

Proceso Batch Continuo IIoT

Carlos Alidio Medina López
Cod: 20191383031

Resumen – El presente documento, es un informe de la programación realizada con Codesys, y la simulación de un PLC para hacer la prueba de conexión MQTT y OPC UA, además de aprender a usar la norma ISA88 para todos los procesos, cuyos procesos dependan de una receta. Se realiza la simulación de una planta de producción químicas, siguiendo los paso a paso de las recetas. Finalmente, mediante la comunicación OPC UA, y mediante condigo de *JavaScript* se permite la comunicación a un navegador Web mediante una IP local.

I. PROYECTO PROCESO BATCH

A. *Proceso Batch*

Para la ejecución del proyecto, se eligió el proceso de un químico, el proceso fue tomado de una planta de producción de una empresa localizada en Bogotá, Colombia. El proceso es usado para un proyecto de grado, el cual se piensa implementar transformación digital. El contrato de confidencialidad no permite ingresar los datos correctamente y no fue posible obtener los diagramas PFD y PI&D. El proceso se divide en cuatro etapas, y la receta es la siguiente:

CARGA DE MATERIAS PRIMAS

1. Verificar que el chiller este encendido y recirculando por el condensador del reactor, temperatura entre -5 y -9°C.
2. A temperatura ambiente cargar el ácido y el Varsol, el Varsol es condensado de la cochada anterior, y complementar con Varsol nuevo.
3. Colocar agitación al 100% por 5 minutos.
4. Con agitación 100% cargar el metal en un lapso de 1 hora.
5. Cerra el pozo y dosificar el agua lentamente en un plazo de media hora.
6. Iniciar la inyección de aire con el soplador a un flujo de 60HZ y a una temperatura entre 62 y 65°C.

REACCION

1. Encender el enfriador si la temperatura excede 65°C. Evitar que supere 70°C.
2. Sin suspender la inyección y desde el inicio de la reacción, realizar el retorno del Varsol y del agua.
3. Después de 16 horas de reacción, adicionar la sustancia y esperar 1.5 horas.
4. Mantener la reacción a 65°C hasta cumplir las 24 horas desde el inicio de la reacción.

SECADO

1. Temperatura a 85 por 20 minutos y la velocidad del agitador a 75%.
2. Temperatura a 92 por 20 minutos.
3. Temperatura a 95 por 40 minutos.
4. Temperatura a 120 por 30 minutos.
5. Recoger el agua destilada.

ENFRIAMIENTO

1. Enfriar el producto mínimo 80°C.
 2. Recircular por el filtro durante 30 minutos.
-

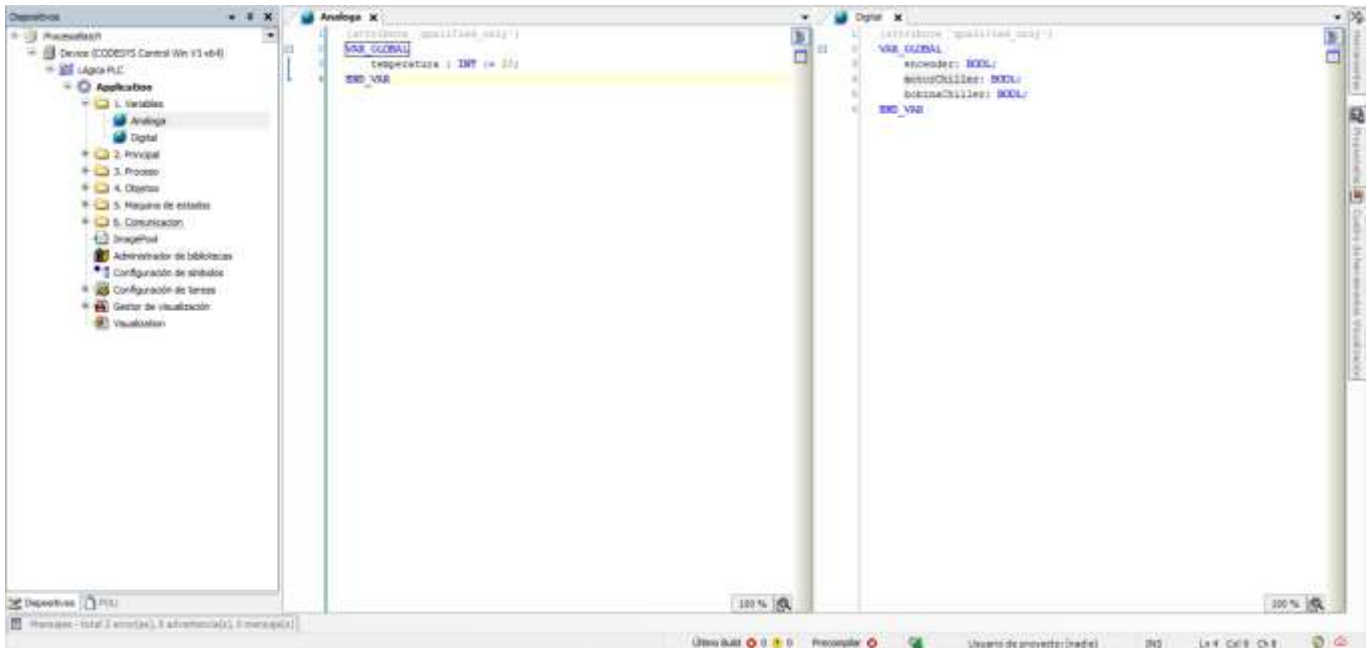
B. Modelo ISA88

Para la realización del proyecto, se dividió en varias secciones, donde tenemos una máquina de estados, los procesos de la planta, las válvulas y actuadores en la planta, como se observa en la siguiente imagen, a continuación, se explicada cada sección en la programación de Codesys.



1. Variables

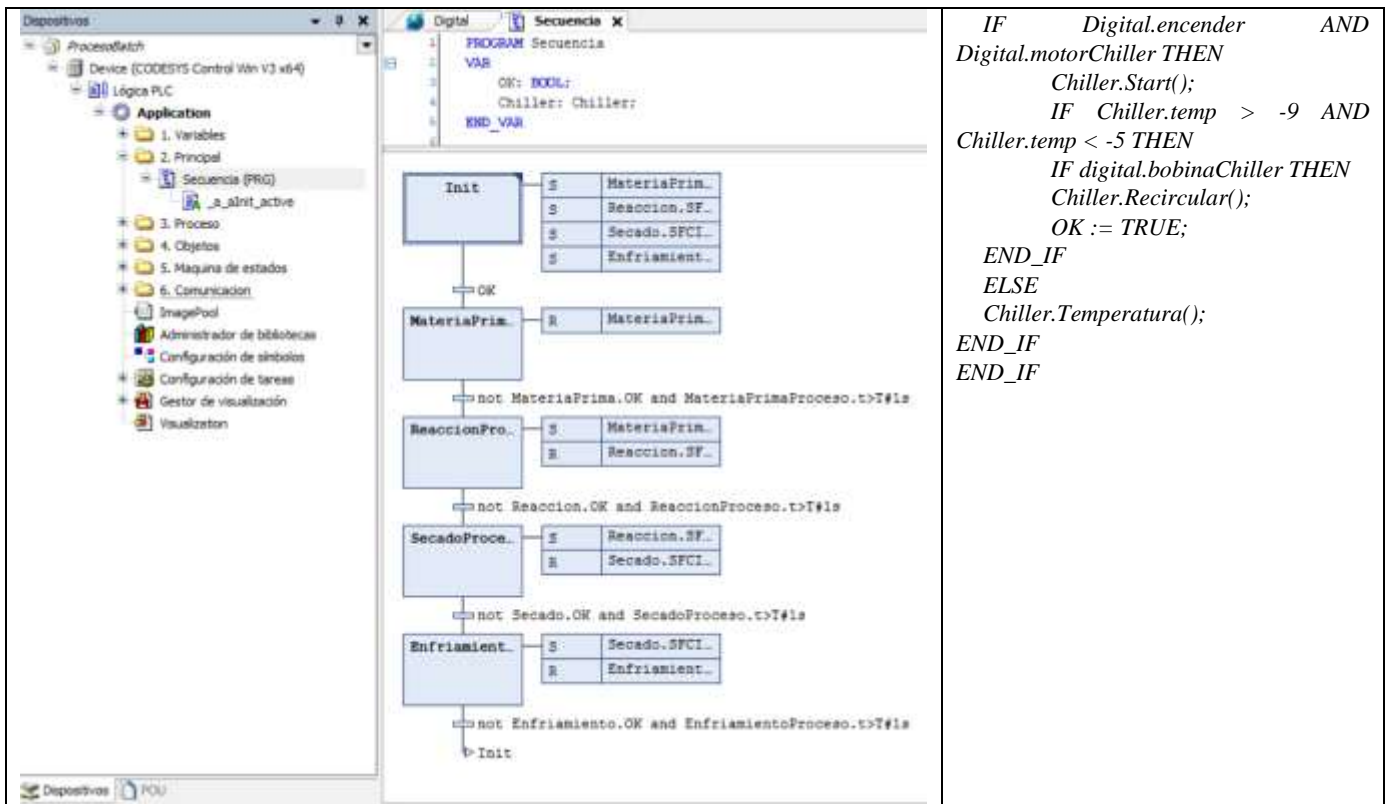
En esta sección se almacenan todas las variables del proyecto, y que funcionen en cualquier otro POU, llamadas variables globales.



Hasta el momento, el proyecto posee una variables analoga global, la cual se asocia a la temperatura del recator; y tres variables digitales globales, las cuales se socian al pulsador de encendido, y la activacion del motor de chiller y la bobina para la circulacion del aire.

