



*ugr*

Universidad  
de Granada

# Análisis del software EPICS para el control y supervisión de procesos industriales

---

TRABAJO FIN DE GRADO  
INGENIERÍA INFORMÁTICA

**Autor**

Juan Carlos Ruiz Fernández

**Director**

Miguel Damas Hermoso







*ugr* | Universidad  
de **Granada**

# Análisis del software EPICS para el control y supervisión de procesos industriales

---

TRABAJO FIN DE GRADO  
INGENIERÍA INFORMÁTICA

**Autor**

Juan Carlos Ruiz Fernández

**Director**

Miguel damas hermoso





# Análisis del software EPICS para el control y supervisión de procesos industriales

Juan Carlos Ruiz Fernández

**Palabras clave:** SCADA, EPICS, WinCC OA, simulación, entorno industrial, acelerador de partículas, requerimientos críticos

## Resumen

---

Con este *Trabajo Fin de Grado (TFG)* se pretende analizar el software *EPICS*, ampliamente usado en la supervisión y control de ambientes científicos como aceleradores de partículas, así como también en el ámbito industrial.

Para este fin se ha desarrollado, en un entorno simulado, un proceso industrial el cual imita la secuencialidad de hechos en un acelerador de partículas.

En base a dicho proceso, se confecciona una interfaz **SCADA** (*Control Supervisor y Adquisición de Datos*, por sus siglas en inglés) que cubra las necesidades de un ámbito tan crítico como el industrial. Entre estas necesidades se puede encontrar la monitorización de los sensores y actuadores, el control del proceso o el manejo de alertas y alarmas, entre otras funciones.

El desarrollo de esta interfaz y el establecimiento de las comunicaciones implica completar, o complementar, el software base *EPICS* con otras herramientas, las cuales también serán analizadas.

En contraposición, se implementa esa misma aplicación *SCADA* con un software comercial como *WinCC Open Architecture (WinCC OA)*.

Al realizar la misma interfaz, dentro de lo posible, se pueden analizar ambos frameworks de control de forma exhaustiva y dar a conocer las mejores y peores características de cada uno.

# Analysis of the EPICS Software for Industrial Process Control and Supervision

Juan Carlos Ruiz Fernández

**Palabras clave:** SCADA, EPICS, WinCC OA, simulation, industrial environment, particle accelerator, critical requirements

## Abstract

---

With this “final grade work” (TFG), the main target is analyze the *EPICS* software, widely used for the supervision and control of scientific environments such as particle accelerators, in the industrial field.

For this purpose, an industrial process has been developed in a simulated environment, mimicking the sequence of events in a particle accelerator.

Based on this process, a **SCADA** (*Supervisory Control and Data Acquisition*) interface is created to meet the real needs of such a critical industrial environment. These needs include sensor and actuator monitoring, process control, and management of alerts and alarms.

The development of this interface and the establishment of communications involve completing or complementing the base EPICS software with other tools, which will be included in the analysis.

In contrast, the same **SCADA** is also implemented in a commercial software such as *WinCC Open Architecture (WinCC OA)*. By creating the same interface, to the extent possible, both programs can be thoroughly analyzed, highlighting their strengths and weaknesses.



Yo, **Juan Carlos Ruiz Fernández**, alumno de la titulación *Grado en Ingeniería Informática* de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77448396J, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

**Fdo:** *Juan Carlos Ruiz Fernández*

Granada a 1 de septiembre de 2023





**D. Miguel Damas Hermoso**, Profesor en el área de *Ingeniería de Computadores, Automática y Robótica* de la Universidad de Granada .

**Informa:**

Que el presente trabajo, titulado ***Análisis del software EPICS para el control y supervisión de procesos industriales***, ha sido realizado bajo su supervisión por **Juan Carlos Ruiz Fernández**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 1 de septiembre de 2023.

**El tutor:**

**Miguel Damas Hermoso**

## Agradecimientos

---

A mi madre por impulsarme hasta llegar donde estoy hoy. A todos mis amigos que han seguido este arduo camino y han sido mis pilares en los peores momentos. A mis compañeros de carrera que se han convertido en amigos y hacen que la competitividad sea un energético desarrollo personal.



# Índice

---

<b>TABLA DE ILUSTRACIONES .....</b>	<b>16</b>
<b>GLOSARIO .....</b>	<b>20</b>
<b>1 INTRODUCCIÓN.....</b>	<b>22</b>
1.1 MOTIVACIÓN.....	22
1.2 OBJETIVOS DEL PROYECTO .....	22
1.3 ESTRUCTURA DE PROYECTO .....	23
1.4 ESTRUCTURA DE DIRECTORIOS DE LOS ARCHIVOS ADICIONALES.....	24
<b>2 CONTEXTO .....</b>	<b>25</b>
2.1 IFMIF-DONES .....	25
2.1.1 EPICS dentro de DONES .....	26
2.2 SCADAS EN LA INDUSTRIA.....	26
2.2.1 WinCC OA en contraposición.....	27
2.3 PROTOCOLO OPC UA .....	27
<b>3 HERRAMIENTAS SOFTWARE.....</b>	<b>28</b>
3.1 FRAMEWORKS DE CONTROL (SCADAS) .....	28
3.1.1 EPICS.....	28
3.1.2 WinCC OA .....	38
3.2 SOFTWARE DE SIMULACIÓN .....	42
3.2.1 Factory IO .....	42
3.2.2 PLCSim .....	42
3.3 ENTORNOS DE PROGRAMACIÓN PLC .....	43
3.3.1 TiaPortal.....	43
3.4 COMUNICACIONES.....	46
3.4.1 KepServer EX.....	46
3.4.2 NetToPLCsim .....	47
3.5 DIAGRAMA DE CONEXIÓN.....	48
<b>4 DEMOSTRADOR .....</b>	<b>49</b>
4.1 PROCESO INDUSTRIAL.....	49
4.1.1 Introducción a los elementos de Factory IO .....	49
4.1.2 Descripción del proceso industrial.....	50
4.2 PROGRAMACIÓN DEL CONTROLADOR.....	55
4.2.1 Autómata .....	55
4.2.2 Estructura de programa .....	57
4.2.3 Programa y variables .....	59
4.3 CONFIGURACIÓN .....	59
4.3.1 Configuración del proceso .....	59
4.3.2 Configuración de la comunicación con los SCADA.....	62
<b>5 DESARROLLO DE LA APLICACIÓN SCADA.....</b>	<b>65</b>
5.1 HARDWARE .....	65
5.2 SOFTWARE .....	65
5.3 JERARQUÍA DE LOS PANELES .....	65

5.4 ESTRUCTURA DE LA INFORMACIÓN .....	66
5.5 PROCESO DE INSTALACIÓN.....	66
5.6 INSTALACIÓN DE MÓDULOS .....	69
5.7 CREACIÓN DEL PROYECTO.....	70
5.8 CONFIGURACIÓN DE DRIVERS .....	76
5.9 CREACIÓN DE LAS VARIABLES DEL PROCESO .....	81
5.10 CREACIÓN DE LA INTERFAZ.....	86
<i>5.10.1 Introducción a las interfaces de edición</i> .....	86
<i>5.10.2 Información central</i> .....	89
<i>5.10.3 Barra Superior – Accesos directos</i> .....	109
<i>5.10.4 Barra Inferior</i> .....	111
<i>5.10.5 Montaje de la pantalla principal – paneles empotrados</i> .....	115
<i>5.10.6 Resto de elementos y configuraciones</i> .....	116
5.11 GESTIÓN DE ALARMAS .....	122
5.12 GESTIÓN DE REMANENCIA DE DATOS .....	126
5.13 APLICACIONES FINALES .....	127
<b>6 PLANIFICACIÓN Y PRESUPUESTO .....</b>	<b>128</b>
6.1 FASES.....	128
<i>6.1.1 Fase inicial</i> .....	128
<i>6.1.2 Fase de análisis de requisitos</i> .....	129
<i>6.1.3 Fase de desarrollo</i> .....	129
<i>6.1.4 Fase de prueba y corrección</i> .....	129
<i>6.1.5 Redacción de la memoria</i> .....	129
6.2 DIAGRAMA DE GANTT .....	131
6.3 PRESUPUESTO.....	132
<i>6.3.1 Recursos</i> .....	132
<i>6.3.2 Costes</i> .....	132
<b>7 ANÁLISIS Y VALORACIONES.....</b>	<b>134</b>
7.1 DISCUSIÓN EN BASE A LOS RESULTADOS .....	134
7.2 IMPRESIONES.....	135
7.3 TABLA RESUMEN .....	138
7.4 CURVAS DE ESFUERZO.....	138
<b>8 CONCLUSIONES Y TRABAJOS FUTUROS .....</b>	<b>140</b>
8.1 CONCLUSIONES .....	140
8.2 LÍNEAS FUTURAS DE TRABAJO .....	141
<b>9 INCIDENCIAS .....</b>	<b>142</b>
<b>BIBLIOGRAFÍA .....</b>	<b>144</b>
<b>ANEXOS .....</b>	<b>146</b>
PANELES FINALES.....	146
<i>EPICS</i> .....	146
<i>WinCC OA</i> .....	152
RECORDS .....	160
DATOS PARA IMPORTAR VARIABLES A WINCC OA .....	163

# Tabla de ilustraciones

---

Ilustración 1 - Esquema simplificado del funcionamiento de la instalación IFMIF-DONES.....	26
Ilustración 2 - EPICS Logo .....	28
Ilustración 3 Estructura simple de EPICS.....	29
Ilustración 4 Componentes de un IOC.....	30
Ilustración 5 Ejemplo de enlazado de registros .....	32
Ilustración 6 WinCC OA Logo .....	38
Ilustración 7 - Arquitectura de WinCC OA.....	38
Ilustración 8 - Estructura en árbol de un Data Point orientado a dispositivo .....	40
Ilustración 9 - Data points Float y Pump .....	41
Ilustración 10 - Factory IO Logo.....	42
Ilustración 11 - Factory IO Escenario con algunos componentes y estaciones .....	42
Ilustración 12 - PLCSim Logo.....	42
Ilustración 13 - Interfaz PLCSim .....	43
Ilustración 14 - TIA Portal V17 Logo .....	43
Ilustración 15 - Segmento 2 del bloque 3PesadoRociadoSalidas en KOP .....	44
Ilustración 16 - Fragmento ejemplo de FUP .....	44
Ilustración 17 - Código del bloque ClasificadoraSCL .....	45
Ilustración 18 - Fragmento del bloque Pesado-Rociado en GRAPH .....	45
Ilustración 19 - Fragmento del segmento 2 de PesadoRociadoSalidas traducido a AWL.....	46
Ilustración 20 - Fragmento del segmento 2 de PesadoRociadoSalidas traducido a AWL.....	46
Ilustración 21 - Panel principal KepServer sin conexiones .....	47
Ilustración 22 - Diagrama de conexión .....	48
Ilustración 23 - Factory IO Elemento Right Positioner .....	49
Ilustración 24 – Demostrador Factory IO Visión general .....	50
Ilustración 25 – Demostrador Factory IO Tipos de piezas, Emitter y Palletizer .....	50
Ilustración 26 - Demostrador Factory IO Centradora .....	51
Ilustración 27 - Demostrador Factory IO Secuencia de Centradora.....	51
Ilustración 28 - Demostrador Factory IO Estampadora .....	51
Ilustración 29 - Demostrador Factory IO Secuencia Estampadora .....	52
Ilustración 30 – Demostrador Factory IO Rociado por peso .....	52
Ilustración 31 - Demostrador Factory IO Secuencia de Rociado por peso .....	53
Ilustración 32 - Demostrador Factory IO Control numérico .....	53
Ilustración 33 - Demostrador Factory IO Secuencia Control numérico robotizado.....	54
Ilustración 34 - Demostrador Factory IO Clasificadora .....	54
Ilustración 35 - Demostrador Factory IO Remover .....	54
Ilustración 36 - Demostrador Factory IO Secuencia de clasificación .....	55
Ilustración 37 – Autómata CPU .....	56
Ilustración 38 – Autómata Módulos DI/DO y DO.....	56
Ilustración 39 - Módulos AI y AO .....	56
Ilustración 40 - Autómata completo .....	57
Ilustración 41 - Función SCALE de PesadoValor en las instrucciones permanentes anteriores del GRAPH PesadoRociado [FB3].....	58
Ilustración 42 - Función UNSCALE en el segmento 1 de PesadoRociadoSalidas [FC3].....	58
Ilustración 43 - Configuración del PLC en FactoryIO.....	60
Ilustración 44 - Asignación de entradas y salidas en FactoryIO .....	60
Ilustración 45 - Indicador conexión correcta entre FactoryIO-PLCSim .....	61

<i>Ilustración 46 - TIA Portal icono de subir progama .....</i>	61
<i>Ilustración 47 - TIA Portal Dialogo de carga.....</i>	61
<i>Ilustración 48 - Interfaz PLCSIM.....</i>	62
<i>Ilustración 49 - Factory IO Funcionando.....</i>	62
<i>Ilustración 50 - Interfaz de NetToPLCsim con una conexión ejemplo .....</i>	62
<i>Ilustración 51 - Configuración del canal en KepServer .....</i>	63
<i>Ilustración 52 - Configuración dispositivo en KepServer.....</i>	64
<i>Ilustración 53 - Panel principal de KepServer con la conexión, el dispositivo y las etiquetas .....</i>	64
<i>Ilustración 54 - Jerarquía de paneles.....</i>	65
<i>Ilustración 55 - Estructura de la información de los paneles.....</i>	66
<i>Ilustración 56 - WinCC OA Instalación Primera ventana .....</i>	68
<i>Ilustración 57 - WinCC OA Instalación Selección de extensiones.....</i>	68
<i>Ilustración 58 - WinCC OA Instalación Progreso .....</i>	69
<i>Ilustración 59 - EPICS Estructura de directorios de la aplicación base.....</i>	71
<i>Ilustración 60 - Aplicación WinCC OA Project Administration .....</i>	71
<i>Ilustración 61 - WinCC OA Project Administration Selección tipo proyecto.....</i>	72
<i>Ilustración 62 – WinCC OA Project Administration Lista de proyectos e ícono de creación .....</i>	72
<i>Ilustración 63 - WinCC OA Project Administration Nombre, ruta, idioma y ejecutable .....</i>	73
<i>Ilustración 64 - WinCC OA Project Administration Usuarios y contraseñas.....</i>	73
<i>Ilustración 65 - WinCC OA Project Administration Certificados.....</i>	74
<i>Ilustración 66 - WinCC OA Project Administration Servidor en local .....</i>	74
<i>Ilustración 67 - WinCC OA &gt; Project Administration &gt; Resumen .....</i>	75
<i>Ilustración 68 - WinCC OA &gt; Project Administration &gt; Creando .....</i>	75
<i>Ilustración 69 - WinCC OA &gt; Project Administration &gt; Lista de proyectos actualizada.....</i>	75
<i>Ilustración 70 - EPICS procesadorPiezasApp/src/Makefile modificado .....</i>	76
<i>Ilustración 71 - EPICS configuración de drivers Estructura de directorios Archivos modificados .....</i>	77
<i>Ilustración 72 - WinCC OA &gt; Console &gt; incorporación del cliente OPC.....</i>	77
<i>Ilustración 73 - WinCC OA Gedi &gt; Interfaz con System Management indicado.....</i>	78
<i>Ilustración 74 - WinCC OA &gt; System Management.....</i>	78
<i>Ilustración 75 - WinCC OA &gt; System Management &gt; Driver OPC .....</i>	78
<i>Ilustración 76 - WinCC OA OPC UA &gt; System Management &gt; Client Configuration &gt; Dirección y suscripción .....</i>	79
<i>Ilustración 77 - WinCC OA &gt; System Management &gt; OPC UA &gt; Client Configuration .....</i>	79
<i>Ilustración 78 - WinCC OA OPC UA &gt; System Management &gt; Client Configuration &gt; Browse .....</i>	80
<i>Ilustración 79 - Ícono módulo PARA .....</i>	83
<i>Ilustración 80 - WinCC OA PARA .....</i>	83
<i>Ilustración 81 - WinCC OA &gt; PARA Crear DPT .....</i>	84
<i>Ilustración 82 - WinCC OA &gt; PARA &gt; Dp-Type Editor Booleano .....</i>	84
<i>Ilustración 83 - WinCC OA ASCII Manager Ícono .....</i>	85
<i>Ilustración 84 - WinCC OA &gt; System Management &gt; Database &gt; ASCII Manager Interfaz .....</i>	85
<i>Ilustración 85 - WinCC OA &gt; PARA Todos los DP insertados y configurados.....</i>	86
<i>Ilustración 86 - Phoebus Interfaz .....</i>	87
<i>Ilustración 87 - WinCC OA Interfaz .....</i>	88
<i>Ilustración 88 - Phoebus Añadir panel .....</i>	89
<i>Ilustración 89 -WinCC OA Añadir panel .....</i>	90
<i>Ilustración 90 -WinCC OA Panel vacío .....</i>	90
<i>Ilustración 91 - Phoebus Imagen propiedades.....</i>	91
<i>Ilustración 92 - Phoebus Panel principal con la imagen general del proceso .....</i>	92
<i>Ilustración 93 - WinCC OA Rectángulo vacío .....</i>	93
<i>Ilustración 94 - WinCC OA Panel principal con la imagen general del proceso .....</i>	93

Ilustración 95 - Phoebus polígono Flecha .....	94
Ilustración 96 - Phoebus Regla cambiar color .....	94
Ilustración 97 - Phoebus Regla Change color y color dinámico .....	95
Ilustración 98 - WinCC OA Polígono flecha .....	95
Ilustración 99 - WinCC OA Zoom sobre Event Script Initialize .....	96
Ilustración 100 - WinCC OA Wizard Initialization .....	96
Ilustración 101 - WinCC OA Initialize wizard change color .....	97
Ilustración 102 - WinCC OA Initialization wizard change color & Data Point Selector .....	97
Ilustración 103 - WinCC OA Color Selector .....	98
Ilustración 104 - Panel principal con indicadores de cinta .....	98
Ilustración 105 - Phoebus Polígono pieza .....	99
Ilustración 106 - Phoebus Reglas Visibilidad .....	99
Ilustración 107 - Phoebus Reglas Visibilidad Record numérico .....	99
Ilustración 108 - Phoebus Regla Color Record numérico .....	99
Ilustración 109 - WinCC OA Polígono Pieza .....	100
Ilustración 110 - WinCC OA Initialization Change visibility .....	100
Ilustración 111 - WinCC OA Initialization Change color pieza clasificadora .....	101
Ilustración 112 - Phoebus visión general con indicadores de cinta, pieza y clasificadora .....	101
Ilustración 113 - WinCC OA visión general con indicadores de cinta, pieza y clasificadora .....	101
Ilustración 114 - Phoebus Action Button .....	102
Ilustración 115 - Phoebus Action button Añadir acción .....	102
Ilustración 116 - Phoebus Action Button Open Display .....	103
Ilustración 117 - Phoebus Accesos por zona .....	103
Ilustración 118 - WinCC OA Accesos por zona .....	104
Ilustración 119 - Phoebus tank .....	104
Ilustración 120 - Phoebus Vision general con tanque .....	105
Ilustración 121 - WinCC OA Pillar tanque .....	105
Ilustración 122 - WinCC OA Pillar configuración .....	105
Ilustración 123 - WinCC OA Vision general con tanque .....	106
Ilustración 124 - WinCC OA Regla tubería llenado .....	106
Ilustración 125 - Phoebus Visión general con tuberías .....	106
Ilustración 126 - WinCC OA Visión general con tuberías .....	107
Ilustración 127 - Phoebus Visión general leds .....	107
Ilustración 128 - WinCC OA STD Leds .....	108
Ilustración 129 - WinCC OA Visión General con indicadores led .....	108
Ilustración 130 - Phoebus Monitor numérico .....	108
Ilustración 131 - WinCC OA STD Objects Values .....	108
Ilustración 132 - WinCC OA Monitor numérico .....	109
Ilustración 133 - Phoebus Barra Superior .....	109
Ilustración 134 - WinCC OA Push Button .....	109
Ilustración 135 - WinCC OA Clicked .....	110
Ilustración 136 - WinCC OA Clicked Panel Functions .....	110
Ilustración 137 - WinCC OA Clicked Panel Functions Parameter .....	110
Ilustración 138 - WinCC OA Clicked Panel Functions Type and Position .....	111
Ilustración 139 - WinCC OA Barra Superior .....	111
Ilustración 140 - Barra inferior Phoebus .....	112
Ilustración 141 - WinCC OA Asistente Mouse Pressed .....	113
Ilustración 142 - WinCC OA Barra inferior .....	113
Ilustración 143 - Phoebus Plots .....	114
Ilustración 144 - Phoebus Script chart configuración .....	114

<i>Ilustración 145 - Phoebus Barra inferior del panel principal .....</i>	114
<i>Ilustración 146 - WinCC OA Históricos.....</i>	114
<i>Ilustración 147 - WinCC OA Trend parametrization configuración.....</i>	115
<i>Ilustración 148 - WinCC OA Trend parametrization configuración tendencia.....</i>	115
<i>Ilustración 149 - WinCC OA Bara inferior con histórico .....</i>	115
<i>Ilustración 150 - Phoebus Panel principal final.....</i>	116
<i>Ilustración 151 - WinCC OA Panel principal final.....</i>	116
<i>Ilustración 152 - Phoebus regla para enabled .....</i>	117
<i>Ilustración 153 - Phoebus Centradora en ejecución automática.....</i>	117
<i>Ilustración 154 - Phoebus Barra inferior en ejecución manual.....</i>	117
<i>Ilustración 155 - WinCC OA Centradora en ejecución automática .....</i>	118
<i>Ilustración 156 - WinCC OA Centradora en ejecución manual.....</i>	118
<i>Ilustración 157 - Phoebus Dial .....</i>	120
<i>Ilustración 158 - WinCC OA Dial Gauge EWO .....</i>	121
<i>Ilustración 159 - WinCC OA Data Point de calculo.....</i>	122
<i>Ilustración 160 - Phoebus Componentes del sistema de alarmas .....</i>	123
<i>Ilustración 161 - WinCC OA Insert Config Alert.....</i>	124
<i>Ilustración 162 - WinCC OA Ventana Alarm Handling .....</i>	125
<i>Ilustración 163 - WinCC OA Alarm and Events Screen .....</i>	125
<i>Ilustración 164 - WinCC OA Archiving.....</i>	127
<i>Ilustración 165 - Diagrama de Gantt .....</i>	131
<i>Ilustración 166 - Curvas de esfuerzo a corto plazo .....</i>	139

# Glosario

---

## A

### Autómata

Puede definirse como un equipo electrónico programable en lenguaje no informático y diseñado para controlar, en tiempo real y en ambiente industrial, procesos secuenciales. Este término es similar al PLC.

## C

### CERN

La Organización Europea para la Investigación Nuclear, comúnmente conocida por la sigla CERN, es una organización de investigación europea que opera el laboratorio de Física más grande del mundo. La organización tiene 22 estados miembros —Israel es el único país no europeo con una membresía plena concedida— y está oficialmente observado por las Naciones Unidas.

### CWS

La estación de trabajo del cliente (Client Workstation) es cualquier ordenador capaz de ejecutar varias herramientas de EPICS y aplicaciones cliente. Son comúnmente las herramientas de interfaz de usuario y almacenamiento de datos. El ordenador puede ser cualquiera de escritorio, servidor o similar funcionando con sistemas operativos regulares (Windows, Linux o macos).

## D

### Data Point

Variable de un proceso que se controla y monitoriza desde wincc OA.

### DONES

International Fusion Materials Irradiation Facility – Demo Oriented neutron Source (IFMIF-DONES) es una infraestructura de investigación novedosa de un solo sitio para probar, validar y calificar los materiales que se utilizarán en futuras plantas de energía de fusión como DEMO (un prototipo de reactor de fusión de demostración). En relación con este proyecto internacional, en diciembre de 2017, Fusion for Energy (F4E) valoró positivamente la propuesta conjunta de España y Croacia para ubicar el IFMIF-DONES en Granada.

### Drivers

Un driver es esencialmente un traductor que permite la comunicación entre un elemento software y otro hardware.

## F

### Framework

Entorno de trabajo o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

## I

### IOC

El controlador de entrada-salida (Input/Output Controller) es el componente servidor de EPICS. Casi cualquier plataforma que pueda soportar los componentes básicos de EPICS puede ser usado como IOC.

### ITER

El ITER,, es un experimento científico a gran escala que intenta producir un plasma de fusión que tenga diez veces más potencia térmica que la potencia necesaria para calentar el plasma. Wikipedia.

## L

### LAN

Local Area Network o red de área local basada en ethernet (o inalámbrico) que permita la comunicación entre los iocs y los cwss.

## O

### OPC

El OPC es un estándar de comunicación en el campo del control y supervisión de procesos industriales, basado en una tecnología Microsoft, que ofrece una interfaz común para comunicación que permite que componentes de software individuales interactúen y compartan datos.

## P

### PLC

Un PLC, o Controlador Lógico Programable (en inglés, Programmable Logic Controller), es un dispositivo electrónico utilizado en la automatización industrial para controlar y supervisar una variedad de procesos y sistemas.

### Process Variable

Variables de proceso que puede representar cualquier atributo o dato.

## R

### Radiación

El fenómeno de la radiación es la propagación de energía en forma de ondas electromagnéticas o partículas subatómicas a través del vacío o de un medio material.

## S

### SCADA

SCADA, acrónimo de Supervisory Control And Data Acquisition es un concepto que se emplea para realizar un software para ordenadores que permite controlar y supervisar procesos industriales a distancia. Facilita retroalimentación en tiempo real con los dispositivos de campo, y controla el proceso automáticamente.

### Softplc

Un softplc es un software que se instala en una electrónica y emula las funcionalidades de un PLC. De esta manera, equipando a un PC con un software de estas características y módulos de Entrada / Salida, se puede conseguir un PLC emulado.

### Software

Se conoce como software, logicial o soporte lógico al sistema formal de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hace posible la realización de tareas específicas, en contraposición a los componentes físicos que son llamados hardware.

# 1 Introducción

---

## 1.1 Motivación

La energía nuclear pretende ser una fuente de energía crucial para la generación de electricidad limpia. Sin embargo, se enfrenta a desafíos significativos, entre ellos, la degradación de materiales debido a la radiación y altas temperaturas del plasma.

La exposición prolongada a la radiación en estos reactores nucleares puede provocar daños y cambios en las propiedades físicas de los materiales, lo que afecta su integridad estructural y longevidad. A su vez, las altas temperaturas a las que se someten los componentes del reactor también pueden acelerar el envejecimiento y la degradación éstos, lo que representa una preocupación para la seguridad, el mantenimiento y viabilidad a largo plazo de las plantas nucleares de fusión.

Abordar estos desafíos es esencial para garantizar la eficiencia y seguridad continua de la energía de fusión en el panorama energético actual y futuro.

El objetivo de IFMIF-DONES, que se implantará en nuestra ciudad Granada, es la construcción de un pequeño acelerador que permita hacer comprobaciones sobre los materiales que se usarán en los futuros reactores de fusión comerciales.

Al momento del nacimiento de este *TFG*, el proyecto DONES se encontraba analizando los sistemas de manipulación remota, comprobando cuáles se utilizan en otros aceleradores de partículas como el CERN o incluso ITER y verificar su validez para sus objetivos.

Estos sistemas deben cumplir unos requerimientos muy críticos, como la precisión en la manipulación o tener una alta disponibilidad.

## 1.2 Objetivos del proyecto

En particular, el objetivo de este *TFG* es examinar y comparar dos tipos de software: EPICS, que se utiliza habitualmente en este tipo de instalaciones científicas, y las herramientas SCADA que se emplean en la industria convencional.

Con este fin, se propone llevar a cabo el control y supervisión de un proceso industrial (proceso gestionado por un autómata programable de Siemens) utilizando tanto EPICS como un SCADA comercial, en este TFG WinCC OA, y recopilar las métricas necesarias para determinar cuál de las dos soluciones es la más adecuada. Estos SCADAS se comunicarán con el proceso industrial mediante un servidor OPCUA, unificando en esta forma la comunicación de éstos con el proceso.

De esta manera, se busca obtener una visión más completa y detallada sobre las características y limitaciones de cada herramienta, para poder seleccionar la que mejor se adapte a las necesidades específicas de cada proyecto científico.

Este análisis comparativo permitirá identificar las fortalezas y debilidades de cada herramienta, su nivel de complejidad, la facilidad de uso, la integración con otros sistemas, la escalabilidad y la capacidad de adaptación a diferentes entornos y situaciones. En definitiva, se espera obtener una evaluación rigurosa y objetiva de estas herramientas, que pueda servir de guía para futuros proyectos científicos y tecnológicos.

En este proyecto, la perspectiva es desde el punto de vista de un informático que se enfrenta a ambos programas desde cero. Sumergiéndose en la experiencia de instalar y utilizar ambas opciones con el objetivo de evaluar las curvas de aprendizaje y los costos asociados. A través de este enfoque, se busca comprender y destacar las dificultades en el aprendizaje y el tiempo que conlleva.

## 1.3 Estructura de proyecto

Este documento se encuentra vertebrado de la siguiente forma:

- **Contexto:** Breve descripción e introducción al proyecto IFMIF-DONES, el papel de EPICS en éste, los SCADAS comerciales disponibles y el papel de WinCC OA.
- **Herramientas software:** Introducción a los software de control y supervisión, además de:
  - Descripción detallada de EPICS y WinCC OA
  - Descripción del software que simula el proceso industrial
  - Entornos de programación
  - Comunicaciones entre los distintos software
- **Demostrador:** El demostrador engloba el conjunto del proceso industrial y su programación con un autómata programable, también simulado. Se encontrará:
  - Descripción de las etapas que conforman el proceso
  - Simulación de éste y el driver de comunicación
  - Programación del autómata
- **Desarrollo de la aplicación SCADA:** Se mostrará cómo se ha formado las aplicaciones finales con cada software. Se irán comparando paralelamente por etapas, como creación de proyectos, configuración de drivers o creación de pantallas, con el objetivo de obtener una visión de las diferencias entre ambos programas.
- **Planificación y presupuesto:** Descripción de cómo se ha desarrollado este proyecto a lo largo del tiempo y un cómputo de cuánto costaría en horas de aprendizaje, implementación y licencias de cada software.
- **Análisis y valoraciones:** Discusión de los resultados e impresiones durante el proceso junto a una tabla que resume, a la vez comparan las características más relevantes y una gráfica que refleja el esfuerzo necesario al iniciar cada etapa.
- **Conclusiones y trabajos futuros:** Conclusiones tras el análisis de las plataformas y futuras ampliaciones de este proyecto.
- **Bibliografía:** Fuentes de información obtenidas para la realización del proyecto.
- **Anexos:** Documentos de interés sobre la realización de este proyecto, como la programación del autómata y glosario de términos.

## 1.4 Estructura de directorios de los archivos adicionales

Junto a este documento se adjuntan los archivos fruto de este análisis, por lo que se van a desglosar para entender cuál es el papel de cada uno:

- **Carpeta 1: EPICS – IOC y Phoebus.** En esta carpeta se encuentran los archivos más relevantes de la creación de la aplicación con EPICS.
  - **1\_1\_IOC.** Proyecto base, para ejecutarlo se necesita tener instalado EPICS y su modulo OPC UA.
  - **1\_2\_Phobus.** Es el programa de creación de la interfaz gráfica junto a los paneles finales. Para ejecutarlo es conveniente resolver sus dependencias.
- **Carpeta 2: WinCC OA – Proyecto.** Proyecto completo de WinCC OA, para ejecutarlo hay que tenerlo instalado (basta con la licencia de evaluación) e incluirlo en el gestor de proyectos.
- **Carpeta 3: Demostrador.** Aquí se recogen los archivos resultantes para ejecutar el demostrador.
  - **3\_1\_ProcesadorPiezas.factoryio.** Archivo de Factory IO que contiene el proceso industrial con su driver y etiquetas configuradas.
  - **3\_2\_1\_ProgramaPLC.pdf.** Archivo pdf para visualizar el código del programa PLC y sus variables.
  - **3\_2\_2\_ProyectoTiaPortal15PLC.** Es el proyecto de tia portal con el programa del PLC junto a la configuración del dispositivo y sus extensiones.
  - **3\_3\_PLCSimulado.plc.** Archivo resultante de PLCSim tras cargar el programa y realizar pruebas. Contiene datos como el número de piezas procesadas hasta la última prueba.
  - **3\_4\_KepServerConfiguracion.opf.** Archivo de configuración de KepServer con todas las etiquetas.
- **Este documento.** Que es la memoria completa del proyecto.

## 2 Contexto

---

Al menos 70 años lleva investigándose la energía nuclear de fusión, aunque en estos últimos veinte años han ocurrido los mayores avances como la fundación de *ITER* en 2007 o el logro de la “ignición”, conseguir obtener más cantidad de energía que la cantidad se aporta, en el *NIF (Instalación Nacional de Ignición)*, por sus siglas en inglés) de EEUU el pasado diciembre. (Bishop, 2022)

El reactor tokamak de *ITER* (*International Thermonuclear Experimental Reactor*) es uno de los proyectos de energía nuclear más ambiciosos y avanzados en la actualidad. *ITER* está ubicado en Cadarache, Francia, y es un esfuerzo colaborativo internacional con la participación de 35 países.

Su objetivo principal es demostrar la viabilidad científica y tecnológica de la fusión nuclear como una fuente de energía limpia y prácticamente inagotable.

El reactor tokamak utiliza un campo magnético para confinar y comprimir el plasma a altas temperaturas (más de 150 millones de grados Celsius) para permitir que los núcleos de hidrógeno se fusionen y generen una cantidad significativa de energía. (*ITER Organization*, s.f.)

Aunque *NIF* consiguiera encontrar esa viabilidad con otro tipo de confinamiento, los tokamak apuntan a que será el tipo de reactor más factible y que usarán los reactores nucleares de fusión comerciales.

Junto a esas temperaturas, habrá grandes flujos de neutrones a alta energía que chocarán con las paredes y éstas se degradarán, por lo que se debe conocer bien como estos materiales se comportarán a lo largo de la vida útil del reactor. Es aquí donde interviene *DONES*.

### 2.1 IFMIF-DONES

La Comisión Europea se propuso construir una planta experimental para estudiar el comportamiento de materiales bajo irradiación. España y Croacia mostraron interés en albergar esta instalación. Aunque finalmente, Croacia apoyó la candidatura española y *Fusion for Energy (F4E)* aprobó ubicar la instalación, llamada *IFMIF-DONES*, en Granada.

El proyecto *IFMIF-DONES*, por sus siglas en inglés *International Fusion Materials Irradiation Facility – DEMO-Oriented Neutron Source*; siendo *DEMO* el futuro sucesor de *ITER* como reactor de demostración, tiene los siguientes objetivos:

---

1.- Obtener datos de la irradiación de materiales para el diseño, licenciamiento, construcción y operación segura del reactor de demostración de energía de fusión (*DEMO*) con las principales características tal y como se definen en la hoja de ruta europea, en un entorno de fusión simulado para anticipar las necesidades de resistencia a la radiación de los materiales estructurales en *DEMO*.

2.- Generar una base de datos con las respuestas de los materiales, usando para ello herramientas computacionales.

3.- Proporcionar una fuente de neutrones que produzca neutrones de alta energía a una intensidad y volumen de irradiación suficientes. Para ello, se diseñará, construirá y operará la nueva Instalación (Instalación *DONES*). (*IFMIF-DONES-España*, 2022)

El funcionamiento del acelerador se puede observar en la siguiente ilustración:

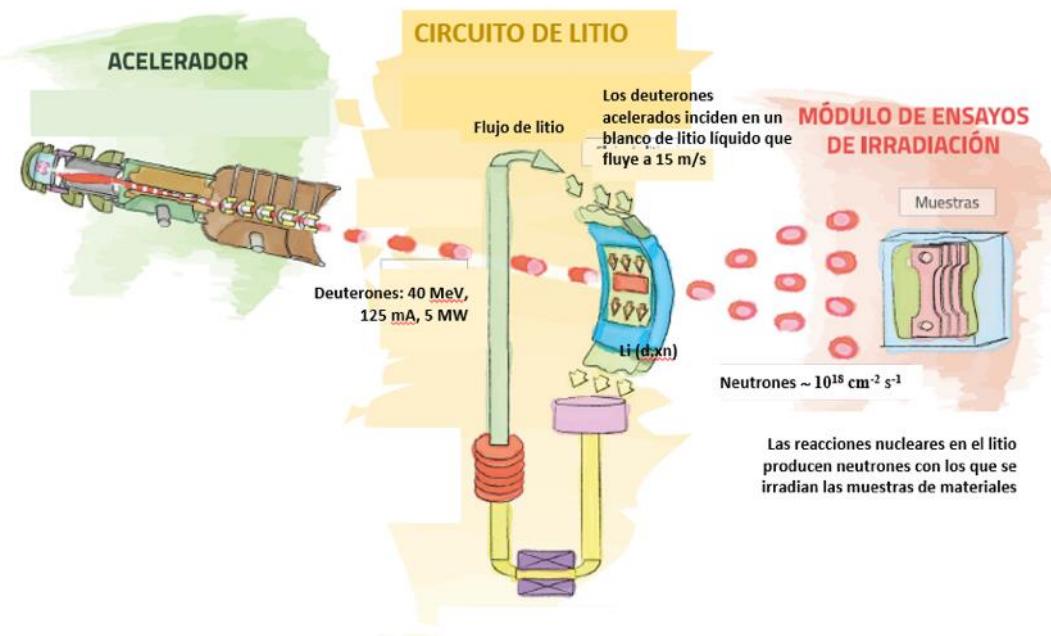


Ilustración 1 - Esquema simplificado del funcionamiento de la instalación IFMIF-DONES (Ciemat)

### 2.1.1 EPICS dentro de DONES

En el *Journal Fusion Engineering and Design* se publicó un volumen donde comentan el estado de la ingeniería y el diseño del proyecto IFMIF-DONES. Escribieron que después de una revisión general sobre el diseño e implementación se presta particular atención a la gestión de datos incluyendo EPICS como estructura de control, además el HMI se estaba desarrollando con *EPICS Control System Studio Display Builder* y *EPICS BEAST*. (Cappelli, y otros, 2021)

## 2.2 SCADAS en la industria

Los SCADA son sistemas de control y adquisición de datos utilizados para supervisar y controlar procesos industriales en tiempo real. La **importancia** de éstos en la industria es **innegable** debido a su amplio impacto en la eficiencia operativa, la seguridad y la toma de decisiones informadas.

Proporcionan una visión completa del estado de la planta, lo que permite a los operadores tomar decisiones informadas y rápidas para mejorar la eficiencia y la productividad.

Además, los SCADA facilitan el mantenimiento predictivo, reduciendo tiempos de parada no planificados y mejorando la seguridad en el entorno de trabajo. Su capacidad para integrarse con otros sistemas y su adaptabilidad los hacen herramientas esenciales para optimizar la gestión y el rendimiento de las operaciones industriales.

Algunos SCADAS comerciales son:

- SIMATIC WinCC OA, de Siemens
- AVEVA, adquirido por Schneider Electric
- iFix, de GE Digital
- MAPS, de Mitsubishi

## 2.2.1 WinCC OA en contraposición

Un coordinador de sistemas de control e instrumentación de IFMIF-DONES, Rubén Lorenzo Ortega, fue el encargado de analizar que sistemas SCADAS implementar en el proyecto y los dos software que tuvo en cuenta fueron *EPICS* y *WinCC OA*, según su perfil de [LinkedIn](#).

## 2.3 Protocolo OPC UA

El protocolo *OPC (OLE for Process Control)* es un estándar de comunicación para la supervisión y control de procesos y está basado en tecnología de Microsoft. La comunicación de este protocolo se basa en la **arquitectura cliente-servidor**. Es utilizado en la industria para facilitar la comunicación entre sistemas y dispositivos en entornos de automatización industrial y control de procesos.

*OPC UA* representa una evolución del estándar original *OPC*, el cual tenía limitaciones al operar con ciertas tecnologías de Microsoft como COM/DCOM.

Se utiliza ampliamente en la industria de la automatización y el control debido a varias razones:

1. **Interoperabilidad:** es un estándar ampliamente aceptado para la comunicación e intercambio de datos entre diferentes dispositivos y sistemas. Permite la interoperabilidad entre componentes de diferentes fabricantes y proveedores, lo que facilita la integración de sistemas heterogéneos.
2. **Arquitectura abierta y escalable:** está diseñado para permitir una fácil incorporación de nuevos dispositivos y sistemas a una red industrial. Esto lo hace adecuado para entornos industriales en constante evolución y crecimiento.
3. **Seguridad:** proporciona seguridad para proteger la integridad y confidencialidad de los datos transmitidos. Utiliza **cifrado, autenticación y mecanismos de firma digital** para garantizar la seguridad de la comunicación entre los dispositivos.
4. **Comunicación independiente de la plataforma:** permite la comunicación entre dispositivos en diferentes plataformas, como Windows, Linux o sistemas embebidos.
5. **Soporte para diferentes tipos de datos:** es capaz de manejar una amplia variedad de datos, desde simples hasta datos complejos y estructurados. Esto lo convierte en una opción flexible y adecuada para una amplia gama de aplicaciones industriales.
6. **Funcionalidades avanzadas:** Proporciona funcionalidades avanzadas, como descubrimiento automático de dispositivos, acceso a datos históricos, notificaciones de eventos, control remoto y gestión de alarmas. Estas características lo hacen ideal para aplicaciones exigentes.

En general, el uso del protocolo *OPC UA* en la industria se debe a su capacidad para habilitar la **comunicación segura, interoperable y confiable** entre los dispositivos y sistemas de automatización, lo que facilita la integración y la implementación de soluciones de control en entornos industriales diversos y complejos.

Por lo tanto, se ha escogido este protocolo porque es ampliamente usado en la industria, junto con ello, según (Valenzuela, y otros, 2022) en el paper "*The IFMIF-DONES remote handling control system: Experimental setup for OPC UA integration*" que habla de la implantación de *OPC UA* como estructura de control en IFMIF-DONES.

Por lo que es el método de unificación en la comunicación de EPICS y WINCC OA con el PLC se ha creado un servidor *OPC UA* que haga de intermediario. De esta forma se puede comparar de forma rigurosa ambos programas.

# 3 Herramientas software

A continuación, se van a explicar detalladamente los software SCADA que se van a analizar y los programas de los que se ha necesitado hacer uso para el correcto desarrollo del proyecto.

## 3.1 Frameworks de control (SCADAs)

### 3.1.1 EPICS

Definiendo EPICS, empecemos por sus siglas: *Experimental Physics and Industrial Control System*. No es un software que se instala y se usa, sino un conjunto de herramientas utilizadas para el control y la adquisición de datos.

Proporciona un marco para la implementación de **sistemas de control distribuidos** con las siguientes capacidades:

- Control remoto y monitoreo de las instalaciones
- Secuenciación automática de las operaciones
- Gestión de sincronización y coordinación de toda la instalación
- Detección de alarmas, reportes y control de accesos.
- Control en lazo cerrado
- Modelación y simulación
- Conversiones de datos y filtrado
- Adquisición de datos
- Análisis de datos, tendencias, recuperación y creación de gráficas
- Protección de accesos



Ilustración 2 - EPICS Logo (EPICS Control Web, s.f.)

