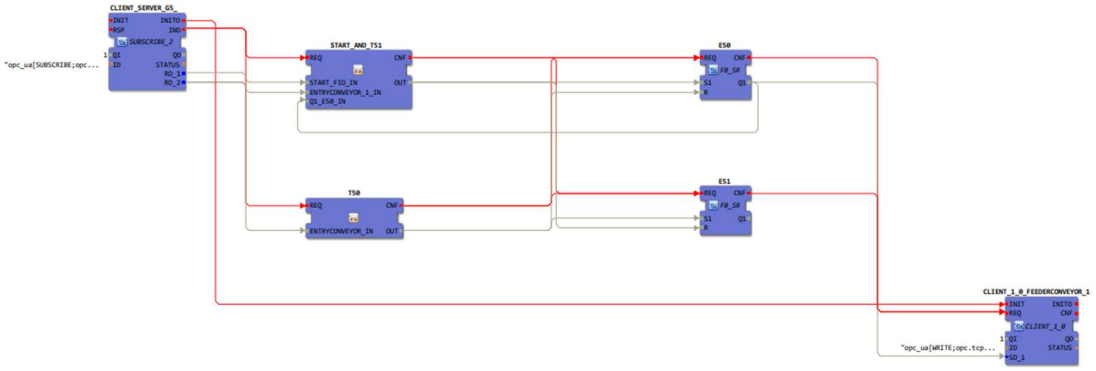

Figura 72: Programación desarrollada para el Grafcet G5 Feeder Conveyor

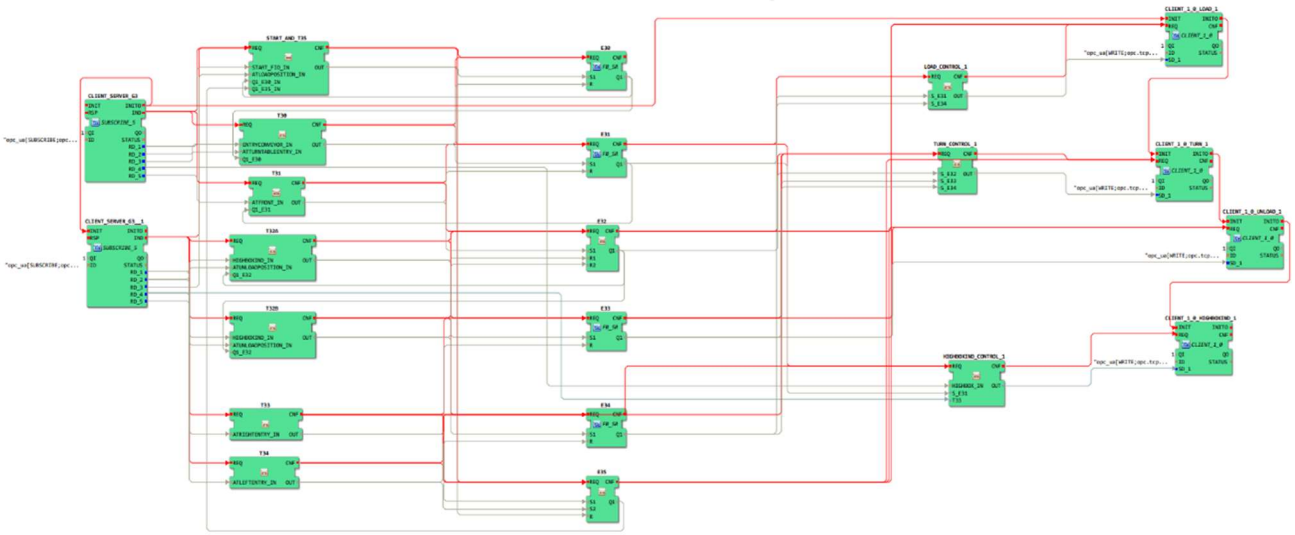

Figura 70: Programación desarrollada para el Grafcet G3 Turn Table

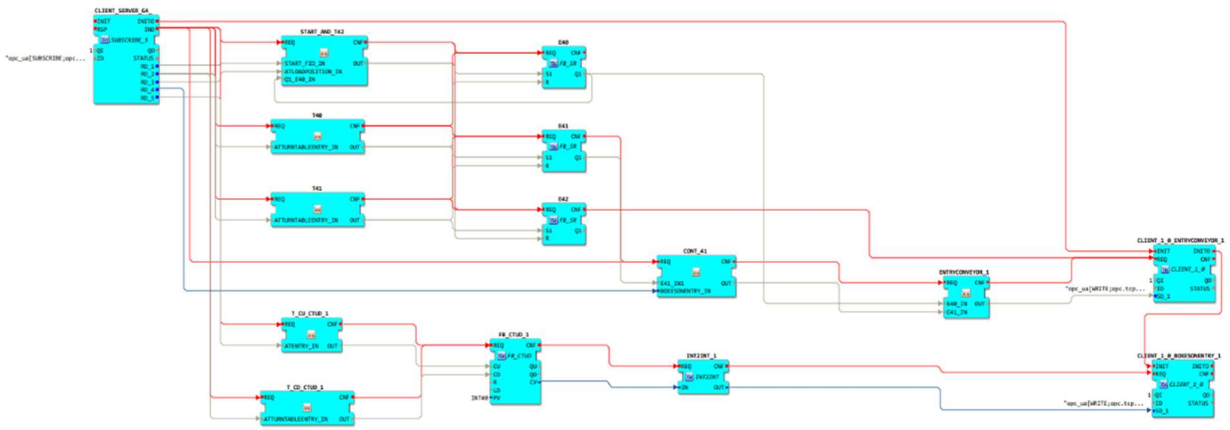

Figura 71: Programación desarrollada para el Grafcet G4 Entry Conveyor