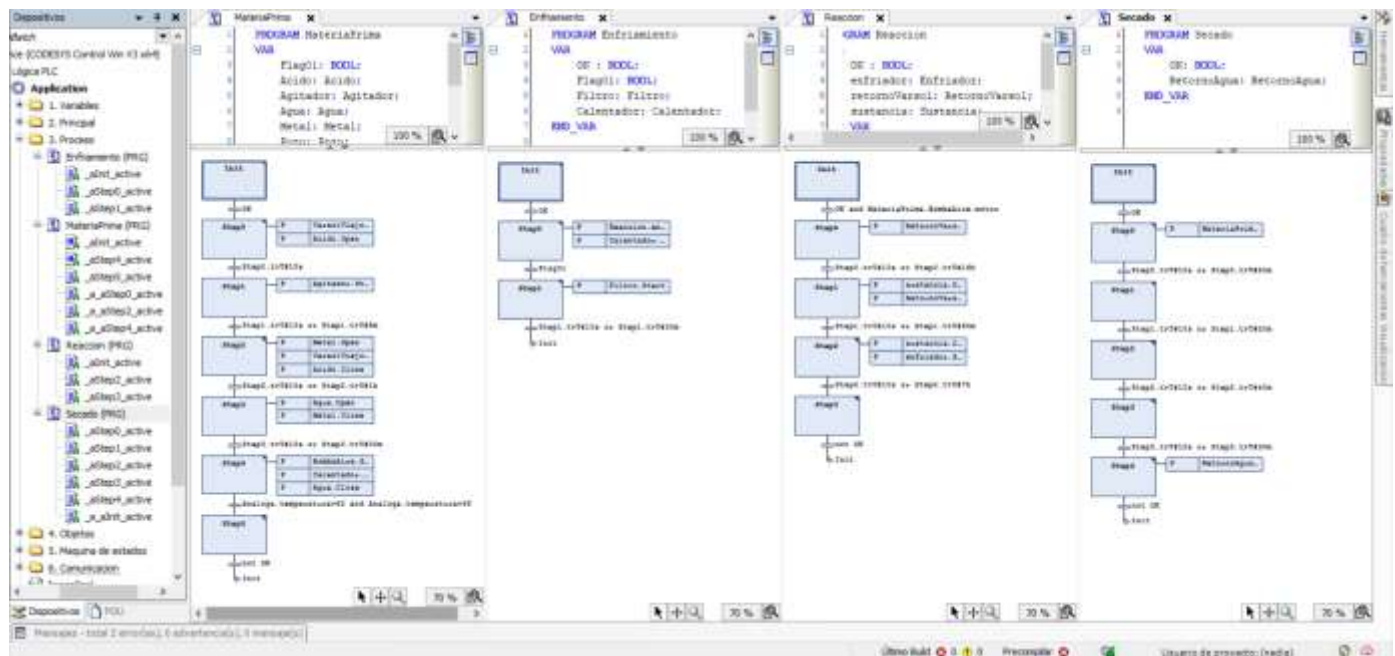
2. Principal

En esta sección se almacena el código principal, el cual ejecutará el proceso de la célula, ejecutando los otros programas de cada proceso de la planta. El programa principal se realizó en Grafset, para lograr una correcta ejecución de la secuencia y visualizar a la hora de simular el proyecto. En la secuencia se logra bloquear los programas que no se usen, ejecutando únicamente el proceso que se requiera, si tener otro proceso corriendo en paralelo.



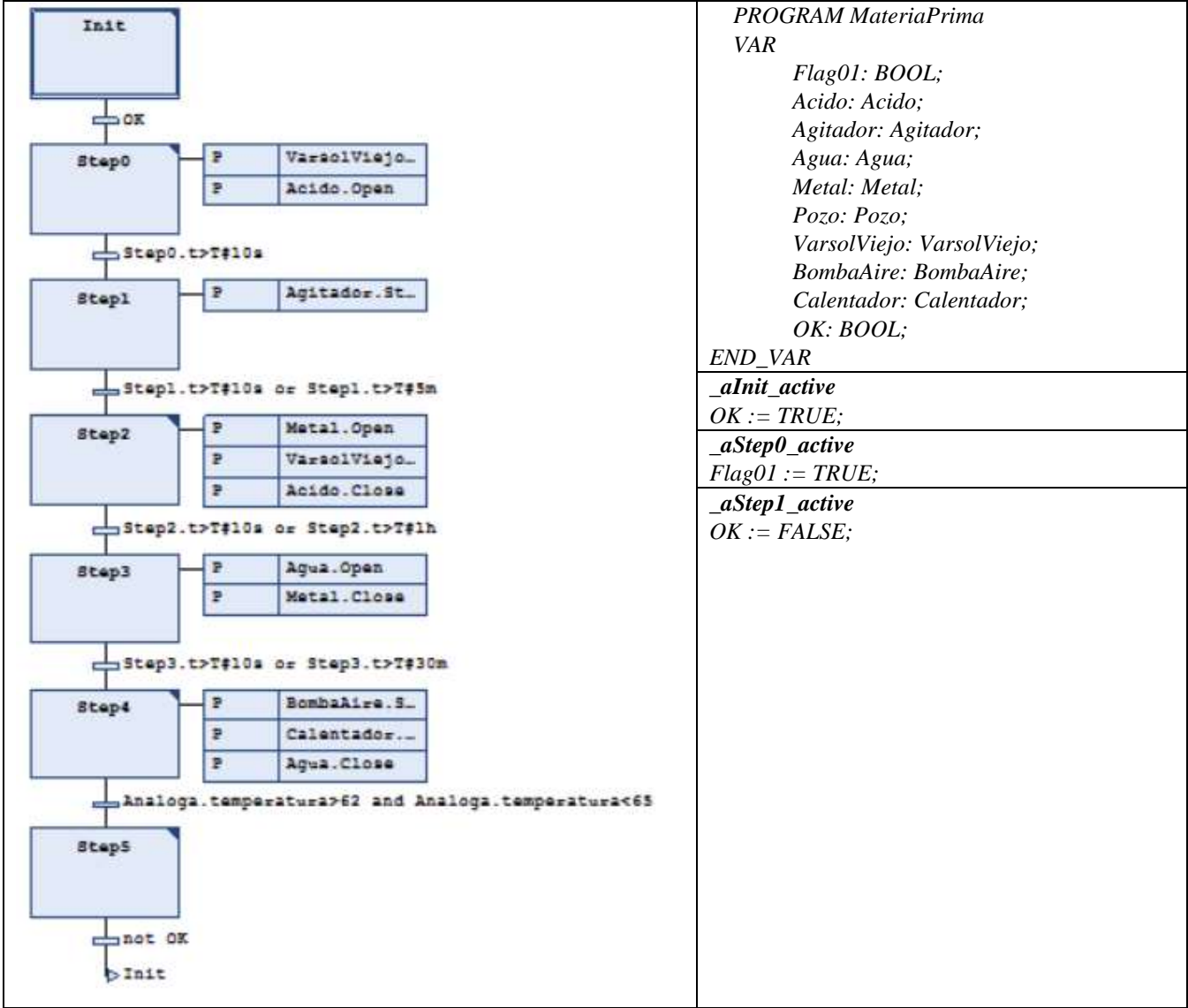
3. Proceso

En esta sección se almacenan los códigos secundarios, los cuales son usados en la sección principal, y cada programa corresponde a un proceso en la planta. En base a la receta dada por el proceso batch, se implementó un código en Grafset para Materia prima, Reaccion, Secado y Enfriamiento. Cada uno tiene una secuencia diferente con respecto a la receta.



A. Materia prima.

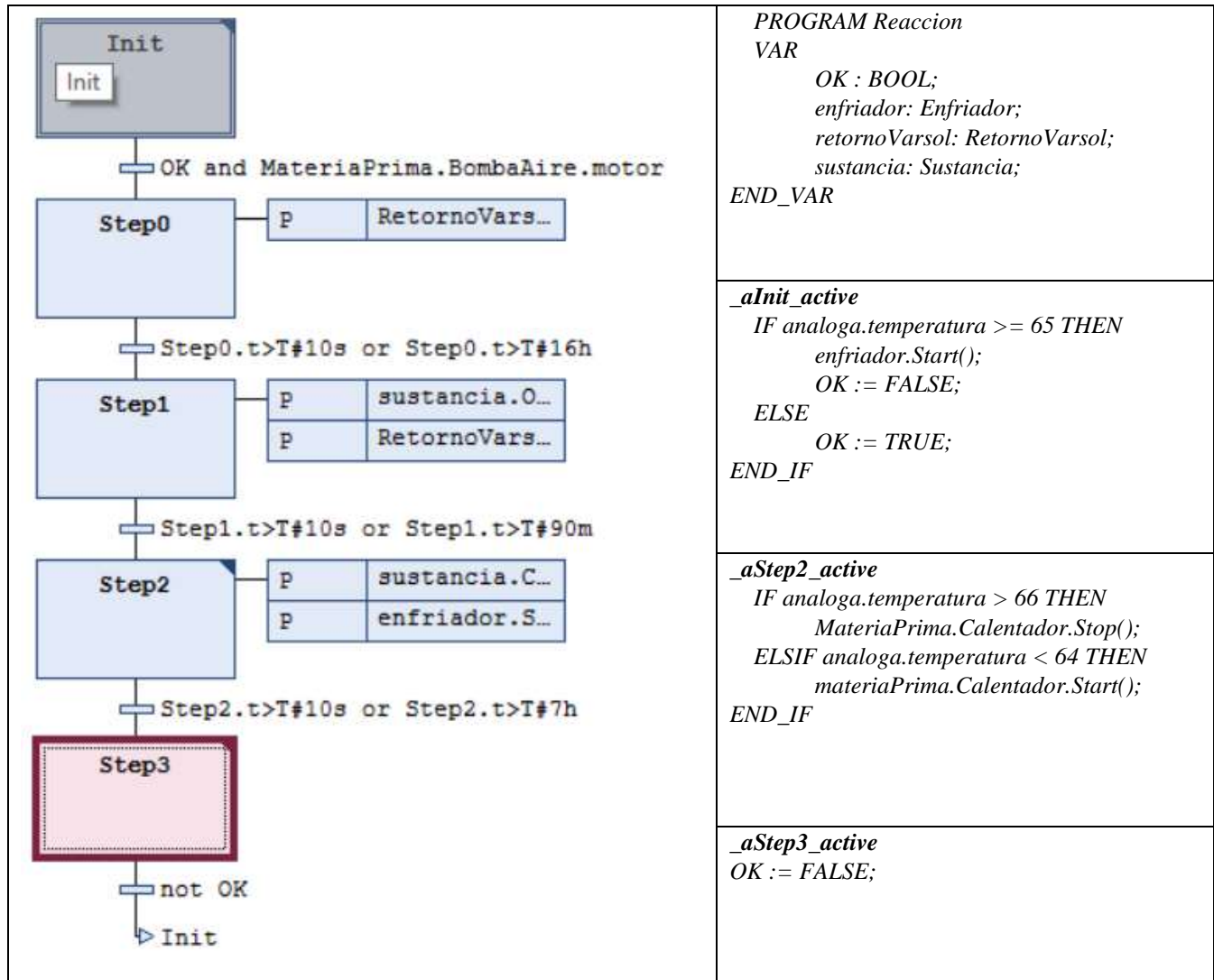
En eta primera etapa, se agregan los insumos necesarios para iniciar el proceso. Comenzamos activando una bandera *OK* para dar inicio a la secuencia, para luego activar las valvulas de *Varsol* y *Acido*. Y Seguidamente se activa el *Agitador* del reactor, esto debe durar 5 minutos. Pasado el tiempo se descactiva las valvyulas de *Varsol* y *Acido*, y se activa la valvula del *Metal*, durante 1 hora. El ultimo ingrediente a agregar meidnate la valvula del *Agua*, y se desactiva la valvula de *Metal*, durante 30 minutos. Y por ultimo se cierra la valvula de *Agua* y se activa la *Bomba de Aire* y el *Calentador* hasta llegar a un tempartura entre 62°C y 65°C en el reactor.