EPICS es completamente **escalable**. Un sistema de grandes dimensiones debe ser capaz de transportar y almacenar grandes cantidades de información, además deberá ser robusta y tolerante a fallos. Uno pequeño debe tener la posibilidad de implementar un sistema de control sin infraestructuras caras y complejas.

Para aplicaciones modernas, la gestión de los datos se ha convertido en un elemento esencial. Debe ser posible almacenar los datos recogidos a largo plazo y poder recuperarlos en su forma original.

La ventaja de ser **software libre** es la activa comunidad extendida a lo largo de todo el mundo dispuesta a ayudar a otros usuarios con sus problemas o discutir sus nuevas ideas.

#### 3.1.1.1 Componentes del sistema

Este software es un conjunto de herramientas que permite la creación de aplicaciones servidores y clientes. Los servidores proveen el acceso a los datos, lectura, escritura local o sobre una red. Leer y escribir se hace comúnmente sobre hardware conectado directamente a los componentes físicos, aunque los datos pueden producirse o usarse en cualquier otro sitio.

Los clientes pueden mostrar, almacenar y manipular los datos. Un software cliente abarca desde (tanto gráficamente como desde la línea de comandos) una interfaz de usuario hasta servicios para la gestión de datos.

Los componentes básicos para un sistema de control basado en EPICS son:

- **IOC:** El controlador de entrada-salida (*Input/Output Controller*) es el componente servidor de EPICS. Casi cualquier plataforma que pueda soportar los componentes básicos de EPICS puede ser usado como IOC.
- **CWS:** La estación de trabajo del cliente (*Client Workstation*) es cualquier ordenador capaz de ejecutar varias herramientas de EPICS y aplicaciones cliente. Son comúnmente las herramientas de interfaz de usuario y almacenamiento de datos. El ordenador puede ser cualquiera de escritorio, servidor o similar funcionando con sistemas operativos regulares (Windows, Linux o MacOS).
- **LAN:** *Local Area Network* o red de área local basada en ethernet (o inalámbrico) que permite la comunicación entre los IOCs y los CWSs.

Siendo un ejemplo de estos componentes la siguiente imagen:

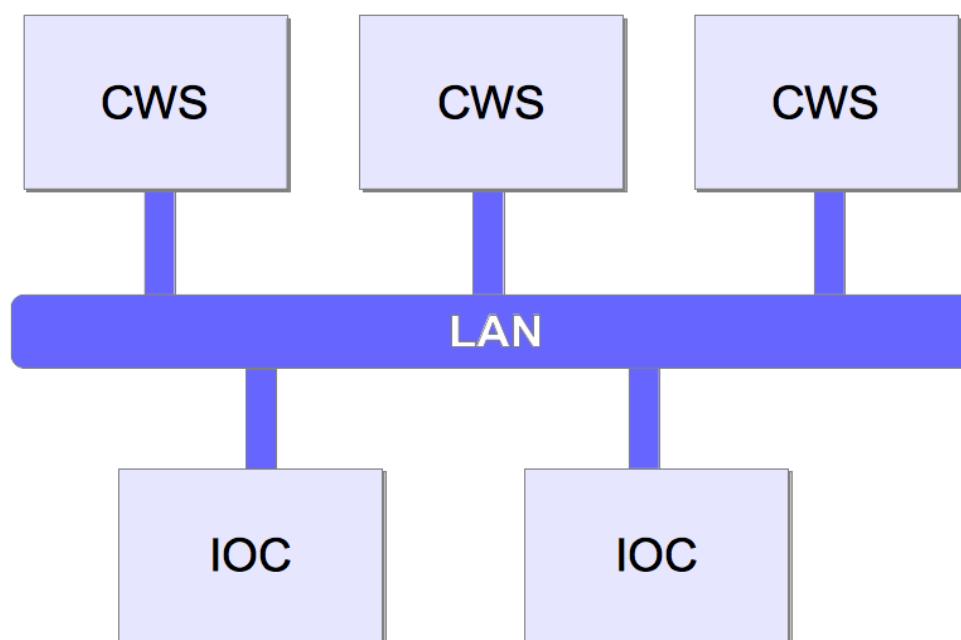


Ilustración 3 Estructura simple de EPICS (EPICS Control, 2019)

### 3.1.1.2 Características básicas

**Basado en herramientas:** Provee un conjunto de herramientas interactivas y componentes para la creación de sistemas de control. Esto disminuye la necesidad de programación específica y homogeneiza las interfaces con los operadores.

**Distribuido:** Se pueden implementar cualquier número de IOCs y CWSs, siempre y cuando no se sature la red de comunicación.

**Basado en eventos:** Todos los componentes de EPICS están diseñados para ser basados en eventos lo máximo posible. Por ejemplo, los clientes en vez de solicitar periódicamente los datos, puede ser solo notificado de los cambios. Permitiendo el uso eficiente de los recursos y mejorando la velocidad de respuesta.

**Alto rendimiento:** Un IOC puede procesar decenas de miles de datos por segundo. Servidores y clientes pueden mantener sistemas con millones de variables de procesos con una carga de red mínima.

**Escalable:** como sistema distribuido, puede escalar desde sistemas basados en un solo IOC y pocos clientes hasta grandes instalaciones con centenares de IOCS y millones de canales de entrada-salida.

**Robustez:** El fallo de un simple componente no hace caer el sistema entero. Cualquier componente puede cargarse o eliminarse sin interferir en el resto del sistema, en el caso de fallo de la red basta con que se retome la red para que todo vuelva a funcionar.

**Basado en variables de proceso:** en contraposición a otros softwares de control, EPICS no es orientado a objetos sino a variables de proceso, *Process Variable (PV)*. Donde un típico PV puede representar cualquier atributo o dato.

### 3.1.1.3 IOC – Input Output Controller

#### 3.1.1.3.1 Componentes

El núcleo de un IOC es una entidad software, un proceso, que contiene los siguientes componentes:

#### 3.1.1.3.2 Base de Datos del IOC:

Una base de datos en memoria persistente que contiene un conjunto de records. Éstos son los que mantienen los PVs mencionados anteriormente.

**Escáneres:** Son los mecanismos para procesar los *records* en la base de datos del IOC.

**Soporte para records:** Cada tipo de *record*, en adelante registros, tiene asociado un conjunto de rutinas de soporte para implementar la funcionalidad asociada al tipo.

**Soporte para dispositivos:** Existen rutinas de soporte que proporcionan entrada-salida de datos a la base de datos.

**Drivers para dispositivos:** Manejan el acceso a dispositivos externos.

**Acceso al canal o pvAccess:** Es la interfaz entre el mundo exterior y el IOC. Provee la interfaz para acceder a la base de datos de EPICS desde la red.

**Secuenciador:** Es una maquina de estados finitos, estrictamente hablando es un módulo externo y no está incluido en el núcleo de EPICS.

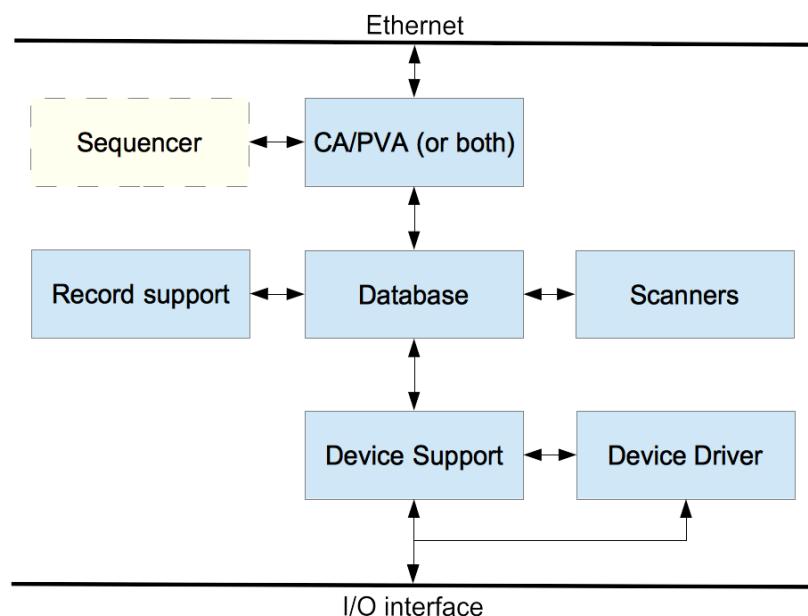


Ilustración 4 Componentes de un IOC (EPICS Control, 2019)

### 3.1.1.4 Base de datos

El núcleo de cada IOC es **una base de datos de procesos**. Esta base de datos reside en la memoria y no se guarda en un disco duro ni en otro dispositivo de memoria permanente. Es diferente de las bases de datos relacionales (SQL) más comunes.

La base de datos define la funcionalidad del IOC: qué datos de proceso proporciona, cómo se manejan y almacenan los datos. Puede contener **cualquier cantidad de registros**, cada uno correspondiente a un tipo específico. Cada tipo de registro define el tipo de datos que maneja y una serie de funciones que determinan cómo se gestionan esos datos.

Los registros incluyen metadatos específicos, también llamados “propiedades”, que se utilizan para configurar y respaldar la operación. Por ejemplo, un registro de entrada analógica (ai) permite leer valores de dispositivos de hardware y convertirlos en unidades de ingeniería deseadas.

También establece **límites de funcionamiento y activación alarmas** cuando se superan esos límites. EPICS admite una amplia variedad de tipos de registros, como *ai* (entrada analógica), *ao* (salida analógica), etc.

Los metadatos, conocidos como “campos”, se utilizan para configurar el comportamiento de los registros. Hay campos comunes a todos los tipos de registros y otros que son específicos de ciertos tipos. Cada registro tiene un nombre único y cada campo tiene un nombre propio. El nombre del registro debe ser único dentro de los IOC conectados a la misma subred TCP/IP para que el software cliente pueda descubrirlo y acceder a su valor y otros campos.

```
record(ai, "Cavity1:T") #type = ai, name = "Cavity1:T"
{
    field(DESC, "Cavity Temperature") #description
    field(SCAN, "1 second") #record update rate
    field(DTYP, "XYZ ADC") #Device type
    field(INP, "#C1 S4") #input channel
    field(PREC, "1") #display precision
    field(LINR, "typeJdegC") #conversion spec
    field(EGU, "degrees C") #engineering units
    field(HOPR, "100") #highest value on GUI
    field(LOPR, "0") #lowest value on GUI
    field(HIGH, "65") #High alarm limit
    field(HSV, "MINOR") #Severity of "high" alarm
}
```

Los registros de la base de datos pueden estar vinculados entre sí. Por ejemplo, los registros pueden obtener información de otros registros, activar el procesamiento de otros registros, habilitar o deshabilitar registros, entre otras funciones.

Al vincular una combinación de registros, la base de datos de **EPICS se convierte en una herramienta de programación**. Utilizando esto, se pueden lograr funciones muy sofisticadas con la base de datos. Además, como esta lógica reside en el IOC, no depende de ningún software cliente para funcionar. Aprovechando esto, muchos **programas clientes pueden ser “ligeros”** y simplemente mostrar o escribir los valores en los registros de la base de datos. La Figura 4 a continuación ilustra un ejemplo sencillo de vinculación de registros: si la temperatura promedio de los dos sensores T1 y T2 supera

los 10 grados, se activa el enfriador. Esta base de datos contiene cuatro registros: dos entradas analógicas (ai), una salida binaria (bo) y un cálculo (calc).

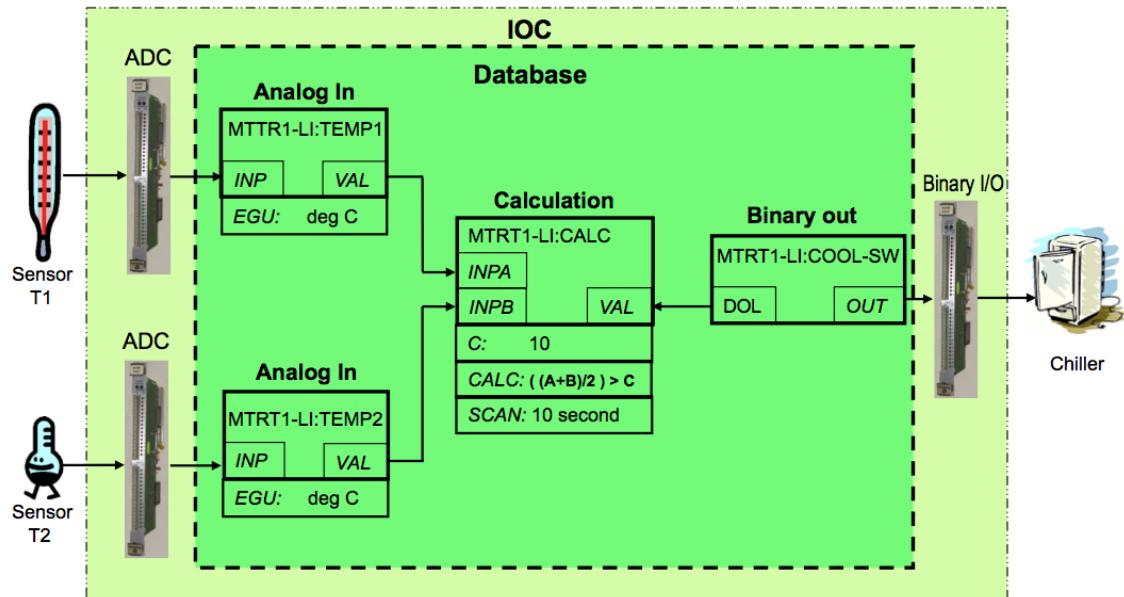


Ilustración 5 Ejemplo de enlazado de registros (EPICS Control, 2019)

#### 3.1.1.4.1 Escaneo

El escaneo de la base de datos es el mecanismo utilizado para procesar un registro. El procesamiento implica que el registro realice su tarea, como por ejemplo leer un canal de entrada/salida, convertir el valor leído a unidades de ingeniería, adjuntar una marca de tiempo al valor o verificar los límites de alarma. La forma en que se manejan los datos cuando se procesa un registro depende del tipo de registro.

Se proporcionan cuatro tipos básicos de escaneo de registros: Periódico, Evento, Evento de E/S (entrada/salida) y Pasivo. Todos estos métodos pueden combinarse en un IOC.

- **Periódico:** Un registro se procesa periódicamente. Se admiten varios intervalos de tiempo, típicamente desde 10 Hz hasta 0.01 Hz. Los rangos son configurables para admitir tasas más altas o más bajas.
- **Evento:** El escaneo por eventos ocurre cuando cualquier componente de software del IOC envía un evento (software EPICS), como un nuevo valor de medición de un sensor de temperatura.
- **Evento de E/S:** El sistema de escaneo por eventos de E/S procesa registros en función de eventos externos, como interrupciones de procesador. Debe estar disponible una rutina de interrupción del controlador del dispositivo IOC para aceptar las interrupciones externas. Sin embargo, un evento de E/S no necesariamente tiene que ser una interrupción en el sentido tradicional de una interrupción de la CPU.
- **Pasivo:** Los registros pasivos no se escanean regularmente ni en función de eventos. Sin embargo, pueden procesarse como resultado cuando se procesan otros registros vinculados a ellos, o como resultado de cambios externos, como nuevos valores establecidos a través de la red utilizando Channel Access.

### 3.1.1.4.2 Soporte a “Records”, dispositivos y drivers

El acceso a la base de datos no requiere conocimientos específicos del tipo de registro, cada tipo de registro proporciona un conjunto de rutinas de soporte que implementan todo el comportamiento específico del registro. Esto permite que los **IOCs admitan un número arbitrario de registros y tipos de registros.**

Del mismo modo, el soporte de registro no contiene conocimientos específicos del dispositivo, lo que brinda a cada tipo de registro la capacidad de tener varios módulos que dan soporte a distintos dispositivos independientes.

El diseño del software IOC permite que una instalación particular e incluso un IOC específico dentro de una instalación elijan un conjunto único de tipos de registros, tipos de dispositivos y controladores. El resto del software del sistema IOC no se ve afectado.

Para tener una visión general de cómo funciona la separación, veamos las tareas del soporte de registro. Cada módulo de soporte de registro debe proporcionar una rutina de procesamiento de registros que será llamada por los escáneres de la base de datos. El procesamiento de registros consiste en una combinación de las siguientes funciones (no todos los tipos de registros necesitan todas las funciones):

- **Entrada:** Leer entradas. Las entradas se pueden obtener, a través de rutinas de soporte de dispositivo, desde hardware, desde otros registros de la base de datos a través de enlaces de base de datos o desde otros IOCs a través de enlaces *Channel Access* (CA) o *pvAccess* (PVA).
- **Conversión:** Conversión de entrada sin procesar a unidades de ingeniería o de unidades de ingeniería a valores de salida sin procesar.
- **Salida:** Escribir salidas. La salida se puede dirigir, a través de rutinas de soporte de dispositivo, hacia el hardware, hacia otros registros de la base de datos dentro del mismo IOC a través de enlaces de base de datos o hacia otros IOCs a través de enlaces CA o PVA.
- **Generación de alarmas:** Verificar y generar alarmas.
- **Monitorización:** Activar monitores relacionados con devoluciones llamada CA o PVA.
- **Enlace:** Activar el procesamiento de registros vinculados.

El mismo concepto se aplica a los módulos de soporte de dispositivo y controlador de dispositivo: cada módulo de soporte debe definir un conjunto de funciones para formar parte del software del IOC.

### 3.1.1.5 Monitores

El mecanismo para enviar notificaciones cuando cambia el valor de una base de datos se llama “monitores de base de datos”. La función de monitorización permite que un **programa cliente sea notificado cuando cambian los valores** de la base de datos **sin tener que consultar** constantemente la base de datos. Estos se pueden configurar para especificar cambios de valor, cambios de alarma y/o cambios de archivo.

Los monitores de base de datos son compatibles con los protocolos estándar de EPICS *Channel Access* y *pvAccess*.

### 3.1.1.6 Protocolos de red

EPICS proporciona acceso transparente a través de la red a las bases de datos del IOC al admitir los siguientes protocolos de red para el intercambio de datos:

### 3.1.1.6.1 Channel Access

Channel Access, CA, se basa en un **modelo cliente-servidor**. Cada IOC provee un servidor CA capaz de establecer comunicación con una cantidad arbitraria de clientes. Los clientes CA están disponibles tanto en los IOCs como en los CWSs y pueden comunicarse con cualquier número de servidores.

#### 1.1.1.1.1.1 Servicios del cliente

Los servicios básicos que provee un cliente CA son:

- **Search:** Localizar los IOCs que contienen variables de proceso seleccionadas y establecer comunicación con cada uno.
- **Get:** Obtener el valor más información adicional opcional para un conjunto seleccionado de variables de proceso.
- **Put:** Cambiar los valores de las variables de proceso seleccionadas.
- **Monitor:** Solicitar al servidor que envíe información solo cuando la variable de proceso asociada cambie de estado. Se pueden solicitar cualquier combinación de los siguientes cambios de estado: cambio de valor, cambio de estado y gravedad de la alarma, y cambio de valor de archivo. Muchos tipos de registros proporcionan factores de histéresis para los cambios de valor.

Además de los valores de las variables de proceso, se puede solicitar cualquier combinación de la siguiente información adicional (“metadatos”):

- **Status:** Estado de la alarma y gravedad.
- **Units:** Unidades de ingeniería para esta variable de proceso.
- **Precision:** Precisión con la que se muestran los números de punto flotante.
- **Timestamp:** Hora en la que se procesó por última vez el registro.
- **Enumeration:** Un conjunto de cadenas ASCII que define el significado de los valores enumerados.
- **Graphics:** Límites altos y bajos para configurar widgets y gráficos en una interfaz gráfica de usuario (GUI).
- **Control:** Límites de control altos y bajos; límites operativos para el registro.
- **Alarm:** Los estados de alarma (HIHI, HIGH, LOW y LOLO) y gravedad para la variable de proceso.

#### Búsqueda del servidor

El CA provee al IOC un servidor residente, el cual está esperando *mensajes de búsqueda CA*. Éstos son mensajes de difusión UDP que son generados por un cliente CA, como por ejemplo cuando una interfaz de operador se inicia, y busca los procesos de variable que usa. El servidor acepta todos los mensajes de búsqueda y revisa si esos PVs se encuentran en su IOC, si los encuentra responde al cliente “*Yo lo tengo*”.

#### Connection Request Server

Una vez localizadas las variables de proceso, el cliente de Channel Access emite solicitudes de conexión para cada IOC que contiene las variables de proceso que el cliente utiliza. El servidor de solicitud de conexión, en el IOC, acepta la solicitud y establece una conexión con el cliente. Cada conexión es gestionada por dos tareas separadas: *ca\_get* y *ca\_put*. Las solicitudes de *ca\_add\_event* resultan en el establecimiento de monitores de la base de datos. El acceso a la base de datos y/o las

rutinas de soporte de registros proporcionan las actualizaciones de valores (monitores) mediante una llamada a `db_post_event`.

#### *Gestión de conexión*

Cada IOC proporciona un servicio de gestión de conexiones. Si un servidor de Channel Access falla (por ejemplo, si el IOC se bloquea), se notifica al cliente y, si un cliente falla (por ejemplo, si su tarea se bloquea), se notifica al servidor. Si un cliente falla, el servidor rompe la conexión. Si un servidor se bloquea, el cliente restablece automáticamente la comunicación cuando el servidor se reinicia.

#### **3.1.1.6.2 pvAccess**

`pvAccess` es el **sucesor moderno** y una **alternativa** a *Channel Access* disponible en EPICS 7. `pvAccess` agrega una serie de capacidades a EPICS que amplían el conjunto de servicios proporcionados por *Channel Access*. Con `pvAccess`, los datos estructurados se pueden **transportar con alta eficiencia** y es capaz de manejar **grandes conjuntos de datos**; esto se ha logrado con una serie de optimizaciones:

- La introspección de la estructura de datos y el transporte de datos se han separado para que la información de estructura solo se transmita una vez por conexión.
- Los monitores envían solo los elementos de una estructura de datos que han cambiado.
- Se han realizado varias optimizaciones internas en la manipulación de datos (reducción de copias, etc.). En las pruebas de aplicación, `pvAccess` ha logrado utilizar entre el **96% y el 99% del ancho de banda teórico disponible de un enlace Ethernet de 10 Gbit**, lo cual se acerca al límite de lo que se puede lograr en la práctica.

#### *Servicios del cliente*

Los servicios básicos del cliente `pvAccess` son similares a *Channel Access*, con un par de **mejoras**:

- **Search:** Localizar los IOC que contienen las variables de proceso de interés y establecer comunicación con cada uno.
- **Get:** Obtener el valor y otra información opcional adicional para un conjunto seleccionado de variables de proceso.
- **Put:** Cambiar los valores de variables de proceso seleccionadas.
- **Add Monitor:** Agregar una devolución de llamada de cambio de estado, similar a *Channel Access*.
- **PutGet:** Cambiar el valor de un PV, procesar el registro EPICS y leer de nuevo el valor en una operación atómica.
- **ChannelRPC:** Un patrón de comunicación de “Llamada de Procedimiento Remoto”. Es similar a *PutGet*, pero la comunicación es asimétrica, es decir, los datos enviados por el cliente (“solicitud”) son diferentes de la estructura de datos que el servidor envía de vuelta. Este patrón se puede describir como una consulta con parámetros.

Para el IOC, un servidor residente del IOC (**`qsrsv`**) proporciona la interfaz para acceder a los registros de la base de datos de procesos. El acceso básico a un único PV proporciona la función equivalente a *Channel Access*. Además, `qsrsv` brinda la posibilidad de crear estructuras de datos que combinan información de diferentes registros de la base de datos en unidades que se transportan como un todo. Desde EPICS 3.16, el núcleo del IOC es capaz de garantizar un acceso atómico a los registros, lo que significa que los datos en la estructura que `qsrsv` proporciona están garantizados como resultado de un solo procesamiento (o, dicho de otra manera, los registros no cambian sus valores mientras `qsrsv` está ensamblando la estructura de datos). Esto también se aplica a las escrituras, lo que significa que todos los valores se escriben en los registros correspondientes antes de que se procesen los registros. De esta manera, se puede garantizar la coherencia de los parámetros para una operación.

### *Búsqueda del servidor*

Al igual que en Channel Access, qsrv espera mensajes de búsqueda. El servidor acepta todos los mensajes de búsqueda (UDP), verifica si alguna de las variables de proceso se encuentra en este IOC y, si se encuentra alguna, responde al remitente con un mensaje de “Yo lo tengo”.

### *Servidor de Solicituds de Conexión*

En *pvAccess*, el proceso de cómo un cliente y un servidor establecen el canal de comunicación es ligeramente diferente a *Channel Access* y consta de dos etapas:

1. La primera etapa consiste en intercambiar datos de introspección. En esta etapa, el servidor comunica al cliente la estructura de los datos que se intercambiarán. Ambos lados pueden crear las estructuras de marcador necesarias para la comunicación.
2. En la segunda etapa, los datos reales pueden intercambiarse utilizando las estructuras de datos asignadas.

### *Gestión de conexión*

*pvAccess* proporciona un servicio de gestión de conexiones similar a *Channel Access*.

(EPICS Control, 2019)

### *3.1.1.7 Módulo de comunicación (Driver)*

Una vez que se ha establecido de manera precisa el concepto de EPICS y se han explicado exhaustivamente los principios fundamentales que rigen su funcionamiento, procederemos a presentar el módulo específico diseñado para brindar el respaldo necesario en cuanto a la conexión con el autómata programable encargado de la gestión del proceso industrial en cuestión.

Pero antes de enseñar el módulo usado, ha de tenerse en cuenta que se planteó previamente otra posibilidad:

#### *3.1.1.7.1 Módulo S7nodave*

El *s7nodave* para EPICS es un soporte de dispositivo basado en las librerías *Asyn* y *libnodave* que se comunica directamente con PLCs S7 (o compatibles).

A diferencia de otros soportes de dispositivo de EPICS para los PLCs S7, este soporte de dispositivo **no requiere ninguna programación especial en el lado del PLC**.

En cambio, los registros de EPICS simplemente especifican la dirección de memoria en el PLC y el soporte de dispositivo utiliza el protocolo ISO-TCP compatible con la mayoría de los PLCs S7 para leer o escribir los datos del canal.

(Marsching, 2013)

#### *3.1.1.7.2 Módulo OPCUA*

Este es el módulo elegido para este proyecto. Fue creado y sigue bajo desarrollo por Ralph Lange, coordinador de sistemas de control en ITER.

Este módulo hace de interfaz con el protocolo *OPC UA* y puede ser instalado tanto en Linux como en Windows.

### *3.1.1.8 Interfaz del operador*

Por último, pero no menos importante, es necesario una interfaz gráfica que facilite el acceso y control del proceso industrial, además de que aporte información de la conexión con éste. Existe un

gran repertorio, tanto de herramientas como de posibilidades, para crear interfaces para EPICS. En su página sobre extensiones (EPICS Control, 2019), se encuentra una lista de las más conocidas:

- **caQtDM**: A Display Manager in the spirit of MEDM (C++, Qt)
- **CS-Studio**: Control System Studio (Java)
- **pyDM**: Python-based Display Manager: (Python, Qt)
- **EDM**: Extensible Display Manager (C++, Motif)
- **MEDM**: Motif Editor and Display Manager (C, Motif)
- **StripTool**: Strip-chart plotting tool (C/C++, Motif)
- **Probe**: Motif Channel Monitoring program
- **React Automation Studio** (Javascript, React)

Para este proyecto se ha decantado por la actual variante de *CS-Studio*, **Phoebus**, la cual elimina ciertas dependencias que hacen que su instalación y uso sean más sencillas que el CSS tradicional.

#### 3.1.1.8.1 Phoebus

*Phoebus* es un marco de trabajo y una colección de herramientas para monitorear y operar sistemas de control a gran escala, como los utilizados en la comunidad de aceleradores. *Phoebus* es una actualización del conjunto de herramientas Control System Studio que elimina las dependencias de Eclipse RCP y SWT (conjunto de componentes para construir interfaces gráficas en Java).

### 3.1.2 WinCC OA

WinCC OA es la abreviación de **SIMATIC WinCC Open Architecture**, un paquete de software diseñado para su uso en automatización. La principal área de aplicación es la operación y control de plantas técnicas utilizando estaciones de trabajo VDU (*visual display unit*) con capacidad gráfica completa.

Esta aplicación requiere, además de la presentación de estados de proceso actuales, la posibilidad de transferir condiciones y comandos al proceso y su equipamiento de control. El usuario utiliza **ratón, teclado y otros dispositivos** de entrada de manera interactiva con una **respuesta inmediata** en la pantalla. Las funciones principales también incluyen la generación de **alertas** en caso de condiciones críticas o superación de un umbral, así como el **archivado histórico** de datos para su posterior **presentación e interpretación**.



Ilustración 6 WinCC OA Logo

Este tipo de sistemas se conocen principalmente como **sistemas de control, sistemas de visualización** o, de acuerdo con el acrónimo angloamericano, SCADA o HMI. SCADA es la abreviatura de *Supervisory Control And Data Acquisition*, que describe muy bien la forma en que opera el paquete de programas. HMI significa *Human Machine Interface*, que se refiere a la interfaz entre el humano y la máquina.

Por lo tanto, WinCC OA es **un software de supervisión para el centro de control o la operación de máquinas**. Se utilizan servidores y estaciones de trabajo basadas en PC como plataforma de hardware. Junto con los sistemas de control de la automatización básica (PLC, DDC, RTU, etc.) y sus sensores y módulos de E/S, **forma un sistema de automatización completo**.

(SIMATIC WinCC OA, 2023)

#### 3.1.2.1 Arquitectura

WinCC OA es un sistema **modular**. Las unidades que gestionan funcionalidades específicas se llaman "**managers**". Estos *managers* tienen su propio proceso, computacionalmente hablando.

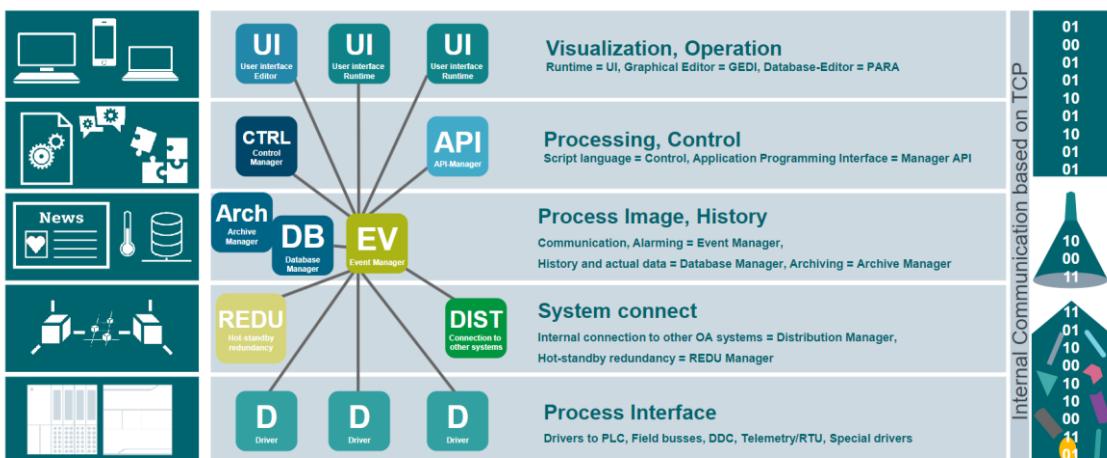


Ilustración 7 - Arquitectura de WinCC OA (SIMATIC WinCC OA, 2023)

##### 3.1.2.1.1 Interfaz de procesos

Los módulos de interfaz de proceso, se llaman **Drivers** (D), forman el nivel más bajo en un sistema WinCC OA. Los *drivers* son programas especiales que manejan la comunicación con el nivel de control y campo. Dado que existen numerosas formas de comunicación diferentes con PLC o nodos de

control remoto, hay varios controladores disponibles. En el caso de este TFG, nos interesa el *driver OPC*.

### 3.1.2.1.2 Imagen de proceso

El **centro de procesamiento central** en WinCC OA se llama **Event Manager** (EV). Esta unidad siempre mantiene **una imagen actualizada de todas las variables** del proceso en la memoria. Cada *manager* que desee acceder a los datos recibe los datos de la imagen del proceso del *event manager* y no tiene que comunicarse directamente con un controlador.

El **Event Manager** es una especie de distribuidor central de datos, el **centro de comunicación de WinCC OA**. Además, este manager también ejecuta el manejo de **alertas** y es capaz de ejecutar diferentes **funciones de cálculo de forma autónoma**.

#### Histórico

El **Data Manager** da soporte al *Event Manager*. El *Data Manager* es el **enlace con una base de datos**. Maneja los datos de configuración de una aplicación que se guardan en la base de datos. Además, los datos históricos, como cambios de valor y alertas, se guardan en una base de datos. Si se desea consultar datos históricos más tarde, la consulta también la ejecuta el *Data Manager* y no la base de datos en sí misma.

### 3.1.2.2 Procesamiento, Control

En WinCC OA, existen **numerosas posibilidades** para implementar **algoritmos y procesamientos propios**. El lenguaje interno de control (**CTRL**) y la interfaz de programación de aplicaciones (**API**, por sus siglas en inglés) son los más importantes.

#### 3.1.2.2.1 CTRL

*Control* es un potente lenguaje de script. Es **interpretado**, por lo que no se requiere compilación. La sintaxis se corresponde con el **ANSI-C** (un lenguaje de programación de alto nivel estandarizado y ampliamente utilizado a nivel internacional) con algunas modificaciones que lo simplifican. Es un **lenguaje de alto nivel procedural** avanzado con soporte de multitarea (procesamiento paralelo de programas individuales, el control del procesamiento es ejecutado por el sistema mismo). El lenguaje proporciona una amplia biblioteca de funciones para tareas de control y técnicas de visualización. *Control* se puede utilizar como un proceso independiente (*manager* de *CONTROL*) para animación y diseño de interfaz de usuario (*manager* de UI), o para procesamiento estandarizado basado en objetos de datos (*manager* de eventos).

#### 3.1.2.2.2 API

La **API** (API de WinCC OA) presenta la forma más potente de extensiones de funciones. Es una biblioteca de clases en **C++** y permite al desarrollador de software **implementar funciones** individuales como un **manager adicional independiente** (sistema de pronóstico, simulación, herramientas de planificación, bases de datos propietarias), entre otras posibilidades.

(ETM Professional Control GmbH, 2023)

### 3.1.2.3 Concepto de DataPoints – Imagen de proceso

Las variables del proceso que deben ser controladas y monitorizadas deben estar presentes en el software de la sala de control. Debe haber una variable que represente el valor de cada estado lógico, cada valor medido o valor establecido dentro del sistema.

A estas variables de proceso se le llaman **Data Points**. También se le suelen dar otros nombres dependiendo de la región o producto como etiquetas, *process variable*, PV, elemento...

A diferencia de los sistemas SCADA comunes que asignan un *data point* a cada variable del proceso, WinCC OA elige otro enfoque: casi toda la información del proceso pertenece lógicamente a un conjunto más o menos complejo, que es **un dispositivo**.

En lugar de crear variables independientes para los valores unidos lógicamente, WinCC OA define puntos de datos estructurados orientados a dispositivos. Los puntos de datos se definen en una estructura de árbol con niveles de ramificación arbitrarios.

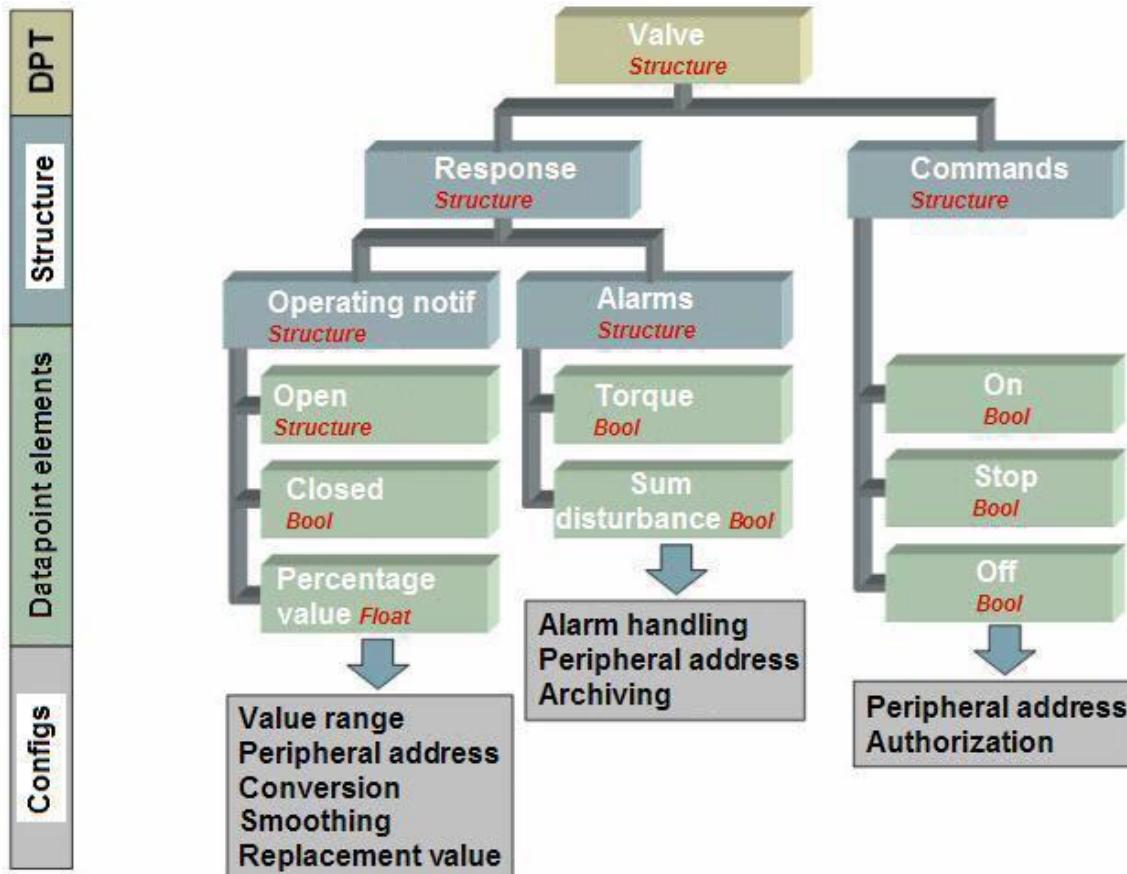


Ilustración 8 - Estructura en árbol de un Data Point orientado a dispositivo (SIMATIC WinCC OA, 2023)

Cada *data point* puede representar un dispositivo real o una combinación lógica de información. Está compuesto por uno o varios *data point elements* estructurados de manera casi arbitraria. Cada *data point* pertenece a su *Data Point Type* y se muestra en el módulo PARA. Al hacer doble clic en el nombre del tipo de punto de datos o en el símbolo "+" delante del nombre, se abre una lista de todos los puntos de datos de ese tipo.

La siguiente imagen muestra dos tipos disponibles de forma predeterminada y sus instancias de *data points*. El *Data Point Type ExampleDP\_Float* tiene **4 data points** en este caso. Este *Data Point Type* consta de solo un elemento de punto de datos y, por lo tanto, solo puede representar una variable de proceso. Esto corresponde al modelo de datos común de la mayoría de los sistemas SCADA, pero

es una excepción en WinCC OA. Este tipo (y también los otros tipos *ExampleDP\**) están disponibles únicamente con fines de **prueba o demostración**.

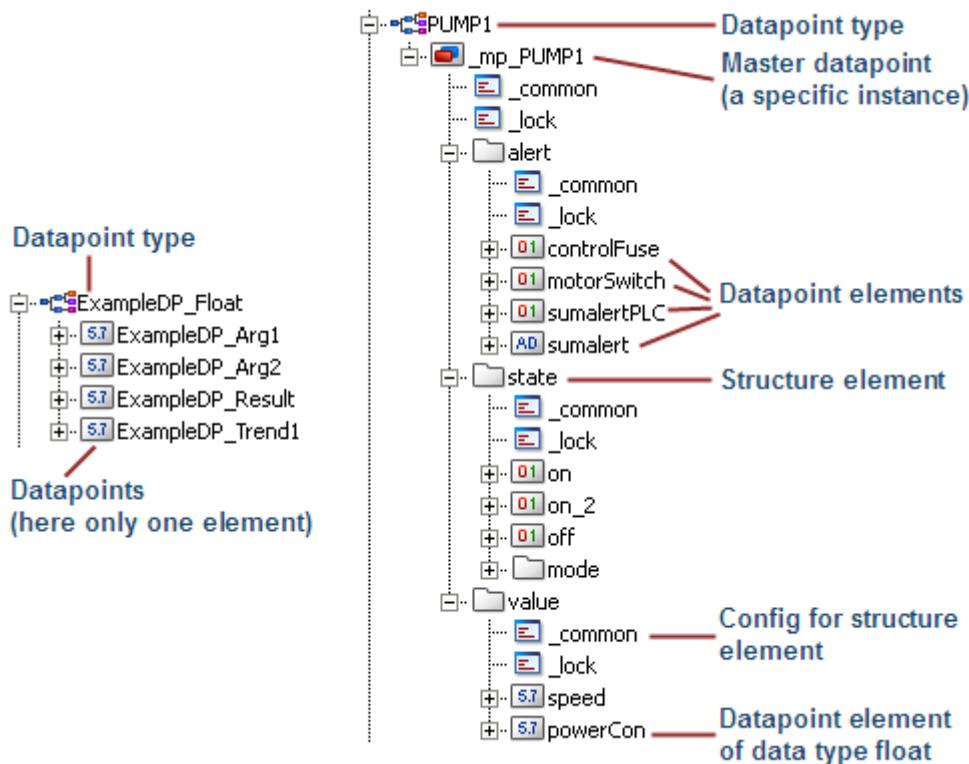


Ilustración 9 - Data points *Float* y *Pump* (SIMATIC WinCC OA, 2023)

El Data Point Type **PUMP1** (mostrado a la derecha) corresponde más a la estructura común en WinCC OA. Numerosas variables de proceso forman parte del *data point* orientado al dispositivo. Al inicio, **PUMP1** tiene solo una instancia *data point*. Incluso esto es una excepción: es un **Master Data Point**, **MP**, que puede mostrar cierta información de configuración para el tipo.

### 3.1.2.4 Conexión con el proceso industrial

Como se ha comentado anteriormente, del mismo modo que EPICS se conectará *WinCC OA* con el proceso mediante el protocolo *OPC UA*.

## 3.2 Software de simulación

### 3.2.1 Factory IO

Factory I/O es un programa de simulación de fábricas en tres dimensiones.

Factory I/O es un software de simulación interactivo utilizado en la industria para crear y simular entornos de fabricación en 3D. Permite a los usuarios diseñar y simular sistemas de automatización industrial, como líneas de producción, máquinas y procesos, en un entorno virtual.