Por último, en la *figura 73,* podemos observar un ejemplo de cómo queda la interfaz del bloque de subaplicación que desarrollamos para las transiciones y la programación interna de las mismas. Para las demás transiciones realizamos el mismo procedimiento con las particularidades de cada transición.

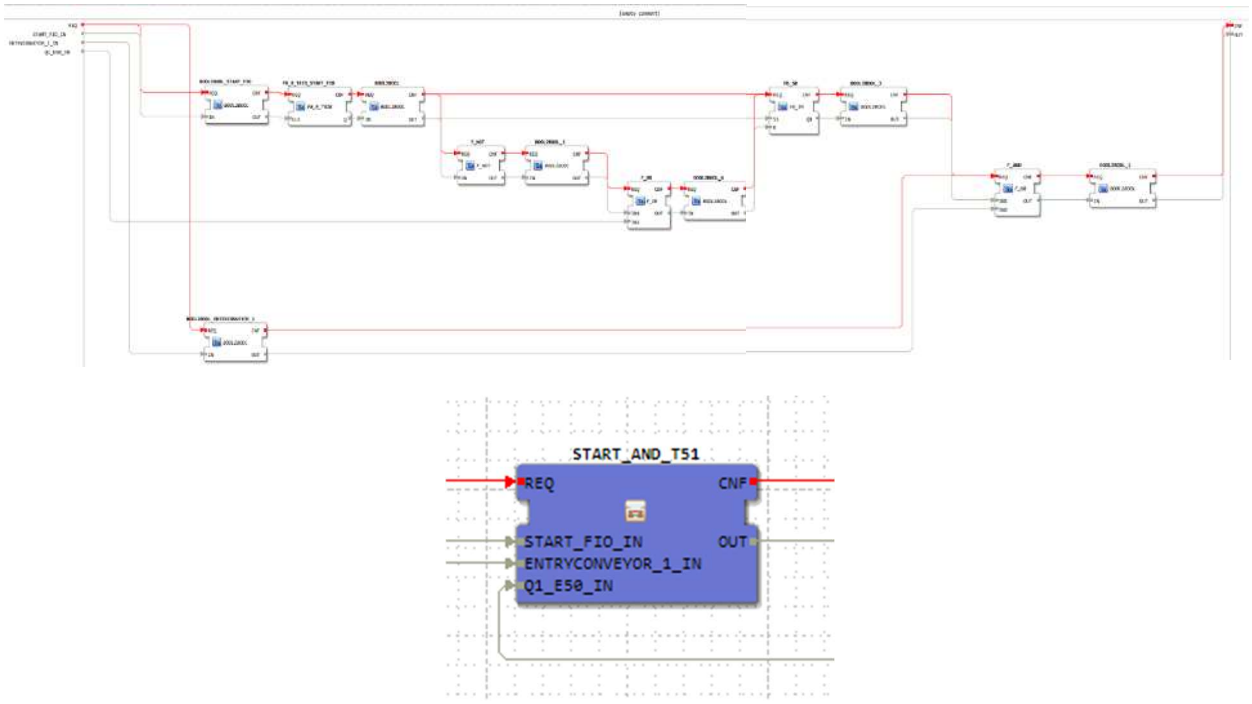

Figura 73: Interfaz y programación interna de la transición del Grafcet 5, Transición T51