B. Reaccion

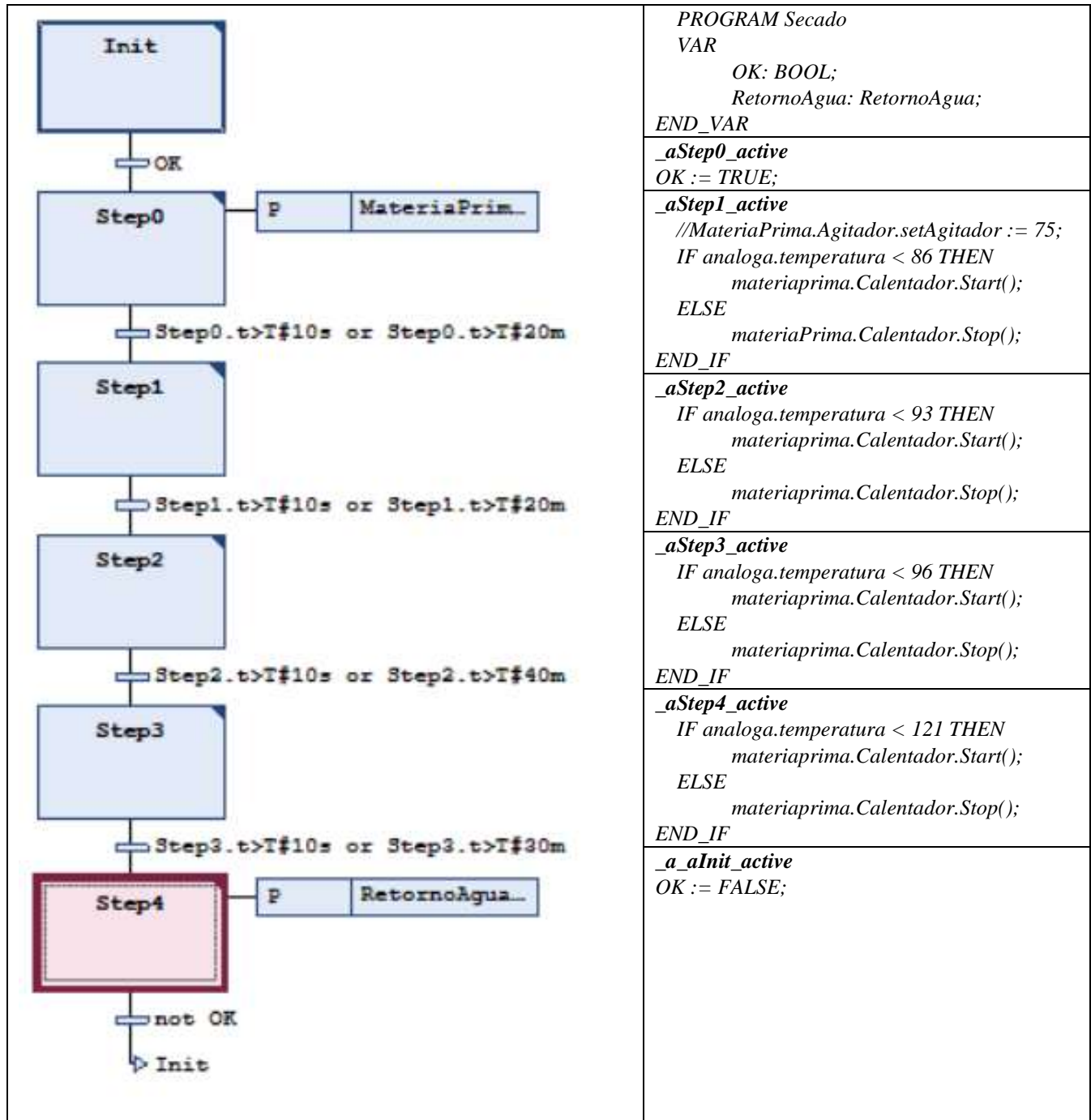
Finaliza la carga de las materias primas, se procede a hacer la reaccion del producto. Tenemos la primera condicion, y en el caso de que la temperatura del reactor suba de 65°C, se debe activar el enfriador, si la temperatura llega hasta 70°C puede ser muy peligroso.

El procedimiento inicia con la activacion del *Retorno del varsol*, al tanque de varsol viejo, este proceso dura entre 16 horas. Continuamos con la activacion de la *Sustancia* y ademas desactivando el *Retorno del varsol*, durante 7 horas. Y por ultimo se desactiva la valvula de *Sustancia* y por cuestion de emergencia, se desactiva el *Enfriador* por si en algun momento del proceso se activo. Todo el proceso debe realizarse con una temperatura de 65°C.



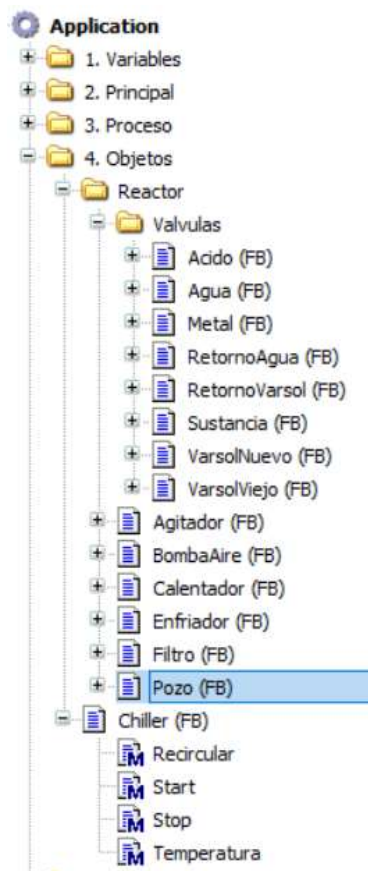
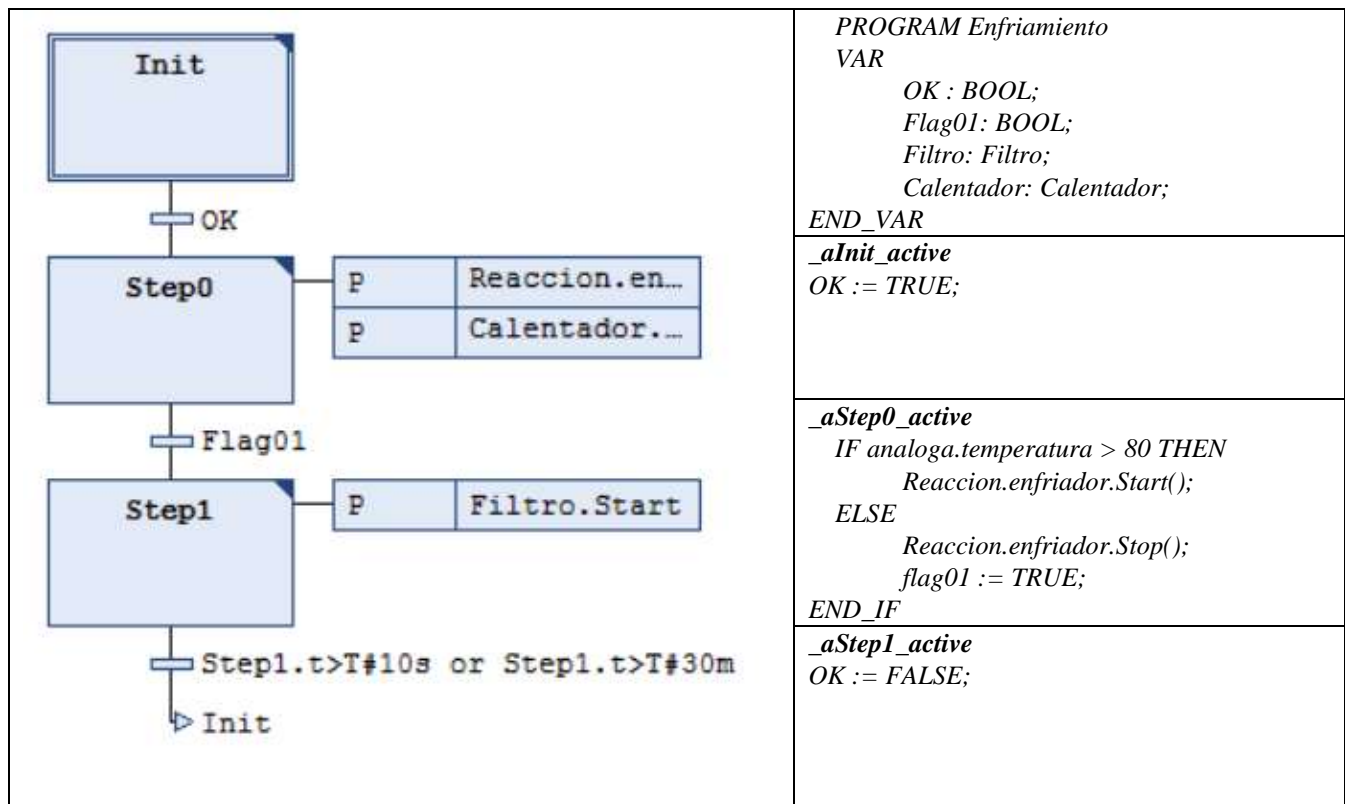
C. Secado