Ilustración 10 - Factory IO Logo

Se utiliza para entrenar a estudiantes, ingenieros y técnicos en el campo de la automatización industrial. Proporciona una forma segura y eficiente de aprender sobre sistemas de control, programación de PLC y procesos de fabricación sin la necesidad de interactuar con equipos físicos reales. Los usuarios pueden crear escenarios, programar lógica de control y observar cómo se comportan los sistemas en tiempo real dentro del entorno virtual.

Sin embargo, también se puede utilizar con microcontroladores o SoftPLC entre muchas otras tecnologías.

Para este caso, este programa es muy adecuado ya que se pretende crear un proceso industrial que se adecúe a los requerimientos de este TFG y así poderlo monitorizar con EPICS y WinCC OA, controlado por un autómata programable.

Factory IO permite una comunicación sencilla con cualquier tipo de autómata, además de la creación de casi cualquier escenario industrial gracias a su amplia gama de objetos, sensores y actuadores.

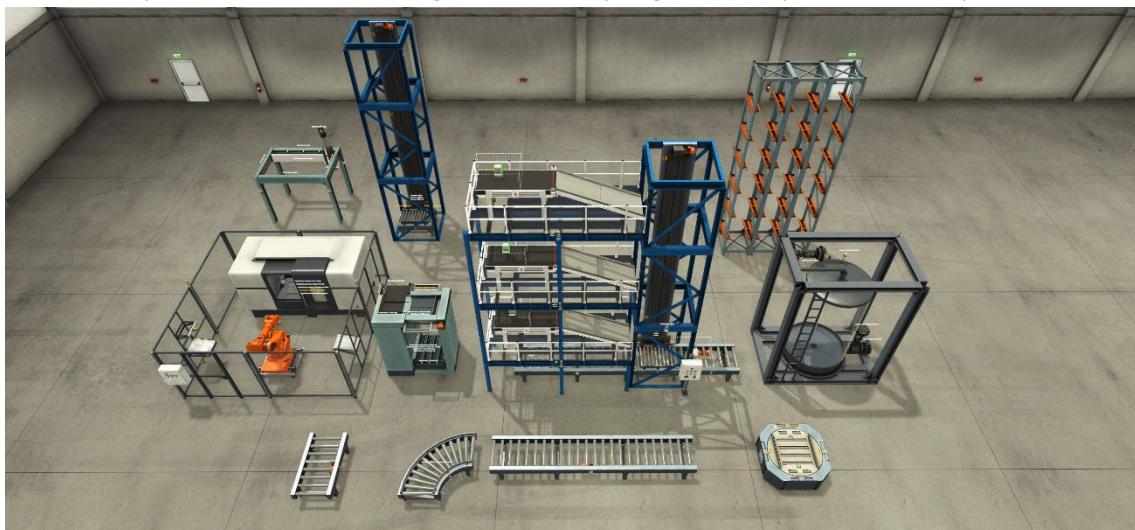


Ilustración 11 - Factory IO Escenario con algunos componentes y estaciones

### 3.2.2 PLCSim

Los PLCs industriales son microcontroladores que se utilizan para automatizar procesos industriales, como máquinas, líneas de producción, sistemas de iluminación, ventilación, sistemas hidráulicos, etc.

Pero son **costosos** debido a múltiples factores. En primer lugar, están diseñados para **resistir condiciones industriales extremas**, lo que requiere componentes de alta calidad y materiales resistentes, aumentando los costos. La necesidad de cumplir con **rigurosas certificaciones y estándares industriales** también contribuye a su costo.



Ilustración 12 -  
PLCSim Logo

Además, el diseño y desarrollo especializado para asegurar la confiabilidad en entornos industriales, junto con el soporte técnico y servicio postventa, agrega gastos significativos.

Por esta razón, interviene **PLCSim**. Es un software de simulación de controladores lógicos programables de Siemens y desarrollado por ellos. Permite simular y probar programas de control en un entorno virtual antes de implementarlos en hardware real. Esto es especialmente útil en el desarrollo y la depuración de sistemas de automatización industrial.

Para este *TFG*, es especialmente favorecedor por permitir que todo el proyecto se gestione en un entorno simulado, facilitando el estudio de los SCADAs.

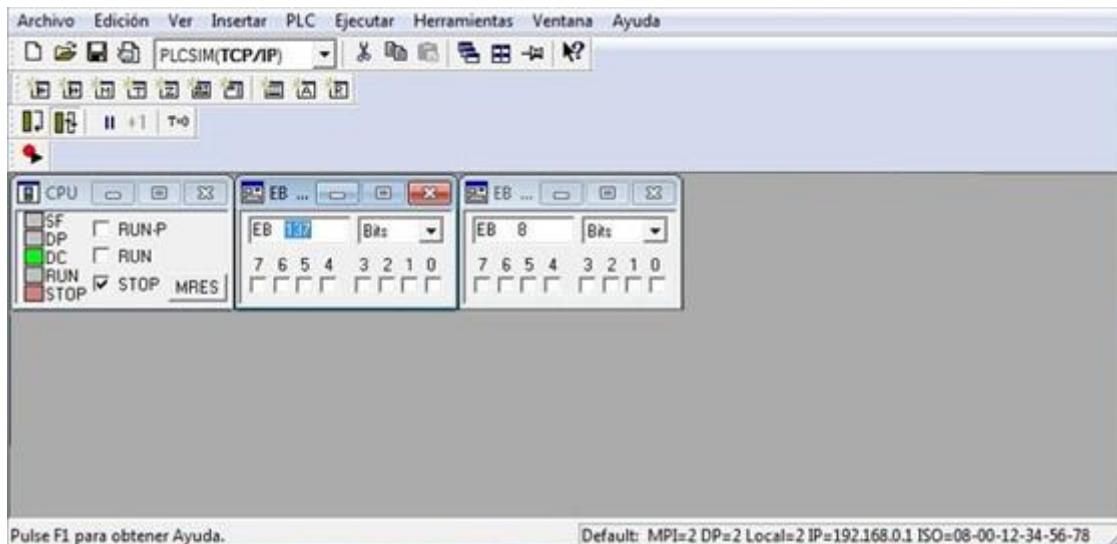


Ilustración 13 - Interfaz PLCSim

### 3.3 Entornos de programación PLC

#### 3.3.1 TiaPortal

TIA Portal (*Totally Integrated Automation Portal*) es un entorno de ingeniería de Siemens diseñado para la programación, configuración y diagnóstico de sistemas de automatización industrial. Es una plataforma integral que abarca desde la planificación y diseño hasta la puesta en marcha y el mantenimiento de sistemas de automatización.



Ilustración 14 - TIA Portal V17 Logo

Ofrece herramientas de programación intuitivas y potentes para diferentes familias de PLCs de Siemens, como SIMATIC S7-1200, S7-1500, S7-300 y otros. Los ingenieros pueden crear programas de control, simular su funcionamiento y cargarlos en hardware real.

También permite la simulación para probar y depurar programas de control en un entorno virtual antes de implementarlos en hardware real. Esto ayuda a identificar y corregir errores antes de la puesta en marcha.

Proporciona una interfaz de usuario unificada que permite a los ingenieros trabajar en diferentes aspectos de un proyecto de automatización en una sola plataforma. Esto agiliza el proceso de desarrollo y facilita la colaboración entre equipos.

### 3.3.1.1 Lenguajes de programación

Existen varios lenguajes de programación utilizados en los PLCs de Siemens: KOP, FUP, SCL, GRAPH y AWL. Cada lenguaje ofrece distintas ventajas y se elige según las necesidades de la aplicación, desde lógica secuencial y modularidad hasta algoritmos complejos y control de bajo nivel. La elección adecuada del lenguaje permite el diseño y la implementación eficiente de sistemas de control industriales.

- **KOP** (Kontaktplan, en alemán)

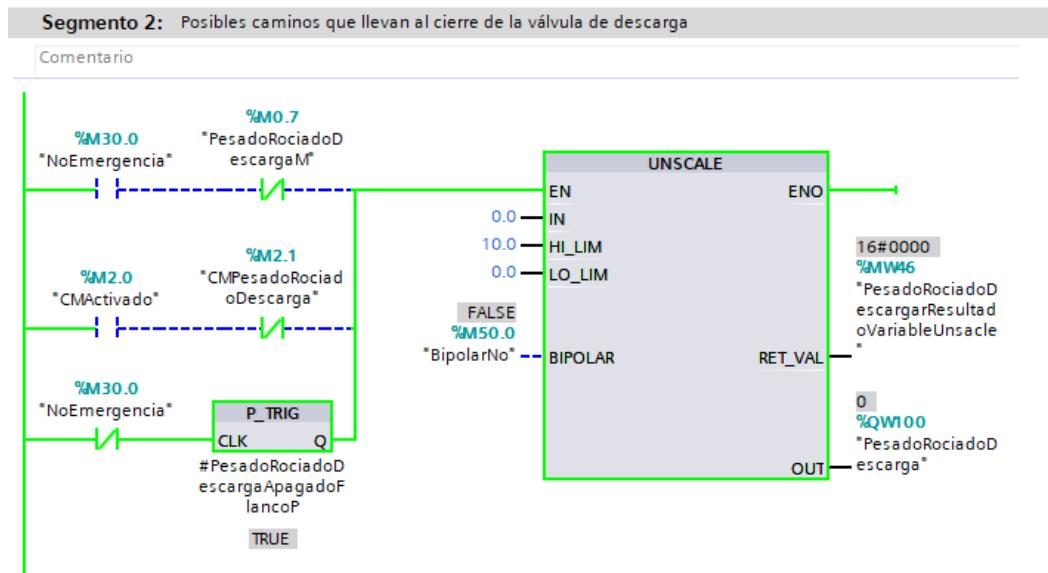


Ilustración 15 - Segmento 2 del bloque 3PesadoRociadoSalidas en KOP

El lenguaje KOP es un lenguaje gráfico **ampliamente utilizado** en la programación de PLCs. Es especialmente útil cuando se trabaja con **lógica de contactos y circuitos de relés**. KOP se basa en la **representación de circuitos eléctricos** mediante **contactos y bobinas**, y es **visualmente intuitivo** para aquellos familiarizados con esquemas de **circuitos eléctricos**. Es adecuado para aplicaciones que implican principalmente lógica secuencial y operaciones discretas.

- **FUP** (Funktionsplan, en alemán):

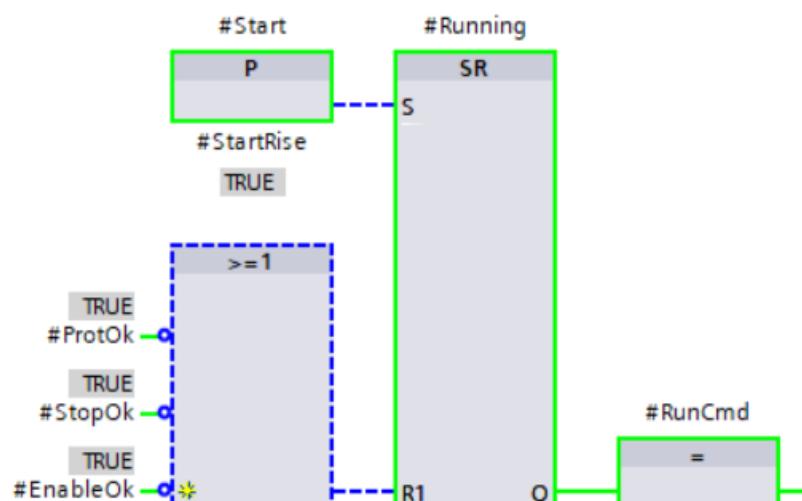


Ilustración 16 - Fragmento ejemplo de FUP

El lenguaje FUP es otro lenguaje gráfico utilizado en la programación de PLCs. Permite representar **funciones y algoritmos** de control mediante bloques interconectados. FUP es útil cuando se necesita modularidad y reutilización de código, ya que los bloques se pueden crear y utilizar en diferentes partes del programa.

- **SCL (Structured Control Language):**



Ilustración 17 - Código del bloque *ClasificadoraSCL*

El lenguaje SCL es un **lenguaje de programación de alto nivel basado en texto**, similar a otros lenguajes de programación estructurados como PASCAL. Es especialmente adecuado para implementar **algoritmos matemáticos complejos y operaciones de manipulación de datos**. SCL ofrece una mayor flexibilidad y un mayor nivel de abstracción en comparación con los lenguajes gráficos.

- **GRAPH (GRAFCET):**

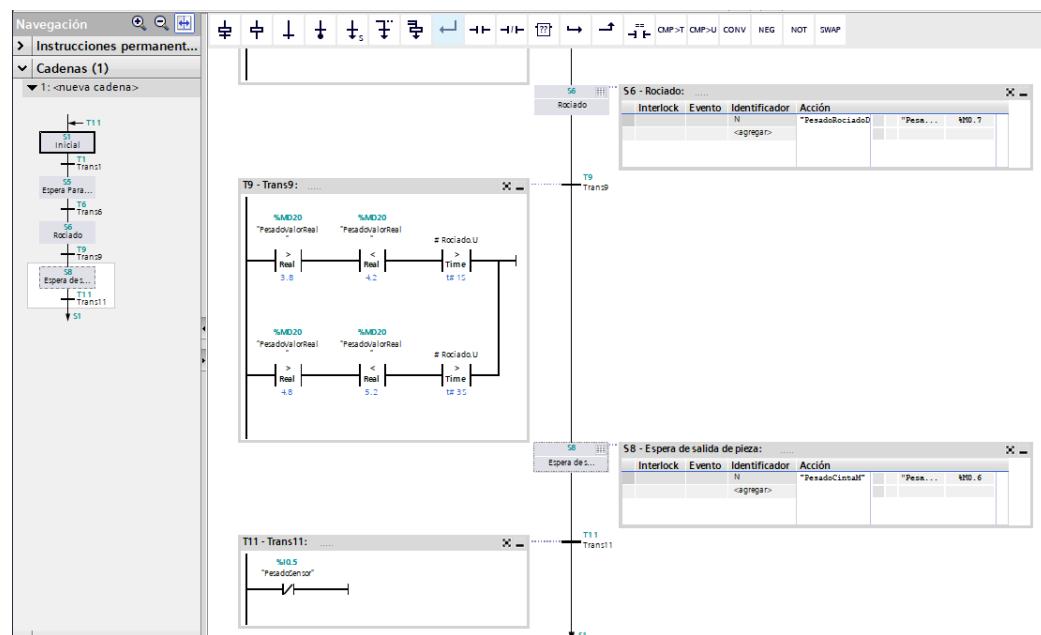


Ilustración 18 - Fragmento del bloque Pesado-Rociado en GRAPH

El lenguaje GRAPH o GRAFCET es un lenguaje gráfico utilizado para modelar y programar sistemas de control secuenciales complejos. Se basa en el concepto de estados y transiciones y es especialmente útil para describir y programar máquinas de estados. GRAPH

proporciona una representación visual clara y una estructura lógica para sistemas de control secuenciales complejos.

- **AWL** (Anweisungsliste, en alemán) o Statement List:

```

1   A      "BipolarNo"           $M50.0
2   =      $L1.0                $L1.0
3   BLD   103                 103
4   A(
5   A      "NoEmergencia"        $M30.0
6   AN    "PesadoRociadoDescargaM"
7   O
8   A      "CMActivado"         $M2.0
9   AN    "CMPesadoRociadoDescarga"
10  O(
11  AN    "NoEmergencia"        $M30.0
12  FP    #PesadoRociadoDescargaApagadoFlancoP
13  )
14  )
15  JNB   Label_1
16  CALL  UNSCALE
17  IN    :=0.0                0.0
18  HI_LIM :=10.0              10.0
19  LO_LIM :=0.0                0.0
20  BIPOLE :=$L1.0             $L1.0
21  RET_VAL :="PesadoRociadoDescargarResultadoVariableUnsacle"
22  OUT   :="PesadoRociadoDescarga" $MW46
23  Label_1 : NOP 0           $QW100

```

Ilustración 19 - Fragmento del segmento 2 de PesadoRociadoSalidas traducido a AWL

El lenguaje AWL o Statement List es un lenguaje de programación textual **similar al ensamblador**. Proporciona un **control de bajo nivel** y se utiliza para programar tareas de **temporización crítica** y operaciones de bajo nivel. AWL permite un acceso directo a características específicas del hardware del PLC y es adecuado para programadores con experiencia en lenguajes de bajo nivel. Sin embargo, debido a esta naturaleza, AWL puede ser más difícil de aprender y de depurar en comparación con los lenguajes gráficos y SCL.

En la ilustración 25, se muestra el mismo segmento que la ilustración 21 pero traducido por TIA PORTAL a AWL.

## 3.4 Comunicaciones

### 3.4.1 KepServer EX

KepServerEx es el servidor *OPC UA* que va a actuar como intermediario entre el PLC y las aplicaciones SCADA. Se va a encargar de recopilar los datos del PLC, como sensores, actuadores y zonas de memoria, para proporcionarlos a las aplicaciones cliente. También va a recibir comandos de estas aplicaciones y transmitirlos al PLC para llevar a cabo acciones específicas.

La principal ventaja que ofrece KepServerEx es que es un software único, es decir, todo lo integra en una misma aplicación con una única interfaz haciendo que su configuración sea rápida y sencilla.

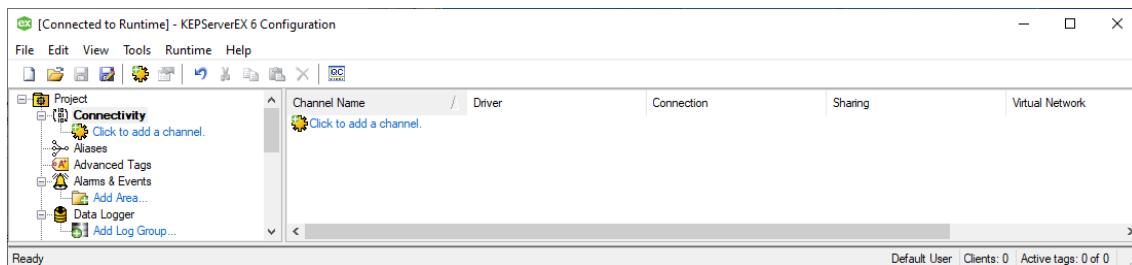


Ilustración 21 - Panel principal KepServer sin conexiones

### 3.4.2 NetToPLCsim

Debido a que **PLCSim** concentra su campo de uso en la simulación de uno o distintos autómatas programables en un ámbito **local**, no contempla la comunicación con agentes externos.

**NetToPLCsim** aporta la **interfaz de comunicación** y abre la posibilidad de incorporarlo a red, además del monitoreo de paquetes.

Según su documentación, la interfaz S7online se encarga de gestionar la comunicación entre las aplicaciones Simatic y los PLCs, manejando las capas OSI 1 a 4 para la comunicación dentro del ecosistema Simatic. Permite que las aplicaciones Simatic se comuniquen con los PLCs utilizando varias capas de transporte subyacentes como TCP/IP, MPI o Profibus. Esta interfaz es accesible a través de la biblioteca de programas "s7onlinx.dll" ubicada en el directorio del sistema Windows.

El papel de NetToPLCsim es actuar como intermediario entre la capa de transporte IP/IsoOnTcp y la interfaz S7online. Se encarga de pasar los datos entre estas capas, permitiendo la comunicación entre las aplicaciones Simatic y Plcsim. Básicamente, representa la capa de transporte IP/IsoOnTcp en este contexto.

### 3.5 Diagrama de conexión

Una vez conocidos todos los elementos software implicados, es hora de tomar una visión general de cómo se va a interconectar cada elemento, con cual y si se halla incrustado dentro de otro. Esto se representa con figuras rectangulares y de colores para facilitar su diferenciación, además de poder visualizar sobre qué sistema operativo se ejecuta.

A modo de resumen, se quiere acceder como operario al proceso industrial (*Factory IO*). Para que *Factory IO* pueda ser accesible y controlable, se conecta con *PLCSim* (encargado de simular el PLC con el programa). *PLCSim* por si solo no tiene capacidades de conexión complejas, por lo que hace uso de *NetToPLCSim* que lo hace accesible a la red.

*KepServer*, que alberga el servidor OPC UA, es el intermediario en las comunicaciones entre el PLC, que almacena los datos, con las plataformas SCADA. Éste accede a *NetToPLCSim* a través de la red y los SCADA de la misma forma a él.

Para terminar, cabe destacar que *Phoebus* se muestra sobre un plano moteado ya que es una extensión de EPICS y no pertenece o se ejecuta con él.

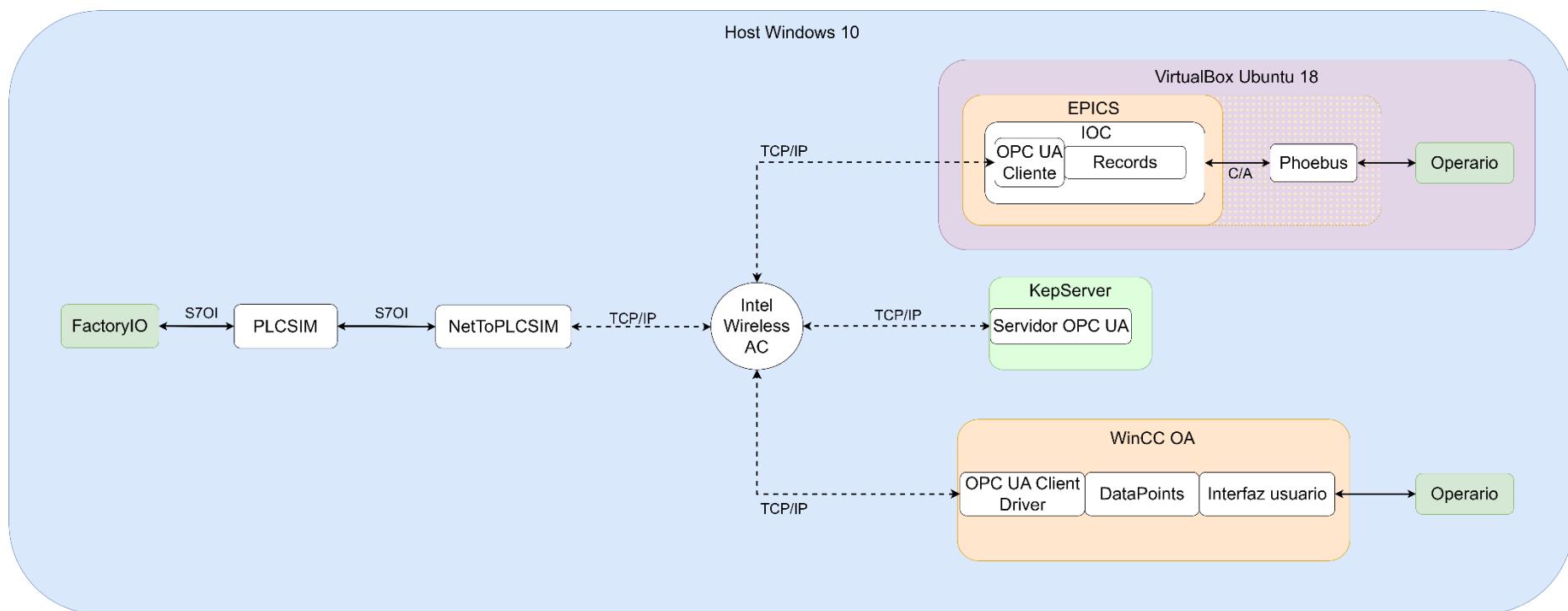


Ilustración 22 - Diagrama de conexión

# 4 Demostrador

## 4.1 Proceso industrial

### 4.1.1 Introducción a los elementos de Factory IO

El objetivo de este punto es mostrar cómo se configuran los elementos de este software.

Cada elemento sobre los que se puede actuar del programa puede tener distintas modalidades, ya sea digital, analógico o combinación de ambas. Dependerá de cada elemento la disponibilidad de esas modalidades.

Por ejemplo, el elemento “*Right Positioner*”:

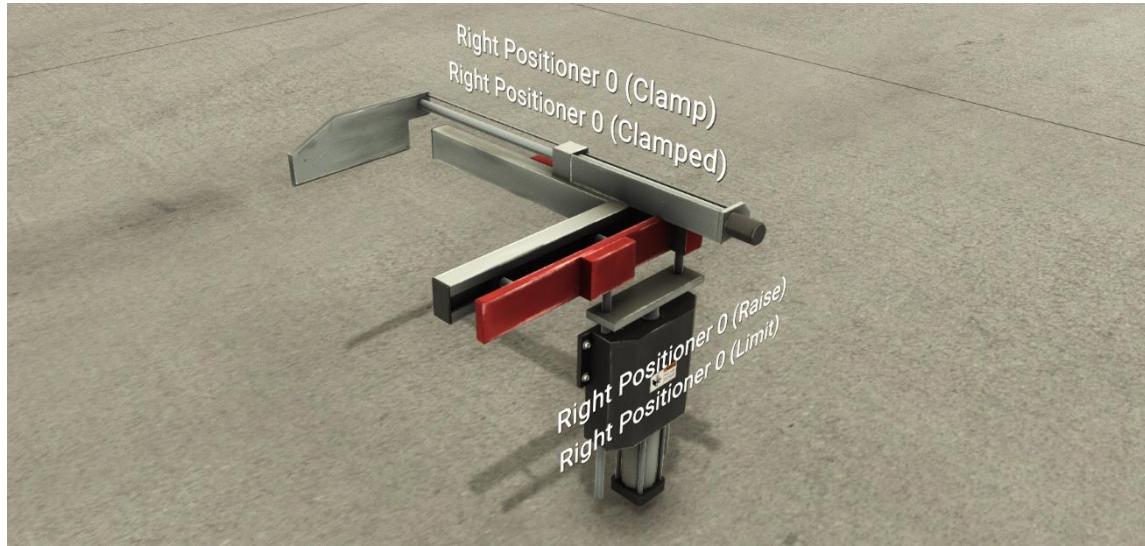


Ilustración 23 - Factory IO Elemento Right Positioner

Este elemento se puede considerar como un brazo que permite centrar objetos entre sus palas. Las etiquetas con texto que lo rodean son los actuadores y sensores de éste. Sus funciones son:

- Actuadores:
  - *Right Positioner (Clamp)*  
Actúa sobre el cilindro neumático horizontal superior de simple efecto normalmente extendido. Es decir, cuando se activa esta etiqueta el cilindro se retrae acercando sus palas.
  - *Right Positioner (Raise)*  
Actúa sobre el cilindro neumático vertical de simple efecto normalmente retraído, lo que implica que al activarse la etiqueta hace que se extienda separando el brazo superior con el cilindro *Clamp*.
- Sensores
  - *Right Positioner (Clamped)*  
Este es un final de carrera que indica que el primer cilindro se ha retraído completamente.
  - *Right Positioner (Limit)*  
Este final de carrera detecta tanto el límite superior como el inferior del segundo cilindro.

Estas etiquetas se les puede cambiar el nombre al que se desee, para este proyecto se ha usado la siguiente nomenclatura:

#### NombreEtapa – Objetivo

Por ejemplo, ese mismo brazo se ha usado en la etapa “Centradora” por lo que el primer actuador queda con el siguiente nombre: **CentradorCerrar**.

### 4.1.2 Descripción del proceso industrial

Con el motivo del proyecto IFMIF-DONES en Granada, el proceso industrial se ha diseñado en etapas imitando la secuencialidad de un acelerador acelerador de partículas.

El proceso, en términos generales, genera tres tipos de piezas, las cuales se centran en la cinta para seguidamente ser estampada en su cara superior. Despues se rocía con un líquido en función de su peso para introducirse posteriormente en un control numérico que la marca. Finalmente se clasifica la pieza en función de su material: verde, azul o gris.

El proceso, tiene la siguiente forma:



Ilustración 24 – Demostrador Factory IO Visión general

#### 4.1.2.1 Etapa 0 – Generador de piezas

En esta etapa se generan tres tipos de piezas: azules, verdes y grises como se puede ver en la siguiente ilustración en la imagen más a la izquierda.

Éstas son generadas por el elemento “**Emitter**”, imagen central, el cual se puede configurar para que aporte distintos tipos de piezas, en algún soporte o su posición y orientación aleatoria.

En este caso solo se generan esas tres, **sin soporte** y con **orientación aleatoria**.

El elemento “**Palletizer**”, imagen de la derecha, es una mera coraza para esconder el emisor de piezas y hacerlo visualmente más real.

Cabe destacar que sobre este proceso no hay ningún control desde el PLC y por eso se considera “Etapa 0”.

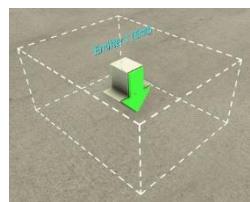


Ilustración 25 – Demostrador Factory IO Tipos de piezas, Emitter y Palletizer

#### 4.1.2.2 Etapa 1 – Centradora

Las piezas vendrán en cualquier orientación y posición, pero para el resto de las etapas es necesario que la pieza a tratar esté bien colocada y alineada en una ubicación determinada de la cinta.

Para ello se ha usado el elemento “**Right positioner**” el cual ya se ha definido anteriormente. Los nombres se han modificado para facilitar la compresión. El actuador *CentradoSubir* eleva el brazo centrador. El actuador *CentradoCerrar* se retrae para colocar la pieza entre sus palas cuando es activado.

En conjunto, tiene un “**Belt Conveyor**” llamado *CentradoCinta* con un sensor del tipo “**Diffuse Sensor**” *CentradoSensor*.

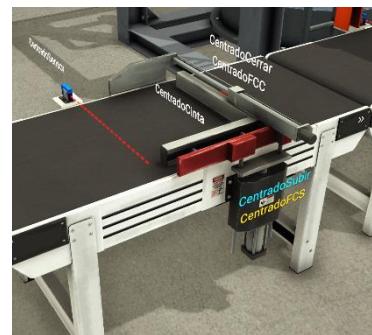


Ilustración 26 - Demostrador Factory IO  
Centradora

##### 4.1.2.2.1 Secuencia

Esta etapa, sigue la siguiente secuencia:

Comienza cuando *CentradoSensor* deja de detectar una pieza, entonces para la cinta, desactiva *CentradoSubir* y espera a que *CentradoFCC*, que es el final de carrera que detecta si éste está completamente retraído o extraído, se active para entonces activar *CentradoCerrar*. Entonces se espera a que *CentradoFCC* se active indicando que ya se ha cerrado por completo, lo que hace que se active de nuevo *CentradoSubir* y *CentradoCinta*.

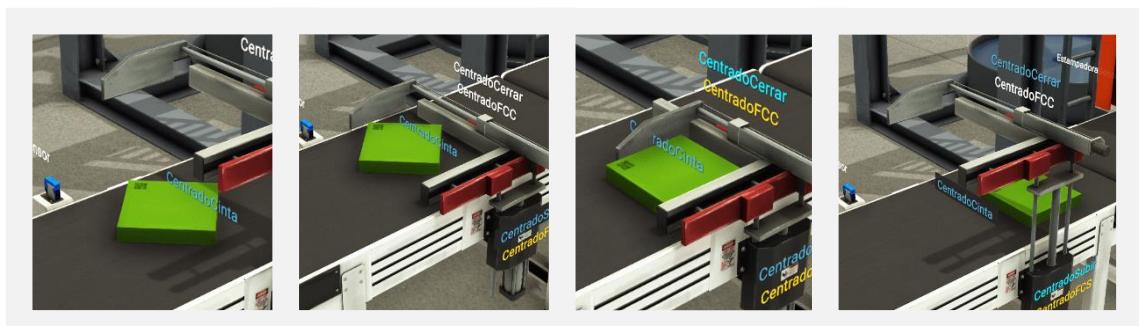


Ilustración 27 - Demostrador Factory IO Secuencia de Centradora

#### 4.1.2.3 Etapa 2 – Estampadora

El proceso de estampado consta de tres elementos activos:

- *EstampadoraCinta* que es del tipo “**Belt Conveyor**”
- *EstampadoraSensor* del tipo “**Diffuse Sensor**”
- “**Pusher**” *EstampadoraActuador*, el cual es un cilindro de simple efecto con dos finales de carrera, uno para la subida *ActuadorFCS* y otro para la bajada *ActuadorFCB*.



Ilustración 28 -  
Demostrador Factory IO  
Estampadora

##### 4.1.2.3.1 Secuencia

El proceso está en espera hasta que *EstampadoraSensor* detecta una pieza, entonces *EstampadoraCinta* se detiene y a la vez *EstampadoraActuador* se activa y baja. Cuando el final de carrera *EstampadoraFCB* se activa, el proceso se reanuda activando *EstampadoraCinta* y subiendo el actuador. El proceso

no espera la siguiente pieza hasta que la actual ha salido completamente del campo de actuación del sensor.

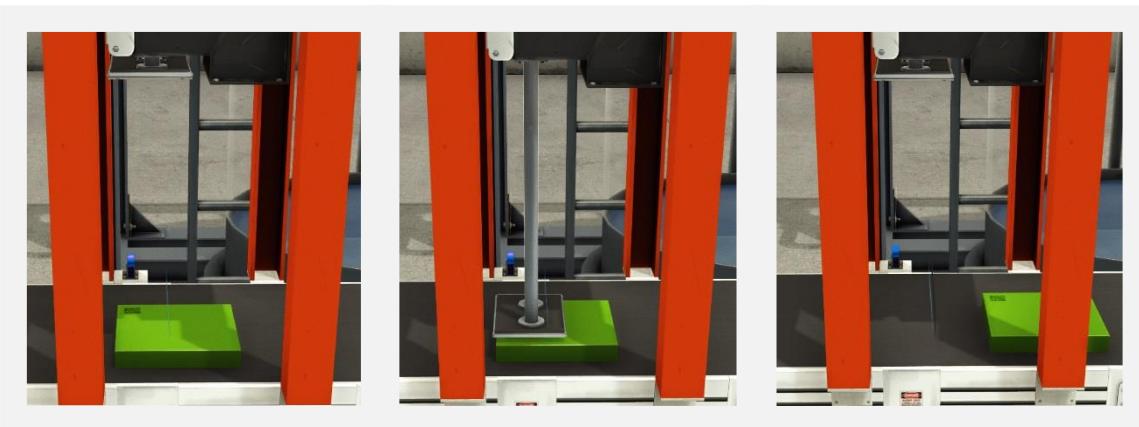


Ilustración 29 - Demostrador Factory IO Secuencia Estampadora

#### 4.1.2.4 Etapa 3 – Rociado por peso

Para el tercer proceso, se pueden dividir en dos sus elementos principales.

Un tanque de líquido, para el que se ha usado el elemento “**Tank**”, el cual lleva:

- Actuadores:
  - **RociadoLlenado**, para llenar el tanque.
  - **RociadoDescarga**, para la descarga.

Ambos son actuadores analógicos en las que se puede modificar su flujo con números decimales del cero al diez.

- Sensores:
  - **RociadoNivel**, para el nivel del tanque.
  - **RociadoFlowSensor**, que mide el flujo del líquido en la descarga.

Ambos también analógicos y decimales.

Por otra parte, se encuentra el módulo de pesado, que contiene:

- “**Conveyor Scale**”, tiene una báscula **PesadoValor** que está configurado para dar valores de cero a diez kilogramos en precisión de un decimal.

Este elemento lleva consigo tambien una cinta **PesadoCinta**, la cual puede avanzar y retroceder, pero solo se hace uso del avance.

Trabaja solidariamente con **PesadoCinta2**, que es una cinta auxiliar salvar el espacio hasta el siguiente proceso.

- **RociadoDescargaLed**, no es más que un indicador luminoso que se activa con **RociadoDescarga**.

##### 4.1.2.4.1 Secuencia

Al igual que los anteriores procesos, éste espera con ambas cintas activadas, **PesadoCinta+** y **PesadoCinta2**, hasta que llega una pieza.



Ilustración 30 – Demostrador Factory IO Rociado por peso

Entonces ambas paran y se da un tiempo arbitrario de espera para que la medida del peso sea fiable. Después de ese tiempo se compara el valor de *PesadoValor* y se inicia un temporizador en función de la pieza. Como las tres piezas tienen pesos distintos, se rocía durante un tiempo predeterminado a cada una: las **verdes** durante **un segundo**, las **azules** durante **dos segundos** y las **grises**, o metálicas, durante **cuatro segundos**.

Una vez pasa el tiempo que corresponda, se reanuda el proceso apagando *RociadoDescarga*, *RociadoDescargaLed* y activando las cintas *PesadoCinta+* y *PesadoCinta2*.

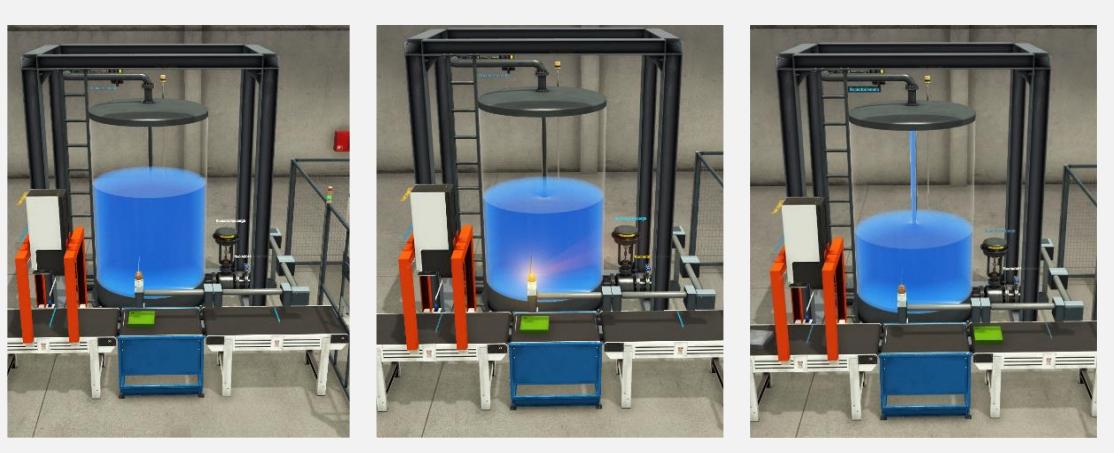


Ilustración 31 - Demostrador Factory IO Secuencia de Rociado por peso

#### 4.1.2.5 Llenado del tanque

El proceso de llenado del tanque no se puede considerar una etapa, es un proceso completamente autónomo e independiente del resto. Pero es conveniente mostrarlo aquí para facilitar la compresión sobre el proceso.

Se han establecido dos límites en el nivel del tanque para que este proceso reaccione:

- Límite superior 80%
- Límite inferior 20%

Si el nivel pasa a estar por debajo del límite inferior, el tanque comienza a llenarse activando la válvula *RociadoLlenado* a su máximo flujo, 10, hasta que se alcanza un nivel del 80% y la desactiva poniéndola al mínimo, 0.

#### 4.1.2.6 Etapa 4 – Control numérico robotizado

Este control número se basa enteramente en el elemento ***"Machining Center"***, el cual acepta el tipo de piezas que se generan en la etapa cero, es decir, *"raw material"*.

Sobre esta estación solo se obtienen datos:

- ***Machining Center (Is Busy)***, se mantiene activa desde que le llega una pieza hasta que la devuelve.
- ***Machining Center (Open)***, se mantiene activa mientras la compuerta de la CNC este abierta
- ***Machining Center (Progress)***, devuelve el progreso de delineado, es un valor analógico.



Ilustración 32 - Demostrador Factory IO Control numérico

- **Machining Center (Has Error)**, devuelve si ha ocurrido algún error, por ejemplo, una pieza en el suelo dentro de su área de acción.

La estación detecta automáticamente las piezas y las coge con el brazo robótico y las introduce en el control numérico, el cual les hace hendiduras. Una vez terminado, el brazo las transporta al lado opuesto de donde fue obtenida.



Ilustración 33 - Demostrador Factory IO Secuencia Control numérico robotizado

#### 4.1.2.7 Etapa 5 – Clasificadora

Por último, las piezas ya procesadas pasan a la última etapa donde se clasifican por materiales. En este caso existen diferentes componentes:

- Sensores:
  - “**Vision Sensor**”, **ClasificadoraCámara**  
Es el principal elemento. Al pasar una pieza por su campo de visión devuelve un código. En nuestro caso las piezas procesadas azul, verde y gris se obtiene 3, 6 y 9 respectivamente.
- Actuadores:
  - Dos “**Belt conveyor**”, **ClasificadoraCinta1 y 2**.
  - Tres “**Pivot Arm Sorter**”, los cuales son cintas en el plano vertical que pueden girarse para desplazar la trayectoria de cualquier objeto. Hay uno para cada pieza.



Ilustración 34 - Demostrador Factory IO Clasificadora

Una vez las piezas están clasificadas, estas desaparecen gracias al elemento “**Remover**”, el cual es el análogo al “**Emitter**”.

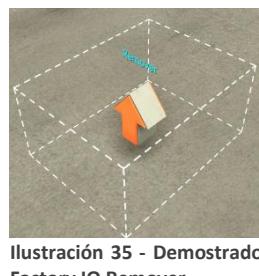


Ilustración 35 - Demostrador Factory IO Remover

#### 4.1.2.7.1 Estados

Solo tiene tres estados, estos son remanentes, es decir, la cámara al leer la pieza devuelve su valor, se compara y se activa la cinta correspondiente. Esta se queda activada hasta la siguiente pieza.

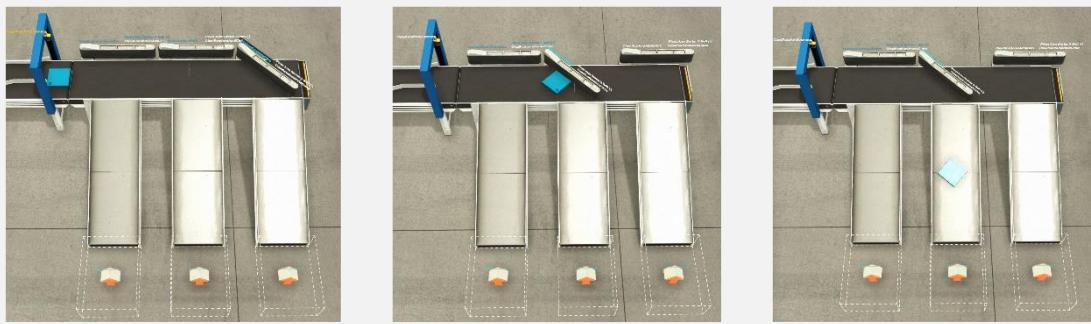


Ilustración 36 - Demostrador Factory IO Secuencia de clasificación

## 4.2 Programación del controlador

### 4.2.1 Autómata

Antes del uso de los PLC (*Programmable Logic Controller*), o también llamados autómatas programables, los sistemas de control se basaban en la lógica cableada utilizando contactores, relés y otros componentes electromecánicos. La transición de las automatizaciones con contactores a los PLC ha brindado ventajas significativas en la industria, entre ellas se destacan: la flexibilidad, reprogramabilidad, programas complejos, la integración simplificada de entradas y salidas (E/S), el monitoreo y diagnóstico en tiempo real. Por lo que el **uso los PLCs** están **ampliamente extendidos**.