Por último, se ha intentado realizar una botonera con órdenes forzadas para el control del sistema por parte del usuario. En la *Figura 74* mostramos el diagrama de Grafcet que controla la ejecución del sistema. También podemos observar en la *Figura 75* la botonera que se ha instalado en el programa de simulación.

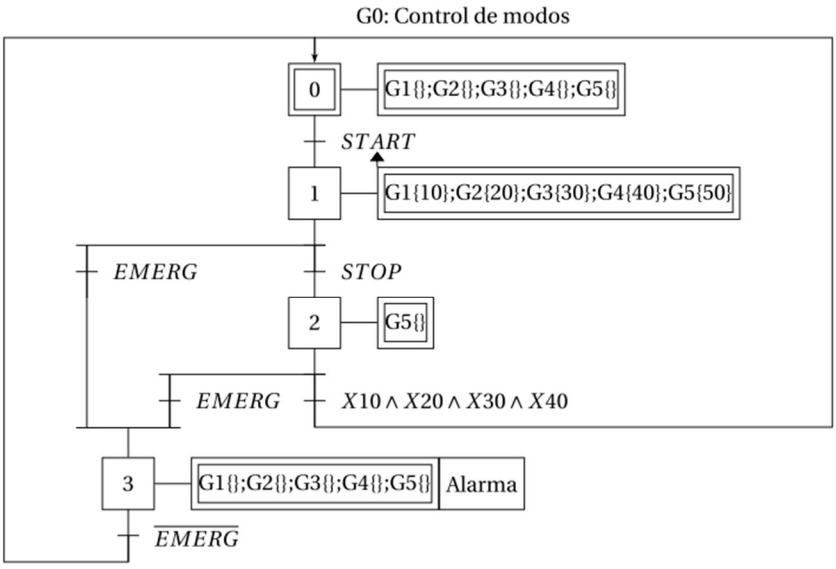

Figura 74: Grafcet de control G0 utilizado por la botonera

Figura 75: Botonera ejemplo sorting by height

La botonera se compone de los siguientes elementos:

- Botones (Start, Stop y Emergencia)
- Luz (Alarma)
- Display (Cuenta piezas)

El funcionamiento del sistema es el siguiente:

El sistema se pone en funcionamiento al pulsar el botón de START. Si queremos para el sistema tenemos dos opciones. Si es una emergencia debemos pulsar el botón de EMERGENCIA, con el que paramos todo el sistema este como esté. Se puede asemejar a un corte de tensión controlado. Por otro lado, si queremos realizar un paro controlado, pulsamos el botón de STOP, con el que paramos la entrada de cajas al sistema y se parará cuando la última caja que quede en la cinta EntryConveyor se acabe de procesar.

En el estado de EMERGENCIA, se activará la alarma de la botonera. Cuando acabemos de resolver la incidencia que nos ha hecho pulsar el botón de emergencia, el sistema no se volverá a poner en marcha hasta que se vuelva a pulsar el botón START.

Teniendo claro el funcionamiento del sistema, se desarrolló la programación del Grafcet de control del sistema, el cual lo podemos ver en la *Figura 76*.