En el proceso de secado, se procese a aumentar la temperatura, para condensar el agua que todavia exsita en el reactor. Inicialmente se aumenta la temperatura hasta 85°C y el Agitador se regula a un 75% de su velocidad, durante 20 minutos, luego se aumenta la temperatura hasta 92°C durante 20 minutos, luego se aumenta la temperatura hasta 95°C durante 40 minutos y por ultimos se aumenta la temperatura hasta 120°C durante 30 minutos. Para finalizar con la activacion de la valvula para Retornar el agua condensada.



D. Enfriamiento

La ultima etapa es enfriar el producto. Se apaga el *Calentador* y se enciende el *Enfriador* hasta llegar a una temperatura de 80°C durante 30 minutos, para finalmente pasar el producto por el filtro.



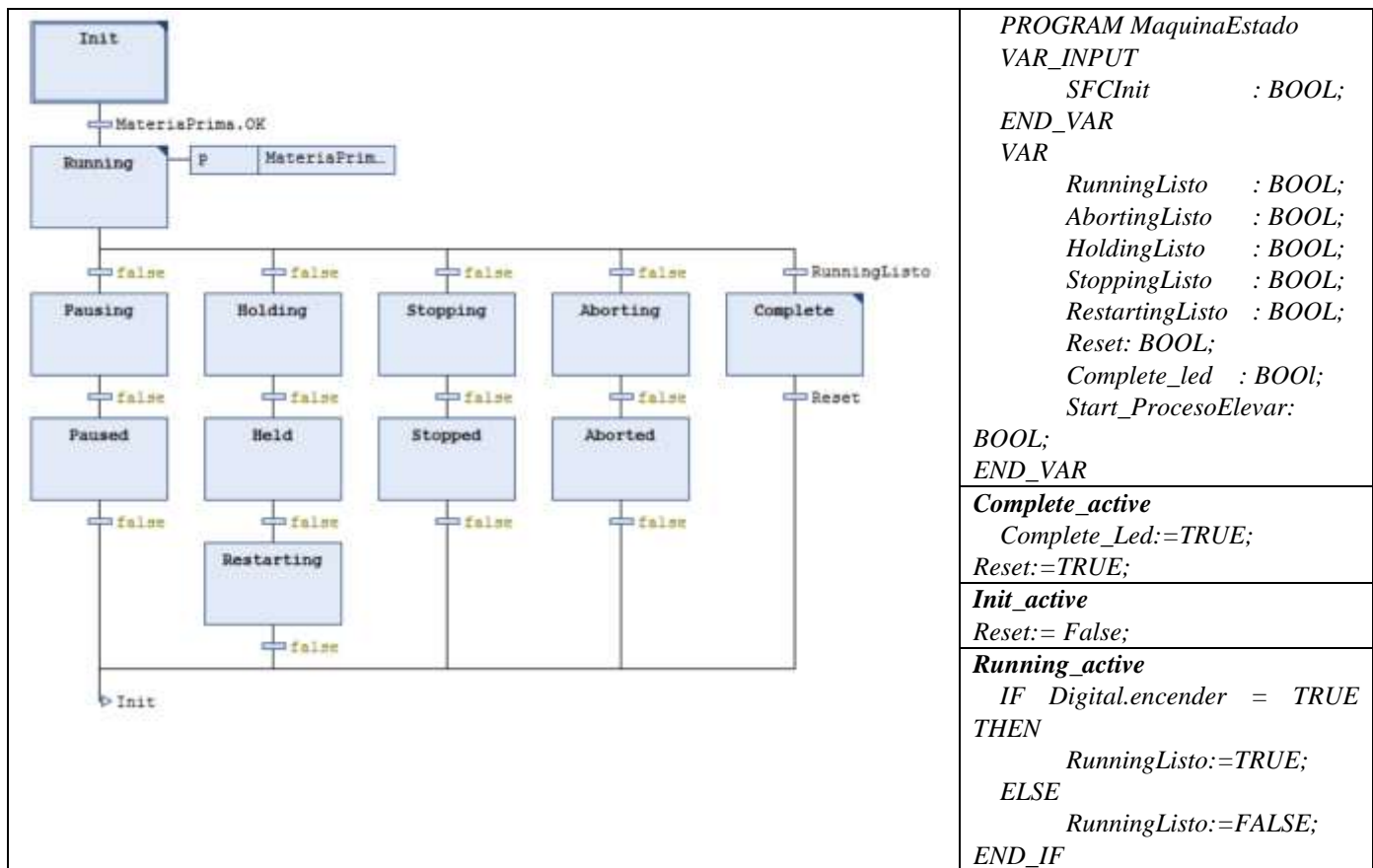
4. Objetos

En esta seccion se almacena los actuadores que trabajan en la planta, en el reactor se asociado las valvulas de *Acido*, *Agua*, *Metal*, *Retorno de Agua*, *Restorno de Varsol*, *Sustancia*, *Varsol Nuevo* y *Varsol Viejo*; ademas del *Agitador*, *Bomba de Aire*, *Calentador*, *Enfriador*, *Filtro*, *Pozo*. Ademas del reactor, se anexa el *Chiller*. Todos los objetos antes mencionados se usan en los procesos.

Cada objeto tiene asociado un metodo, los cuales cumplen cierta funcion, en el caso de las valvulas, solo tiene dos metodos: Abrir y cerrar. En otros caso hay otros metodos asociados los cuales puedan variar la velocidad o un set de temperatura.

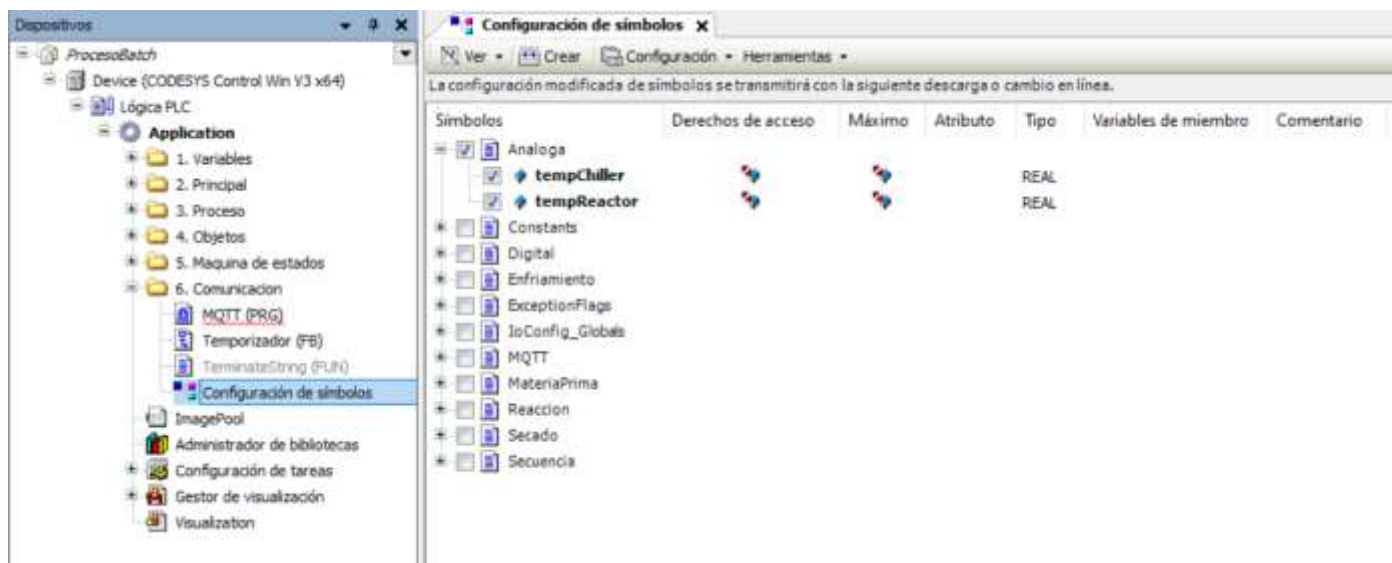
5. Maquina de estados

Maquina de estado donde se ejecuta el tiempo de ejecucion del proceso, recomendado por la noma ISA88.



C. Conexion OPC UA

Codesys tiene disponible en sus herramientas, librerías para realizar la conexión mediante el protocolo OPC UA. Para ello se usa el POU *Configuración de símbolos*, que permite crear una conexión y elegir las variables que se desean comunicar. Codesys tiene asignado el puerto 4840 para la conexión por OPC UA, con Node Red se conectó por medio de TCP IP.



Para verificar el envío de los datos por OPCUA, se usa el software *Unified Automation UaExpert*, el cual nos permite verificar el endpoint de conexión con el servidor, y los nodos disponibles en el servidor. *UaExpert* no realiza una búsqueda de todos los

servidores conector al ordenador, en este caso requerimos de los servidores locales. En la conexión podemos observar el endpoint, el cual es el punto de enlace al servidor, en el caso del proyecto, con *Codesys*, el endpoint es *opc.tcp://localhost:4840*.