Actualmente existe cierta variedad en modelos y marcas de PLCs, pero la **serie S7-300** de Siemens es de las **más usadas** y hay varias razones de porque ha sido ampliamente adoptada:

1. **Trayectoria y experiencia:** La serie S7-300 ha estado en el mercado durante mucho tiempo y ha demostrado su confiabilidad y rendimiento en numerosas aplicaciones industriales a lo largo de los años. **Siemens tiene una sólida reputación** como fabricante de PLC y ha acumulado una gran base instalada con la serie S7-300.
2. **Escalabilidad:** Los controladores S7-300 son altamente escalables, lo que significa que se pueden adaptar a una amplia gama de aplicaciones, desde proyectos pequeños y sencillos hasta sistemas de automatización más complejos y extensos. Esto permite a los usuarios elegir el tamaño y las capacidades del controlador que mejor se adapten a sus necesidades específicas. Incluso extenderlo en el tiempo si fuera necesario.
3. **Modularidad:** El sistema S7-300 se basa en **un diseño modular**, lo que permite una mayor flexibilidad en la configuración del controlador. Los **módulos de E/S**, las **interfaces de comunicación** y otras expansiones se pueden agregar o intercambiar según los requisitos de la aplicación, lo que facilita la adaptación y la expansión del sistema.
4. **Soporte y recursos:** Siemens proporciona un amplio soporte técnico y una gran cantidad de recursos, como documentación, manuales y comunidades de usuarios, para ayudar a los programadores y usuarios a aprovechar al máximo los PLC S7-300. Esto facilita el aprendizaje, la resolución de problemas y el desarrollo de soluciones efectivas.
5. **Compatibilidad y conectividad:** Los controladores S7-300 son compatibles con una amplia gama de interfaces de comunicación, lo que les permite integrarse fácilmente en diferentes entornos industriales y sistemas de control. Esto garantiza la interoperabilidad con otros

dispositivos y facilita la conectividad con redes industriales y sistemas de supervisión y control. Es de los pocos en los que se permite su programación en **STEP7**, interfaz clásica, como en **TIA-Portal**, la más actualizada y con más soporte.

Por lo tanto, para este proyecto se ha escogido un modelo de la serie **S7-300** por ser el más extendido, usado y no es necesariamente de los más actuales. Pese a esto último, es viable para desarrollar cualquiera de los dos proyectos que se van a analizar en este trabajo.

#### 4.2.1.1 Modelo

Esta serie de autómatas se distribuyen en diferentes modelos, algunos de los cuales tienen módulos de entradas y salidas ya integrados. Pero para mostrar la modularidad de estos controladores, se ha optado por un modelo compacto, el cual solo viene provisto de interfaces de comunicaciones. En nuestro caso nos interesa la interfaz *Profinet*, la cual nos va a permitir conectarnos con *NetToPLCSim* y abrir la comunicación con el servidor OPC de *KepServer*.

El modelo en cuestión es la **CPU 315-2 PN/DP** y su referencia **6ES7 315-2EH14-0AB0**.

#### 4.2.1.2 Módulos

En el proceso industrial se encuentran elementos de control digital y analógico, tanto de entrada como de salida. Por lo que necesitaremos dos tipos de módulos.

##### 4.2.1.2.1 Entradas y salidas digitales

Para este caso hay que evaluar la cantidad de entradas y salidas que se necesitan, ya que existen distintos tipos de módulos con diferentes características y cantidad de salidas. Las características en si se van a despreciar, ya que al ser una simulación no hace falta darle importancia a que sea un módulo de relés o transistores ni a su intensidad o voltajes máximos soportados.

Para este proceso se necesitan 15 entradas y 19 salidas. Por lo que, sopesando distintos módulos de solo entradas, solo salidas y ambas a la vez se ha optado por un módulo de 16 entradas y 16 salidas, en la ilustración 19 se representa a la izquierda. Y para cumplir con las 3 salidas que faltan se usa un módulo de solo salidas, 8 patillas.

Sus nombres y referencias, respectivamente, son:

- Nombre: **DI 16/DO 16x24VDC/0.5<sup>a</sup>**, referencia: **6ES7 323-1BL00-0AA0**
- Nombre: **DO 8x24VDC/0.5<sup>a</sup>\_1**, referencia: **6ES7 322-8BF00-0AB0**

##### 4.2.1.2.2 Entradas y salidas analógicas

A entradas y salidas analógicas, con FactoryIO, nos referimos a números enteros o con decimales (*Int* y *Float*). Y se tienen:

Entradas analógicas	Salidas Analógicas
PesadoValor (Float)	PesadoRociadoDescarga (Float)
PesadoRociadoNivel (Float)	PesadoRociadoLlenado (Float)
PesadoRociadoFlowSensor(Float)	PiezasVerdes(Int)
ClasificadoraCamara(Int)	PiezasMetalicas(Int)
MachiningCenterProgress (Int)	PiezasAzules(Int)



Ilustración  
37 -  
Autómata  
CPU

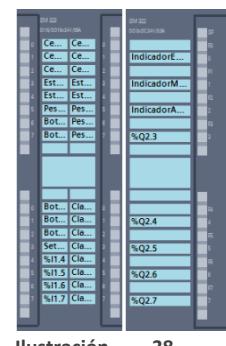


Ilustración  
38 -  
Módulos  
DI/DO y DO

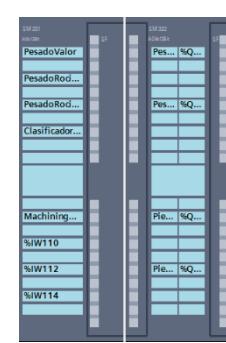


Ilustración 39 - Módulos  
AI y AO

El recuento de las patillas necesarias es de 5 entradas y 5 salidas. En este caso, para diferenciar el anterior, se ha usado un módulo independiente para las entradas y otro para las salidas porque no existe un módulo conjunto que satisfaga esa necesidad. El de entradas tiene 8 patillas y el de salidas 8.

Sus nombres y referencias, respectivamente, son:

- Nombre: **AI 8x12BIT**, referencia: **6ES7 331-7KF02-0AB0**
- Nombre: **AO 8x12BIT**, referencia: **6ES7 332-5HF00-0AB0**

La visión general del autómata con sus módulos quedaría de la siguiente forma:



Ilustración 40 - Autómata completo

#### 4.2.2 Estructura de programa

Gracias a que se ha diseñado el proceso de forma que cada etapa sea independiente, se ha programado acorde a este hecho. Cada etapa está programada en su propio “bloque”, incluyendo tambien un bloque para el control de estados (Emergencia, automático o manual) y el contaje de piezas.

Pero a su vez se ha subdividido cada bloque en dos, uno para el control de la secuencia y otro para la gestión de salidas. El motivo de ello es simplificar y reducir el tamaño de los programas, además de facilitar la gestión de éstas según el estado en el que se encuentre todo el proceso.

Existe un bloque principal, llamado **Main**, el cual solo se encarga de reunir y organizar todos estos bloques.

Por lo que, en forma de lista, se va a mostrar el como queda ese bloque principal con el resto de los bloques:

- Main
  - 1CentradorSecuencia
  - 1CentradorSalidas
  - 2EstampadoraSecuencia
  - 2EstampadoraSalidas
  - 3PesadoRociadoSecuencia
  - 3PesadoRociadoSalidas
  - 4ClasificadoraSecuencia
  - 4ClasificadoraSalidas
  - 5Llenado
  - 6ContajePiezas
  - 7Control

#### 4.2.2.1 Lenguajes usados

Para este proyecto se han mezclado el uso de estos lenguajes de programación, dejando excluidos FUP y AWL:

**KOP** para la estructura principal de programa (*Main [OB1]*), estados de emergencia (*Control[FC8]*), llenado del tanque (*Llenado [FC6]*) y gestión de las salidas de cada proceso por su cercanía a los diagramas de circuitos, permitiendo una programación muy visual.

**SCL** para el proceso de clasificado de piezas (*ClasificadoraSCL [FC4]*).

**GRAPH** para gestionar con facilidad la secuencialidad de los procesos de *Centrador*, *Estampadora* y *PesadoRociado*.

#### 4.2.2.2 Tratamiento de los elementos analógicos

Para las entradas analógicas donde FactoryIO devuelve un entero no hace falta ningún tratamiento especial. Pero en el caso de los que devuelven un numero decimal como *PesadoValor*, *RociadoNivel* y *RociadoFlowSensor*, hay que transformarlos con las funciones *SCALE* y *UNSCALE*.

*SCALE* convierte la entrada que se le proporciona en un rango seleccionado por el usuario. En el caso de *PesadoValor* devuelve valores desde 0KG hasta 10KG. Además, la entrada puede ser tratada como bipolar o unipolar, es decir, con posibles números negativos o no. El estado de la operación puede ser recogido a parte con *RET\_VAL* y el propio resultado de la operación con *OUT*. A continuación, se muestra el caso de *PesadoValor*:

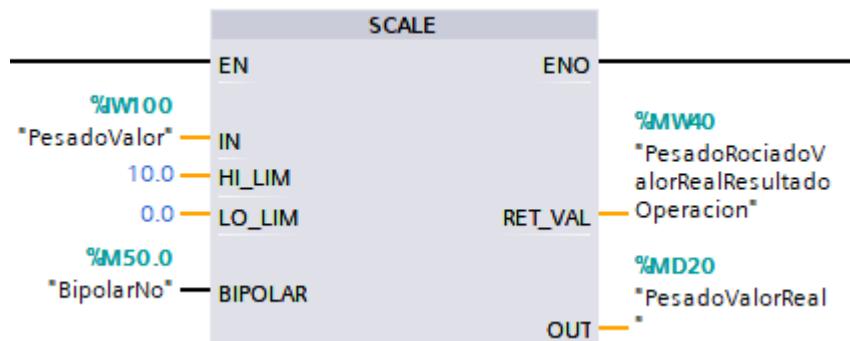


Ilustración 41 - Función *SCALE* de *PesadoValor* en las instrucciones permanentes anteriores del GRAPH *PesadoRociado* [FB3]

*UNSCALE* transforma el numero decimal que se le introduzca en uno entero. El rango que se le inserta debe ser en el que trabaja el decimal de entrada. Por ejemplo, para *PesadoRociadoDescarga* que

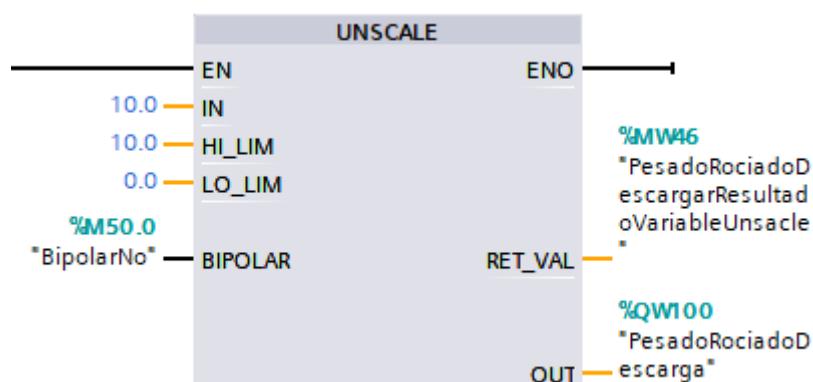


Ilustración 42 - Función *UNSCALE* en el segmento 1 de *PesadoRociadoSalidas* [FC3]

funciona en el rango 0.0 hasta 10.0, donde 10.0 es donde la válvula está dejando pasar la mayor cantidad de líquido, la forma de abrirla sigue como la siguiente ilustración:

De igual forma, se puede tratar como bipolar, recoger el estado de la operación y en este caso se ha devuelto directamente el valor a la salida con *OUT*.

### 4.2.3 Programa y variables

El programa y las variables se incluyen tanto en el archivo PDF **3\_2\_1\_ProgramaPLC.pdf** y en el proyecto de Tia Portal **3\_2\_2\_ProyectoTiaPortal15PLC**, ambos en la carpeta número 3.

## 4.3 Configuración

### 4.3.1 Configuración del proceso

#### 4.3.1.1 Configuración Factory IO

Una vez definido todo el proceso, sus variables de entrada y salida junto con la simulación del PLC, es hora de comunicar FactoryIO con *PLCSIM*.

Una de las grandes ventajas de este software es que ya tiene integrada la comunicación con distintos PLCs de distintas marcas, e incluso mediante protocolos como modbus o como OPC Client DA/UA. Entre las comunicaciones con autómatas de Siemens encontrados: LOGO!, S7-200/300/400, S7-1200/1500 o *PLCSIM*. Es decir, brinda la posibilidad de conectarse a PLCs reales o simulados.

Con NetToPLCsim se podría crear una comunicación como si fuera un PLC real, pero para simplificar esa comunicación, que sea más rápida y no saturar esa comunicación se ha optado por la comunicación directa con *PLCSIM*, como se mostró en el diagrama de conexión.

Una vez escogida la forma, se debe configurar las características del PLC:

- Reconexión automática
- Modelo: 200, 300 o 400
- Cómo se tratan los datos numéricos (las entradas y salidas analógicas de los módulos usados usan WORD, es decir, dos bytes)
- La cantidad de entradas y salidas digitales y analógicas.

En la ilustración 44 queda la configuración adaptada al PLC:

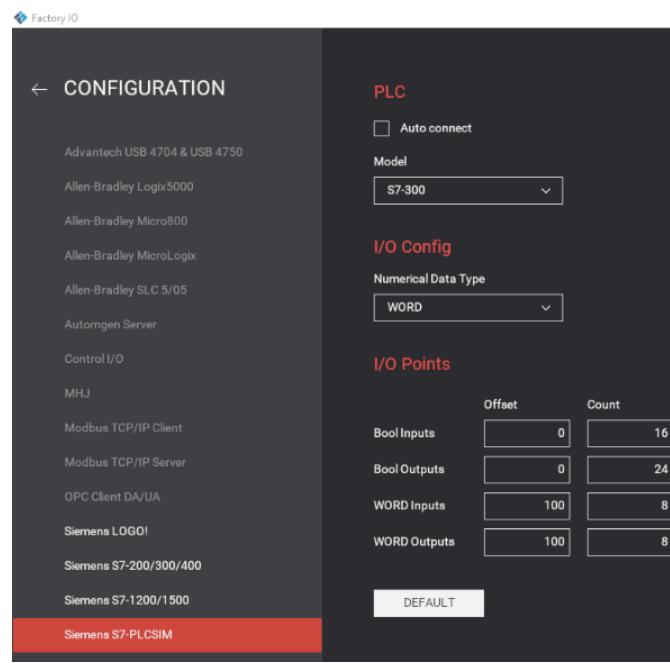


Ilustración 43 - Configuración del PLC en FactoryIO

#### 4.3.1.1.1 Asignación de etiquetas

Con esto ya aparece un PLC representado con esa cantidad de patillas, donde se le asignan las entradas y salidas del proceso de forma tan sencilla como arrastrándolas a su dirección correspondiente.

Esta pantalla, en tiempo de ejecución, también se pudo visualizar qué entradas están siendo activadas y qué salidas están activadas o no.

Por lo que, en este caso, quedarían de la siguiente forma:

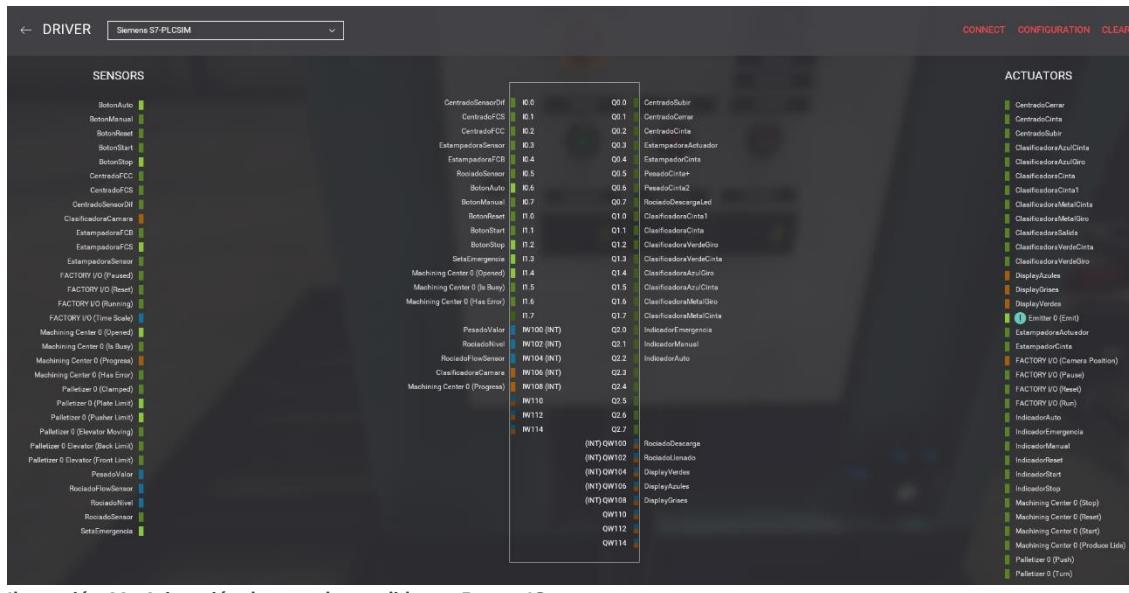


Ilustración 44 - Asignación de entradas y salidas en FactoryIO

Además, si la conexión es correcta con *PLCSim* aparecerá este símbolo:

#### 4.3.1.2 Configuración *PLCSim*

Para hacer que *PLCSim* funcione basta con iniciarla. Abierto de primera no se encuentra ningún programa cargado. Para cargarle un programa, hay dos formas:

- Subir el programa desde *TIA Portal* como si de un PLC real se tratase
- Cargar un archivo *.plc*, el cual genera el propio *PLCSim* cuando tiene un programa cargado y almacena el estado de todas las zonas de memoria y el programa en sí.

Para cargar un programa por primera vez, teniendo abierto tanto *PLCSim* como *TIA Portal* con el proyecto deseado, se pulsa el icono de subida al PLC señalado en rojo en la siguiente ilustración:

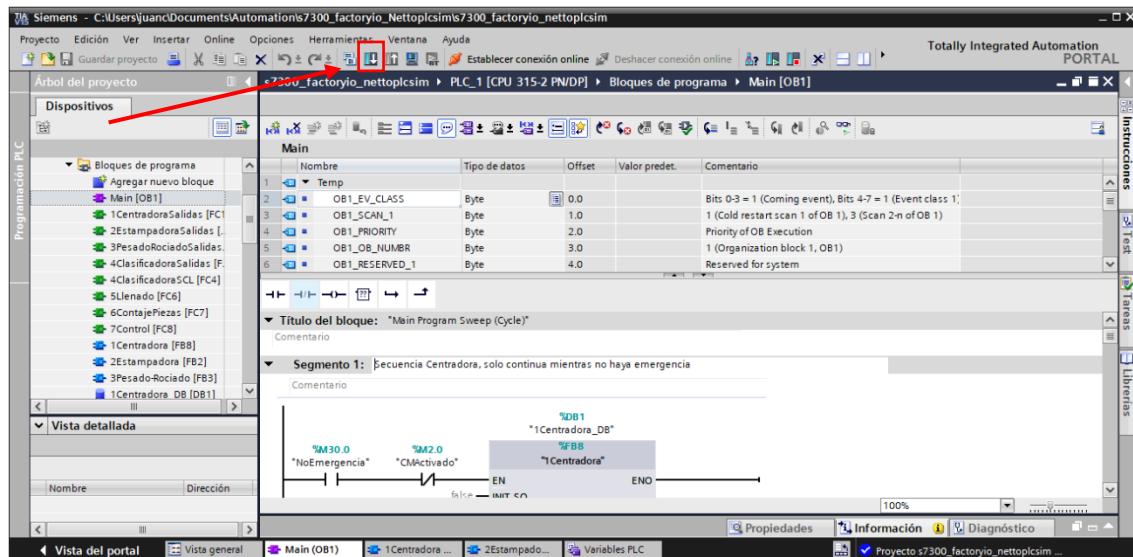


Ilustración 46 - *TIA Portal* icono de subir programa

Entonces el programa se compilará, en el caso de que no se haya hecho ya y saldrá una ventana de dialogo donde se puede seleccionar los bloques de programa. Para cargar todo, la mejor opción es seleccionar en el desplegable “Cargar con coherencia” y así se cargarán todos una vez se pulse “Cargar”:

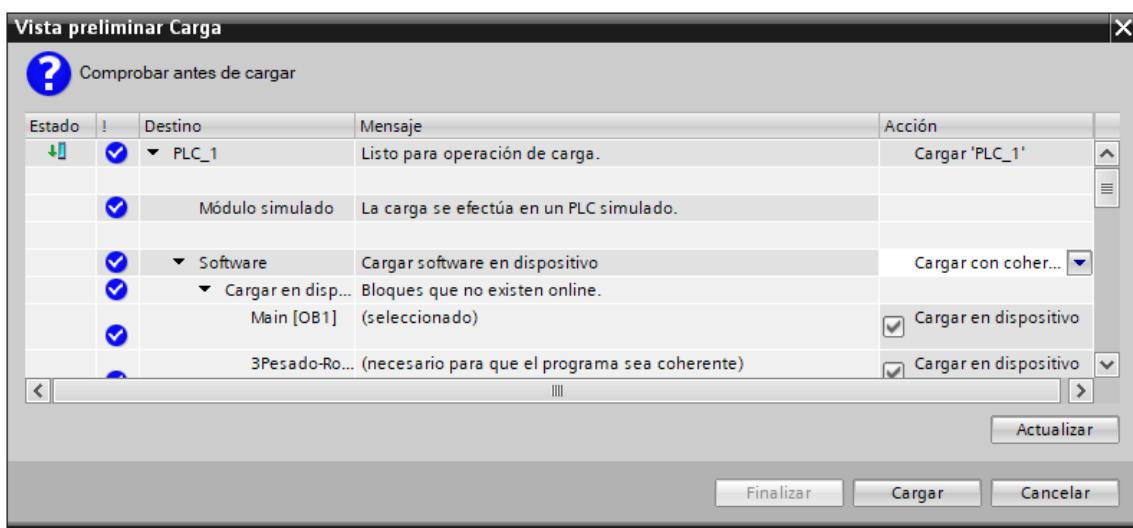


Ilustración 47 - *TIA Portal* Dialogo de carga

Entonces, volverá a la pantalla anterior y se podrá ver si ha habido algún error.

PLCSim hará un amago de reiniciarse y ya se tendrá PLC simulado con el programa deseado cargado. Desde su ventana se pueden añadir subventanas para controlar o visualizar cualquier zona de memoria, con distintos tipos de formato (bits, bytes, decimal...).

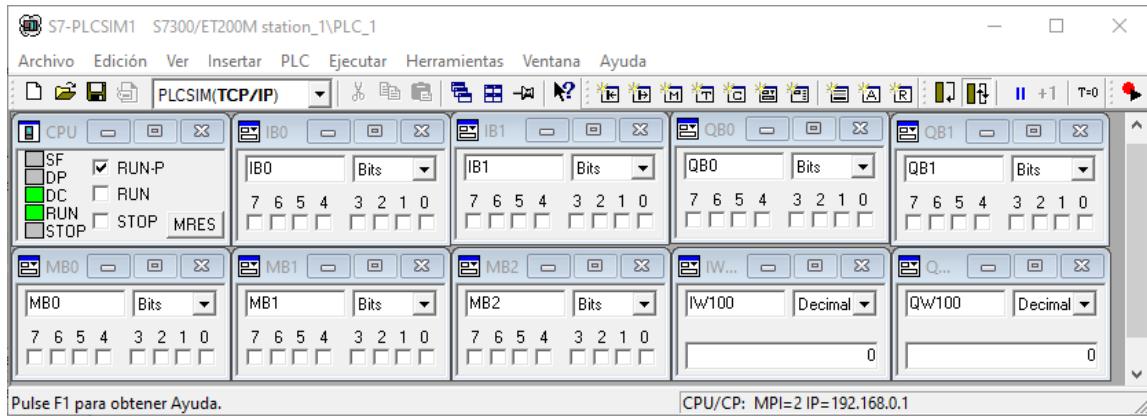


Ilustración 48 - Interfaz PLCSIM

#### 4.3.1.3 Simulación del proceso

Una vez con el programa cargado y las etiquetas colocadas adecuadamente en Factory IO, se puede simular para comprobar el correcto funcionamiento. Para ello, basta darle al botón comúnmente llamado “Play” en la barra superior. Entonces el proceso comenzará a funcionar:



Ilustración 49 - Factory IO Funcionando

#### 4.3.2 Configuración de la comunicación con los SCADA

##### 4.3.2.1 Configuración NetToPLCsim

Entonces, para que KepServer pueda acceder a la memoria del PLC, hay que configurar NetToPLCsim.

En este caso no se usa para validar el programa PLC sino para comparar y analizar el uso en el ámbito industrial de los SCADAS.

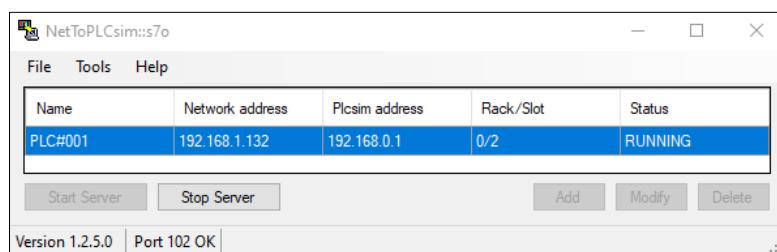


Ilustración 50 - Interfaz de NetToPLCsim con una conexión ejemplo

- Se puede configurar el nombre, el cual puede ser arbitrario y no afecta a nada más.
- *Network address* es la dirección IP que se le asigna al elegir uno de los interfaces de red accesibles por este software.
- *PLCsim address* es la dirección que tiene la instancia de *PLCsim*, como se puede apreciar en la ilustración 29 en la esquina inferior derecha.
- *Rack/Slot* es la ranura en la que se ha configurado la CPU. *Status* es el estado en el que se encuentra la conexión.

#### 4.3.2.2 Configuración KepServer

##### 4.3.2.2.1 Canal

Primero hay que crear un canal, donde hay que indicar el **driver (Siemens TCP/IP Ethernet)**, asignarle un **nombre** como “**ProcesadorPiezas**” y asignarle la **misma interfaz de red que en NetToPLCsim**.

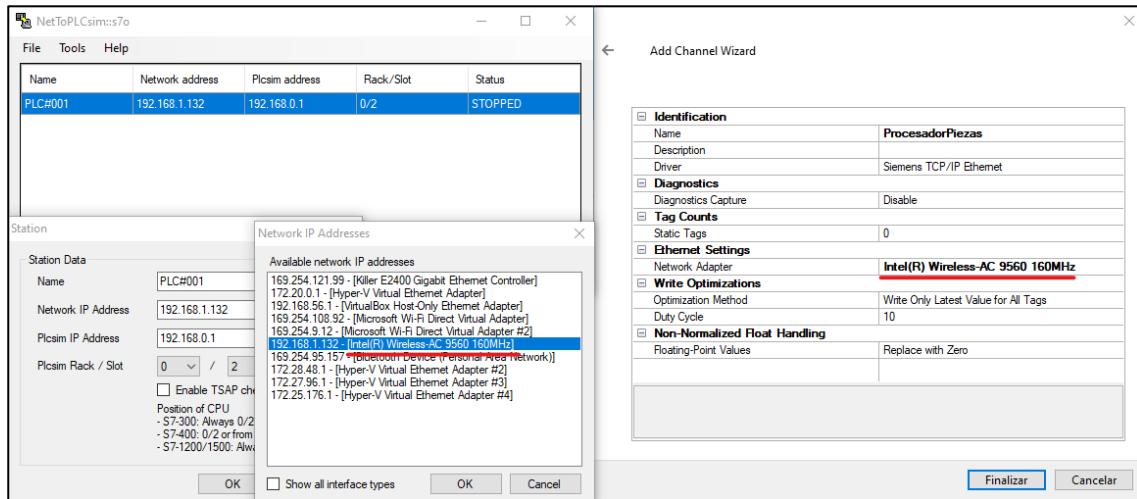


Ilustración 51 - Configuración del canal en KepServer

#### 4.3.2.2.2 Dispositivo

Una vez con el canal creado hay que crear un dispositivo, como elementos que no vamos a dejar por defecto, hay que asignarle un nombre como “**S7300**”, el **modelo** de PLC para el driver (**S7-300**), la

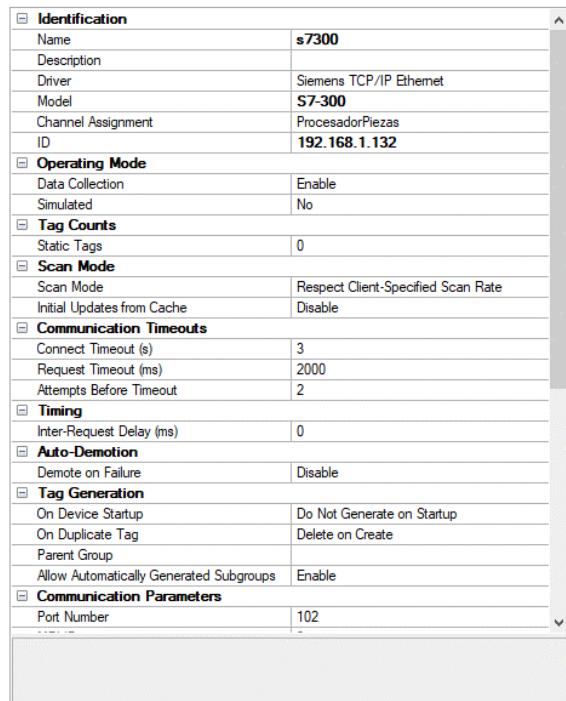


Ilustración 52 - Configuración dispositivo en KepServer

dirección **IP** que deberá ser la que indica **NetToPLCsim** (192.168.1.132) y el **puerto 102**. El resto de las configuraciones, como *Communication TimeOuts* o *Scan Mode* son interesantes para tener en cuenta.

#### 4.3.2.2.3 Etiquetas

Ahora viene el punto de unión de cara a los datos. Hay que crear los *Tags* en KepServer teniendo en cuenta los siguientes parámetros:

- **Nombre:** debe ser único.
- **Dirección:** por ejemplo, *I0.5* o *QW100*.
- **Tipo de dato:** binario, palabra, entero, coma flotante... .

Se pueden crear individualmente, en grupo con ayuda de la aplicación donde te permite escribir texto estático con rangos numéricos, importar las etiquetas desde un proyecto con *Tia Portal Exporter* o importalas desde un archivo *.csv*. Una vez importadas, la ventana principal quedaría así:

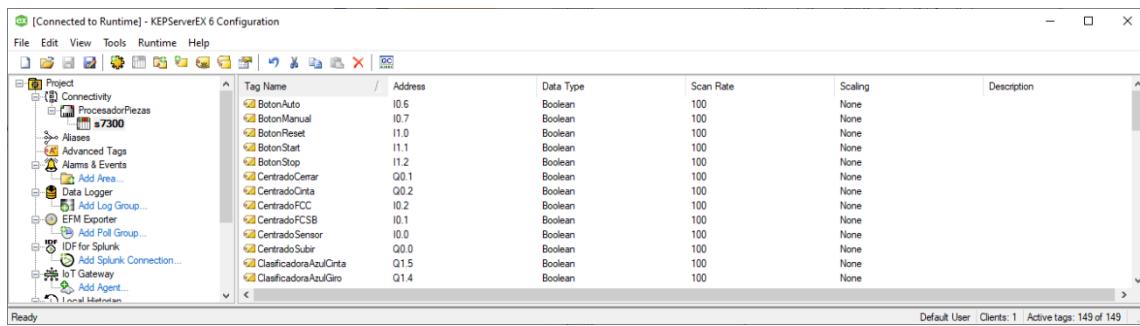


Ilustración 53 - Panel principal de KepServer con la conexión, el dispositivo y las etiquetas

# 5 Desarrollo de la aplicación SCADA

Para poder analizar de forma exhaustiva estos software, se va a documentar desde el proceso de instalación, pasando por sus dependencias y módulos hasta la propia gestión de usuarios y alarmas en cada SCADA.

El desarrollo de cada una se mostrará de forma paralela, es decir, por cada apartado se mostrará como se realiza con cada software. De esta forma se facilita la comprensión de las carencias y virtudes, permitiendo un análisis más profundo y objetivo.

Antes de ello, hay que tener en cuenta el sistema y sus características donde se ha montado cada uno para luego especificar la jerarquía de paneles y sus estructuras.

## 5.1 Hardware

Portátil MSI GF62-8re con las siguientes características:

- **Procesador:** Intel i7-8750H 6 núcleos, 12 hilos.
- **Memoria:** 16 GB RAM DDR4.

Máquina virtual, ejecutándose dentro del computador anterior:

- **Procesador** con 6 hilos
- **Memoria** de 4GB RAM

## 5.2 Software

Como sistema operativo anfitrión, **Windows 10 pro**.

Como sistema operativo auxiliar en máquina virtual en *Virtual Box* con **Ubuntu 18.04**.

## 5.3 Jerarquía de los paneles

Esta jerarquía consta de un panel principal, llamado *Main*, el cual lleva una visión general y simplificada del proceso. En el existen diversos accesos directos a los subprocesos.

Por debajo de esta jerarquía, se encuentran el resto de los paneles de los subprocesos. La jerarquía quedaría así:

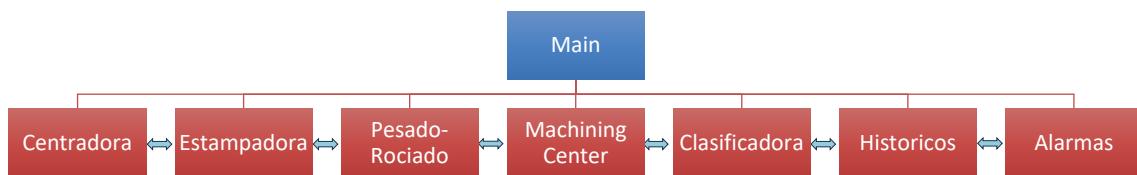


Ilustración 54 - Jerarquía de paneles

## 5.4 Estructura de la información

Para mantener la homogeneidad entre paneles, se van a crear elementos comunes a todos ellos y algunos modificables para aumentar su utilidad.

Habrá una barra superior donde haya accesos directos a todos los paneles, además de ciertos iconos.

La barra inferior constará de dos partes, la parte izquierda contendrá elementos comunes como indicadores del estado de la máquina, controles sobre dicho estado, y elementos varios que diferirán entre las aplicaciones. La parte derecha se reservará para el control y supervisión de los sensores y actuadores de cada subprocesso. Eso quiere decir que en la visión general no habrá controles sino elementos con más datos generales.

La parte central se reservará para representar lo que toque en cada ventana con objetos geométricos, indicadores, leds o históricos.



## 5.5 Proceso de instalación

### 5.5.1 EPICS BASE

EPICS puede ser instalado en **diferentes sistemas operativos**, como Linux, Windows o MacOS. Según el sistema operativo, tendrá dependencias diferentes. En este caso, Linux con Ubuntu en la máquina virtual comentada. Según la documentación oficial, los requisitos son:

- *Make*
- *C++*
- *Libreadline*
- *Git* (opcional, solo si se prefiere descargar el repositorio de esa forma)

1. `mkdir $HOME/EPICS`

2. `cd $HOME/EPICS`
3. `git clone—recursive https://github.com/epics-base/epics-base.git`
4. `cd epics-base`
5. `make`

El proceso de instalación sigue así:

Ahora queda exportar las siguientes variables, añadiéndolas preferiblemente al `.bashrc`:

```
export EPICS_BASE=${HOME}/EPICS/epics-base
export EPICS_HOST_ARCH=$( ${EPICS_BASE}/startup/EpicsHostArch )
export PATH=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}:$PATH
```

#### 5.5.1.1.1 Tiempo

El comando `make` se ha replicado **seis veces** para tomar cierta medida en el tiempo que toma su instalación, estos son los resultados:

1. 710 (11 minutos, 50 segundos)
2. 622
3. 845
4. 539
5. 977
6. 326
7. 452

Una media de 638,72 segundos, es decir, **10 minutos y 38 segundos** para el software base.

#### 5.5.1.2 Módulo OPCUA

Prerrequisitos necesarios y recomendables para que el módulo se pueda instalar y usar:

- EPICS Base 3.15.5 o superiores (EPICS 7 tambien es soportado)
- El módulo `qtest`
- Compilador C++ que soporte el estándar C++11
  - Además, *Unified Automation C++ Based OPC UA Client SDK*, que tambien tiene estos requisitos:
    - *CMake 3.12*
    - *OpenSSL*
    - *LibXML2*

El proceso de instalación quedaría:

1. `sudo apt install libssl-dev` (~ 10,062 segundos)
2. `sudo apt install cmake` (~ 8,989 segundos)
3. `sudo apt install libxml2` (~ 6,986 segundos)
4. `cd /home/juanca/uasdkcpp[...]`
5. `cmake CMakeLists.txt`
6. `make`

#### 5.5.2 WinCC OA

WinCC OA puede ejecutarse en diversos sistemas operativos, Windows principalmente y algunos Linux como *RedHat 8 Enterpris*, *Oracle Linux*, *Debian* con Docker y otros.

Por comodidad y habitualidad al sistema operativo Windows, se instalará en él.

WinCC OA para este SO se distribuye con un solo archivo ejecutable (.exe), por lo que el primer paso es ejecutarlo y saldrá la siguiente ventana:

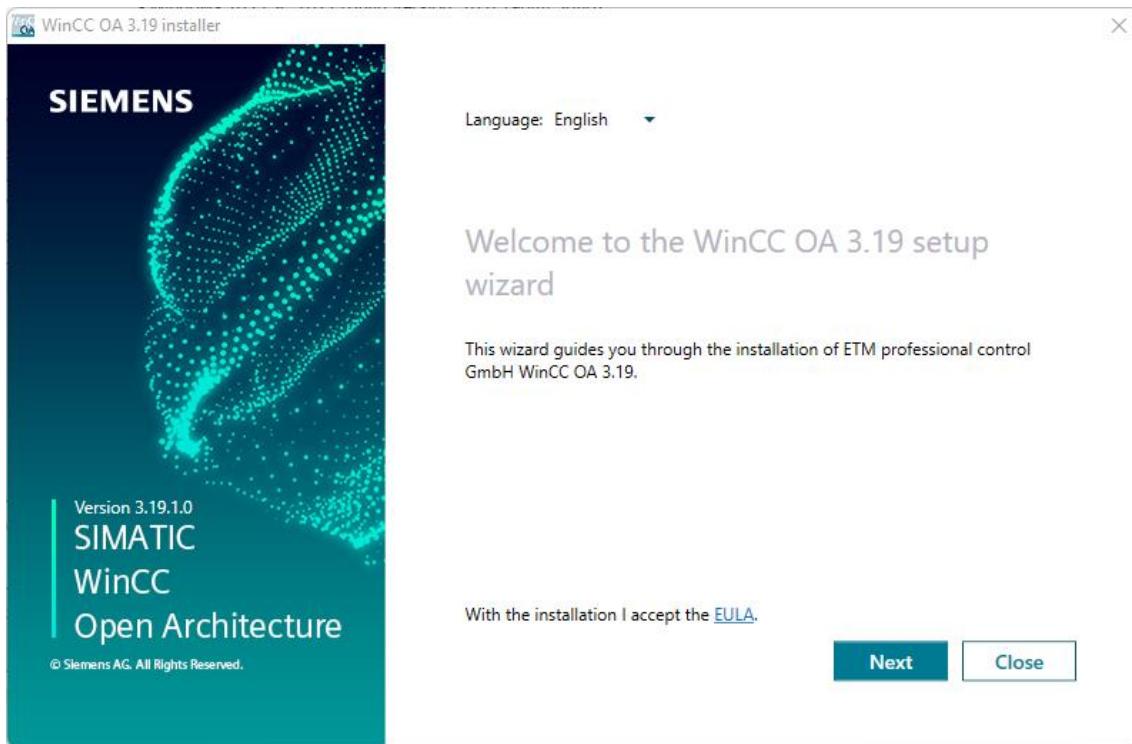


Ilustración 56 - WinCC OA Instalación Primera ventana

Al darle a siguiente, saldrá otra ventana donde se puede seleccionar componentes adicionales al programa, directorio donde se instalará, directorio de proyectos y un computo de cuanta memoria se usará:

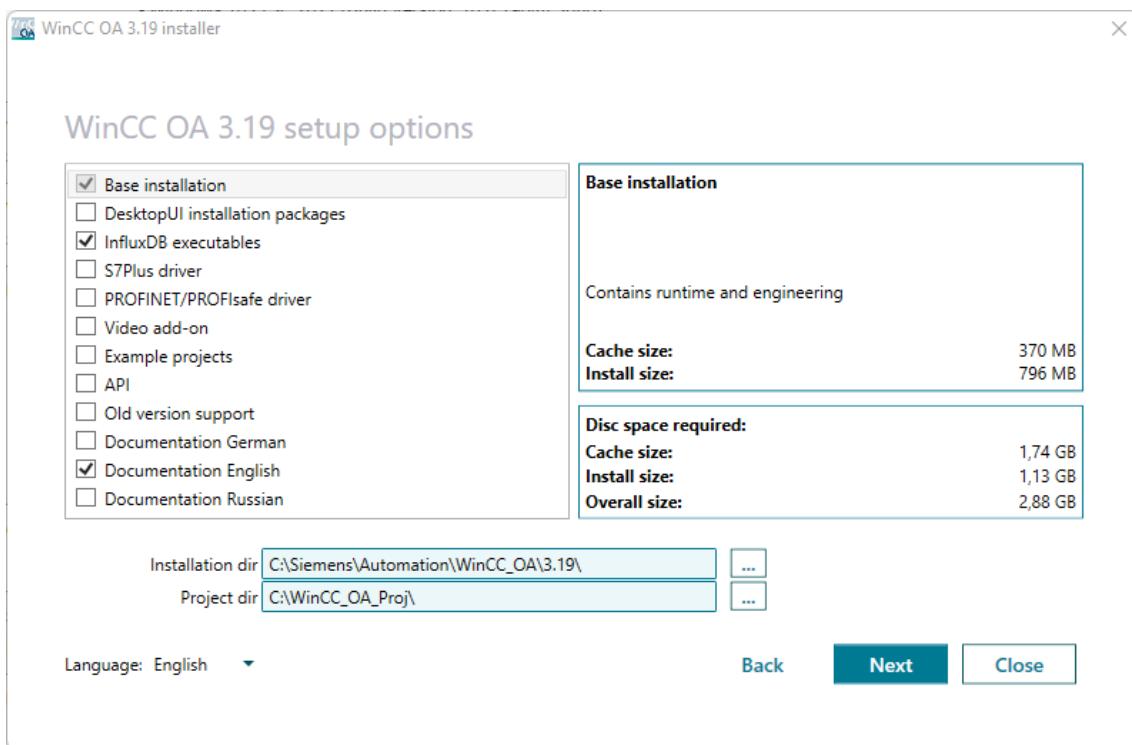


Ilustración 57 - WinCC OA Instalación Selección de extensiones

Al darle de nuevo a siguiente, ya comenzará el proceso de instalación con la siguiente ventana:

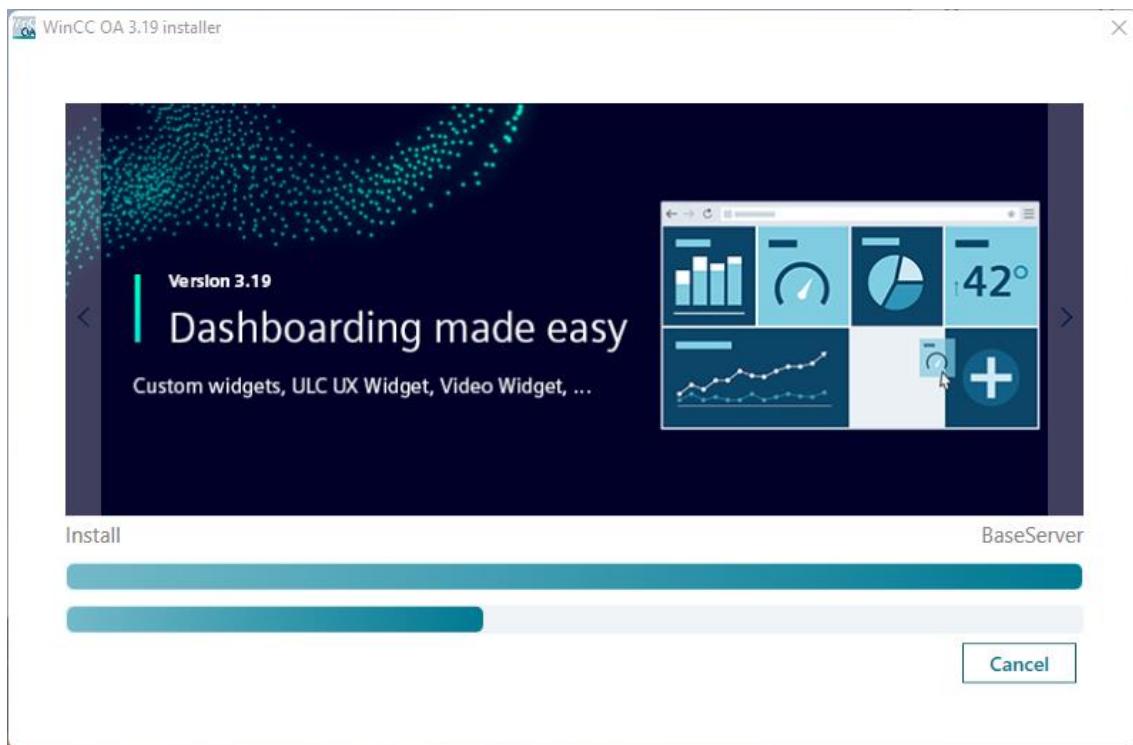


Ilustración 58 - WinCC OA Instalación Progreso

El proceso termina en 2 minutos con 7 segundos de media, encontrándose listo para usar.