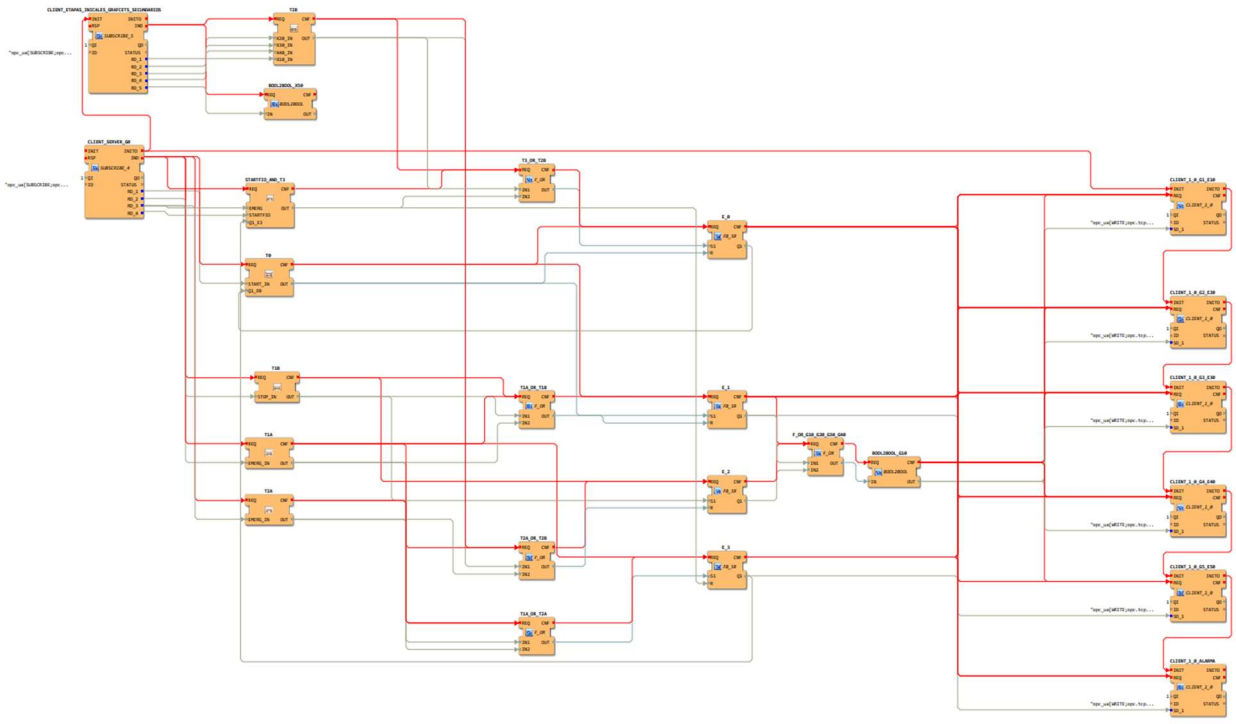

Figura 76:Programación desarrollada para el Grafcet G0 control

A la hora de realizar las pruebas en el sistema nos hemos dado cuenta que con los bloques básicos de funciones que nos proporciona 4diac, no estamos preparados para realizar unas activaciones de Grafcets jerárquicos, por los siguientes motivos.

- Podemos detectar la activación del Grafcet, el problema viene cuando tenemos que hacer la desactivación, porque el dato que utilizamos para activar la etapa inicial del Grafcet lo tendremos a 0, esto hará que no entre la etapa inicial del Grafcet y se quede en la última etapa con el error que conlleva.
- También hemos detectado que los pulsadores, como hay que estar observándolos todo el tiempo para ver si se pulsan o no, carga computacionalmente el runtime y los eventos de 4diac de manera que producen errores de cola de eventos llena. Lo que conlleva un bloqueo del sistema

En el artículo [8] de la bibliografía nos habla de cómo realizar la programación de estos Grafcets jerárquicos en la norma IEC 61499. En este caso utiliza bloques con un evento y un dato controlado que sirve para esta ejecución. A su vez, se

crea el ECC del bloque para tratar el dato y el evento de forma correcta, para que se comporte acorde a las reglas que establecen los diagramas de Grafcet. Por lo tanto, es muy difícil con bloques estándar adecuar este tipo de programación a la IEC61499, porque no podemos modificar el ECC de los bloques de funciones.

Así concluimos que crear Grafcets jerárquicos es prácticamente imposible con los bloques estándar que incluye la norma, pero creando bloques para realizar estas funciones sí que es posible realizarlo. Incluso se podría estudiar la posibilidad de crear un bloque que se encargara de gestionar los forzados y ser portable para todos los sistemas que se puedan desarrollar en esta norma. Aunque no es objeto de este trabajo la creación de bloques de funciones.