Con la ayuda de *UaExpert*, logamos encontrar los valores de los nodos, para permitir la conexión y la recopilación de los datos. Como se puede observar, desde *Codesys* se usa la variable *tempChiller*, la cual corresponde a la temperatura en el Chiller, y la variable *tempReactor*, la cual corresponde a la temperatura en el interior del reactor. En *UaExpert* el nodo relacionado con la temperatura del chiller, se llama *ns=4;s=/var/CODESYS Control Win V3 x64.Application.Analoga.tempChiller* y el nodo que corresponde a la temperatura del reactor se llama *ns=4;s=/var/CODESYS Control Win V3 x64.Application.Analoga.tempReactor*.

Server Settings - OPCUAServer@Kroloz-Asus

Configuration

Configuration Name:

Server Information

Endpoint Url:

Reverse Connect: ☐

Security Settings

Security Policy:

Message Security Mode:

Authentication Settings

☒ Anonymous

☐ Username: ☐ Store

☐ Password:

☐ Certificate: ...

☐ Private Key: ...

Session Settings

Session Name:

OK Cancel

Address Space

No Highlight

- Root
 - Objects
 - DeviceSet
 - Server
 - Auditing
 - CODESYS Control Win V3 x64
 - Resources
 - Application
 - DeviceManual
 - DeviceRevision
 - GlobalVars
 - Analoga
 - tempChiller
 - tempReactor
 - HardwareRevision
 - Manufacturer
 - Model

Attributes

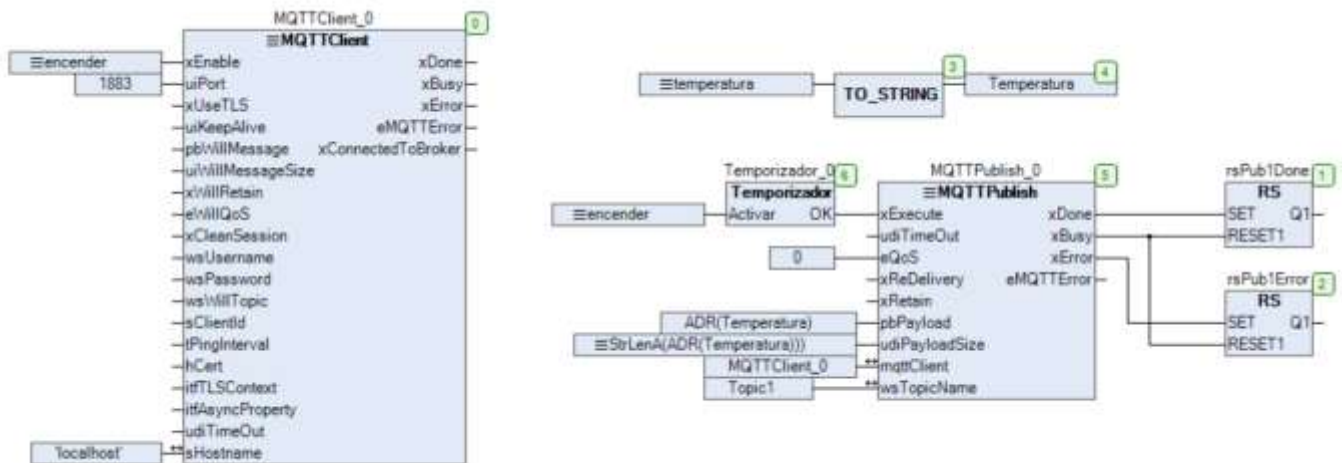
Attribute	Value
NodeId	ns=4;s=/var/CODESYS Control Win V3 x64.Application.Analoga.tempChiller
NamespaceIndex	4
IdentifierType	String
Identifier	/var/CODESYS Control Win V3 x64.Application.Analoga.tempChiller
NodeClass	Variable
BrowseName	4, "tempChiller"
DisplayName	"en-Us", "tempChiller"
Description	BadAttributeIdInvalid (0x803)
WriteMask	BadAttributeIdInvalid (0x803)
UserWriteMask	BadAttributeIdInvalid (0x803)
RolePermissions	BadAttributeIdInvalid (0x803)

Attributes

Attribute	Value
NodeId	ns=4;s=/var/CODESYS Control Win V3 x64.Application.Analoga.tempReactor
NamespaceIndex	4
IdentifierType	String
Identifier	/var/CODESYS Control Win V3 x64.Application.Analoga.tempReactor
NodeClass	Variable
BrowseName	4, "tempReactor"
DisplayName	"en-Us", "tempReactor"
Description	BadAttributeIdInvalid (0x803)
WriteMask	BadAttributeIdInvalid (0x803)
UserWriteMask	BadAttributeIdInvalid (0x803)
RolePermissions	BadAttributeIdInvalid (0x803)

D. Conexión MQTT

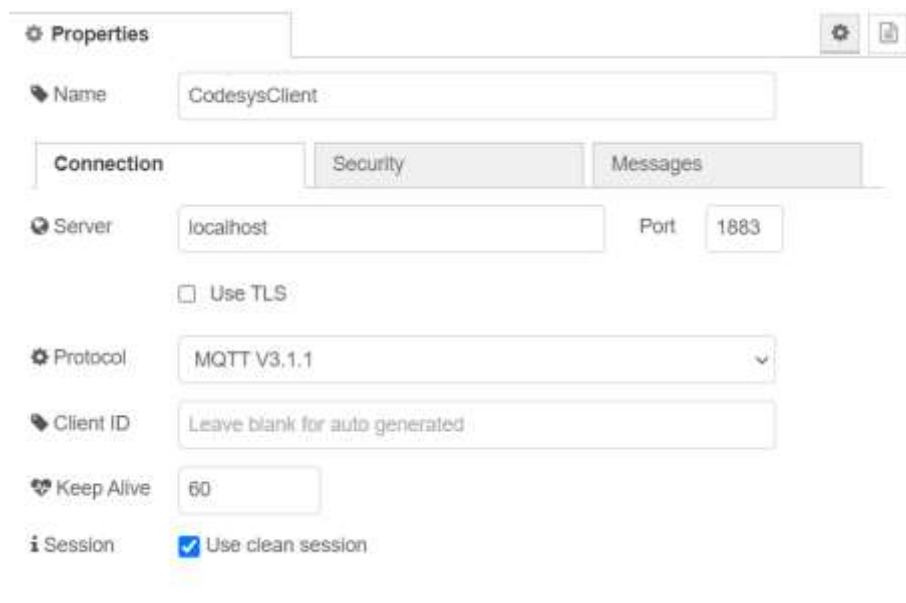
Para la conexión por medio del protocolo MQTT, se instala la librería IIOT, que tiene el código necesario para usar el protocolo MQTT, inicialmente, se crea un bloque de MQTT_Client, permitiendo la conexión por medio de un Puerto y un HostName. Para el proyecto se usa el puerto 1883 y el HostName: localhost o 127.0.0.1, igual que en el ejemplo anterior, la IP es la misma.



La comunicación MQTT requiere para el envío y la recepción de la información de Publicar, para enviar información, y Suscribirse para recibir información. Para el proyecto se realizó únicamente la publicación de la información. Se creó un tópico con el nombre de Temperatura, este nombre es clave, ya que si se quiere obtener esa información exacta, el tópico debe ser igual en todas las plataformas.



En codesys se genera un temporizador de 2 segundos, para el envío de la información. En Node Red, se crea una conexión a un servidor, en este caso localhost, con el puerto 1883 como se había dicho anteriormente.