## 5.6 Instalación de módulos

### 5.6.1 EPICS

Para completar EPICS, se necesita el módulo que hace de driver para la comunicación OPC UA (el ya comentado *OPC UA Module*) y la extensión del gestor de interfaces de usuario (*Phoebus* de CS-Studio).

#### 5.6.1.1 OPCUA

Este módulo no está exento de requisitos, los cuales son:

- Compilador C++ con el estándar C++11.
- *Unified Automation C++ Based OPC UA Client SDK*
- EPICS Base 3.15 o EPICS 7
- *Gtest module* si se quiere usar los test de Google.

Por lo que el primer paso es instalar, la única dependencia que falta, el cliente SDK de *Unified Automation*. Para poder descargar cualquier versión, incluso la de evaluación, hay que registrarse. Una vez hecho, se podrá descargar el archivo .zip con los archivos necesarios para su instalación.

##### 5.6.1.1.1 Instalación OPC UA Client SDK

Al descomprimirse, se encuentra un *README* donde se encuentran las dependencias que deben estar instaladas:

- CMAKE 3.12

- OpenSSL
- LibXML2

Además, recomienda poner los archivos en  `${HOME}/sdk/`.

Para generar el instalador de las dependencias e instalarlo debemos ejecutar los siguientes comandos:

1. `sudo apt-get install cmake`
2. `sudo apt-get install libssl-dev`
3. `sudo apt-get install libxml2-dev`
4. `cmake CMakeLists.txt`
5. `make`

Una vez instaladas las dependencias y el propio `sdk`, toca configurar la instalación del módulo OPC UA.

Para ello es necesario indicarle el directorio, con su ruta, tanto de EPICS como del `sdk`. Cada uno se le indica en un archivo distinto.

Para la dirección de EPICS, puede ser un archivo compartido para el resto de los módulos, es decir, si tenemos la carpeta “support” y dentro de ella los diferentes módulos, dentro de esa carpeta madre se crea un archivo llamado “`RELEASE.local`” y se le añade la dirección de esta forma: “`EPICS_BASE:=/home/juanca/EPICS/epics-base`”.

Para el `sdk` se debe configurar el archivo “`CONFIG_SITE`”, ubicado en la carpeta “`configure`” dentro del módulo, añadiendo “`UASDK = /home/juanca/sdk`”.

Solo faltaría ejecutar `make` en la carpeta principal del módulo para instalarlo.

### 5.6.1.2 Phoebus

Phoebus se puede descargar desde la página oficial de CS-Studio o desde su repositorio de github, para este segundo caso lo descargamos en la carpeta `home`:

1. `git clone --recursive https://github.com/ControlSystemStudio/phoebus.git`

#### 5.6.1.2.1 Dependencias

Se necesita:

- JDK11 o superior, recomiendan OpenJDK
- Maven 3.x o ant

1. `sudo apt-get install openjdk-11-jdk`

2. `sudo apt-get install maven`

Entonces ya se puede ejecutar el script “`Phoebus.sh`”, el cual abre su interfaz.

### 5.6.2 WinCC OA

En el caso de WinCC OA todos los módulos y drivers vienen incluidos en el ejecutable. Por lo que en este punto ya se encuentran instalados.

## 5.7 Creación del proyecto

### 5.7.1 EPICS – Creación de un IOC

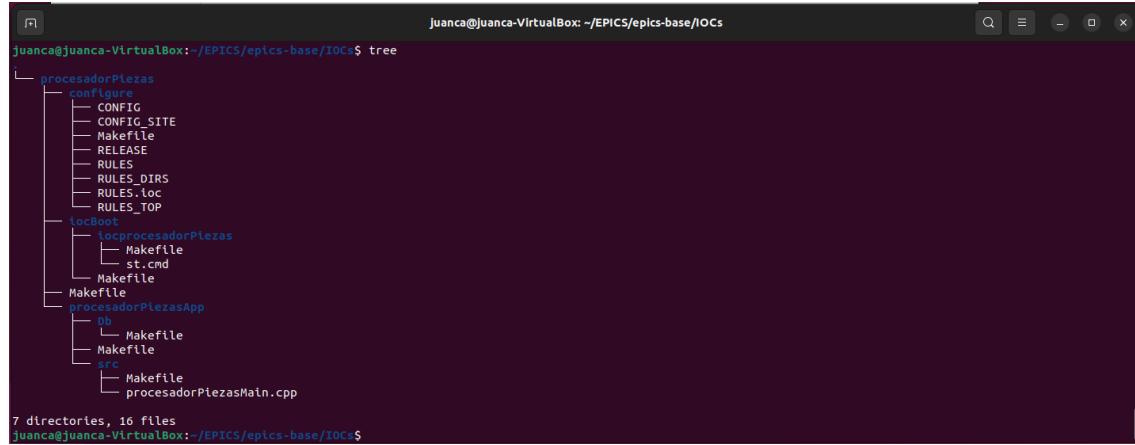
Como ya hemos estado viendo, EPICS no tiene ninguna interfaz gráfica para su configuración. Para crear proyectos, no cambia ese hecho, pero facilita un script llamado “`makeBaseApp.pl`” el cual

genera varias estructuras de directorios y archivos, en función del tipo de aplicación que se desee crear.

Dentro de la ubicación que deseemos, en este caso `~/EPICS/epics-base/IOCs/procesadorPiezas` ejecutaremos los siguientes comandos:

1. `makeBaseApp.pl -t ioc procesadorPiezas`
2. `makeBaseApp.pl -i -t ioc procesadorPiezas`

Y lo que queda es la siguiente estructura de directorios:



```
juanca@juanca-VirtualBox:~/EPICS/epics-base/IOCs$ tree
└── procesadorPiezas
    ├── configure
    │   ├── CONFIG
    │   ├── CONFIG_SITE
    │   ├── Makefile
    │   ├── RELEASE
    │   ├── RULES
    │   ├── RULES_DIRS
    │   ├── RULES_toc
    │   └── RULES_TOP
    ├── iocBoot
    │   └── iocprocesadorPiezas
    │       ├── Makefile
    │       └── st.cmd
    └── Makefile
        └── procesadorPiezasApp
            ├── nb
            │   └── Makefile
            ├── Makefile
            └── src
                └── Makefile
                    └── procesadorPiezasMain.cpp

7 directories, 16 files
juanca@juanca-VirtualBox:~/EPICS/epics-base/IOCs$
```

Ilustración 59 - EPICS Estructura de directorios de la aplicación base

En este caso, la ejecución del script es instantánea ya que lo único que hace es crear esa estructura de directorios.

Tambien hay que tener en cuenta que realmente esto no es un proyecto sino un IOC base, a éste habrá que añadirle posteriormente las configuraciones y archivos necesarios para que pueda hacer uso del módulo de comunicación.

### 5.7.2 WinCC OA

Para WinCC OA existe una aplicación que se encarga de la creación y administración de proyectos, *WinCC OA Project Administration*.



Ilustración 60 - Aplicación WinCC OA Project Administration

Una vez dentro, se muestra una lista con todos los proyectos disponibles. Para crear uno nuevo, se puede usar la combinación de teclas *CTRL + N* o pulsar el ícono dentro del recuadro en rojo de la siguiente ilustración:

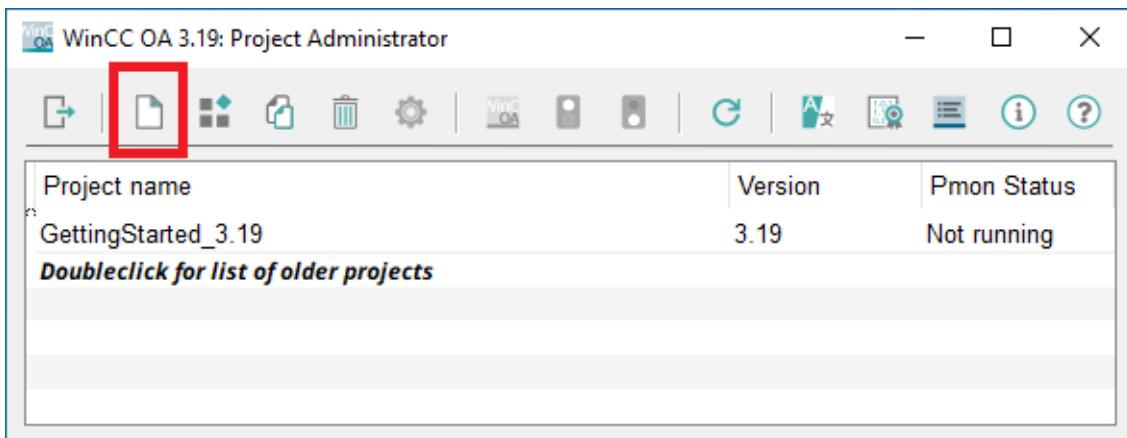


Ilustración 62 – WinCC OA Project Administration Lista de proyectos e ícono de creación

Al pulsar el ícono se abre una nueva ventana donde se puede elegir el tipo de proyecto. El que más se adecua a la aplicación de este TFG será el proyecto estándar:

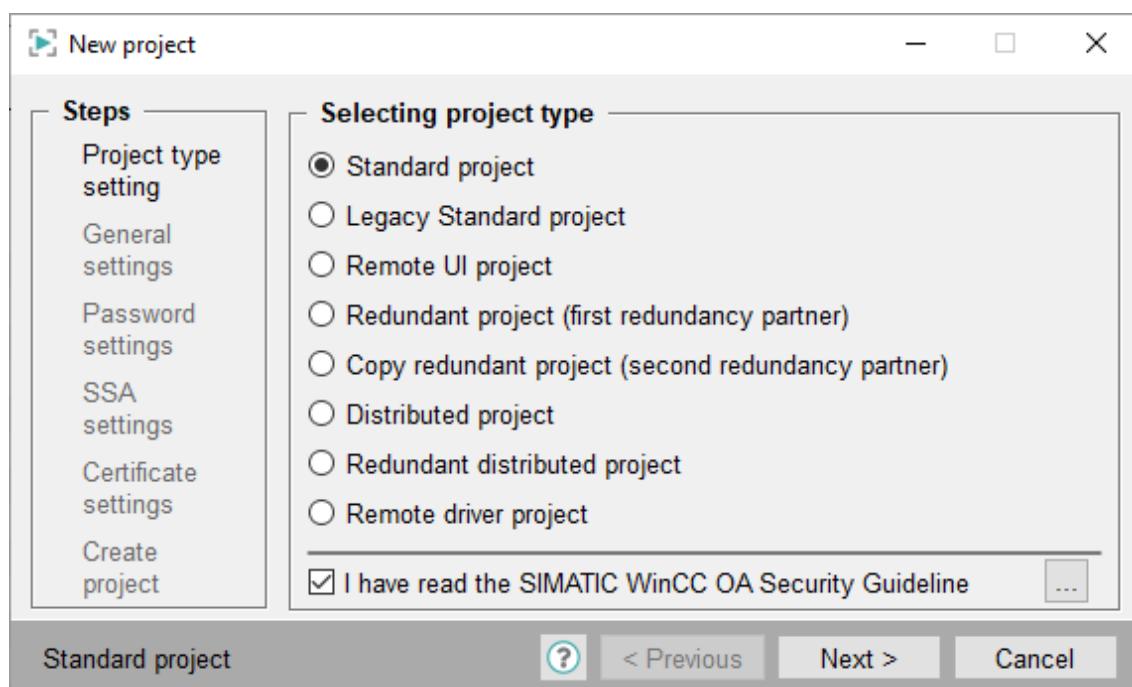


Ilustración 61 - WinCC OA Project Administration Selección tipo proyecto

Luego hay que asignarle un nombre, *ProcesadorPiezas*, un idioma, *Spanish – Spain*, una ruta y que sea ejecutable:

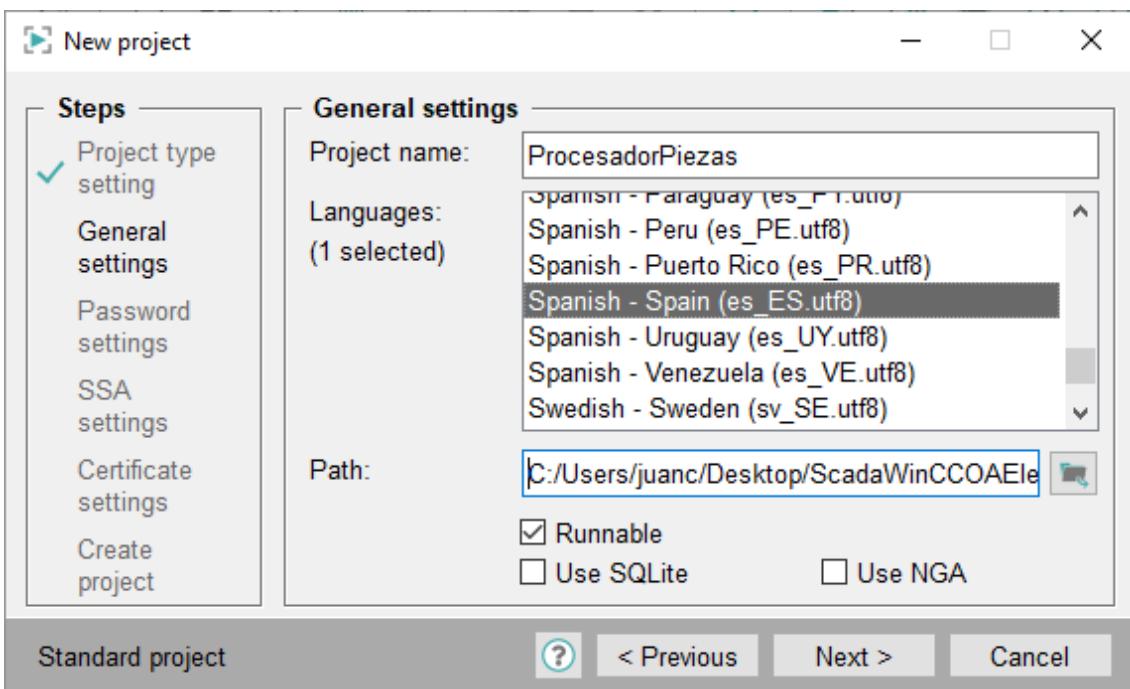


Ilustración 63 - WinCC OA Project Administration Nombre, ruta, idioma y ejecutable

El siguiente paso es la asignación de contraseñas para los usuarios por defecto y los usuarios opcionales. Como se puede apreciar en la siguiente ilustración las contraseñas siguen la tipología:

*tipo usuario – “Password!1”*

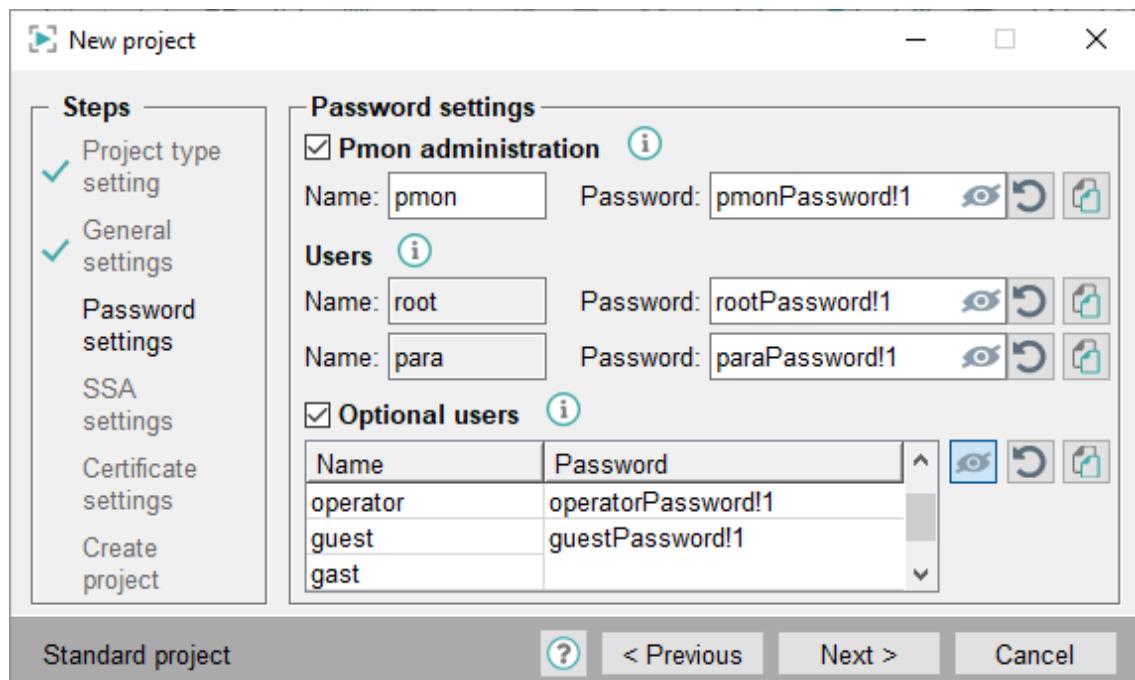


Ilustración 64 - WinCC OA Project Administration Usuarios y contraseñas

Como penultimo paso, se selecciona si se quiere el servidor en local:

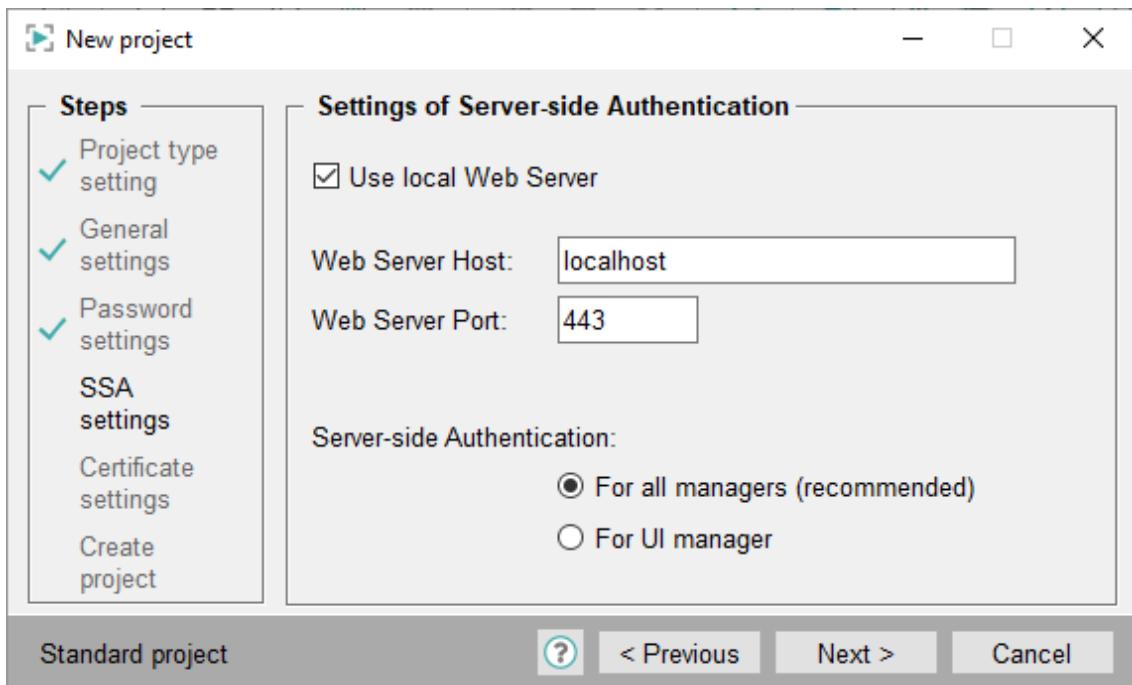


Ilustración 66 - WinCC OA Project Administration Servidor en local

Por último, hay que asignar una contraseña a los certificados para el proyecto:

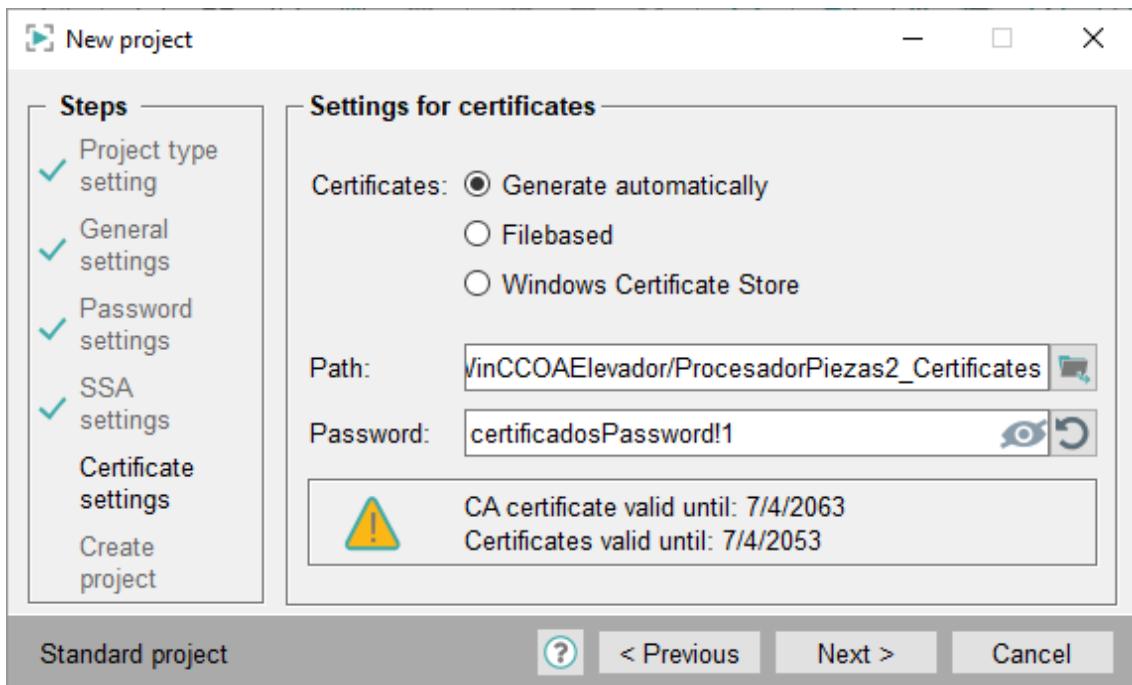


Ilustración 65 - WinCC OA Project Administration Certificados

Antes de la creacion, aporta un resumen de todo lo seleccionado y el tema de los iconos para el proyecto:

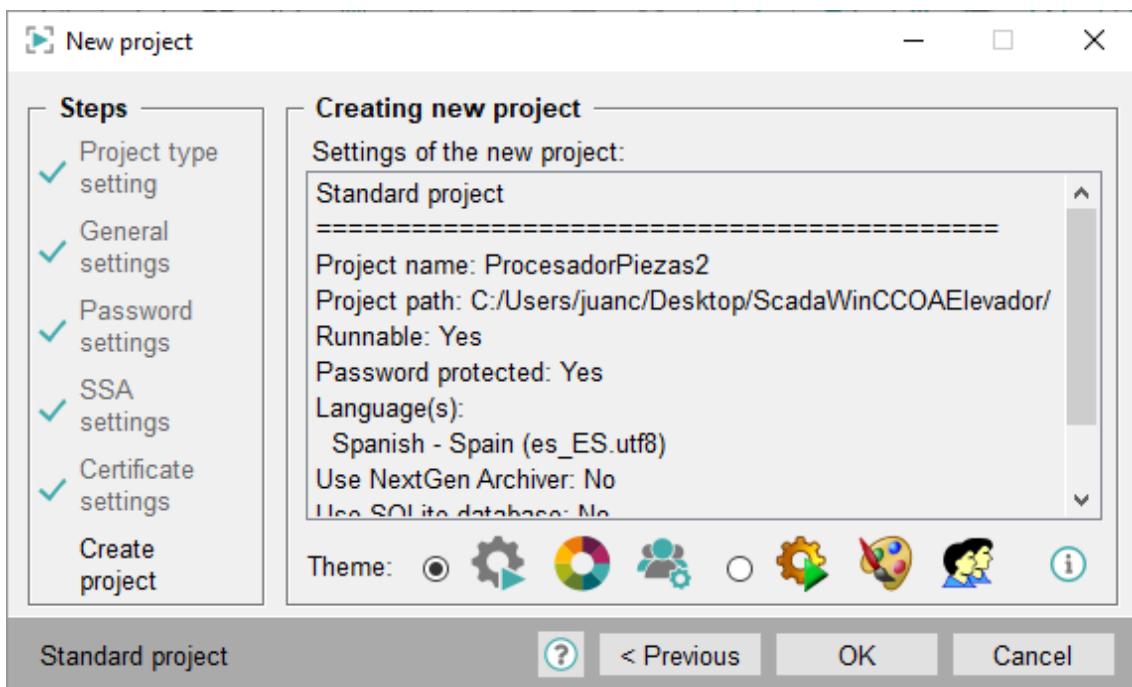


Ilustración 67 - WinCC OA > Project Administration > Resumen

Dándole a **OK**, comenzara la creación del proyecto:

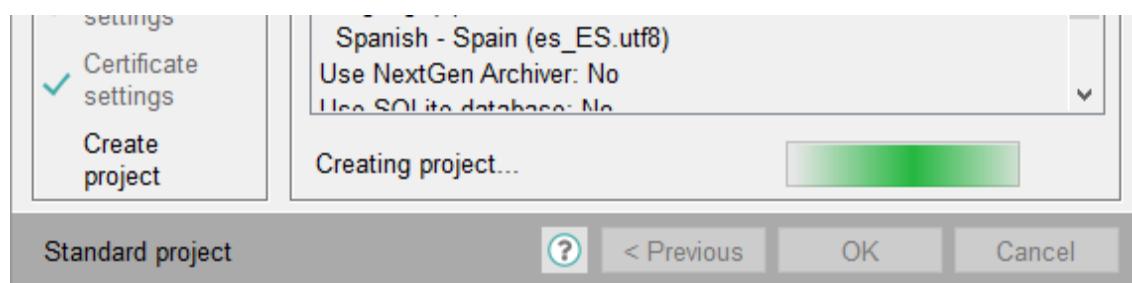


Ilustración 68 - WinCC OA > Project Administration > Creando

La cual ha tardado **26,04 segundos** en ejecutarse.

Repetiendo este proceso 6 veces con la misma configuración obtenemos:

1. 26,04
2. 24,57
3. 23,98
4. 27,32
5. 32,76
6. 28,11

Dado una media de **27,13 segundos**.

En este paso ya se puede ver el proyecto en la lista:

Project Administrator		
Project name	Version	Pmon Status
GettingStarted_3.19	3.19	Not running
ProcesadorPiezas	3.19	Not running

Ilustración 69 - WinCC OA > Project Administration > Lista de

## 5.8 Configuración de drivers

### 5.8.1 EPICS

Para que el IOC que creas en el anterior apartado pueda hacer uso del driver OPCUA, se deben añadir tres cosas:

- Tener en un RELEASE.local la ruta hasta el módulo.
- Incluir el archivo “*opcua.dbd*” a la hora del montaje del IOC
- Incluir la librería “*opcua*”

La entrada en el archivo RELEASE.local tiene esta forma:

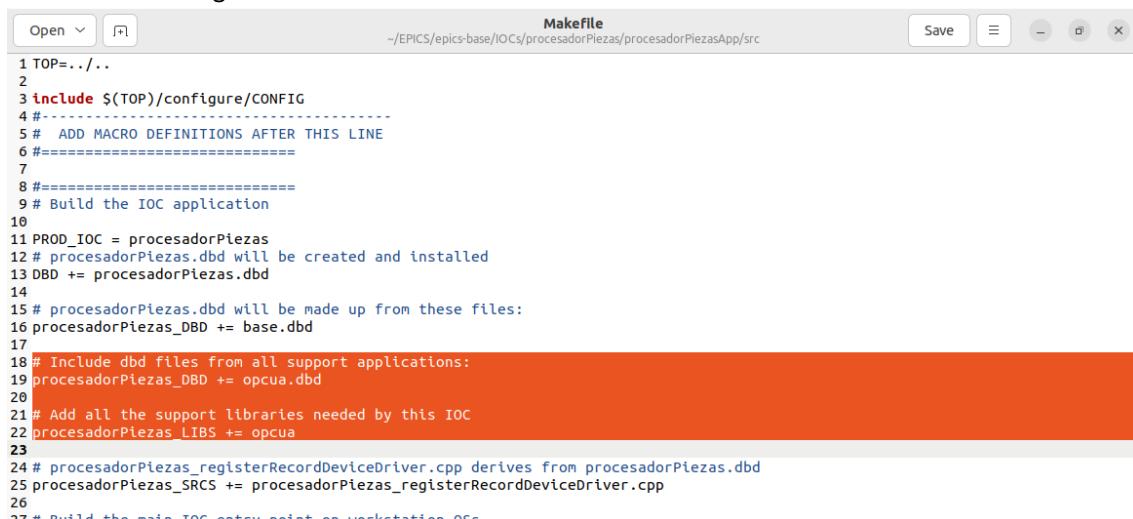
**OPCUA = /home/juanca/EPICS/support/opcua**

Alternativamente a ese archivo, se puede modificar el *configure/RELEASE* del IOC añadiendo esa misma entrada.

Para los dos últimos apartados hay que modificar el archivo *procesadorPiezasApp/src/Makefile* añadiéndole las siguientes líneas:

```
procesadorPiezas_DB += base.dbd  
procesadorPiezas_LIBS += opcua
```

Quedando de la siguiente forma:



```
Open + Makefile ~/EPICS/epics-base/IOCs/procesadorPiezas/procesadorPiezasApp/src Save ... ×  
1 TOP=../..  
2  
3 include $(TOP)/configure/CONFIG  
4 #-----  
5 # ADD MACRO DEFINITIONS AFTER THIS LINE  
6 #####  
7  
8 #####  
9 # Build the IOC application  
10  
11 PROD_IOC = procesadorPiezas  
12 # procesadorPiezas.dbd will be created and installed  
13 DBD += procesadorPiezas.dbd  
14  
15 # procesadorPiezas.dbd will be made up from these files:  
16 procesadorPiezas_DB += base.dbd  
17  
18 # Include dbd files from all support applications:  
19 procesadorPiezas_DB += opcua.dbd  
20  
21 # Add all the support libraries needed by this IOC  
22 procesadorPiezas_LIBS += opcua  
23  
24 # procesadorPiezas_registerRecordDeviceDriver.cpp derives from procesadorPiezas.dbd  
25 procesadorPiezas_SRCS += procesadorPiezas_registerRecordDeviceDriver.cpp  
26  
27 # Build the main IOC entry point on workstation nse.
```

Ilustración 70 - EPICS procesadorPiezasApp/src/Makefile modificado

Para hacerse una idea jerárquica de estos archivos, se muestra la siguiente imagen, marcándose en amarillo y subrayado en rojo lo que se ha modificado y en verde subrayado la alternativa al RELEASE.local:

```

juanca@juanca-VirtualBox:~/EPICS/epics-base/IOCs$ tree
.
+-- procesadorPiezas
|   +-- configure
|   |   +-- CONFIG
|   |   +-- CONFIG_SITE
|   |   +-- Makefile
|   |   +-- RELEASE
|   |   +-- RULES
|   |   +-- RULES_DIRS
|   |   +-- RULES.ioc
|   |   +-- RULES_TOP
|   +-- tocBoot
|       +-- tocprocesadorPiezas
|           +-- Makefile
|           +-- st.cmd
|       +-- Makefile
|   +-- procesadorPiezasApp
|       +-- Db
|           +-- Makefile
|       +-- Makefile
|       +-- src
|           +-- Makefile
|           +-- procesadorPiezasMain.cpp
|   +-- RELEASE.local
.
7 directories, 17 files

```

Ilustración 71 - EPICS configuración de drivers Estructura de directorios Archivos modificados

## 5.8.2 WinCC OA Driver manager y OPC Client

Antes de ejecutar el proyecto, de cara a edición, desde la consola *WinCC OA Console* del proyecto se debe incorporar el *manager* para *OPC UA Client*:

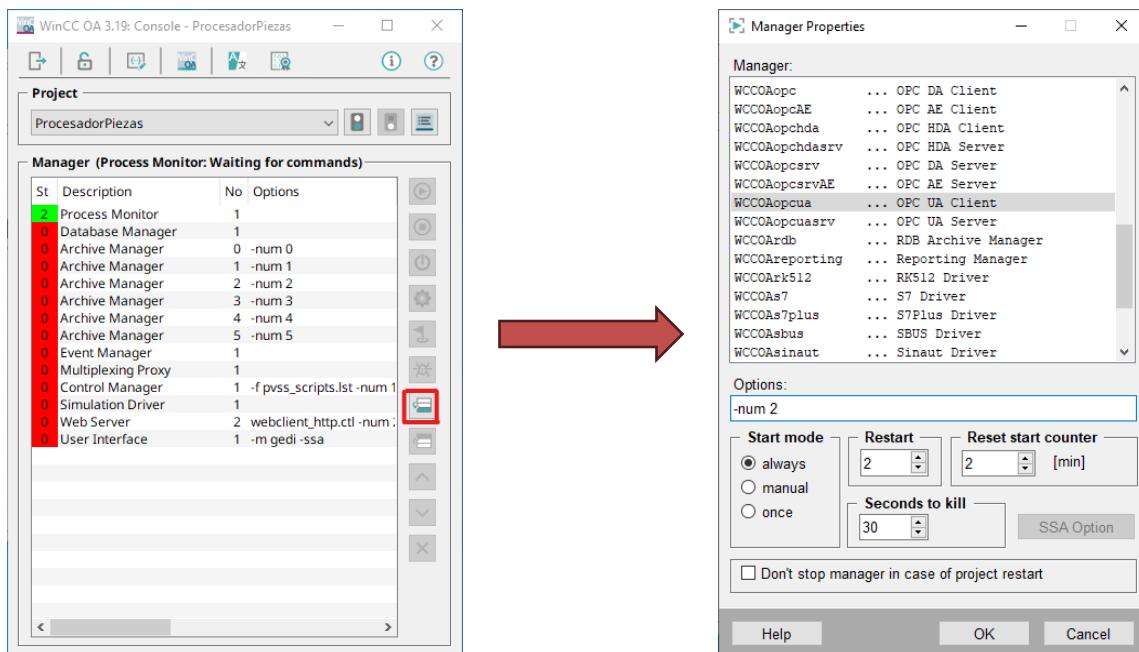


Ilustración 72 - *WinCC OA > Console > incorporación del cliente OPC*

En las opciones, se le debe asignar un número el cual no es arbitrario y se debe tener en cuenta despues para la configuración de la sesión OPC.

También se configura como quiere que se comporte el *manager*, como cuando se inicia, cada cuanto reiniciarse en caso de caer o el tiempo que puede estar bloqueado antes de cerrarse.

Para ejecutar el proyecto hay que clicar sobre el icono justo a la derecha del desplegable donde pone el nombre del proyecto y esperar a que se inicie *Gedi*, que es la interfaz para creación y configuración

de paneles. En él se enlaza los objetos gráficos con los *Data Points*, los cuales se crean en el módulo *Para*.