5.3. COMPARATIVA DE LAS SOLUCIONES DESARROLLADAS SOBRE LA NORMA IEC 61499

En este apartado realizaremos una comparativa de las soluciones desarrolladas con las metodologías propuestas.

El primer punto por tratar es que la metodología con eventos actualmente no está preparada para poder desarrollar todas las soluciones extrapolando directamente los diagramas de Grafcet que es lo que proponemos en este trabajo. Por el hecho de la falta de bloques de función para eventos para seguir la metodología propuesta.

En segundo lugar, hay que comentar que todos los ejemplos se pueden realizar mediante la programación empírica, pero tenemos el problema que llega un momento en que el volumen de transiciones y etapas que están plasmadas en el ejemplo son tantas que se hace inviable seguir con este tipo de programación porque la cantidad de cables y bloques es tal que no permite una trazabilidad para interpretar el programa. Aunque se realizaran subaplicaciones el volumen de bloques sería tal que tampoco sería viable para desarrollar un programa interpretable por otros programadores.

En tercer lugar, hay que destacar que la metodología con datos ha sido la única que nos ha permitido alcanzar el objetivo del proyecto. Nos permite la flexibilidad necesaria tanto para la realización de las transiciones como las etapas. También podemos destacar que la portabilidad de la estructura del Grafcet es inmediata. Esto facilita la interpretación del programa y la programación que se tiene que realizar, ya que, si el Grafcet desarrollado es correcto, solo hay que cambiar transiciones y etapas por bloques de función que realicen la misma función.

Por último, hay que mencionar que, entre la metodología con datos y eventos, si se desarrollan los bloques de función necesarios para que en las dos opciones se puedan desarrollar todos los ejemplos, puede haber una gran diferencia a la hora de utilizar los bloques con eventos ya que, a nivel computacional, el tráfico de datos es menor porque la información se envía en un mismo “tipo de canal” como es el evento. Aunque algunos bloques tienen que enviar datos, son minoritarios y utilizados para detección de sensores y señales para actuadores.

6. CONCLUSIONES

6.1. OT 1. Adecuación y programación de los diagramas de Grafcet de aplicaciones centralizadas en la norma IEC 61131 a la norma IEC 61499.

En general todos los ejemplos programados han sido factibles mediante la metodología desarrollada. Siguiendo la estructura marcada por los diagramas de Grafcet y respetando las normas marcadas en este proyecto para la creación de la lógica de las transiciones. Por lo tanto, podemos afirmar que se pueden desarrollar la mayoría de los diagramas de Grafcet utilizando la norma IEC 61499.

Como bien hemos comentado en todo el proyecto se puede desarrollar utilizando los bloques denominados como datos.

Con respecto a los bloques por eventos, es posible que en un futuro si se siguen desarrollando bloques para su total portabilidad se pueda aplicar la metodología descrita.

También cabe la posibilidad del desarrollo de una metodología válida solo para los bloques de eventos, pero queda fuera del alcance de este proyecto y su estudio queda para trabajos futuros.

6.2. OT 2. Utilización de protocolos de comunicación aptos para la industria 4.0 y simulación.

Se ha podido comprobar que utilizando el protocolo de comunicación OPC-UA se puede hacer tanto la comunicación de los equipos como la simulación del proyecto. Esto abre la posibilidad de desarrollar gemelos digitales en la industria y tener en tiempo real el proceso bajo supervisión. Con ello el director técnico o incluso los responsables de producción y mantenimiento, con un simple vistazo a la simulación pueden saber cómo se está comportando el sistema y actuar en consecuencia.

El uso del protocolo OPC-UA nos permite tener un servidor en la nube en el que poder comprobar todas las variables que se encuentren en este servidor. Esto es interesante para crear una estructura de trabajo empresarial en que todos los departamentos puedan acceder a los datos del sistema que se les permita y así agilizar los procesos de intercambio

de información entre departamentos, ya que, al tener la información a su alcance, cualquier duda puede ser consultada en ese mismo momento.

Actualmente, en la mayoría de las empresas se utilizan sistemas 4.0 para su gestión, ya sea mediante servicios cloud o con servidores locales para tener un mayor control de los datos que se almacenan en el servidor. Con ello queremos decir que es esencial que la norma permita de una forma fácil realizar este envío de datos al servidor. Como bien hemos comentado, es totalmente factible realizar este envío de datos de forma relativamente sencilla.