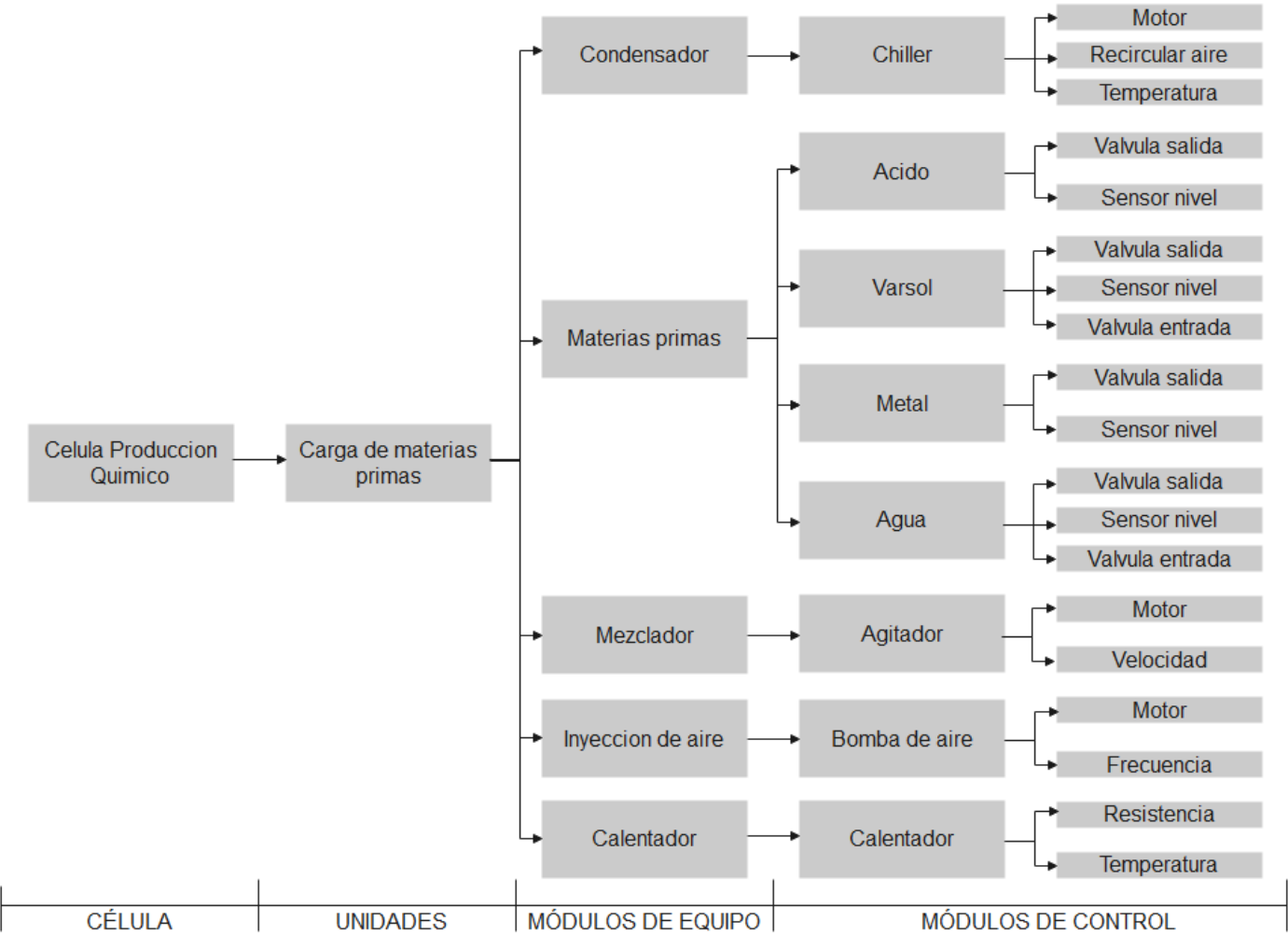
E. HMI

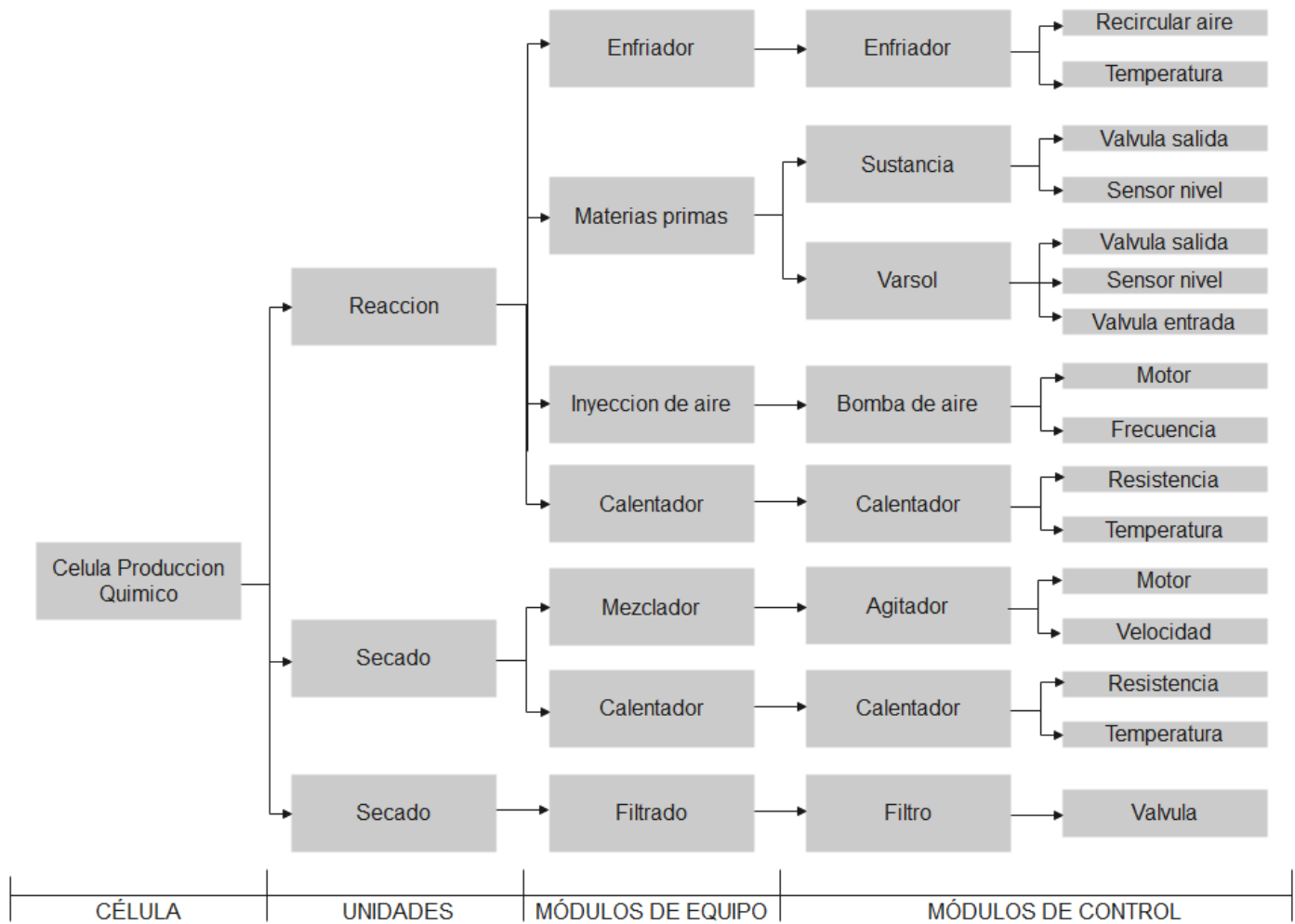
Para observar la conexión entre los objetos y la programación realizada por *Codesys*, se realiza una interfaz humano maquina (HMI) para observar el proceso.



II. DIEGRAMAS

En la explicación del proyecto de proceso batch, se nombro la norma ISA88, en este apartado usaremos un diagrama donde se puede observar claramente el orden de los objetos y proceso en el proyecto. El diagrama se divide en niveles, los cuales son, el nivel de la célula, las unidades, los módulos de equipos y los módulos de control, donde en los módulos de control encontramos los objetos usados en el proyecto y en las unidades, las diferentes etapas del proceso completo.





III. BACKEND

OPCUA y MQTT son protocolos de mayor importancia para la industria 4.0, nos permite la conexión con servidores, servicios en la nube y dispositivos IOT. Todos los códigos que corre bajo un servidor, o una pagina web, que no involucre la interfaz de usuario, se llama Backend, de los lenguajes mas comunes es *JavaScript*, el cual permite la conexión por varios protocolos entre el PLC y el FrontEnd, servicios en la nube o servidores NoSQL.

Mediante el protocolo OPCUA, implementaremos un código *JavaScript* mediante la conexión del EndPoint y los nodos, la información se obtuvo mediante el software *UaExpert* anteriormente.

EndPoint	opc.tcp://localhost:4840
Nodo 1	ns=4;s= var CODESYS Control Win V3 x64.Application.Analoga.tempChiller
Nodo 2	ns=4;s= var CODESYS Control Win V3 x64.Application.Analoga.tempReactor

El propósito del código, es recoger los datos de *Codesys* y mostrar en graficas en un *HTML* mediante librerías de gráficas, además de importar los datos obtenido a *MongoDB*. Inicialmente importamos la librería útiles para cumplir el propósito del código. El archivo se llama *package.json*.

```
{
  "name": "app1",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
```

```

    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "chalk": "^4.1.2",
    "express": "^4.17.1",
    "mongodb": "^4.1.3",
    "node-opcua": "^2.50.0",
    "socket.io": "^4.2.0"
  }
}

```

A continuación, implementamos un código en *HTML* para configurar la visualización en el navegador y la importación de las librerías de gráficos y los códigos de *JavaScript* a usar, este archivo es el principal para lograr observar todos el código realizado en el BackEnd. El archivo se llama *index.html*.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ProcesoBatch</title>
  <script src="javascript/libraries/RGraph.common.core.js"></script>
  <script src="javascript/libraries/RGraph.line.js"></script>
  <script src='/socket.io/socket.io.js'></script>
  <script src="javascript/libraries/RGraph.common.dynamic.js"></script>
  <script src="javascript/libraries/RGraph.gauge.js"></script>
</head>
<body>
  <h1>Produccion Quimico</h1>
  <h2>Temperatura reactor</h2>
  <canvas id="cvs_line" width="950" height="400">
    [No canvas support]
  </canvas>

  <h2>Temperatura chiller</h2>
  <canvas id="cvs_gauge" width="300" height="150">
    [No canvas support]
  </canvas>

  <script src="graph.js"></script>
</body>
</html>

```

Ahora, se escribirá el código importante, se encarga de los eventos para la sincronización entre los datos obtenidos por *Codesys* y la visualización o con *MongoDB*. El archivo se llama *app.js*.

```

/*#####

```



```

Importacion de modulo, debes estar previamente instalados
#####*/
const express = require("express");
const {cyan, bgRed, yellow} = require("chalk");
const listen = require("socket.io");
const MongoClient = require('mongodb').MongoClient;
const {AttributeIds, OPCUAClient, TimestampsToReturn} = require("node-opcua");
//const mqtt = requiere

/*#####
Creacion de constantes para la comunicacion y la DB
#####*/
//OPCUA

const endpointURL = "opc.tcp://Krlolz-Asus:4840";
const nodeIDToMonitor1 = "ns=4;s=|var|CODESYS Control Win V3 x64.Application.Analoga.tempReactor";
const nodeIDToMonitor2 = "ns=4;s=|var|CODESYS Control Win V3 x64.Application.Analoga.tempChiller";

//Aplicacion WEB

const port = 3700;

//Mongo DB

const uri =
"mongodb+srv://KrlolzMedina:Krlolz1216@procesobatch.hubhv.mongodb.net/myFirstDatabase?retryWrites=true&w=majority";
const clientmongo = new MongoClient(uri, {useNewUrlParser: true});

/*#####
El codigo principal va en la funcion async
#####*/

(async() => {    //await
  try{
    //Crear el cliente OPCUA
    const client = OPCUAClient.create();

    //Avisar cuando se esta intentado reconectar
    client.on("backoff", (retry, delay) => {
      console.log("Retrying to connect to ", endpointURL, " attemp ", retry);
    });