Pero el siguiente paso es abrir el *System Management* para terminar la configuración OPC, el cual se puede acceder desde el ícono del engranaje indicado dentro del recuadro rojo:

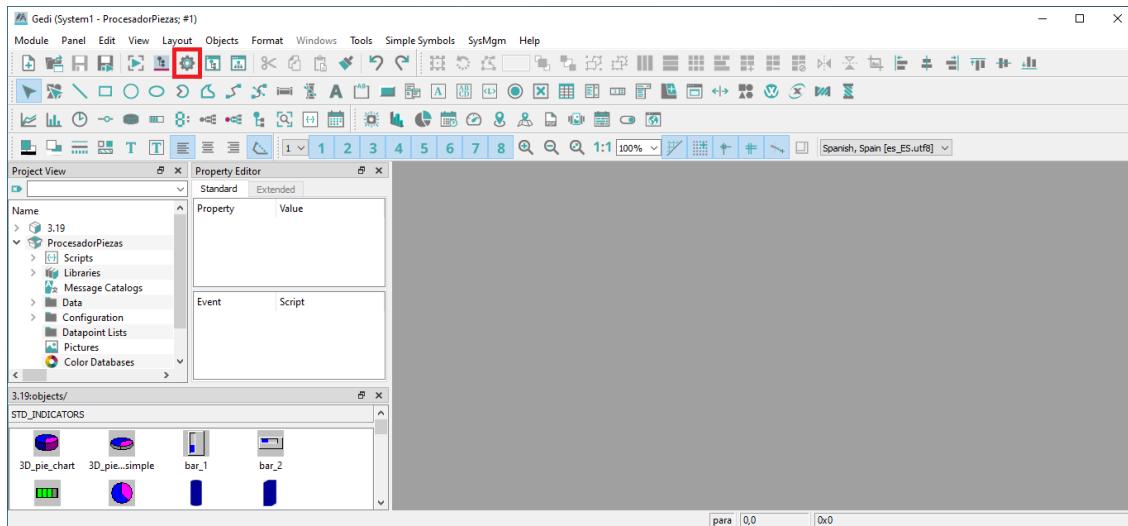


Ilustración 73 - WinCC OA Gedi > Interfaz con System Management indicado

Una vez abierto, se selecciona el *Driver OPC*:

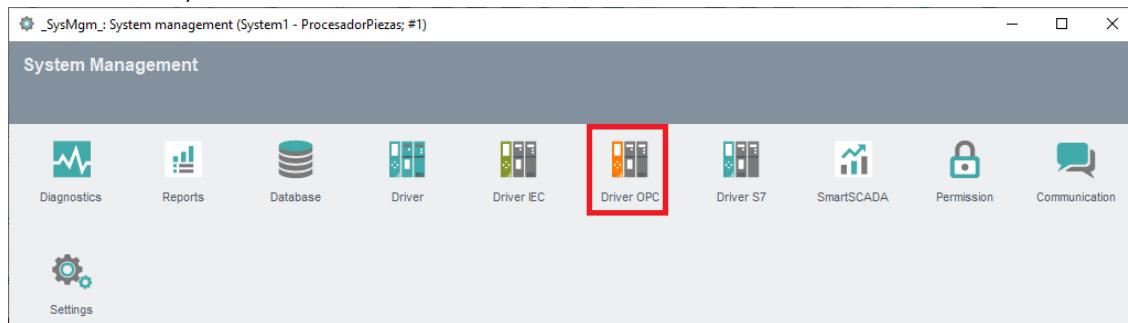


Ilustración 74 - WinCC OA > System Management

Y dentro también se selecciona *OPC UA Driver*:

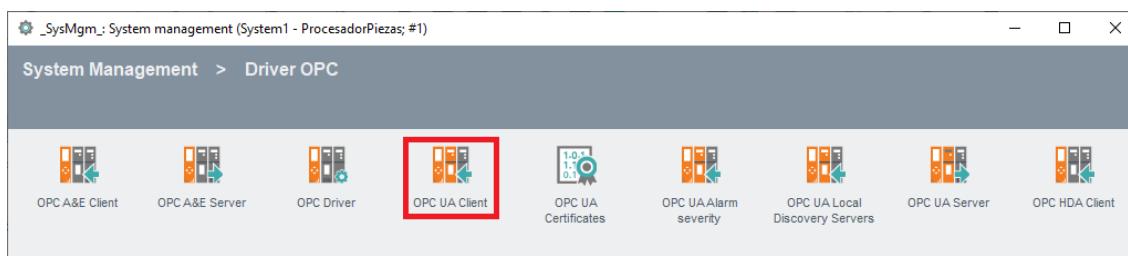


Ilustración 75 - WinCC OA > System Management > Driver OPC

Entonces aparecerá el siguiente cuadro, donde se deberá dar a crear para configurar la conexión con el servidor OPC en KepServer:

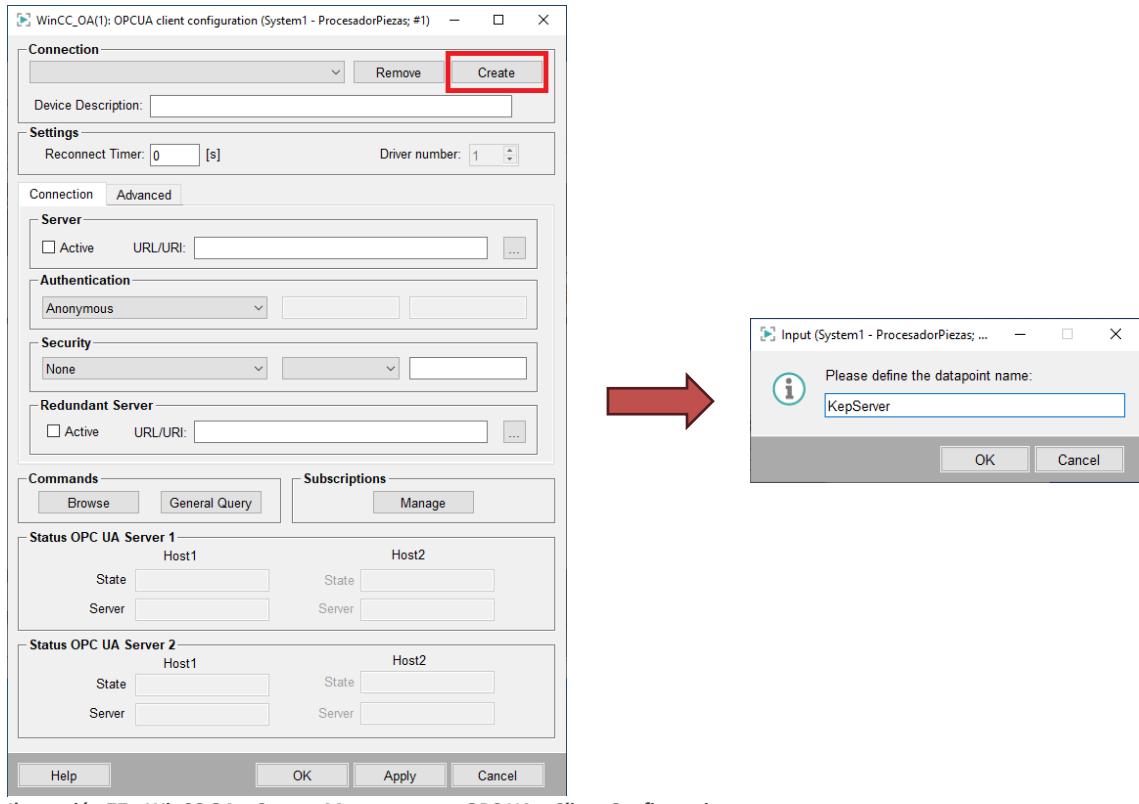


Ilustración 77 - WinCC OA > System Management > OPC UA > Client Configuration

Entonces volverá a la ventana de la imagen de la izquierda de la ilustración 56 y se rellenará con la dirección del servidor OPC en KepServer. Después, la misma ventana, para configurar una suscripción se pulsará el botón *Manage* dentro del recuadro *Subscriptions* y de igual forma se le crea con el botón *Create*:

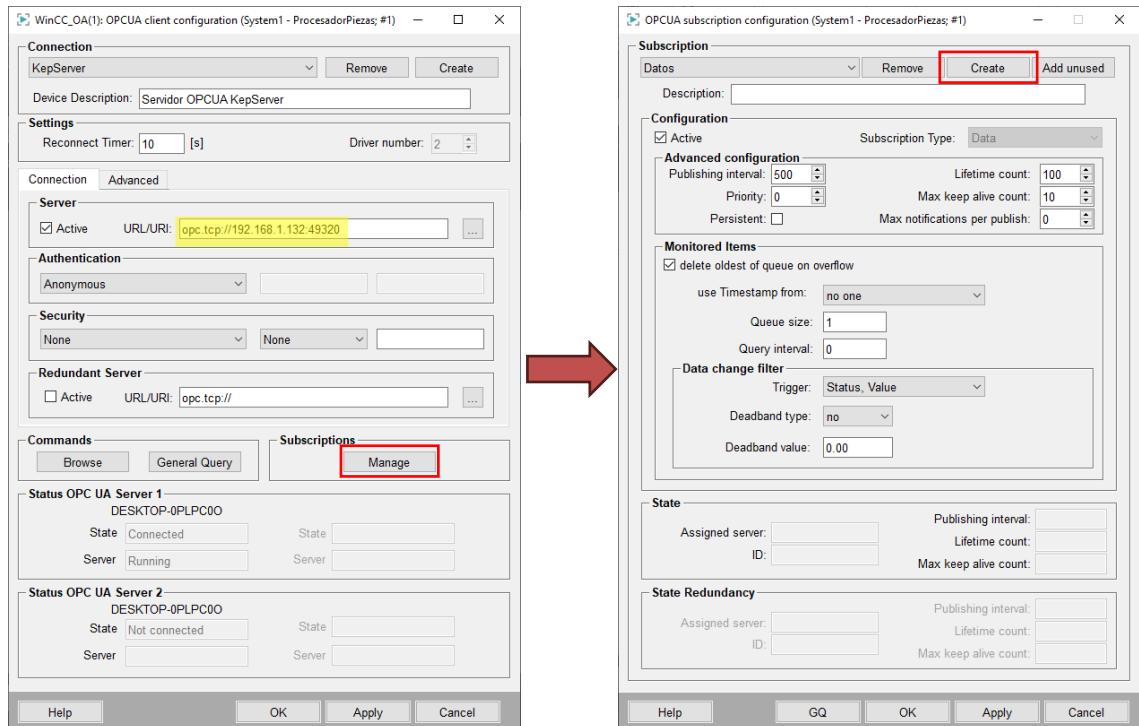


Ilustración 76 - WinCC OA OPC UA > System Management > Client Configuration > Dirección y suscripción

Si el resto de los componentes software se encuentran en funcionamiento, se puede hacer una rápida comprobación de conexión pulsando el botón *Browse*, dentro del recuadro *Commands*, de la ventana de la configuración del cliente y acceder a los valores que proporciona el servidor:

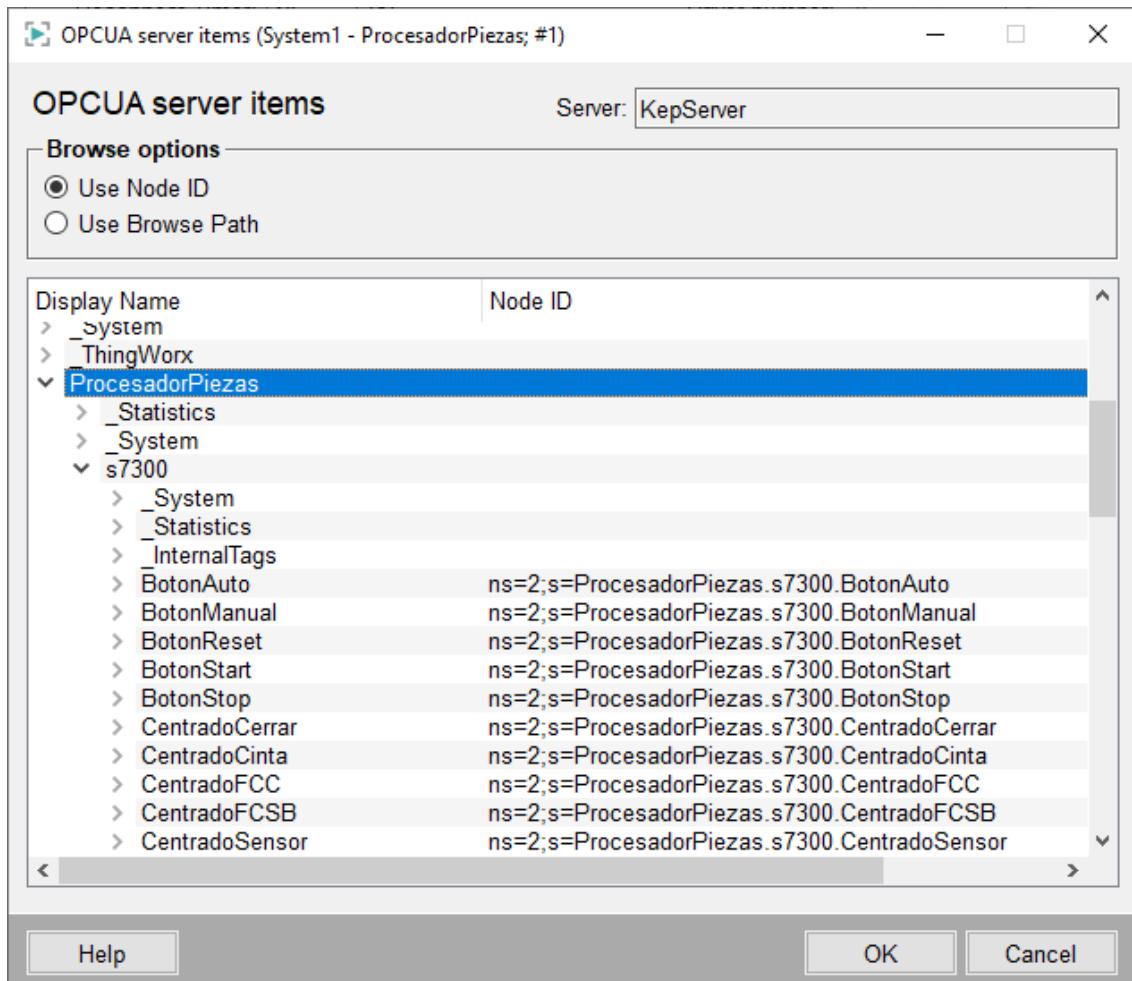


Ilustración 78 - WinCC OA OPC UA > System Management > Client Configuration > Browse

## 5.9 Creación de las variables del proceso

### 5.9.1 EPICS Records

Las variables de proceso en EPICS se configuran con los ya comentados *records*. Existe una extensión llamada VDCT (*Visual Database Configuration Tool*) que es una herramienta gráfica de configuración de las bases de datos de EPICS. Con ella los *records* pueden ser creados, movidos, relacionados de forma sencilla. Además, muestra de forma jerárquica y ordenada a éstos y tambien señala la direccionalidad de los datos con flechas.

Es apreciable que su interfaz se encuentra bastante anticuada. Aunque se sigue manteniendo, las actualizaciones son escasas, de media una cada año por lo que muestra su portal de GitHub.

Debido a que el proceso de instalación y aprendizaje sobre esta herramienta iba a conllevar más tiempo que la creación manual, no se ha usado.

En este punto se pueden exportar las etiquetas desde KepServer, tratarlas para extraerle solo el nombre y así poder crear los records con dichos nombres.

Cada tipo de dato tiene una estructura ligeramente diferente, teniendo en cuenta que tendrán en común la configuración de OPCUA. Por lo que se van a mostrar:

#### 5.9.1.1 Binary input (bi)

```
record(bi, "[NOMBRE-RECORD]") {
    field(DTYP, "OPCUA")
    field(INP, "@SUB1 ns=2;s=[DIRECCION]. [ETIQUETA]")
    field(SCAN, "I/O Intr")
}
```

Para facilitar la memorización de los records, [ETIQUETA] y [NOMBRE-RECORD] serán iguales. [DIRECCION] tiene que incorporar el nombre del canal “*ProcesadorPiezas*” y el nombre del dispositivo “*s7300*”. Qedaría tal que:

```
record(bi, "ClasificadoraCinta") {
    field(DTYP, "OPCUA")
    field(INP, "@SUB1 ns=2;s=ProcesadorPiezas.s7300.ClasificadoraCinta")
    field(SCAN, "I/O Intr")
}
```

#### 5.9.1.2 Binary Output (bo)

```
record(bo, "[NOMBRE-RECORD]") {
    field(DTYP, "OPCUA")
    field(OUT, "@SUB1 ns=2;s=[DIRECCION]. [ETIQUETA]")
}
```

#### 5.9.1.3 Analog Input (ai)

```
record(ai, "[NOMBRE-RECORD]") {
    field(DTYP, "OPCUA")
    field(INP, "@SUB1 ns=2;s=[DIRECCION]. [ETIQUETA]")
    field(SCAN, "I/O Intr")
```

```
}
```

#### 5.9.1.4 Multi-bit Binary Input (mbbi)

```
record(mbbi, "[NOMBRE-RECORD] ") {
    field(DTYP, "OPCUA")
    field(INP, "@SUB1 ns=2;s=[DIRECCION]. [ETIQUETA]")
    field(SCAN, "I/O Intr")
}
```

Conociendo entonces la estructura que debe tener cada tipo de dato, los nombres de esos datos, es bastante simple crear un programa auxiliar que cree un archivo con todos los *records* configurados de esta forma.

En este caso se ha usado **Python** por su simplicidad. El código de dicho programa es el siguiente:

```
# Listas con los nombres de las etiquetas extraídas de Kepserver
inputs = ["BotonAuto", "BotonManual", [...] ]
output = ["CMActivado", "CMCentradoCerrar", [...] ]
inputsAnalogicos = ["PesadoRociadoFlowSensor", "PesadoRociadoDescarga", [...] ]

# Preambulos, mitad texto y post texto a las sustituciones de las variables analógicas
preAnalogico = "record(mbbi, \""
midAnalogico = "\") {\n\tfield(DTYP, \"OPCUA\")\n\tfield(INP,  \"@SUB1\nns=2;s=ProcesadorPiezas.s7300."
postAnalogico = "\")\n\tfield(SCAN, \"I/O Intr\")\n}\n\n"

# Preambulos, mitad texto y post texto a las sustituciones de las variables binarias
# de entrada
preInput = "record(bi, \""
midInput = "\") { \n\tfield(DTYP, \"OPCUA\")\n\tfield(INP,  \"@SUB1\nns=2;s=ProcesadorPiezas.s7300."
postInput = "\")\n\tfield(SCAN, \"I/O Intr\")\n}\n\n"

# Preambulos, mitad texto y post texto a las sustituciones de las variables binarias
# de salida
preOutput = "record(bo, \""
midOutput = "\") {\n\tfield(DTYP, \"OPCUA\")\n\tfield(OUT,  \"@SUB1\nns=2;s=ProcesadorPiezas.s7300."
postOutput = "\")\n}\n\n"

archivo = open('salidas.txt', 'a')

for i in inputs:
    archivo.write(preInput + i + midInput + i + postInput)
    archivo.write("\n\n")

for i in output:
    archivo.write(preOutput + i + midOutput + i + postOutput)
    archivo.write("\n\n")
```

```

for i in inputsAnalogicos:
    archivo.write(preAnalogico + i + midAnalogico + i + postAnalogico)

archivo.close()

```

El archivo resultante con todos los *records* se adjunta en el anexo 2 ([02](#)).

### 5.9.2 WinCC OA Data Points

El módulo PARA de WinCC OA es una interfaz gráfica para editar los *Data Point Types* y *Data Points*. Esta interfaz constituye una herramienta con la que se puede acceder a la base de datos interna y hacer modificaciones simultáneamente.



Ilustración 79 - Icono módulo PARA

Para acceder al módulo, se pulsa el icono de la ilustración 60 dentro de la ventana GEDI.

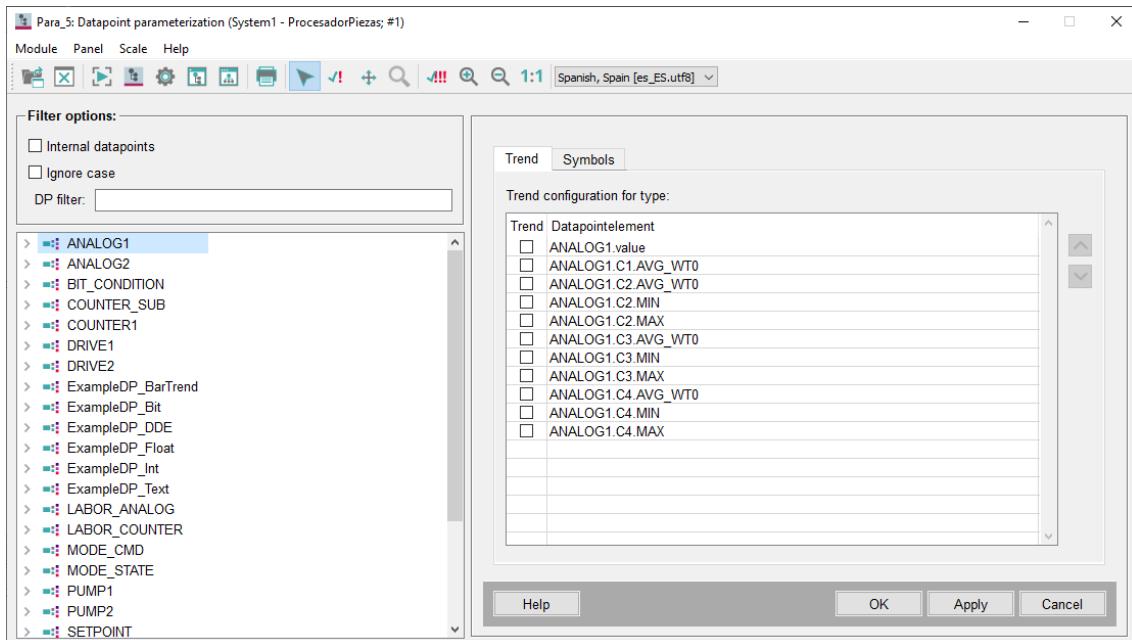


Ilustración 80 - WinCC OA PARA

Una vez dentro, se muestran los *Data Point Types* por defecto y dentro de ellos los *Data Points*.

Pero antes, hay que definir los tipos de datos básicos que se van a necesitar:

- Booleanos, como dato mayoritario
- Enteros y en coma flotante

Para poder adaptar cómodamente las necesidades de este proyecto, se van a crear dos *Data Point Types*: **Booleano** y **Double**.

Para crearlos, desde el módulo PARA, se pulsa el botón derecho sobre la lista y se selecciona “*Create Data Point Type*”:

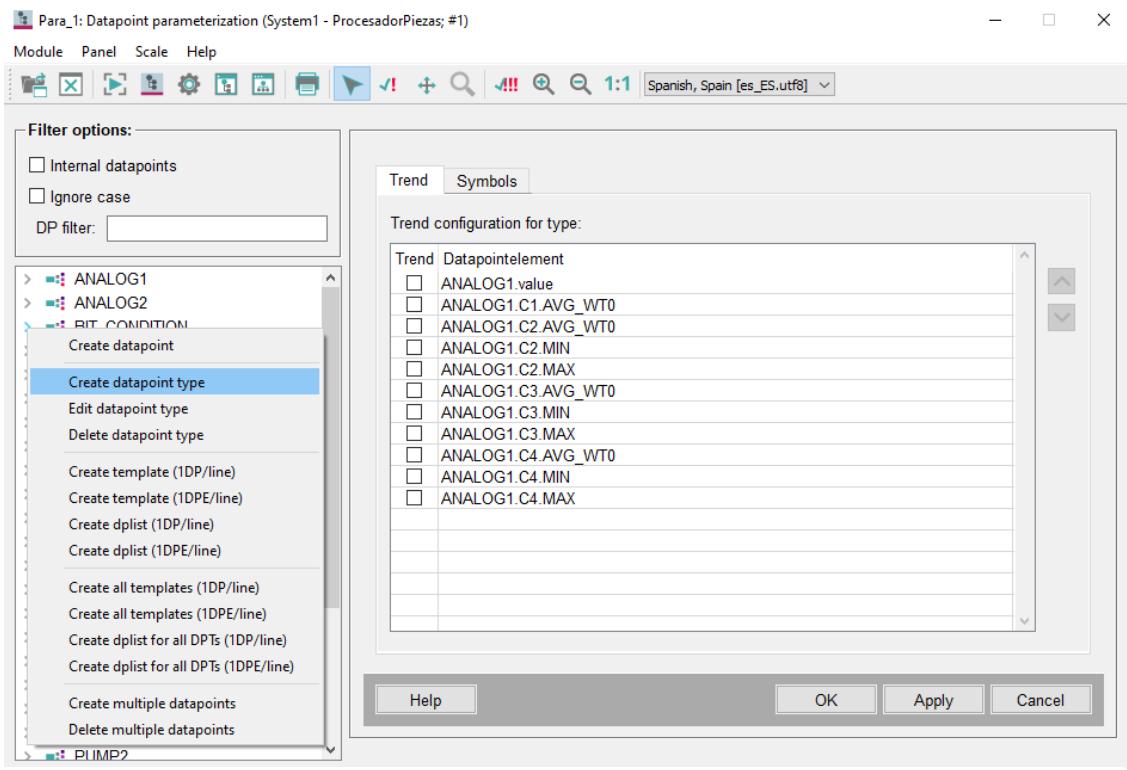


Ilustración 81 - WinCC OA > PARA Crear DPT

Entonces saldrá una nueva ventana con el icono de una carpeta a la cual se le cambia al nombre requerido y pulsando botón derecho saldrán unos desplegables donde se seleccionaran el tipo de dato o estructura. En la siguiente imagen se muestra el caso de **Booleano**:

De igual forma se hace con **Double** pero seleccionando en este caso *float*.

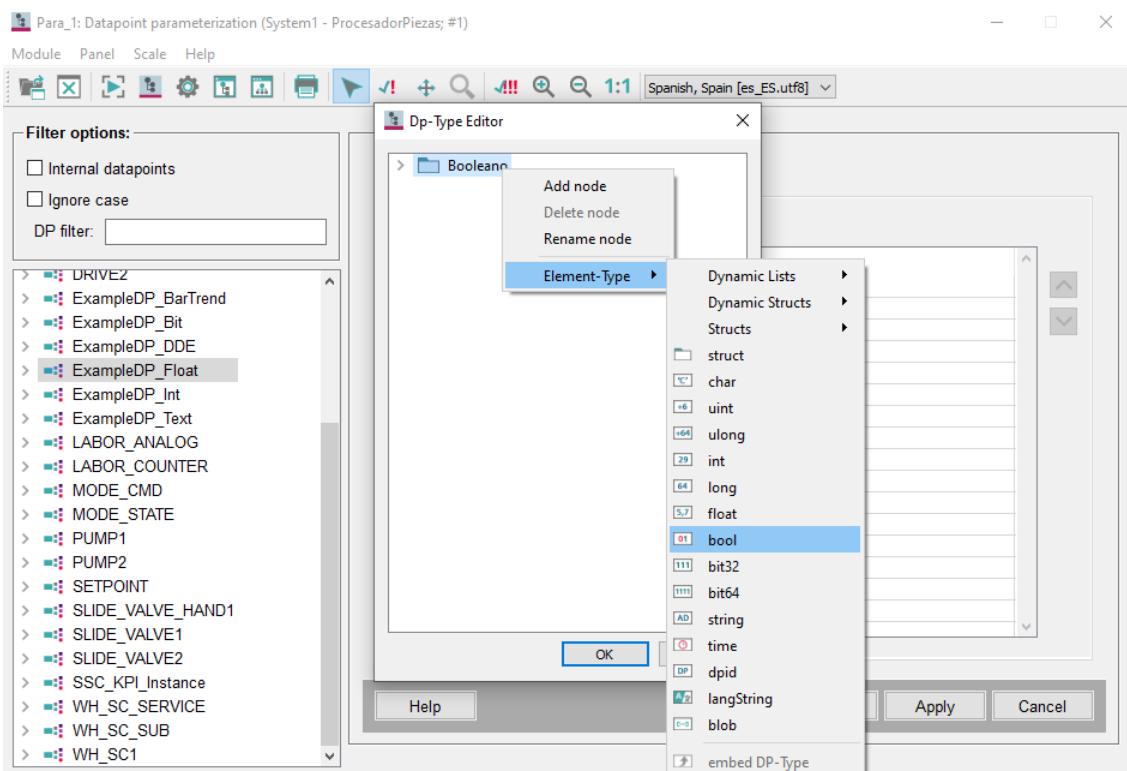


Ilustración 82 - WinCC OA > PARA > Dp-Type Editor Booleano

Es en este momento donde se van a crear todos los *Data Points*, los cuales se pueden crear individualmente, en grupo con ayuda de un formateador de texto integrado en WinCC OA o importarlos desde un archivo con *ASCII Manager* integrado tambien en el programa.

Se va a usar la misma metodología que con EPICS, con un programa Python se quiere formatear los datos de la siguiente forma:

DpName	TypeName	ID
BotonAuto	Booleano	501
CentradoCerrar	Booleano	506
PesadoRociadoLlenado	Double	556
[...]		
# PeriphAddrMain		
Manager/User	ElementName	TypeName _address.._type _address.._reference
UI (1)/0	BotonAuto.	Booleano 16 "KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.BotonAuto"
UI (1)/0	BotonManual.	Booleano 16 "KepServer\$atos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.BotonManual"
[...]		

Tabla 1 - Formateo de variables para importar en WinCC OA con ASCII Manager

El archivo correspondiente a la tabla 1 se puede encontrar en el anexo 4 ([0](#)).

Para importar ese archivo hay que acceder al *ASCII Manager* desde:

*GEDI > System Management > Database > ASCII Manager*

Su icono es el siguiente:



Ilustración 83 - WinCC OA ASCII Manager Icono

Entonces se abrirá su interfaz, la cual sirve tanto para importar como para exportar los *Data Points*, *Data Points Types* y configuraciones.

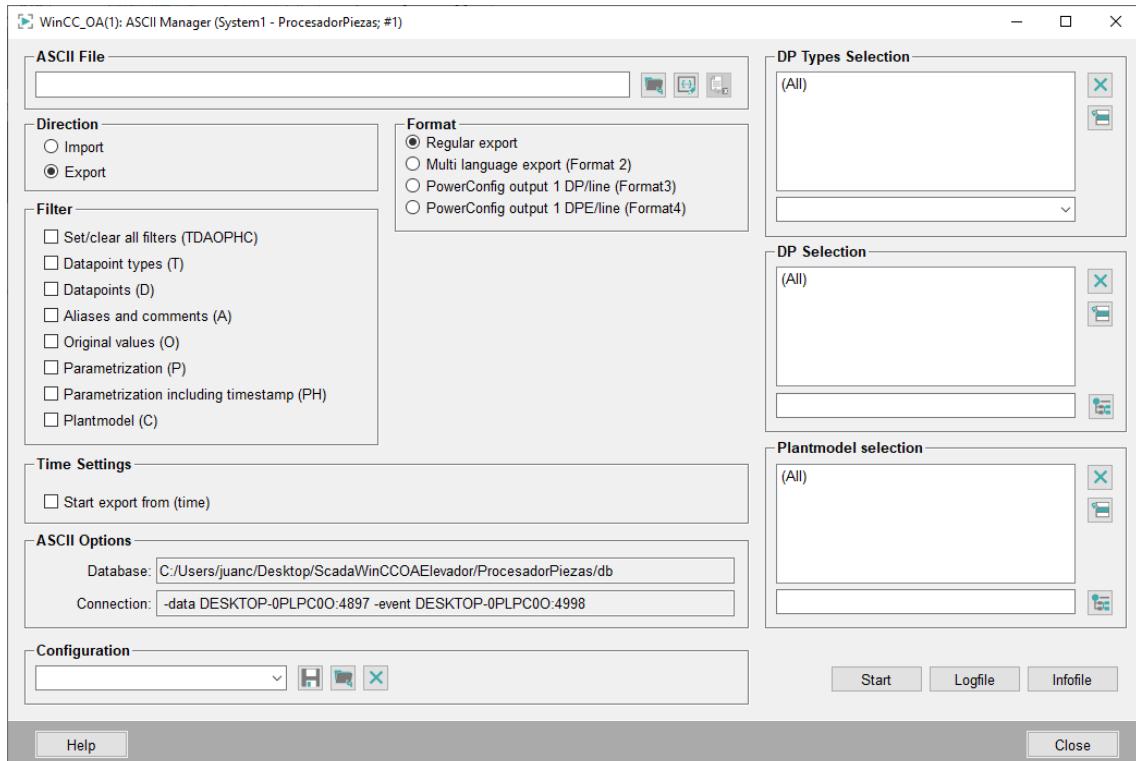


Ilustración 84 - WinCC OA > System Management > Database > ASCII Manager Interfaz

A la derecha del recuadro en blanco está el icono de la carpeta para seleccionar el archivo, que debe estar dentro de la carpeta “*dplist*” del proyecto de WinCC OA. Después en “*Direction*” se selecciona “*import*” y ya se puede dar a “*start*”.

Entonces ya se tendrá, y podrá visualizar dentro del módulo PARA, todos los *Data Points*:

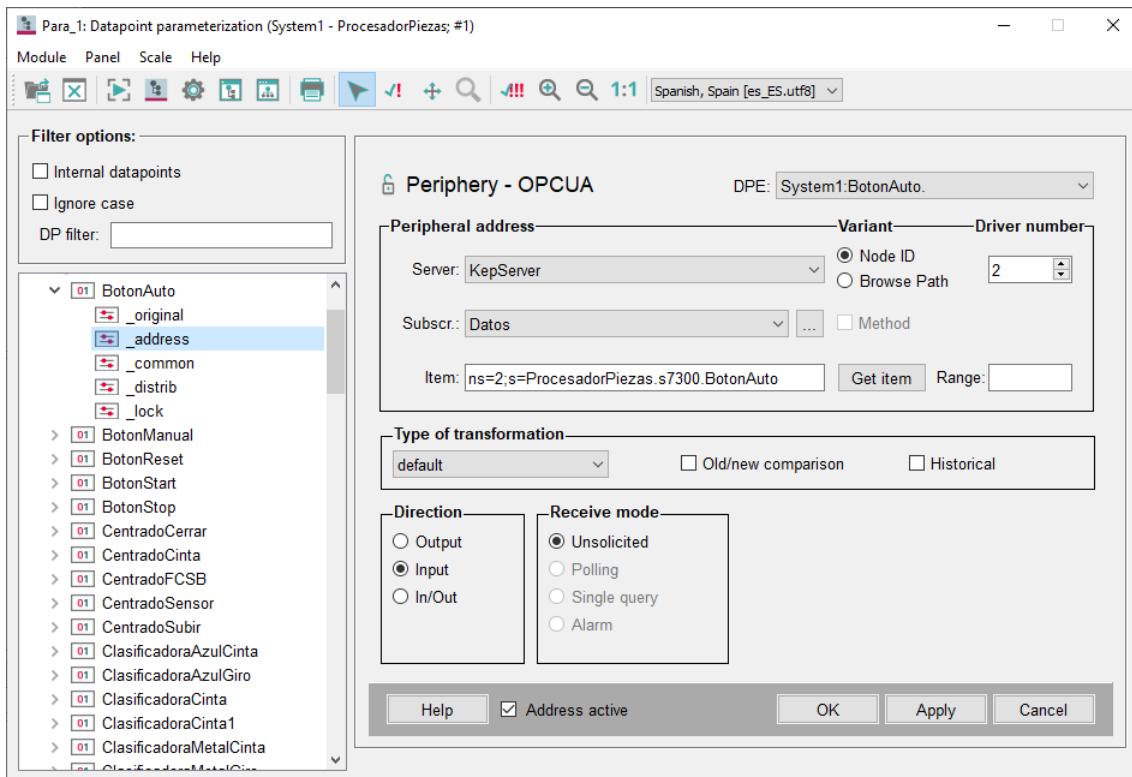


Ilustración 85 - WinCC OA > PARA Todos los DP insertados y configurados

## 5.10 Creación de la interfaz

La creación de un SCADA conlleva el uso de numerosos elementos, en su mayoría repetidos. Por lo que en esta sección se va a explicar como se ha creado solo el **panel principal**. Los elementos que se hayan usado en otros paneles y no en éste, se explicaran seguidamente.

La estructura de esta sección se compone de:

- **Introducción a las interfaces de edición**, donde se mostrarán las principales diferencias entre ambos programas.
- **Información central**, como se crean los objetos de la parte central del panel. Corresponde al recuadro de *Visualización*, véase el apartado [5.4](#).
- **Barra superior**, correspondiente a los accesos directos, común a todos los paneles.
- **Barra inferior**, correspondiente a la visualización y control tanto del estado de la maquina como de cada estación.
- **Resto de elementos**, con los elementos que no se han tratado en los apartados anteriores.

### 5.10.1 Introducción a las interfaces de edición

Antes de empezar a mostrar cómo se crearía una interfaz y el uso de los elementos más importantes, se va a mostrar una leve introducción a las interfaces de cada programa.

### 5.10.1.1 EPICS – Phoebus

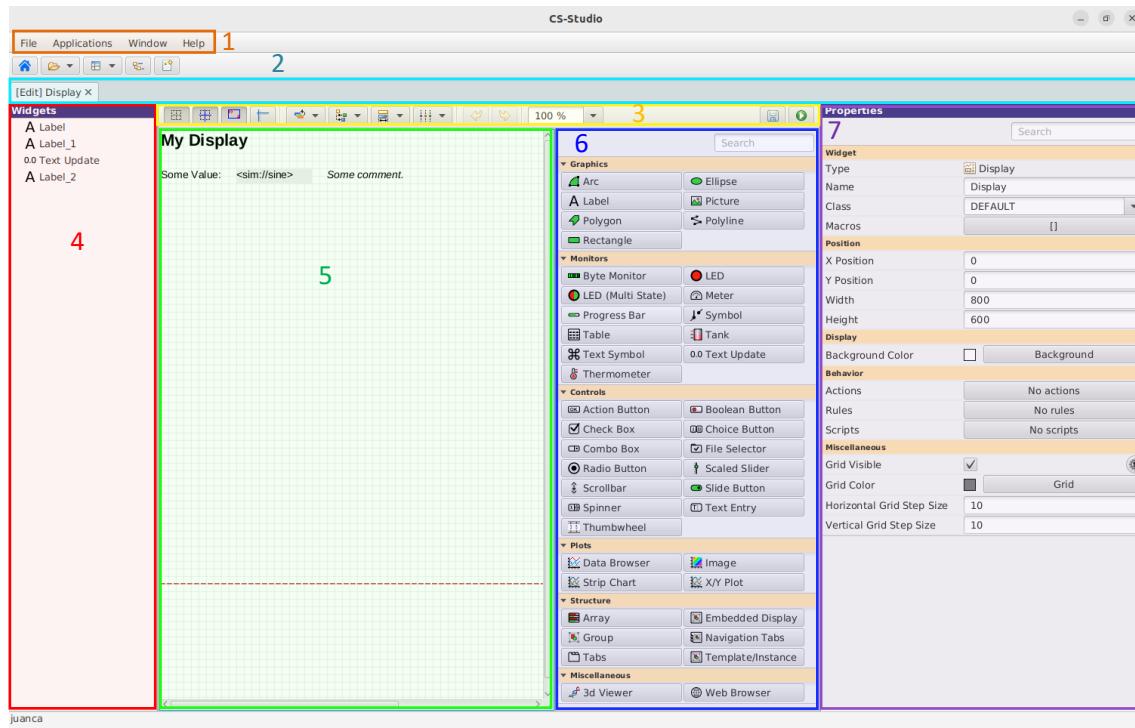


Ilustración 86 - Phoebus Interfaz

En la anterior ilustración, se muestra el editor de Phoebus con un panel abierto. Se han colocados recuadros numerados y de distintos colores para facilitar la su división.

En primer lugar, el menú superior remarcado en [1] naranja que contiene accesos a ciertos elementos básicos de edición:

- *File*, intuitivamente contiene accesos para abrir archivos .bob y abrir ejemplos o guardar el archivo actual.
- *Applications*, es de los más útiles ya que contiene los accesos directos para añadir objetos y pantallas relacionados con alarmas, depuración, paneles, escaneo y otras utilidades.
- *Window*, para la configuración de la distribución de los paneles abiertos y sus configuraciones.
- *Help*, información sobre la aplicación y la alguna documentación del programa.

En segundo, se encuentra un grupo de pestañas con los paneles abiertos marcado en [2] cian.

En tercer lugar y marcado en [3] amarillo, contiene herramientas para el manejo de la visualización del panel. Como por ejemplo básico el zoom, mostrar coordenadas del cursos con guías, guardar, ejecutar o el centrado de objetos entre los mas importantes.

Marcado en [4] rojo, es el panel donde se reúnen en orden de visualización los objetos por sus nombres. En él se pueden arrastrar para reordenar o cambiar el nombre de cada objeto.

El panel activo, marcado en [5] verde, se puede ver los objetos que contiene y con una línea roja discontinua su área de visualización.

A la derecha del anterior, marcado en [6] azul es el panel con todos los objetos disponibles, los cuales trataremos los más importantes más adelante.

En [7] morado, es la ventana de configuración del objeto seleccionado. En el momento de la captura, al no haber seleccionado ninguno, se muestra la configuración del panel. En esta ventana se configurarán las conexiones con los *records*, animaciones, reglas o scripts (Python 2.8 o Java Script). Se puede decir que es de las ventanas más importantes.

### 5.10.1.2 WinCC OA – GEDI

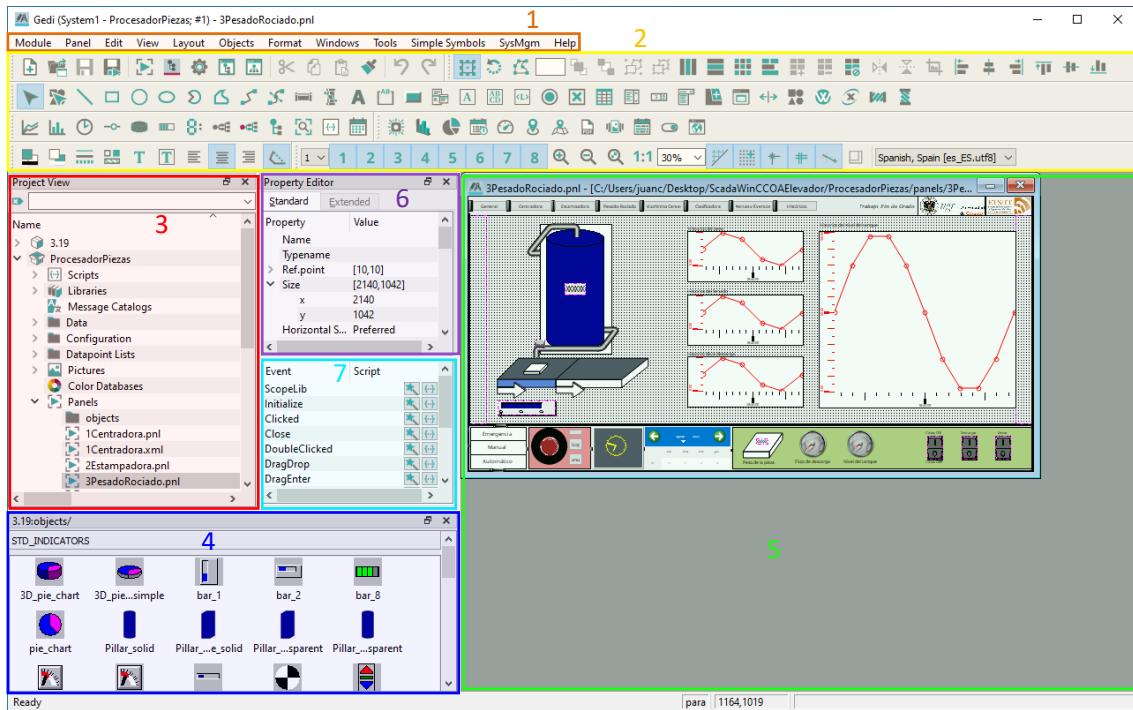


Ilustración 87 - WinCC OA Interfaz

En la anterior ilustración se muestra la interfaz de WinCC OA, remarcada de igual forma que con EPICS.

En primer lugar, marcado en [1] naranja, se encuentra la barra de menú estándar mucho más completa que en Phoebus. Resumidamente, cada menú contiene:

- **Module**, donde se puede abrir los módulos *Gedi* (actual), *PARA* (interfaz gráfica de edición de los *Data Points* y *Data Point Type* ), *Vision* (ejecución de proyecto) o gestión del sistema entre otros.
- **Panel**, reúne las herramientas básicas sobre la gestión de paneles (abrir, crear, guardar, imprimir, abrir cualquier panel del proyecto)
- **Edit**, herramientas básicas de copiar, pegar, cortar, deshacer, rehacer y gestión de SCRIPTS.
- **View**, gestión básica de la visualización del panel, además de módulos externos como el visor de *DPS*.
- **Layout**, configuración detallada de la disposición de las ventanas del módulo *Gedi*.
- **Objects**, acceso a la mayoría de los objetos como los primitivos (geometrías), widgets, y especiales.
- **Format**, gestión de los colores, grosor de línea o la alineación de texto.
- **Windows**, para los paneles abiertos.
- **Tools**, herramientas varias a las que no daremos importancia en este proyecto.
- **SysMng**, para la gestión del sistema mas a bajo nivel como drivers o depuración.
- **Help**, accesos directos a la documentación del programa desglosada por módulos.

En segundo lugar, marcado en [2] amarillo, es una recopilación de los accesos directos más importantes y visualizados por iconos del apartado anterior.

A la izquierda, marcado en [3] rojo, se halla la ventana de visualización de los archivos del proyecto.

Debajo de ese panel, marcada en [4] azul, es una ventana auxiliar con accesos directos a la librería de objetos llamada *STD\_Objects*. Los cuales se comentarán en apartados posteriores.

Siguiendo una espiral, a la derecha y como ventana más grande marcada con [5] verde, es un espacio dedicado a las ventanas de los paneles abiertos. Éstas pueden superponerse, tener tamaños y zoom distintos.

Ahora como ventana central marcada con [6] morado, es la ventana dedicada a mostrar las propiedades del objeto seleccionado. Entre esas propiedades se encuentra el nombre, posición en la ventana, tamaño, color de fondo, color principal, fuente... entre otros.

Finalmente, marcada en [7] cian, como ventana más importante de cara a la configuración y conexión con los *DPS* se encuentra la gestión de los *SCRIPTS* del objeto seleccionado. Esos *SCRIPTS* configuran el comportamiento del objeto, de esto se hablará en apartados posteriores.

## 5.10.2 Información central

Una vez introducidos los interfaces de cada programa y sus posibilidades, se va a mostrar cómo se ha realizado el apartado de visualización del panel principal.

### 5.10.2.1 Creación de un panel

Éste es el elemento base que agrupa los elementos que se quieren visualizar, además es el que se almacena como archivo.

#### 5.10.2.1.1 EPICS

En Phoebus, para crear un nuevo panel hay que dirigirse a la pestaña superior **Applications > Display > New Display** tal y como se muestra en la siguiente ilustración:

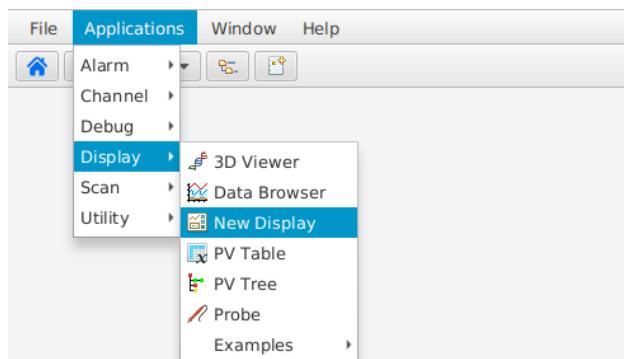


Ilustración 88 - Phoebus Añadir panel

Entonces se abrirá una ventana para seleccionar su ubicación y su nombre, este último se rellenará con el nombre *VisionGeneral*.

#### 5.10.2.1.2 WinCC OA

En el caso de WinCC OA, hay diversas formas para crear un nuevo panel.

- Pulsar el botón derecho del ratón sobre la carpeta *Panels* de la ventana de visualización de archivos de proyecto.