6.3. OT 3. Programación de aplicaciones en modo distribuidas adecuadas a la norma IEC 61499.

La norma IEC 61499 fue concebida para el desarrollo de aplicaciones distribuidas, en este caso, hemos demostrado que es viable la programación de este tipo de sistemas industriales. Incluso resulta más fácil la programación de varios Grafcets, uno por dispositivo, que centralizar la aplicación en uno solo. Este tipo de sistemas también proporciona una solución con equipos menos potentes que visto desde el punto de vista económico puede resultar interesante, ya que en muchas instalaciones el equipo de control está muy saturado de información y te obliga a ir a un equipo superior. Este equipo superior puede encarecer el presupuesto y con este nuevo concepto, si separamos el sistema industrial en varios procesos con equipos más económicos, podemos tener la misma solución a un precio más competitivo.

Como hemos comentado en la introducción de este trabajo, una de las ventajas que ofrece la IEC 61499 es la programación online de los sistemas, pudiendo transferir la programación al sistema sin tener que pararlo. Hay que matizar que cuando pasamos el programa al dispositivo de control, machacamos las variables que tiene guardadas en el sistema en local/servidor y puede hacer que nos den fallos. Por lo tanto, aunque es una herramienta que nos agiliza las pruebas sobre los sistemas, se debe realizar de forma controlada para no causar incongruencias en el sistema.

6.4. OT 4. Estudio de la capacidad para adecuar la forma de programar de la norma IEC 61131 a la IEC 61499.

El último objetivo propuesto en el trabajo hace referencia a las dificultades que se puede encontrar un programador con experiencia en la norma IEC61131 cuando tiene que realizar un proyecto en la nueva norma IEC 61499.

Para nuestra sorpresa, hemos comprobado que no hay ninguna dificultad para trasladar los conocimientos que se tienen de la norma IEC61131 a la 61499, salvo la interfaz de usuario del programa y el concepto de evento y dato.

Al desarrollar la metodología a partir de los diagramas de Grafcet se ha facilitado la portabilidad de los conocimientos del programador, esto permite que el programador sólo tenga que plasmar en unos bloques las interpretaciones que se tiene en el Grafcet.

Por lo tanto, en este objetivo podemos concluir que la capacidad de adecuación de programación de una norma a otra es factible y no habrá inconveniente al plantearle a un programador con experiencia sistemas industriales para que los programe con esta nueva norma.

Como conclusión general podemos decir que los objetivos que se habían planteado en el proyecto se han alcanzado con éxito. La norma IEC 61499 tiene una proyección muy amplia para poder implantarse en los sistemas industriales. Con la ayuda de este proyecto entre otros se está dando visión a nivel educativo que la norma tiene la suficiente capacidad para poder abarcar las complejidades que se encuentran en el mundo industrial. A su vez en este proyecto, se ha desarrollado una metodología para facilitar la portabilidad de los diagramas de Grafcet a la norma IEC 61499, con esto favoreceremos la inclusión de los programadores que están acostumbrados a trabajar con la norma por excelencia en la automatización de sistemas industriales como es la IEC 61131, ya que no supone un gran cambio y su adaptabilidad es relativamente rápida.

7. TRABAJOS FUTUROS

Con lo que respecta a trabajos futuros que se pueden desarrollar a partir del trabajo propuesto podemos desarrollar varias, líneas de investigación.

- Desarrollo de bloques de eventos para poder realizar funciones lógicas con los eventos.
- Continuación de la metodología descrita en este trabajo, desarrollando pruebas con diferentes protocolos de comunicación y ejemplos propuestos con características similares a los encontrados en la industria.
- Desarrollo de una metodología para la creación de bloques de función propios para la programación de aplicaciones. En este caso, el objetivo principal sería desarrollar un manual de procedimientos, dónde se recojan las principales características que hay que configurar en los bloques de función para poder tener una funcionalidad completa. Esto permitiría al programador realizar programaciones muy compactas y bloques de funciones completamente portables a otros tipos de sistemas similares, con lo que reduciríamos el tiempo de trabajo de los programadores a la hora de realizar una portabilidad de programación de un sistema a otro.