    //Mostrar las URL cuando logre conectar
    console.log(" connecting to ", cyan(endpointURL));
    await client.connect(endpointURL);
    console.log(" connected to ", cyan(endpointURL));

    //Iniciar la sesion para interactuar con el servidor MQTT

```

```

const session = await client.createSession();
console.log("Sesion iniciada", yellow);

//Crear una suscripcion
const subscription = await session.createSubscription2({
  requestedPublishingInterval: 200,
  requestedMaxKeepAliveCount: 20,
  publishingEnabled: true,
});

//-----
//Se incia monitoreo de la variable del servidor MQTT
//-----

//Crear el item con su NodeID y atributo
const itemToMonitor1 = {
  nodeId: nodeIDToMonitor1,    //Variable a monitorear
  attributeId: AttributeIds.Value
};

const itemToMonitor2 = {
  nodeId: nodeIDToMonitor2,    //Variable a monitorear
  attributeId: AttributeIds.Value
};

//Definir los parametros de monitoreo
const parameters = {
  samplingInterval: 50,    //Tiempo de muestreo
  discardOldest: true,
  queueSize: 100
};

//Crear el objeto de monitoreo
const monitoredItem1 = await subscription.monitor(itemToMonitor1, parameters, TimestampsToReturn.Both);
const monitoredItem2 = await subscription.monitor(itemToMonitor2, parameters, TimestampsToReturn.Both);

//-----
//Crear la aplicacion WEB
//-----

//Crear la aplicacion
const app = express();
app.set("view engine", "html");

//Definir el directorio de estaticos
app.use(express.static(__dirname + '/'));    //Definir el directorio de estaticos
app.set('views', __dirname + '/');

```

```

//Definir como se responde cuando el navegador solicita entrar
app.get("/", function(req, res){
    res.render('index.html'); //Aqui se llama la pagina HTML que se va a utilizar
});

//-----
//Se crea un objeto listen para enviar los datos a la aplicacion WEB
//io.socket --> "real-time bidirectional event-based communication"
//-----

//Asociar el puerto a la app WEB
const io = listen(app.listen(port));

//Esperar la conexion
io.sockets.on('connection', function (socket){

});

//Mostrar el URL para entrar a la aplicacion WEB
console.log("Listening on port " + port);
console.log("visit http://localhost:" + port);

//-----
//Conexion a la base de datos
//-----

//Conectar el cliente
await clientmongo.connect();

//Conectarse a la coleccion con los datos del mongoDB atlas
const collection = clientmongo.db("mydb").collection("mycollection");

//-----
//Definimos que hacer cuando la variable monitoreada "cambie"
//-----

monitoredItem1.on("changed", (dataValue) => {
    //Escribir en la base de datos
    collection.insertOne({
        valor: dataValue.value.value,
        time: dataValue.serverTimestamp
    });

    io.sockets.emit("reactor", {
        //El mensaje contiene:
        value: dataValue.value.value, //Valor de la variable
        timestamp: dataValue.serverTimestamp, //Tiempo
        //nodeId: nodeIdToMonitor, //NodeID del nodo OPCUA
    });
});

```

```

        //browseName: "Nombre" //Nombre de busqueda
    });
});

monitoredItem2.on("changed", (dataValue) => {
    io.sockets.emit("chiller", {
        //El mensaje contiene:
        value: dataValue.value.value, //Valor de la variable
        timestamp: dataValue.serverTimestamp, //Tiempo
        //nodeId: nodeIDToMonitor, //NodeID del nodo OPCUA
        //browseName: "Nombre" //Nombre de busqueda
    });
});

//-----
//Salir al presionar CTRL+C
//-----

let running = true;
process.on("SIGINT", async() => {
    if (!running){
        return; //Avoid calling shutdown twice
    }
    console.log("Shutting down client");
    running = false;
    await clientmongo.close();
    await subscription.terminate();
    await session.close();
    await client.disconnect();
    console.log("Done");
    process.exit(0);
});
}
catch(err){
    //-----
    //Aquí ponemos que pasa si al intentar lo anterior, hay un error
    //-----

    console.log(bgRed.white("Error" + err.message));
    console.log(err);
    process.exit(-1);
}
})(); //La funcion se estara ejecutando

```

Finalmente, se crea un archivo, cuyo código usa los datos obtenidos de *app.js* y mostrar la información en graficas en el *HTML*. Se usan las librerías de *RGraph*. El archivo se llama *graph.js*.

```

/*-----
Creacion de variables para las graficas

```

```

-----*/

//Variables para los objetos de graficas
let Pline = null;
let Gauge1 = null;

//Variables para los datos
let data_line = [];

//Obtener los canvas
canvas1 = document.getElementById("cvs_line");

const numvalues = 1000;
for(let i=0; i<numvalues; ++i){
    data_line.push(null);
}

/*-----
Se utiliza la funcion onload para crear o inicializar
las graficas cuando se carga la pagina
-----*/
window.onload = function(){
    //Parametrizar la grafica
    Pline = new RGraph.Line({
        id: 'cvs_line',
        data: data_line,
        options: {
            //xaxisLabels: ['Aug 13, 2012','Sep 9 2013','Oct 6 2014'],
            marginLeft: 75,
            marginRight: 55,
            filled: true,
            filledColors: ['#C2D1F0'],
            colors: ['#3366CB'],
            shadow: false,
            tickmarksStyle: null,
            xaxisTickmarksCount: 0,
            backgroundGridVlines: false,
            backgroundGridBorder: false,
            xaxis: false,
            textSize: 16
        }
    })
    Pline.draw();

    Gauge1 = new RGraph.Gauge({
        id: 'cvs_gauge',
        min: -10,
        max: 25,
        value: 19,

```

```

    options: {
        centery: 120,
        radius: 130,
        anglesStart: RGraph.PI,
        anglesEnd: RGraph.TWOPI,
        needleSize: 85,
        borderWidth: 0,
        shadow: false,
        needleType: 'line',
        colorsRanges: [[-10,-5,'#000099'], [-5,8,'#0000FF'],
[8,15,'#0099FF'],[15,20,'#FFCC33'],[20,25,'#FF6633']],
        borderInner: 'rgba(0,0,0,0)',
        borderOuter: 'rgba(0,0,0,0)',
        borderOutline: 'rgba(0,0,0,0)',
        centerpinColor: 'rgba(0,0,0,0)',
        centerpinRadius: 0
    }
}).grow();
}

/*-----
Funciones necesarias para actualizar las graficas
-----*/

function drawLine(value){
    if(!Pline){return}
    RGraph.Clear(canvas1);
    data_line.push(value);

    if(data_line.length > numvalues){
        data_line = RGraph.arrayShift(data_line); //Esto descarta el primer valor del
arreglo
    }

    Pline.original_data[0] = data_line;
    Pline.draw();
}

/*-----
Conectar al socket y leer el mensaje
-----*/

//Conexion
const socket = io.connect('http://localhost:3700');

socket.on("reactor", function (dataValue){
    drawLine(dataValue.value);
    //Gauge1.value = dataValue.value;

```



```

    //Gauge1.grow();
  });

socket.on("chiller", function (dataValue){
  //drawLine(dataValue.value);
  Gauge1.value = dataValue.value;
  Gauge1.grow();
});