- Pulsar el acceso directo con el icono  , en la primera barra debajo del menú superior.
- Pulsar la pestaña **Panel > New Panel**.
- Con el atajo de teclado **Ctrl + N**

En el caso homologo al usado en EPICS, seria:

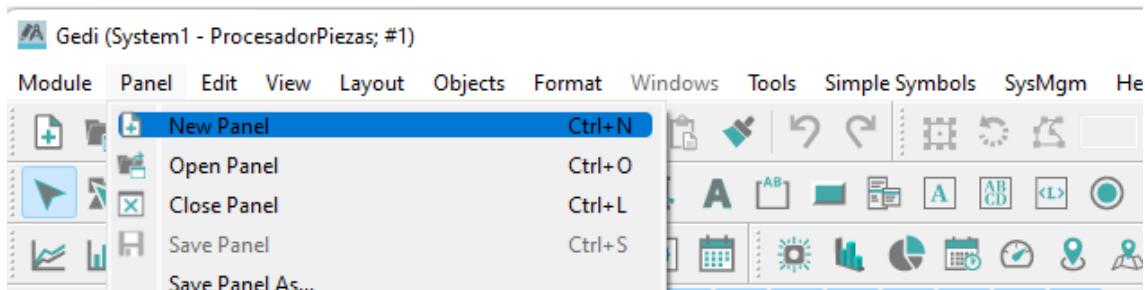


Ilustración 89 -WinCC OA Añadir panel

Esto añade la siguiente ventana vacía:

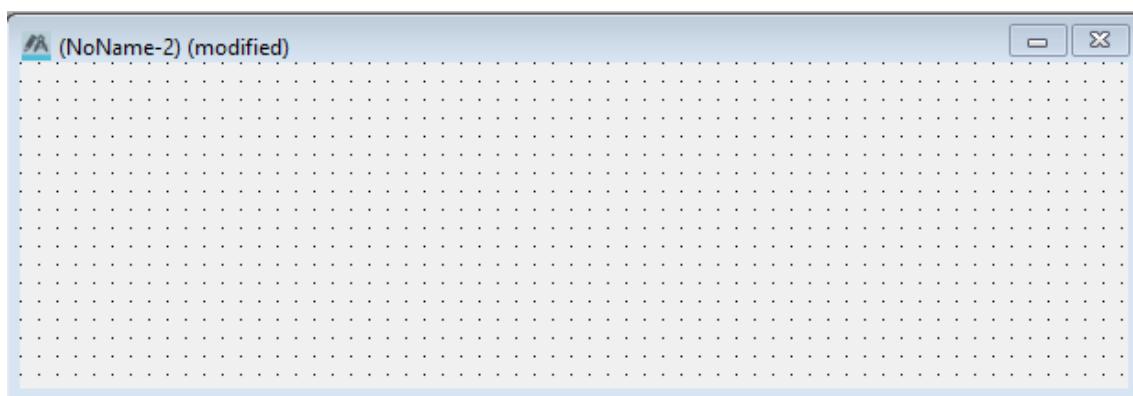


Ilustración 90 -WinCC OA Panel vacío

### 5.10.2.2 Añadir imagen

Siguiendo la estructura comentada anteriormente, en la parte de visión del panel principal se va a colocar una captura del proceso en vista aérea.

#### 5.10.2.2.1 EPICS

En Phoebus existe un elemento llamado *Picture*, con el icono  , para introducir imágenes. Para ello hay dos formas: arrastrar y soltar desde el icono, lo que creara un objeto de este tipo con un tamaño predeterminado o seleccionando el elemento y con el cursor en cualquier parte de la pantalla, mantener pulsado y arrastrar para crear del tamaño que va desde que se pulso hasta que se soltó.

Si se pulsa este objeto, en la ventana de propiedades *Properties* más a la derecha, se mostrará todas sus posibles configuraciones. Como en la siguiente ilustración:

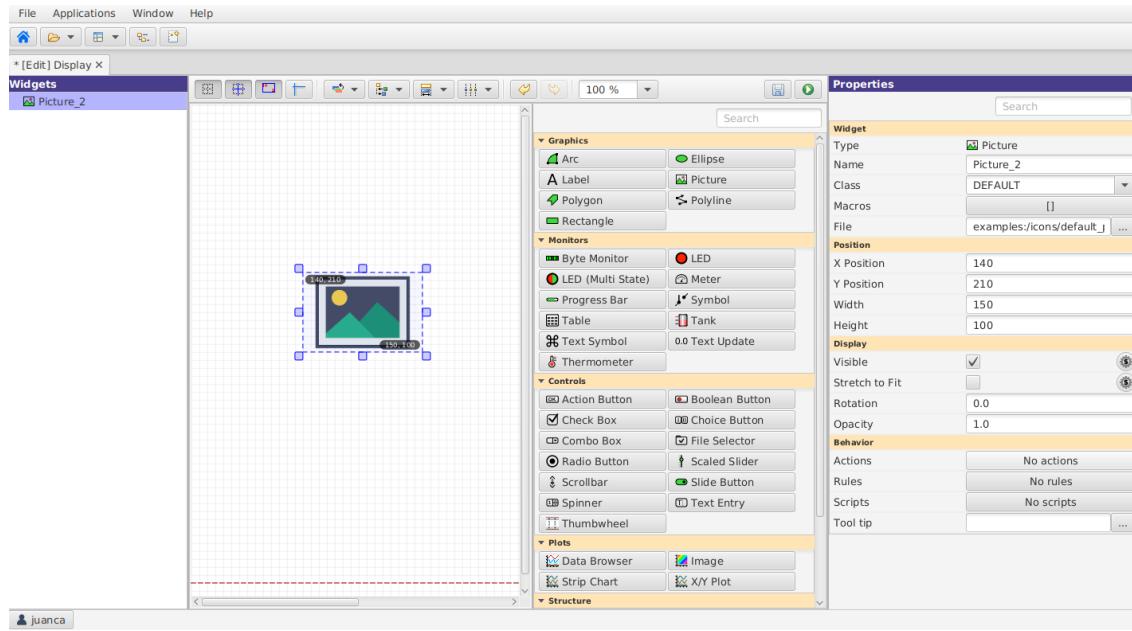


Ilustración 91 - Phoebus Imagen propiedades

Se aprovecha este momento para comentar las distintas propiedades de este elemento, en su mayoría son comunes a otros elementos.

- **Widget**
  - **Type**, corresponde al tipo de objeto (*Picture*)
  - **Name**, es el nombre del objeto
  - **Class**, clase del objeto. En este caso no ocurre, pero podría haber distintos tipos del mismo objeto.
  - **Macros**, son los acortamientos, renombramientos o alias que le puede dar el desarrollador de la aplicación a los *records*. Se introducen manualmente y solo sirven para este objeto.
  - **File**, este apartado es específico de este objeto. En él se selecciona el archivo correspondiente a la imagen que se desea que contenga.
- **Position**
  - **X Position** e **Y Position**, contienen las coordenadas absolutas en pixeles de la esquina superior izquierda. Esto se considera el origen.
  - **Width** y **Height** respectivamente son la anchura y altura relativas al origen en pixeles.
- **Display**
  - **Visible**, que el objeto sea visible o no.
  - **Stretch to Fit**, si se marca estira o comprime la imagen según el tamaño del objeto. Si se deja sin marcar la imagen mantiene su aspecto y su tamaño varía en función del objeto.
  - **Rotation**, rotación del objeto.
  - **Opacity**, opacidad del objeto.
- **Behaviour**
  - **Actions**, se le pueden asignar acciones como ejecuciones de scripts, comandos, abrir ficheros o abrir ventanas. En el caso de los widgets, la acción se ejecuta pulsando el botón derecho en tiempo de ejecución y pulsando sobre la acción.

Como ya se tratará en apartados posteriores, los *Action Buttons* ejecutan directamente la acción al pulsar sobre ellos.

- **Rules**, quizás es la propiedad más útil de cara a la creación del SCADA. Las reglas del objeto sirven para cambiar cualquier otra propiedad de éste en función de uno o varios *records*. Además, se puede hacer cualquier operación con los *records*.
- **Scripts**, permite la ejecución de scripts en función de uno o más *records*.

Comentados las principales características comunes a la mayoría de los objetos, ahora es el momento de modificarlo adecuadamente para que contenga la imagen general del proceso. Para ello se selecciona el archivo en el apartado *File*, la imagen se puede encontrar en **cualquier lugar**. Y no menos importante, el nombre se configura con uno más representativo como “Visión general”. Por lo que quedaría así:

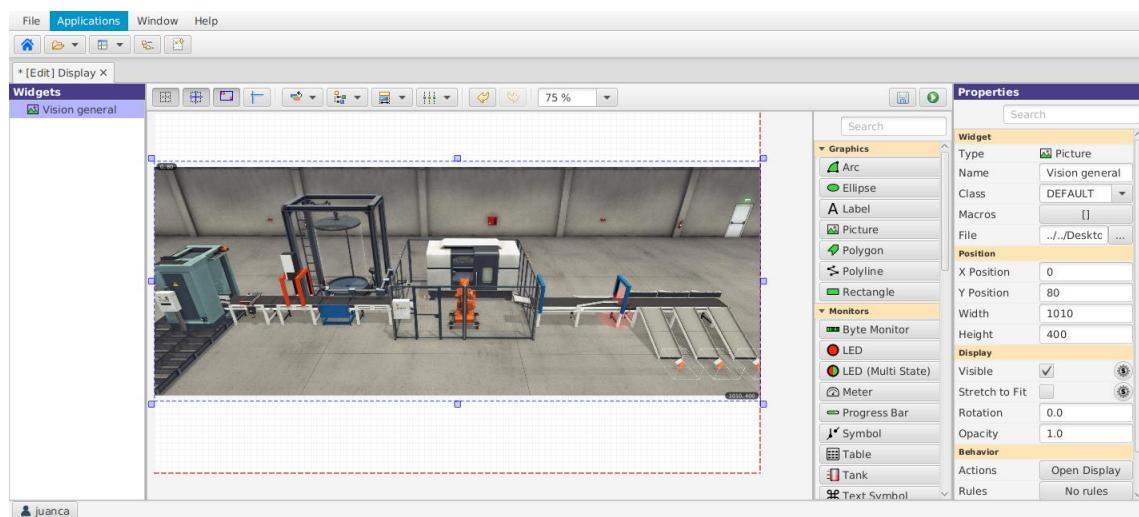


Ilustración 92 - Phoebus Panel principal con la imagen general del proceso

Un detalle en la pestaña de la ventana actual es que aparece un asterisco, el cual simboliza que hay cambios que no se ha guardado.

#### 5.10.2.2.2 WinCC OA

En WinCC OA no existe un elemento propio para albergar una imagen. Pero las amplias configuraciones de casi todos los objetos de este programa lo permiten. Por lo que se va a usar un rectángulo y luego se le cambiará el relleno con la imagen deseada.

Para ello, se selecciona el elemento del rectángulo y se coloca de cualquier manera en la pantalla ya que luego se modificará el tamaño a la imagen.

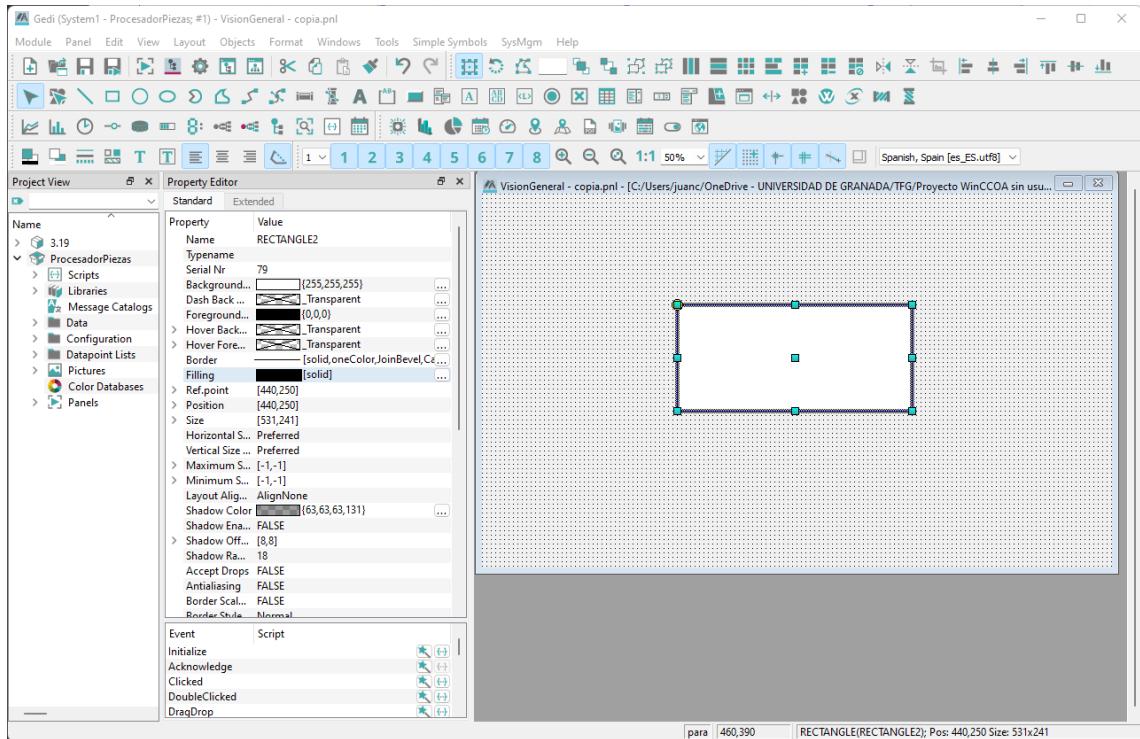


Ilustración 93 - WinCC OA Rectángulo vacío

En la ventana de propiedades se puede vislumbrar la cantidad de ellas que tiene el objeto.

Entre las propiedades principales se encuentran las típicas de posición, tamaño, color o visibilidad. Pero son la minoría, ya que también hay propiedades como estilo de borde, tamaño máximo, sombra, foco, capas, radio de borde, rotación de imagen y de acciones como permitir su selección, arrastrar, soltar...

Profundizando en el grupo de propiedades de color, hay un apartado llamado *Filling*, en él se puede configurar como se rellenará el objeto. Al pulsarlo, sale una nueva ventana donde permite elegir entre: **Outline** (sin relleno), **Solid** (color sólido), **Pattern** (patrón por archivo), **Hatch** (patrón geométrico) o **Gradient** (color gradual). Dicho esto, lo que interesa para crear la imagen que visualiza el proceso entero es el *Pattern*.

Dentro de él se puede seleccionar el archivo de la imagen y el tipo de encuadrado (*Fit* que deforma la imagen al objeto y *Tiled* donde la imagen es estática y es el objeto el que tiene que adaptarse a mano a la imagen). Escogiendo este último y cambiando el tamaño del objeto, queda:



Ilustración 94 - WinCC OA Panel principal con la imagen general del proceso

### 5.10.2.3 Indicadores de cinta

Con el objeto polígono se creará una flecha para indicar el estado de cada cinta en ambos programas. Esta flecha tendrá un rojo oscuro cuando esté la cinta apagada y un verde **parpadeante** cuando este encendida.

#### 5.10.2.3.1 EPICS

Para comenzar el proceso de creación de un polígono primero hay que seleccionar su ícono, entonces pulsar doblemente sobre cualquier espacio y empezar a crear el resto de los puntos pulsando donde quiera. Para terminar, hay que pulsar escape.

Durante el proceso, el relleno del polígono no acompaña al ratón, **no es fluido**, solo se puede visualizar el resultado temporal cada vez que se añade un punto nuevo.

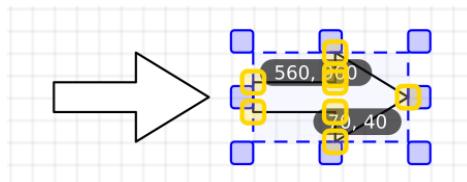


Ilustración 95 - Phoebus polígono Flecha

Para objetos pequeños como éste, al seleccionarlo la interfaz no ajusta los puntos de edición y no permite ver cómodamente el objeto.

Para configurar el comportamiento del polígono y que reaccione a los cambios de un *record* hay que configurarle una regla. Para ello se pulsa el botón *Rules* que está al final de la ventana de propiedades. Entonces se abrirá una nueva ventana donde se pueden gestionar todas las reglas del objeto.

Para crear una regla hay que pulsar el botón *Add* y asignarle un nombre y una propiedad a modificar, en este caso *background\_color*. Seguidamente en la siguiente columna se le coloca el nombre del *record* precedido por el protocolo de comunicación, en este caso *Channel Access*, de esta forma: **ca://[NombreRecord]**. Para configurar el comportamiento según esta variable, Phoebus lo renombra por numeración entonces para acceder a su valor se usa como una variable llamada **pv0**. En la última columna se configura este comportamiento, se escribe como texto plano, sin ninguna ayuda y con expresiones booleanas. A la derecha de la expresión booleana se selecciona el color deseado, pero sólo hay colores estáticos.

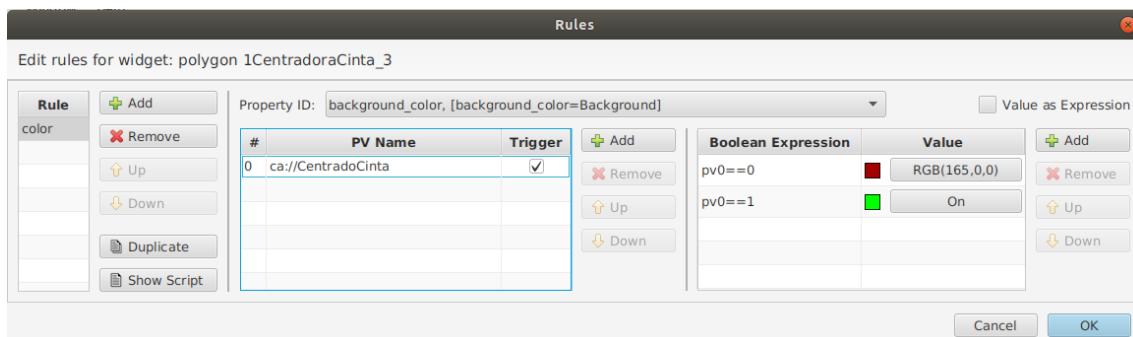


Ilustración 96 - Phoebus Regla cambiar color

Pero existe la posibilidad de generar colores dinámicos manualmente. Phoebus tiene integrado distintos valores simulados y se pueden añadir como una variable más. Para acceder a estas variables

se accede poniendo **sim://** y saldrán las disponibles. Se escogerá **sim://flipflop(1)** siendo **1** el periodo del valor. Entonces, para crear un color dinámico solo cuando la cinta esté encendida quedaría:

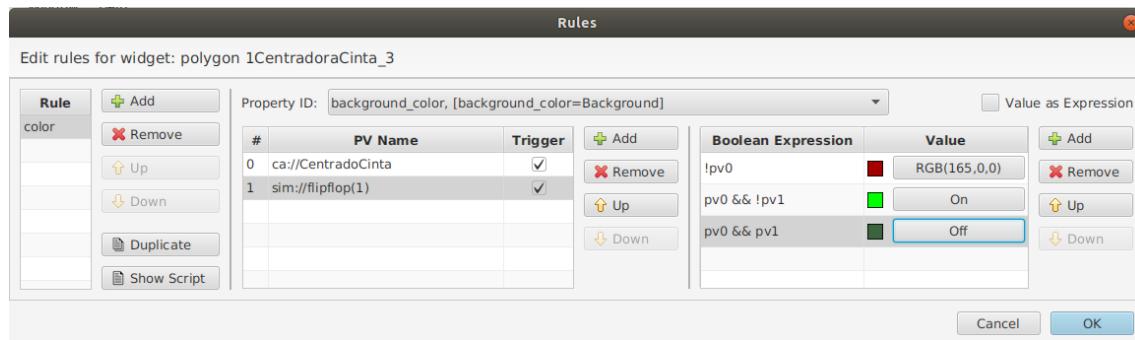
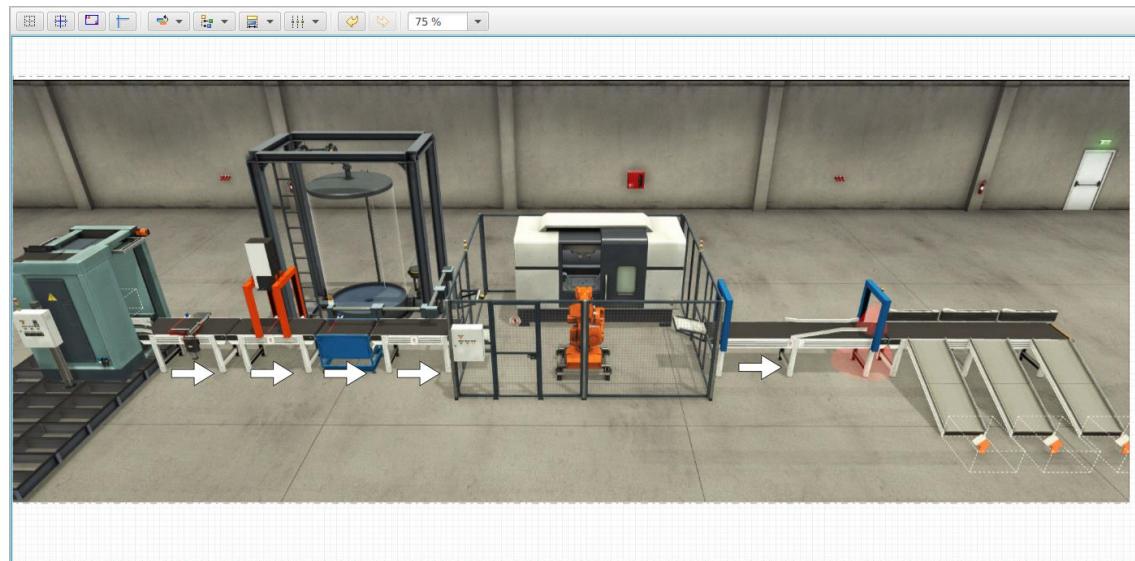


Ilustración 97 - Phoebus Regla Change color y color dinámico

Hecho esto con un objeto, se puede multiplicar por cada estación que se necesite modificando solamente el *record* de la ilustración anterior.

Quedando así:



### 5.10.2.3.2 WinCC OA

Al usar la herramienta de polígono basta con pulsar en cualquier lugar donde poner el primer punto, entonces desde ese punto hasta el ratón se trazará una línea hasta que se vuelva pulsar nuevamente. La zona interior del polígono se ira pintando con forme se mueva el ratón. Para terminar, hay que pulsar doblemente.

Cabe destacar que este proceso es **fluido**.

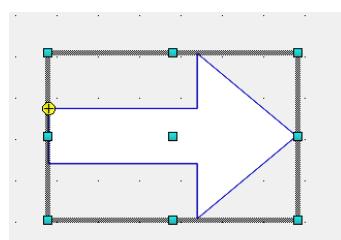


Ilustración 98 - WinCC OA Polígono flecha

Además, los polígonos se pueden editar para añadirle o quitarle puntos, o modificarle los existentes, pulsando el icono  . Esta edición se realiza por defecto sobre una cuadricula virtual, por lo que los puntos originales pueden no volver a ponerse donde estaban. Entonces si se quisiera colocar con precisión, se mantiene pulsado la tecla *Alt* para no usar la cuadrícula.

Para hacer que el polígono reaccione a los estados de cualquier *Data Point*, se puede realizar de dos formas: Realizando manualmente un script o con la ayuda del *wizard* que proporciona la aplicación.

Para hacerlo con el asistente, hay que dirigirse a la ventana de *Event Script*, en la fila de *Initialize* y pulsar sobre el icono de varita mágica para invocar el asistente:

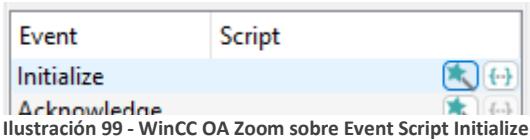


Ilustración 99 - WinCC OA Zoom sobre Event Script Initialize

En la ventana del asistente, se pueden configurar acciones sobre el polígono como su visibilidad, posición, color, rotación... entre otros. En este caso solo se va a cambiar el color.

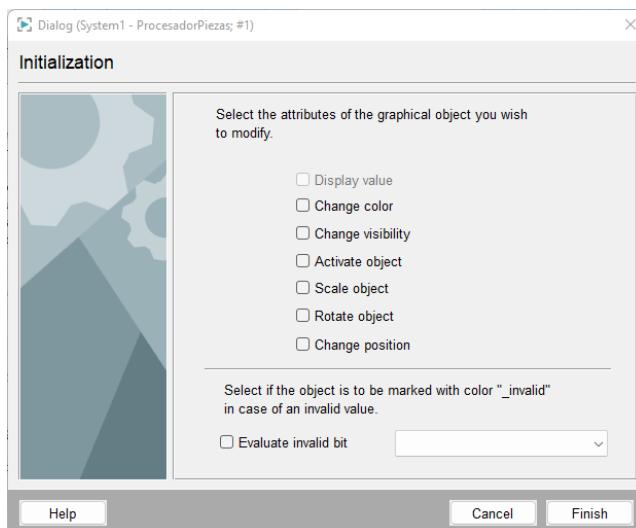


Ilustración 100 - WinCC OA Wizard Initialization

Al pulsar *Change color* cambiará a la siguiente ventana, donde pedirá elegir si el color frontal (borde y/o texto), el de fondo (relleno) o ambos. Nos interesa solo el de relleno, por lo que se marca solo esa opción.

En la parte inferior hay otro recuadro, donde se puede elegir como reaccionará el objeto al *Data Point*, ahora mismo solo interesa la primera *Value-dependent* para visualizar su valor:

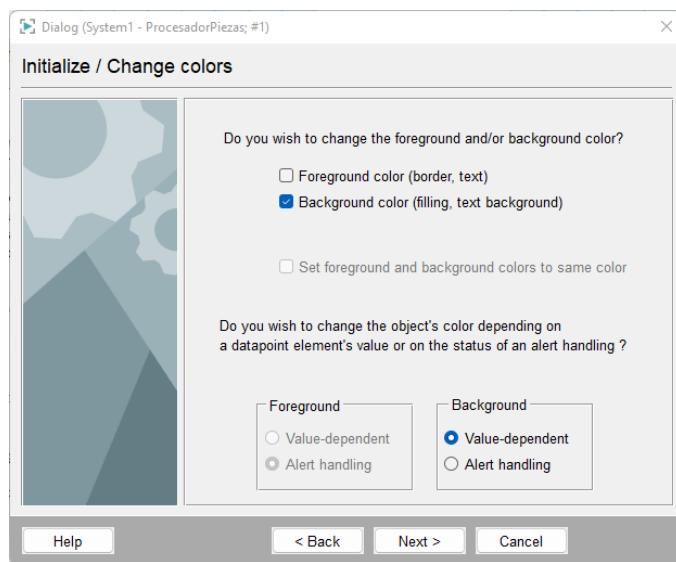


Ilustración 101 - WinCC OA Initialize wizard change color

Al darle a siguiente, se podrá elegir el *Data Point Selector* pulsando el icono *Data Point Selector* situado a la izquierda del recuadro de texto blanco:

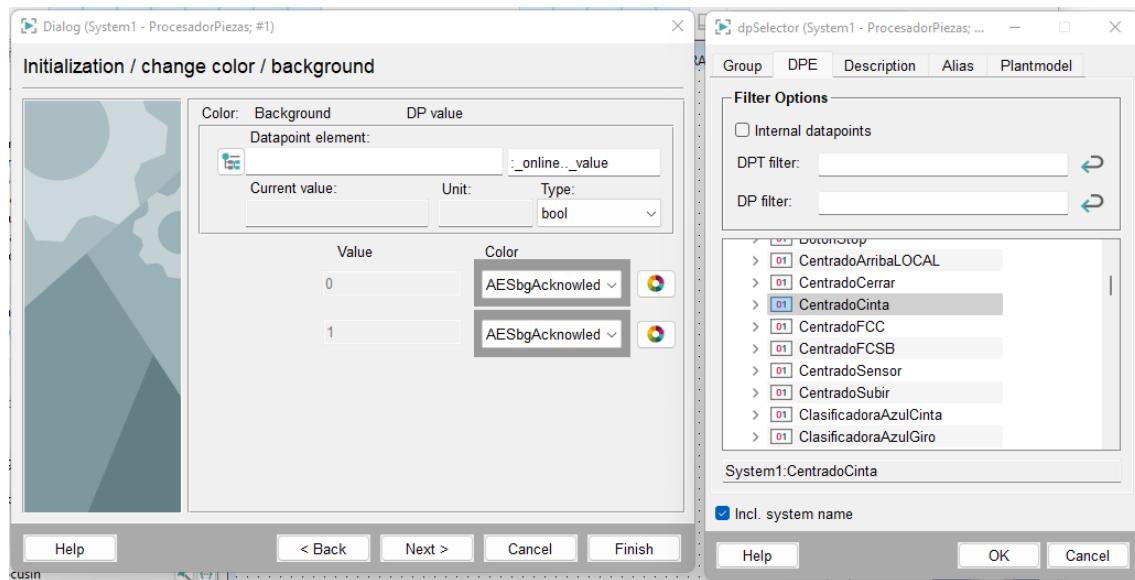


Ilustración 102 - WinCC OA Initialization wizard change color & Data Point Selector

Una vez escogido el *DP*, al ser un tipo booleano, permite escoger dos colores para los dos estados posibles. Al pulsar sobre el ícono del selector de colores aparecerá otro asistente con una amplia gama de colores, tanto **estáticos** como **dinámicos**.

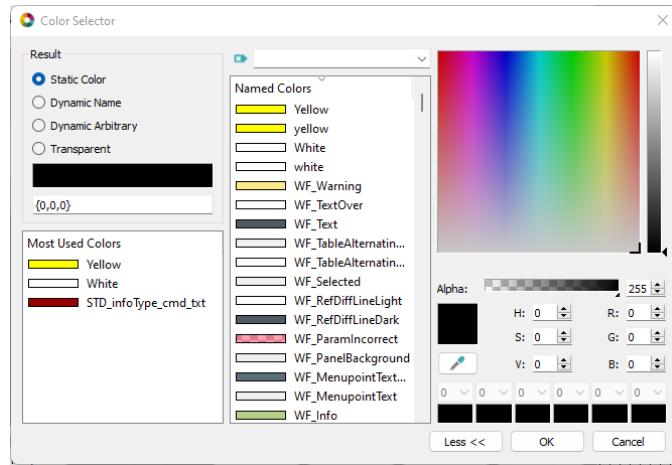


Ilustración 103 - WinCC OA Color Selector

Para cuando la cinta este apagada se escogerá el color rojo oscuro estático y para cuando este encendida se cogerá un verde parpadeante.

Al darle a finalizar, se volverá a la ventana del editor GEDI con el objeto de que se estaba editando.

Entonces, teniendo la primera flecha, se puede duplicar en cada lugar que se necesite modificando solo el DP con el que reacciona. Quedando así la pantalla principal:



Ilustración 104 - Panel principal con indicadores de cinta

#### 5.10.2.4 Indicadores de Pieza

Otro uso de los polígonos va a ser representar las piezas sobre la cinta cada vez que un sensor las detecta, además, en el proceso de clasificado cambiará de color en función de la lectura de la cámara.

Otra configuración manejada con frecuencia ha sido el cambio de visibilidad de objetos en función de una o varias variables. Un ejemplo claro de este hecho es son las piezas colocadas en cada estación para indicar por donde va la pieza.

### 5.10.2.4.1 EPICS

De igual forma que el polígono anterior, se crea otro con la apariencia de una pieza tal que así:

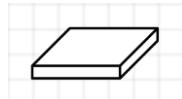


Ilustración 105 - Phoebus Polígono pieza

Lo común a todas las piezas de todos los procesos es que se va a modificar su visibilidad, por lo que se va a crear una regla de la misma forma que antes y con los records acordes a cada proceso. La regla queda de esta forma:

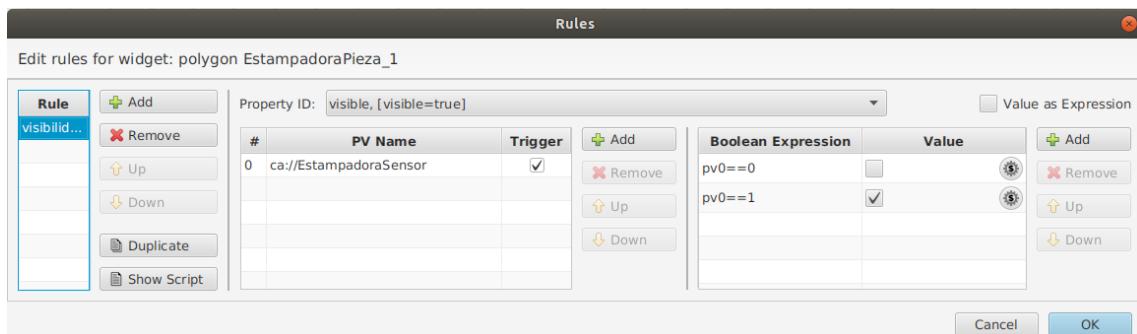


Ilustración 106 - Phoebus Reglas Visibilidad

En la última estación, la clasificadora, la pieza debe reaccionar con un *record* de tipo numérico. Tanto para su visibilidad como para el color. El tratamiento es ligeramente distinto en la expresión booleana: igualar su valor al número deseado de esta forma. Entonces las reglas de visibilidad y color para esta pieza quedan:

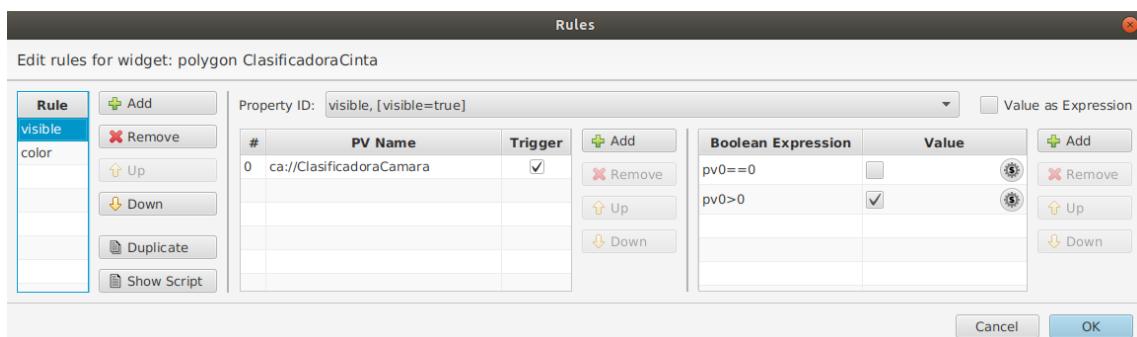


Ilustración 107 - Phoebus Reglas Visibilidad Record numérico

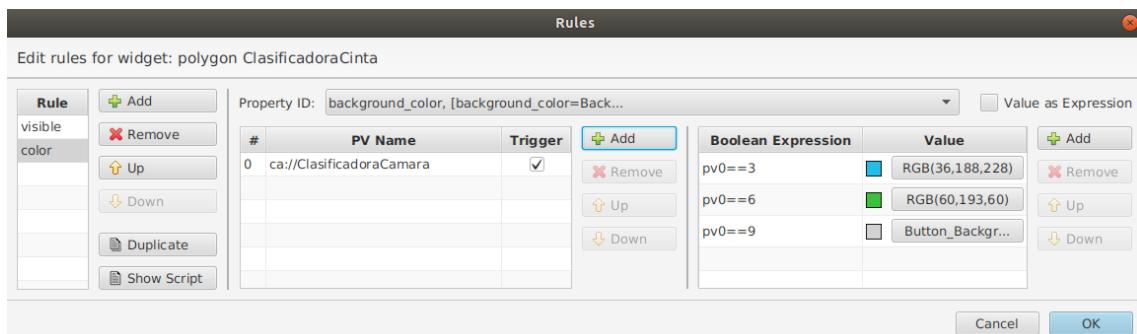


Ilustración 108 - Phoebus Regla Color Record numérico

Estas reglas se ejecutan paralelamente ya que reaccionan al mismo *record*.

#### 5.10.2.4.2 WinCC OA

La figura queda de siguiente forma:

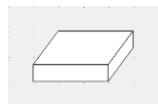


Ilustración 109 - WinCC OA Polígono Pieza

En este caso, solo se modificará su visibilidad. Para ello se pulsa tambien sobre el icono del asistente en *Initialization* y se selecciona *Change visibility*. Entonces aparecerá la misma ventana de selección del *DP* pero con la configuración adaptada a visibilidad:

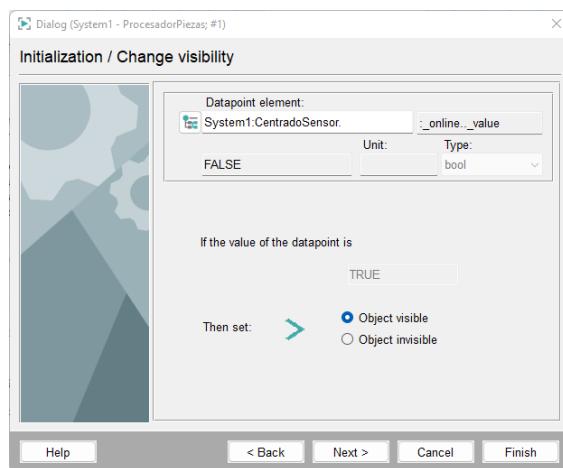
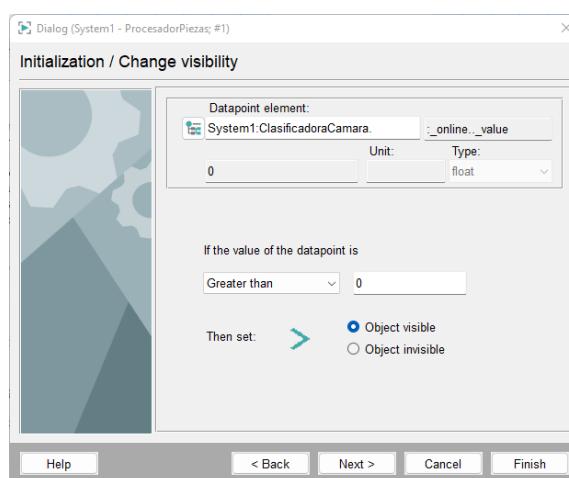


Ilustración 110 - WinCC OA Initialization Change visibility

Al darle a finalizar, ya se puede multiplicar para colocar una en cada estación cambiando el *DP* acordemente.

En la última estación, Clasificadora, debe reaccionar diferente ya que el *DP* es numérico y será entonces en función del código que lea la cámara. La configuración cambia sutilmente, en el caso de la visibilidad muestra un desplegable con tipos de comparaciones y a su derecha el valor pivote:



Para la otra regla, se le da a *next* se selecciona *Change color*. Al ser un valor numérico, la configuración cambia y se pueden tener varios estados. Por cada comparación se puede elegir un tipo de comparación diferente, además de su valor. Cabe destacar que en el asistente tiene un límite de comparaciones, pero eso se puede solventar accediendo al script que genera éste.

En este caso interesa que quede así:

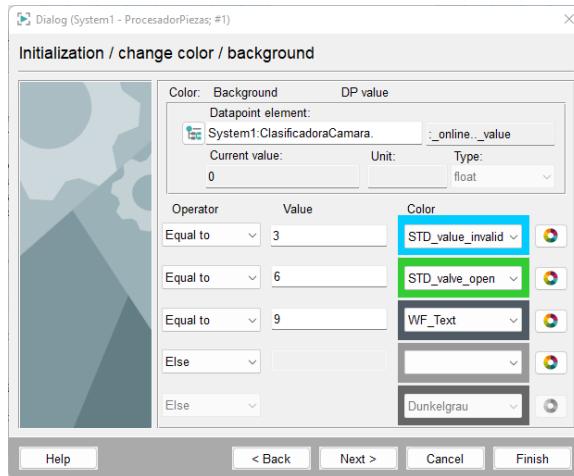


Ilustración 111 - WinCC OA Initialization Change color pieza clasificadora

#### 5.10.2.5 Indicadores de desviadores

En la estación de clasificado se encuentran tres cintas verticales que se giran para sacar la pieza correspondiente. Para representarlos se usa también el elemento de polígono en ambos programas. Como no hay ninguna configuración adicional, solo se muestra los resultados:

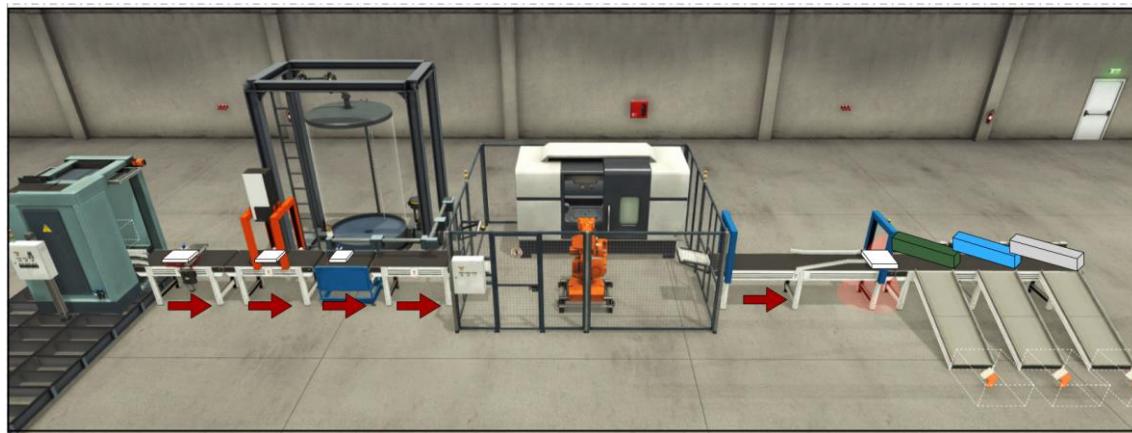


Ilustración 112 - Phoebus visión general con indicadores de cinta, pieza y clasificador



Ilustración 113 - WinCC OA visión general con indicadores de cinta, pieza y clasificador

### 5.10.2.6 Accesos por zona

A lo que se refiere los accesos por zona es poder acceder al panel de una estación cuando se pulse encima de ella en la imagen de la visión general. Realmente lo que habrá es un objeto transparente que ejecute esta acción.