8. BIBLIOGRAFÍA

[1] Roberto Sanchís, Julio Ariel Romero y Carlos Ariño. *Automatización industrial.* Universitat Jaume I, 2010

[2] INTRODUCCIÓN AL IEC61499, Esteban Querolz 2014

[3] Dr.Alois Zoitl. Recuperado de: *www.iec61499.com*

[4] IEC. IEC 61131 – programmable controllers, part 3: Programming languages, 2013

[5] IEC. IEC 61499: Function Blocks for industrial-process measurement and control systems, 2012

[6] Eclipse 4diac. Eclipse 4diac – the opensource environment for distributed industrial automation and control systems, 2020

[7] Oscar Miguel-Escrig, Julio-Arial Romero Pérez, Bianca Weismayr, and Alois Zoitl. Distributed implementation of Grafcets through IEC 61499. In *2020 25th IEEE Int. Conf. on Emerging Technologies and Factory Automation, 2020.*

[8] Bianca Weismayr, and Alois Zoitl, Oscar Miguel-Escrig, and Julio-Arial Romero Pérez. Distributed implementation of Hierarchical grafcets through IEC 61499. In *2021 26th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA),* pages 1-8, 2021

[9] Esteban Querolz, Julio Ariel Romero, Antonio M.Estruch, and Fernando Romero. Norma IEC 61499 para el control distribuido. Aplicación al CNC. *Acta de las XXXV Jornadas de Automática, 3-5 septiembre de 2014, Valencia,* pages 1-8, 2014

[10] Oscar Miguel-Escrig, Julio-Arial Romero Pérez. Modelos de grafcet y aplicaciones distribuidas en la norma IEC 61499. Un caso de estudio. 2021.

[11] Andrés Tendero Vegas, Oscar Miguel-Escrig, Julio-Arial Romero Pérez. Integración de Factory IO y 4diac-forte para la validación de software de control en la norma IEC61499, pages 1-7, 2022.

[12] John Conway. *IEC 61499:* La norma de automatización industrial para la portabilidad que permite aprovechas las ventajas de la Industria 4.0. Schneider Electric, 2021.

PLIEGO DE CONDICIONES

Las condiciones en la que se enmarca la investigación se pueden especificar en dos apartados distintos.

El primer apartado es el correspondiente a los medios hardware utilizados. En los que se incluyen los equipos informáticos utilizados.

Estos equipos están formados por los siguientes elementos:

- AMD Ryzen 5 1400 Quad-Core 3.20GHz
- 8 GB de RAM
- Windows 10 Pro
- NVIDIA GeForce GTX 1050 Ti
- Monitor AOC G2490VXA

El segundo apartado está compuesto por las versiones del software utilizado. Estos son los siguientes:

- Factory io es con la versión OPC-UA & MODBUS
- UAExpert version 1.2
- Paquete de Microsoft Office
- 4DIAC en su version 2.0.1
- Runtime Forte para Windows con OPC-UA

Hay que indicar que todos los softwares vienen configurados para su uso salvo el runtime de Forte, ya que es un software programable para distintos usos y por lo tanto hay que configurarlo y compilarlo según necesidades. En este proyecto *forte* fue configurado por una entidad externa, la cual nos proporcionó el ejecutable para volcar la programación desarrollada en 4DIAC.

PRESUPUESTO

Los costes del proyecto se detallan en la siguiente tabla.

	UNIDADES	€/u	TOTAL
PROYECTO			**15395**
CAPITULO 1 : EQUIPOS INFORMÁTICOS			**3000**
PC	2	1500	3000
CAPITULO 2: LICENCIAS SOFTWARE			**395**
FACTORY IO	1	395	395
CAPITULO 3: RECURSOS HUMANOS			**12000**
ESPECIALISTA DOCTOR	60	50	3000
ESPECILISTA TÉCNICO	300	30	9000

Tabla 2: Presupuesto del proyecto

Las especificaciones de los equipos informáticos y softwares utilizados son las mencionadas en el apartado *Pliego de condiciones.*

Por último, los recursos humanos utilizados se calculan en base a cinco días laborables a la semana con una media de cuatro horas de dedicación al proyecto cada día.

Con todo esto el presupuesto final del proyecto es de **quince mil tres cientos noventa y cinco euros.**