//Atender el evento

```

IV. FRONTEND

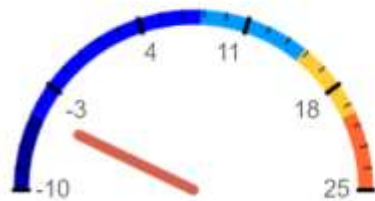
Todo el código anterior, tiene como objetivo final visualizar los datos de *Codesys* en una pagina WEB y en *MongoDB*, a continuación, se mostrara el fruto del código. Se genero un IP cuyo servidor es local desde el mismo ordenador. La IP generada es <http://localhost:3700/> que podemos acceder desde cualquier navegador WEB.

Produccion Quimico

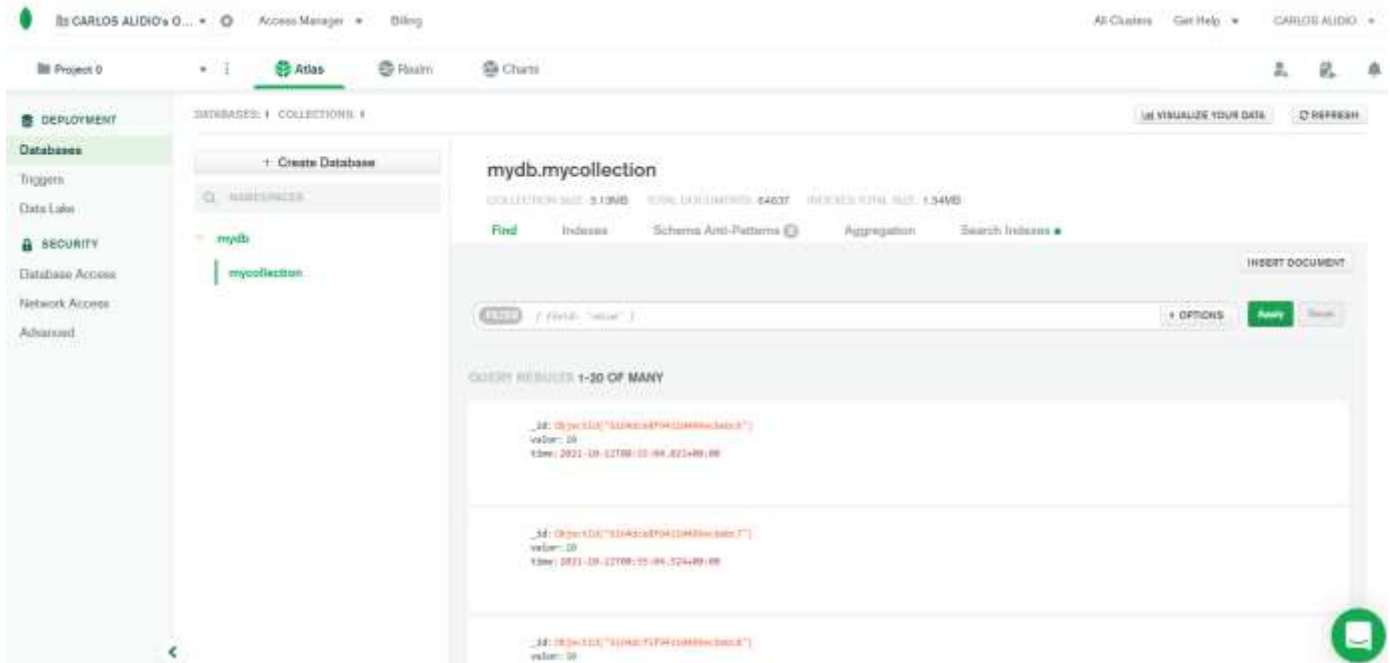
Temperatura reactor



Temperatura chiller



Y para *MongoDB* observamos el historial de los datos almacenamos.



V. AZURE IOT

Azure es una de las nubes industriales más importantes, de la empresa Microsoft. De todos los servicios que tiene Azure, un servicio se encarga de la conexión de dispositivos IoT. Estos dispositivos pueden ser cualquiera, sea un bombillo, un tomacorriente, un motor, entre varios dispositivos, que requieran de un lenguaje de programación y tenga acceso a internet. Básicamente los dispositivos IoT usan protocolos MQTT y OPCUA, para permitir la conexión ya sea desde tópicos o nodos. La conexión por internet se realiza mediante el protocolo *HTTPs*, es el protocolo mas usado para la conexión por medio de internet, pero con mayor seguridad, permitiendo la mayor confianza de la comunicación de los datos y la conexión a los dispositivos.

Con Azure IoT, creamos un grupo de recursos, para agrupar los dispositivos de IoT que creamos.

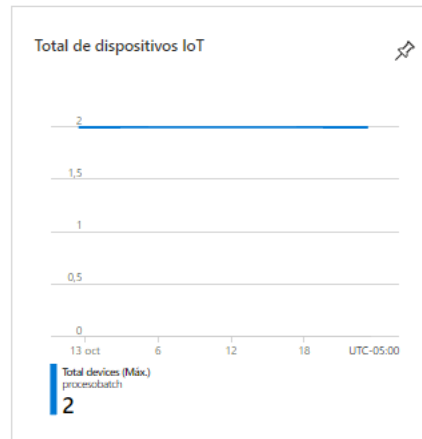
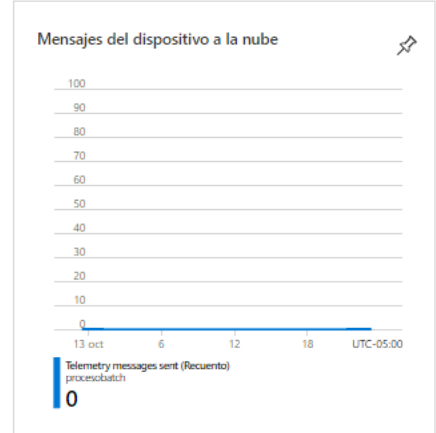


Desde allí podemos monitorear los dispositivos creados.

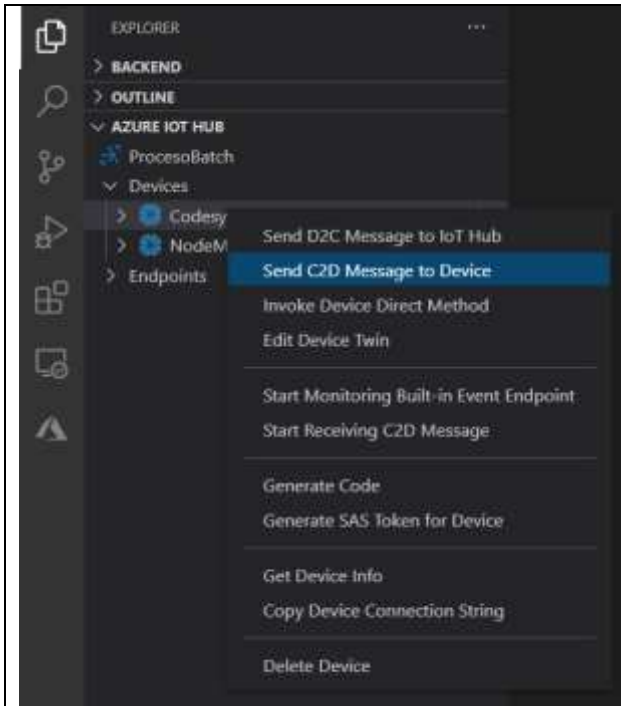
Mostrar datos del último período de: 1 Hora 6 Horas 12 Horas 1 Día 7 Días 30 Días

Uso de IoT Hub

- Mensajes usados hoy: 6
- Cuota diaria de mensajes: 8000 ⓘ
- Dispositivos IoT: 2



Para enviar mensajes a los dispositivos creados, se usa el Software *Visual Studio Code* que posee librería para permitir la conexión.

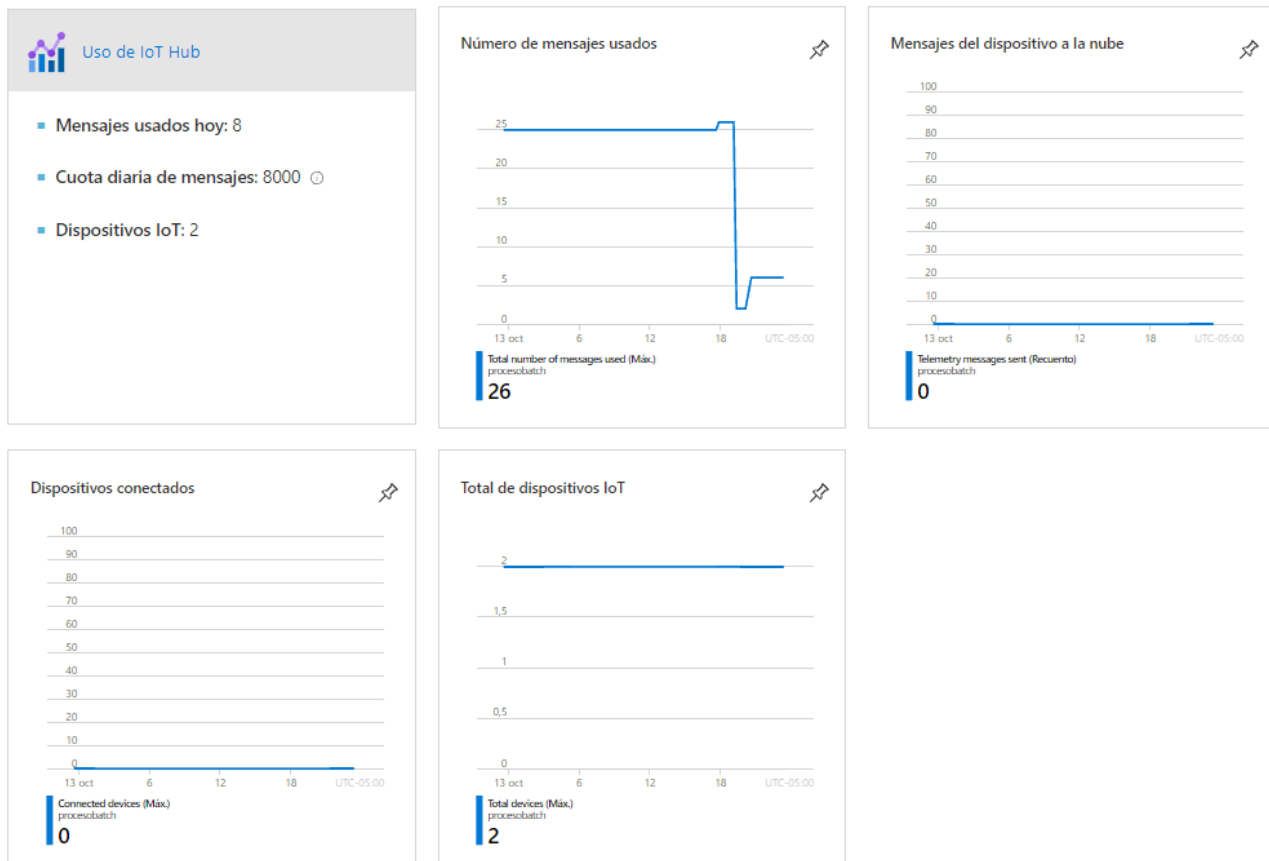


PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE

```
[C2DMessage] Sending message to [CodesysPLC] ...
[C2DMessage] [Success] Message sent to [CodesysPLC]
[C2DMessage] Sending message to [CodesysPLC] ...
[C2DMessage] [Success] Message sent to [CodesysPLC]
Subscription selected: Azure para estudiantes
IoT Hub selected: ProcesoBatch
[C2DMessage] Sending message to [CodesysPLC] ...
[C2DMessage] [Success] Message sent to [CodesysPLC]
[C2DMessage] Sending message to [CodesysPLC] ...
[C2DMessage] [Success] Message sent to [CodesysPLC]
```

Desde *Send C2D Message to Device* Podemos enviar mensajes a Azure, lo que se observara en el monitoreo, se nota la cantidad de información enviada, la fecha y hora de los datos enviados.

Mostrar datos del último período de: 1 Hora 6 Horas 12 Horas 1 Día 7 Días 30 Días



VI. CONCLUSIÓN

La norma ISA88, permite una nueva forma de programación para todos los procesos continuos que dependan de una receta, o sea por lotes. La programación en forma de celular permite mantener un orden único para todo el proceso, lo que impide errores convencionales con la programación de un solo archivo, además de permitir encontrar con facilidad los errores y localizar en el instante de ejecución del proyecto. Además de lograr usar dos protocolos muy útiles en la vida cotidiana como los son el OPC UA y el MQTT, que permite la conexión a varios servidores y a la nube, permitiendo la unión entre las tecnologías de la industria 3.0 y la industria 4.0.