#### 5.10.2.6.1 Phoebus

Para ello, Phoebus tiene un tipo de botón específico, el *Action Button*, el cual puede ejecutar scripts, algún comando o escribir en algún *record*. Para acceder a ello, hay que pulsar el objeto y pulsar en el apartado *Actions* que aparece en su barra de propiedades:

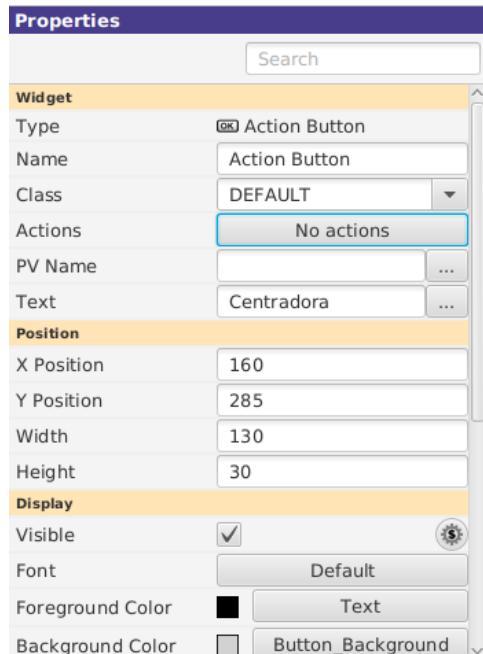


Ilustración 114 - Phoebus Action Button

Al pulsar sobre el botón *No actions*, se abrirá una ventana sin ninguna acción configurada hasta ahora. En el desplegable de *Add* se muestran las posibles acciones, en este caso, se requiere la de *Open Display*:

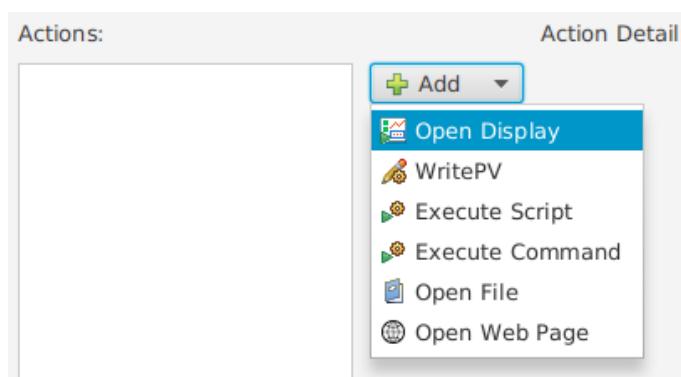


Ilustración 115 - Phoebus Action button Añadir acción

Entonces aparecerán las configuraciones para esa acción, donde se incluye la de seleccionar un archivo de panel a abrir. Entonces quedaría así:

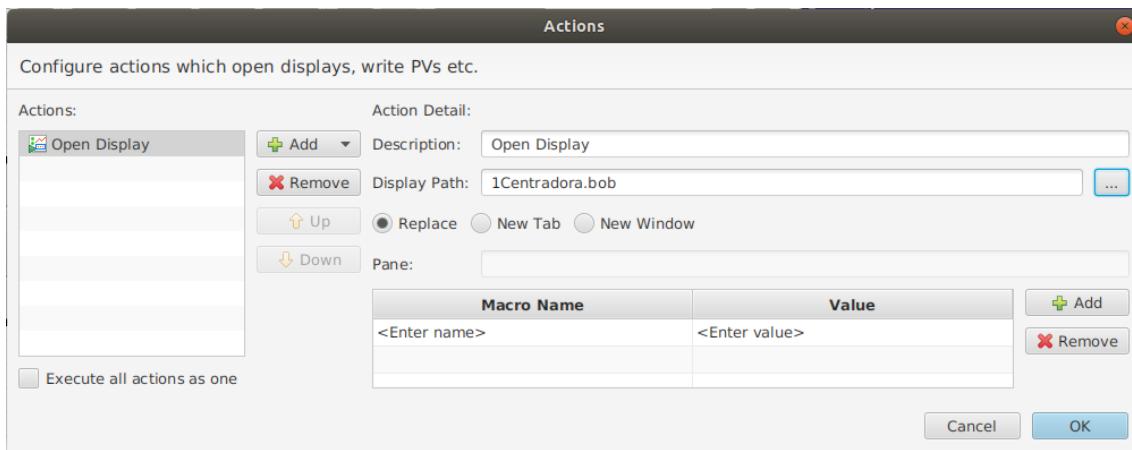


Ilustración 116 - Phoebus Action Button Open Display

Falta configurar su trasparencia para que no estorbe a la vista, pero Phoebus no tiene ninguna propiedad especial que reconozca que el ratón esta encima de ese objeto para cambiar su trasparencia y que sea más visible. Por lo que a la hora de ejecución se podrá pulsar, pero sin animación alguna. Para poder visualizar la zona de “acción” para cada estación, en la siguiente captura se han seleccionado los objetos:

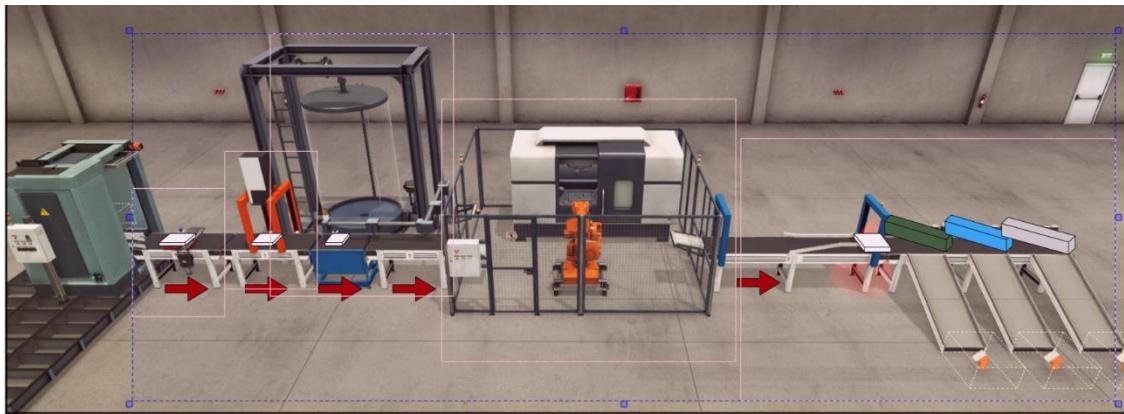


Ilustración 117 - Phoebus Accesos por zona

#### 5.10.2.6.2 WinCC OA

En WinCC OA se tiene la ventaja de cualquier objeto puede ser usado como pulsador, entonces para recrear estos accesos directos se va a usar un rectángulo simple, al que se le pone como color de fondo transparente y sin bordes.

En el evento *Clicked*, se le coloca la función vista anteriormente para abrir el panel que le corresponda. Una forma fácil y rápida de abrir la configuración de este script es pulsar dos veces sobre el objeto.

Lo nuevo en este caso es la propiedad *Hover Background Color*, la cual permite establecer un color de fondo diferente cuando se pasa el ratón por encima. Al pulsar sobre la propiedad se abrirá la ventana de selección de color, donde se escogerá el color cian y se bajará el *Alpha* aproximadamente al 10% para poder visualizar la estación de fondo. En tiempo de ejecución, colocando el ratón encima de la estación Estampadora, queda así:



Ilustración 118 - WinCC OA Accesos por zona

Basta con replicarlos por cada estación y modificarle el panel al que redirigen. Además, estos elementos se han usado en cada panel para avanzar o retroceder en el orden de las estaciones de forma intuitiva.

#### 5.10.2.7 Tanque

Como la gestión de líquidos en la industria es muy común, se colocó un tanque deliberadamente en el proceso industrial para comprobar las opciones que aporta cada programa. Aunque ambos proporcionan la posibilidad de cambiar la escala de los objetos en función de una variable numérica y podría crearse un tanque personalizado, también proporcionan un objeto predefinido para un tanque.

##### 5.10.2.7.1 EPICS

El elemento *tank*, es una barra en dos dimensiones, que se puede configurar para que muestre una escala, se pueda poner en vertical u horizontal o sus colores. Tiene esta forma:

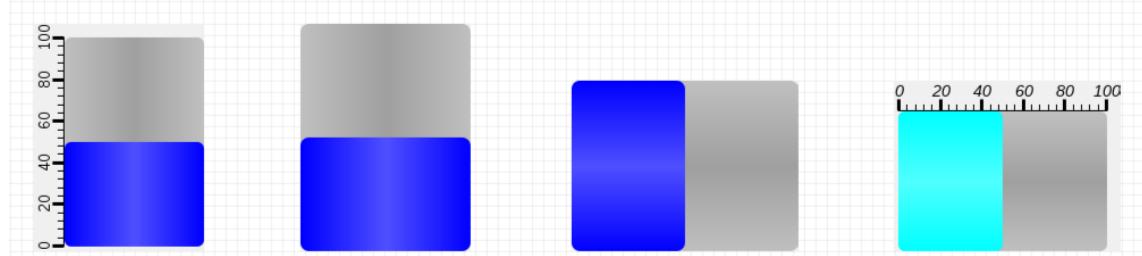


Ilustración 119 - Phoebus tank

La intención es colocarlo encima de la imagen del tanque del proceso, por lo que se escogerá la segunda opción de la ilustración anterior (vertical y sin escala).

Como es un objeto predefinido, en el apartado de propiedades tiene un apartado para indicarle el *record* correspondiente. Para terminar su configuración se ajusta la escala que debe representar, es decir, indicarle los valores mínimos y máximos que representa el *record*.

Pues colocado en su lugar final, se muestra:

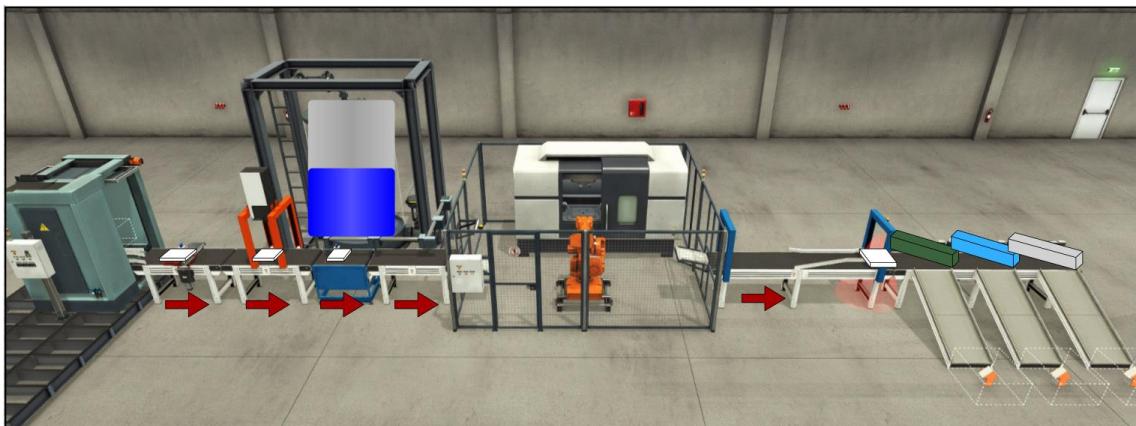


Ilustración 120 - Phoebus Vision general con tanque

#### 5.10.2.7.2 WinCC OA

En WinCC OA, el objeto que puede representar un tanque se llama *Pillar* y se encuentra en la librería de objetos estándar (*STD\_Objects*). Hay diferentes tipos, circulares o cúbicos, con transparencia o sin transparencia. Los dos primeros son:

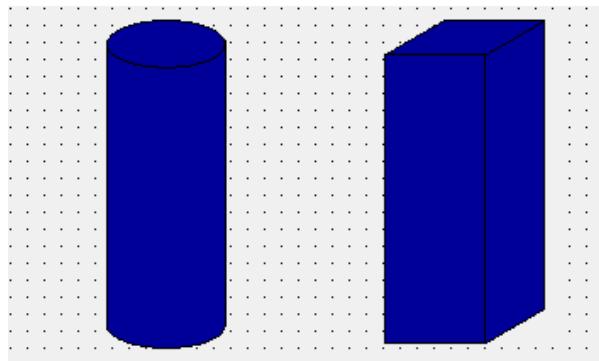


Ilustración 121 - WinCC OA *Pillar* tanque

Para asignarle un *DP* hay que pulsarlo dos veces para que salga su ventana de configuración. En ella se encuentra el *DP Selector* para buscar con facilidad entre los disponibles, ajustes como indicar la invalidez del valor por color, mostrar la alarma del *DP* si la tiene y rango propio si lo tiene o manual. Además del color, en este caso se ha optado por un cian con cierta transparencia para que se aprecie ligeramente el tanque real tras el.

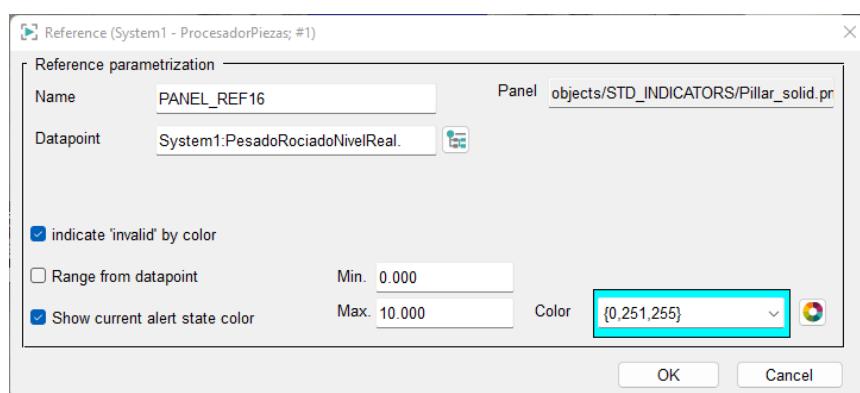


Ilustración 122 - WinCC OA *Pillar* configuración

Una vez configurado, su apariencia en el editor queda así:



Ilustración 123 - WinCC OA Vision general con tanque

### 5.10.2.8 Tuberías

Como el tanque debe llenarse y descargarse, se pueden introducir unas tuberías para simular este hecho e indicar cuando ocurre.

#### 5.10.2.8.1 EPICS

Phoebus no tiene un elemento propio para una tubería, pero se puede recrear con una polilínea haciendo que el grosor de ésta parezca una tubería, aunque las esquinas no se puedan redondear.

Una vez creadas las tuberías, para configurarlas es añadir una regla de color como se ha hecho anteriormente. En este caso las variables de llenado y descarga son numéricas, pero si se quiere que cuando este activo se cambie de color y parpadee hay que añadirle la variable simulada que ya se ha usado. A la hora de la expresión booleana, se pueden mezclar numéricas y booleanas sin problema quedando así:

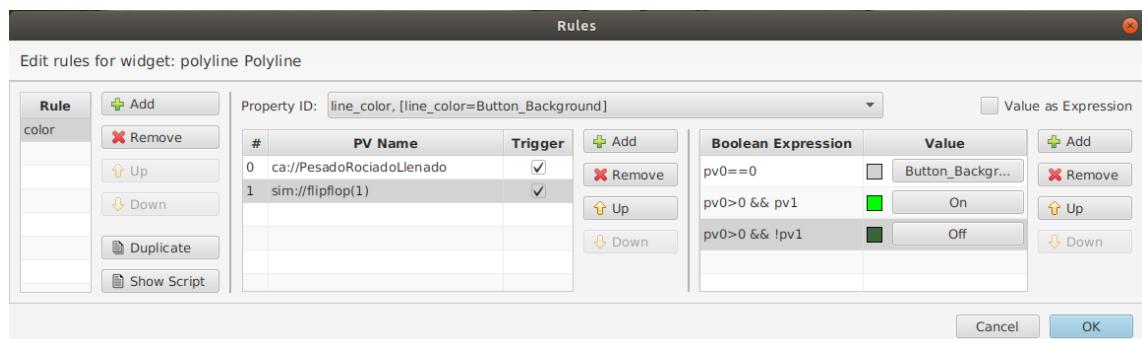


Ilustración 124 - WinCC OA Regla tubería llenado

Entonces, configuradas ambas y colocadas en sus respectivos sitios, el panel se ve de esta forma:

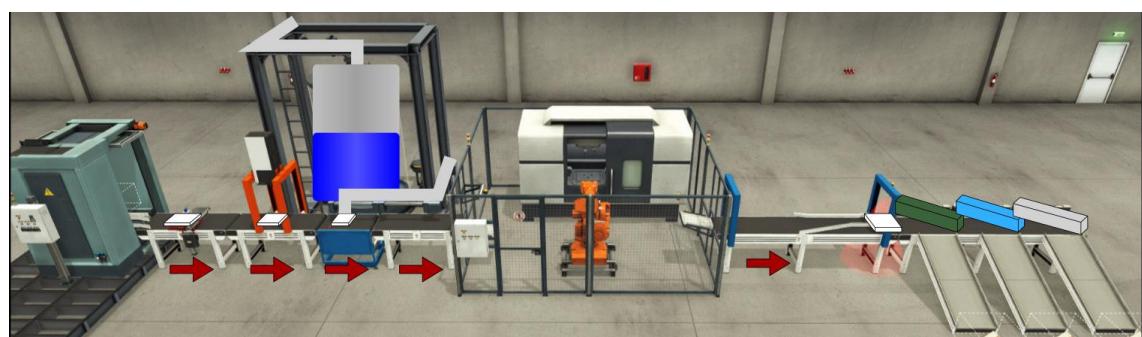


Ilustración 125 - Phoebus Visión general con tuberías

### 5.10.2.8.2 WinCC OA

En este caso si existe un objeto propio para crear tuberías, se representa con el icono . Este objeto ya tiene una apariencia predefinida para aparentar una tubería, redondea las esquinas y se crea como si fuera una polilínea. Una vez creadas y colocadas en su lugar correspondiente, se configuran mediante el asistente, o el script, del evento *Initialize*. Entonces, queda de la siguiente forma:

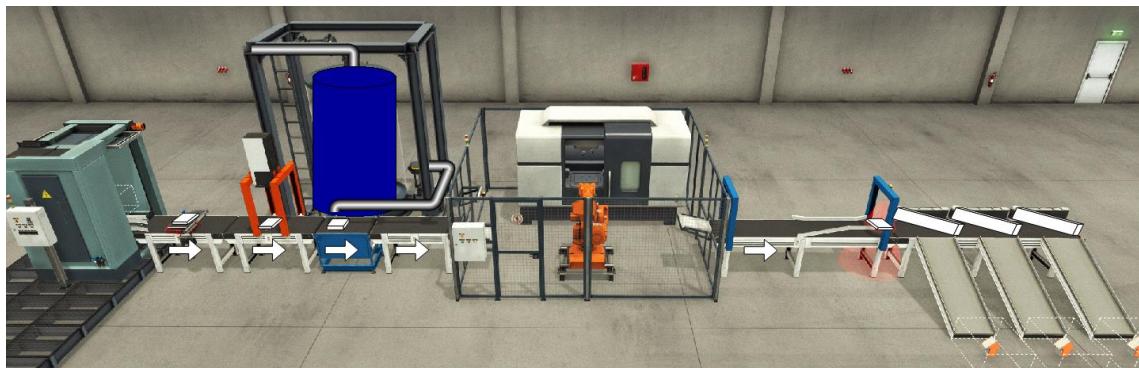


Ilustración 126 - WinCC OA Visión general con tuberías

### 5.10.2.9 Indicadores booleanos

#### 5.10.2.9.1 EPICS

En Phoebus se encuentran tres tipos de indicadores que representen bits, estos son **Byte Monitor** que no es mas que una lista de bits individuales, **Led simple** y **Led Multi State** para representar varios estados, es decir, valores numéricos.

Los tres tienen la misma apariencia y lo único que se puede personalizar son los colores.

Como se pretende representar el indicador de descarga que tiene el proceso del tanque y cuando *Machining Center* está ocupado, conviene usar el led simple.

Su configuración es bastante simple, hay que poner el *record* correspondiente y configurar los colores. Como el led de descarga es naranja, se modifica acordemente.

Por lo que se visualiza así:

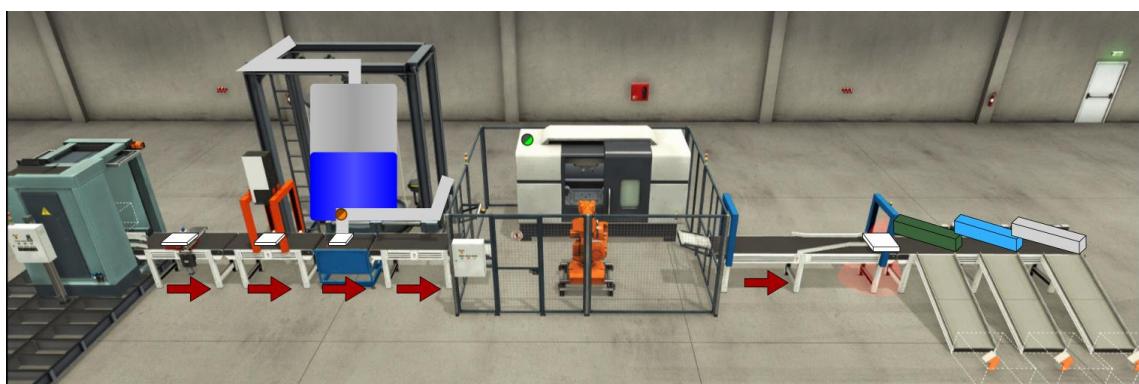


Ilustración 127 - Phoebus Visión general leds

#### 5.10.2.9.2 WinCC OA

De igual manera que en el anterior apartado, los objetos que representan valores booleanos como leds son del tipo *STD\_objects*. Hay gran variedad de apariencias de estos objetos:

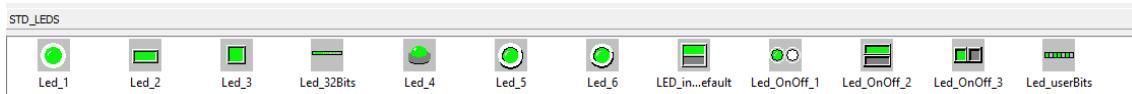


Ilustración 128 - WinCC OA STD Leds

Para darle un poco de perspectiva, se usará el *Led\_4*. Su configuración es muy parecida a la del tanque, pero teniendo en cuenta que es un valor booleano. Con ello, queda así:



Ilustración 129 - WinCC OA Visión General con indicadores led

### 5.10.2.10 Monitor numérico

Como en ambos programas no se ha colocado escala al tanque, se va a introducir un monitor numérico que muestre el nivel exacto de éste.

#### 5.10.2.10.1 EPICS

El elemento que permite visualizar de forma numérica una variable es ***Text Update***. Su configuración es bastante básica, aparte de poder incluir una variable para mostrar, se puede modificar su alineación, fuente, formato o la precisión. Quedaría de la siguiente forma dentro del tanque:

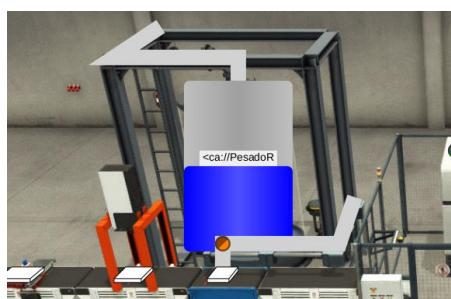


Ilustración 130 - Phoebeus Monitor numérico

#### 5.10.2.10.2 WinCC OA

Aunque en WinCC OA cualquier objeto se puede modificar fácilmente para realizar esta tarea, se usará los *STD\_Objects*. Para este propósito existen ciertas variantes para mostrar texto y son:

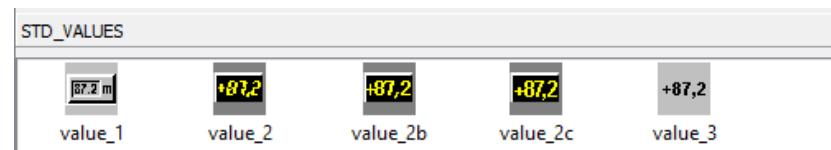


Ilustración 131 - WinCC OA STD Objects Values

El más parecido al de Phoebeus es *value\_3*.

Su configuración es realmente sencilla y sigue la misma filosofía que la mayoría de estos elementos. Al pulsarlo dos veces permite seleccionar el *DP* a mostrar. Aunque a diferencia de Phoebus este elemento no se puede configurar la precisión o formato del número, sino que el propio elemento se adapta al tipo de dato que sea el *DP*.

Quedaría entonces:



Ilustración 132 - WinCC OA Monitor numérico

### 5.10.3 Barra Superior – Accesos directos

La barra superior, al encontrarse sin modificaciones en todos los paneles, se va a crear en un nuevo panel para luego incluirlo en el resto.

#### 5.10.3.1.1 EPICS

Una vez creado el nuevo panel que englobe estos accesos directos y ciertos iconos con los métodos explicados anteriormente, hay que configurar los botones para que actúen como atajos.

Se usará el mismo *Action Button* que en el apartado [5.10.2.6.1](#), con la salvedad que en las propiedades no se configurará como transparente.

Añadiendo ciertos detalles de apariencia y los iconos, la barra superior queda así:



Ilustración 133 - Phoebus Barra Superior

#### 5.10.3.1.2 WinCC OA

Se usará el elemento *Push Button* ya que tiene incorporadas animaciones como cambiar de color cuando se pasa por encima el ratón o cambiar su apariencia al pulsarse.

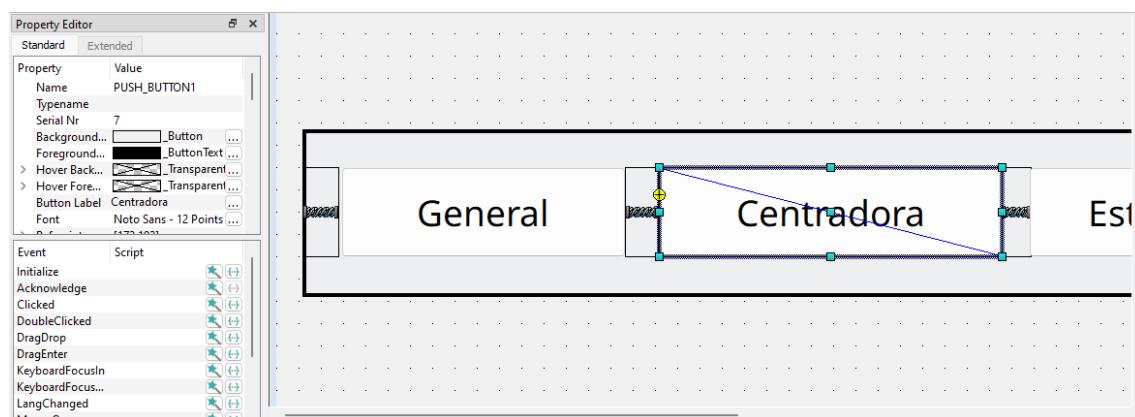


Ilustración 134 - WinCC OA Push Button

Para configurar el botón se invocará al asistente, pero esta vez con la acción *Clicked*. Saldrá la ventana para poder seleccionar entre tres opciones: leer o escribir de *DPs*, funciones con paneles o cambiar de capa.

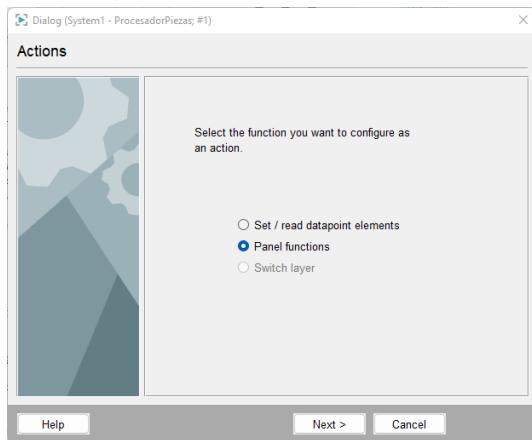


Ilustración 135 - WinCC OA Clicked

Al seleccionar *Panel functions*, se pedirá que función ejecutar: abrir, cerrar o cerrar módulo. En este caso, abrir.

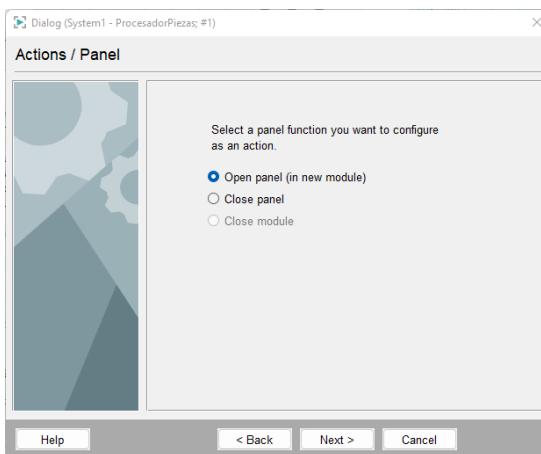


Ilustración 136 - WinCC OA Clicked Panel Functions

Al continuar, se seleccionará el archivo de panel a abrir.

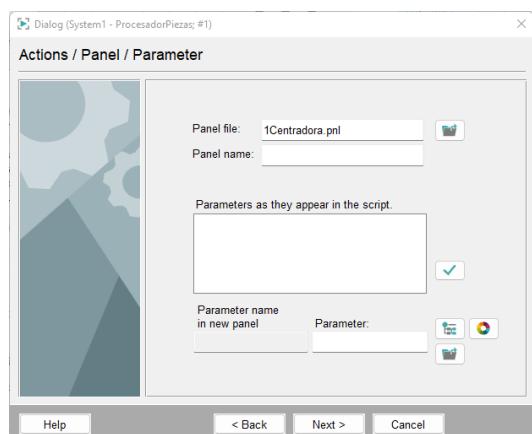


Ilustración 137 - WinCC OA Clicked Panel Functions Parameter

Para finalizar, otra ventana aparecerá para establecer el cómo abrir el panel. Si como una ventana, como ventana raíz o en un nuevo módulo. Como en este caso se quiere que se reemplace, se seleccionala segunda opción llamada *Root Panel in own module*.

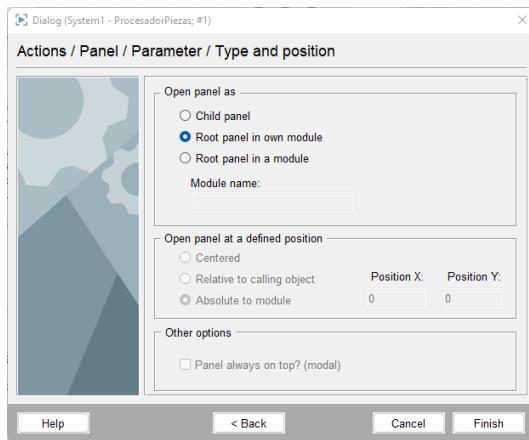


Ilustración 138 - WinCC OA Clicked Panel Functions Type and Position

Así ya se tiene el botón configurado con el asistente para abrir un panel. Bastaría multiplicarlo cuantas veces se quiera y modificar el panel que se abre. Este proceso puede ser más rápido si se accede directamente al script del evento *Clicked*, ya que se ve de esta manera:

```
// SimpleCtrlScriptStart {valid}
main()
{
    EP_childPanelOn();
}

void EP_childPanelOn()
{
    RootPanelOn("1Centrador.pnl","",makeDynString(""));
}
```

El asistente lo que hace es generar un script con las funciones adecuadas a la configuración aportada en las ventanas anteriores. Analizando este pequeño script se puede ver que la función que ejecuta el abrir una ventana y sus parámetros es *RootPanelOn([NombreArchivo],[NombrePanel],[Parametros])*;

Por lo que el script se podría reducir a:

```
main()
{
    RootPanelOn("1Centrador.pnl","",makeDynString(""));
}
```

Entonces el realmente simple solo ir modificando el nombre del panel, más fácil todavía porque la interfaz del script tiene una herramienta para buscar paneles en el directorio del proyecto e introduce la cadena de caracteres automáticamente.

Añadiendo algunos detalles de apariencia e iconos, la barra superior queda así:



Ilustración 139 - WinCC OA Barra Superior

#### 5.10.4 Barra Inferior

La barra inferior se va a dividir en dos como se mostró en la estructura de la información, la parte izquierda para la supervisión y control sobre el proceso general y la parte derecha para supervisión y control de la estación correspondiente.

La parte izquierda va a contener un indicador de los estados (Emergencia, automático o manual) y un control sobre esos estados (seta de emergencia, *start*, *stop* y manual) además de algunos widgets que variarán en función de la aplicación.

La parte derecha, se reservará un espacio para añadir la supervisión y control dependiendo de la estación en la que se encuentre. Es decir, ese espacio estará vacío, siendo en cada panel donde se añadan los elementos necesarios para cumplir este propósito. Esta parte en la pantalla principal, como no requiere control adicional, se añadirá más objetos de información como una ventana que muestre un conteo de piezas procesadas y un histórico de los estados del proceso.

La parte de supervisión de los estados del proceso se conforma de tres rectángulos con texto en su interior, de colores oscuros (verde, rojo, ámbar) siendo éstos los que cambian a uno del mismo tono pero más brillante en función de las variables correspondientes a los estados.

Por lo que, como objetos aun no comentados son los botones que actúan sobre alguna variable y los históricos.

#### 5.10.4.1 Botones

Estos botones deben simular pulsadores como los “físicos” del proceso industrial, ya que se pretenden que sean sus equivalentes en el SCADA. Para ello, se debe usar algún elemento que simule un pulsador, es decir, que mientras se mantenga pulsado la variable se ponga a 1 y cuando se suelte a 0.

##### 5.10.4.1.1 EPICS

En EPICS existe el elemento **Boolean Button** el cual está destinado a ser tanto un interruptor como un pulsador, esta característica se modifica en sus propiedades. Para asignarle un *record* se sigue el procedimiento habitual de escribirle el nombre con el canal de acceso en sus propiedades.

Por lo que estos pulsadores se configuran para *start*, *stop* y manual. Pero la seta de emergencia, en vez de ser pulsador, se configurará como interruptor para simular su comportamiento real. Además de configurarlo como transparente para colocarlo delante de la imagen de la seta. Para añadir una animación que acompañe este hecho, el contorno que engloba la dicha imagen parpadeará cuando esté activado.

Por lo que la barra inferior queda de la siguiente forma:



Ilustración 140 - Barra inferior Phoebus

##### 5.10.4.1.2 WinCC OA

En WinCC OA, hay distintas formas de realizar pulsadores. La más simple es usar un elemento de la librería *STD*, pero como su apariencia no es muy atractiva para este tipo de botón se usará el elemento *Push Button*.

Para configurarlo correctamente como un pulsador, su configuración puede parecer tediosa ya que hay que modificar su comportamiento al presionar (*Mouse Pressed*) y al dejar de presionar (*Mouse Release*). Aunque usando el asistente es bastante simple, primero se da a elegir entre leer o escribir en un *DP* o funciones de panel para luego elegir el *DP* a modificar y su valor. Entonces, al presionar ese valor se pondrá a 1 y al soltar se pondrá a 0.

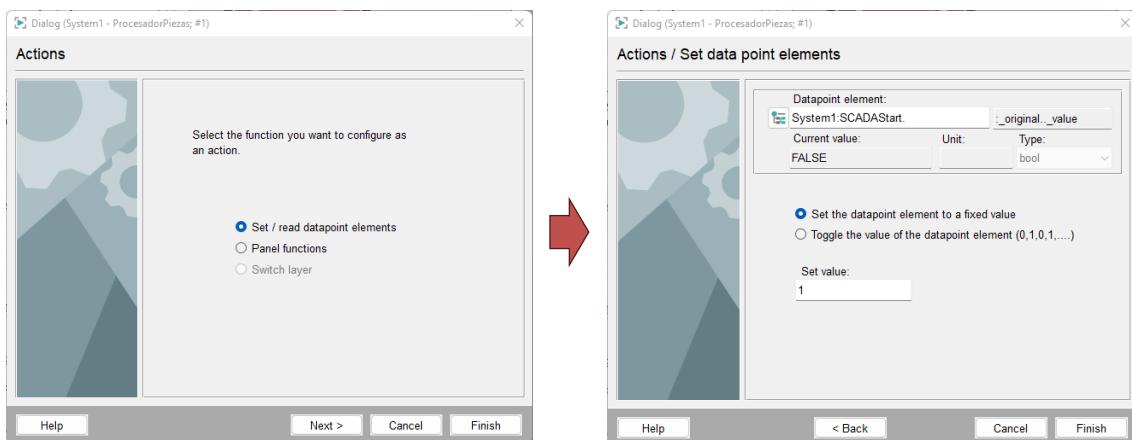


Ilustración 141 - WinCC OA Asistente Mouse Pressed

Con ello se tiene cómo configurar los botones *start*, *stop* y *manual*. Para la seta se hará un tratamiento diferente.

Como debe simular un interruptor, para aparentar que se queda presionada y luego hay que girarla para liberarla, se modificará el evento *clicked* directamente del objeto circular que contiene la imagen de la seta. Este evento se configurará en *toggle value* para que actúe como interruptor.

El script correspondiente a este evento es mucho mas simple de lo que aparenta:

```
main()
{
    bool bo;
    dpGet("System1:SCADEmergencia._online._value", bo); // Leer el valor actual
    dpSet("System1:SCADEmergencia._original._value", !bo); // Escribir el valor puesto
}
```

Igual que en Phoebus, para añadir una animación que refleje que está pulsada el contorno de la imagen parpadeará en rojo.



Ilustración 142 - WinCC OA Barra inferior

#### 5.10.4.2 Históricos

En el apartado anterior se ha mostrado la barra inferior que se colocará en cada estación, pero en el panel principal no habrá ningún tipo de control por lo que ese hueco se llenará con un histórico.

Los históricos son los elementos que recuperan y muestran los estados de una o varias variables a lo largo del tiempo.

##### 5.10.4.2.1 EPCIS

En Phoebus existen diferentes tipos de elementos que muestran por pantalla de forma gráfica los valores de alguna variable. Son los siguientes:



Ilustración 143 - Phoebus Plots

Los que interesan en este momento son *Data Browser* y *Strip chart*, que en esencia actúan igual solo que el primero se configura mediante un archivo y el segundo desde sus propiedades.

La segunda opción se configura de manera sencilla, en sus propiedades se pueden añadir tantas variables como se quieran y cada una puede configurarse de manera distinta.



Ilustración 144 - Phoebus Strip chart configuración

En la anterior ilustración se puede ver un extracto de esas propiedades, mostrando la cantidad de trazas de variables y la configuración de la primera. Siendo *NAME* lo que se mostrará, la variable a representar, tipo de representación (barras, puntos, líneas(STEP) ), color y grosor de la línea.

En la pantalla de edición, se aprecia así:

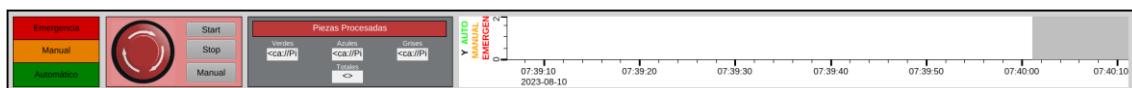


Ilustración 145 - Phoebus Barra inferior del panel principal

#### 5.10.4.2.2 WinCC OA

En WinCC OA existen varios tipos de representación gráfica de las variables a lo largo del tiempo, algunas más simples en su configuración y otras que requieren de programación a nivel de script. Las tres de la izquierda son las más simples, y la que se va a usar es *Trend Parametrization* que la primera por la izquierda de la siguiente ilustración.

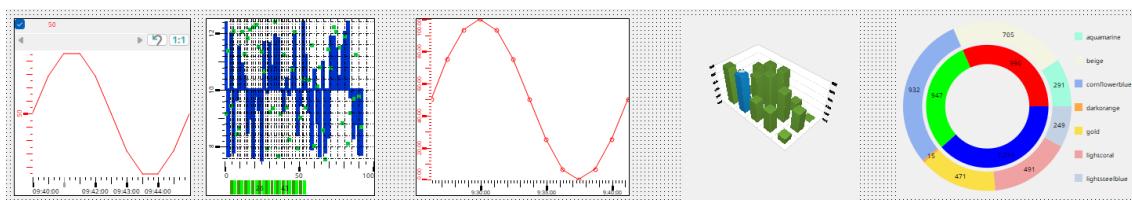


Ilustración 146 - WinCC OA Históricos

Al pulsar doblemente sobre el objeto, saldrá una ventana con la configuración del elemento:

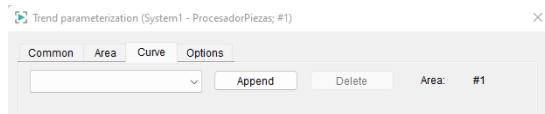


Ilustración 147 - WinCC OA Trend parametrization configuración

Para añadir variables, se pulsa el botón *Append* el cual hace aparecer otra ventana emergente con un selector de *DP*. Una vez escogido, se cierra la ventana y la anterior se llena con la configuración de la tendencia:

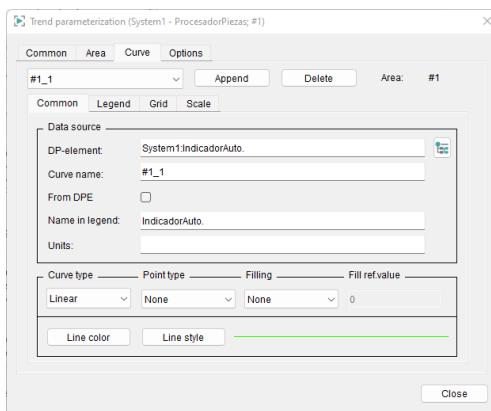


Ilustración 148 - WinCC OA Trend parametrization configuración tendencia

Como se puede comprobar, cada tendencia tiene su propia configuración independiente del resto.

Una vez añadidas el resto de las variables, la barra inferior de la pantalla principal queda de la siguiente forma:



Ilustración 149 - WinCC OA Bara inferior con histórico

## 5.10.5 Montaje de la pantalla principal – paneles empotrados

Hasta aquí se encuentra completados las tres partes que conforman la pantalla principal: **Barra superior, visión general y barra inferior**. Como se han realizado en paneles distintos para favorecer la modularidad, es momento de reunirlos todos en un nuevo panel, el principal.

### 5.10.5.1 EPICS

En Phoebus existe el elemento **Embedded Display**, el cual permite buscar un archivo de panel *.pnl* para que lo muestre. Entre sus configuraciones propias de objeto, se encuentra como se ajustará el contenido. Se puede elegir para que la ventana se redimensione en función del objeto, que se recorte o que el propio elemento se ajuste a las dimensiones del panel. Esta ultima opción es la que interesa ya que todos los paneles se han realizado con las dimensiones adecuadas para que la proporción sea perfecta.

El panel principal, queda así:



Ilustración 150 - Phoebus Panel principal final

#### 5.10.5.2 WinCC OA

En WinCC OA existen dos modos de introducir un panel dentro de otro. El primero es seleccionando la herramienta *Panel Reference*, con icono , el cual abre un selector de archivo directamente dentro de los paneles disponibles del proyecto. Otra forma aún más fácil es arrastrar directamente el panel que se quiera desde la ventana izquierda de archivos de proyecto.

En contraposición a Phoebus, este objeto no se puede configurar. Se puede dimensionar a medida, pero el panel interior se deforma acorde al tamaño del elemento.

El panel principal queda así:



Ilustración 151 - WinCC OA Panel principal final

#### 5.10.6 Resto de elementos y configuraciones

##### 5.10.6.1 Control

Como es común en los SCADA industriales, los procesos deben tener un modo manual para que pueda ser manejado desde la interfaz bajo un modo seguro. Es decir, en la programación del

autómata se permite cierto control del proceso desde eventos externos, pero siempre controlado por éste para evitar situaciones de peligro.

En este apartado se van a comentar los métodos de control sobre las estaciones en el modo manual.

#### 5.10.6.1.1 EPICS

Para Phoebus se va a usar los elementos *Boolean Button* para accionar las variables, como ya se ha explicado anteriormente. La diferencia esta vez reside en que estos botones no estarán siempre habilitados ya que el autómata ignora las acciones manuales si el modo manual no está activado.

Para ello, aparte de configurar el *record* adecuado y que se comporte como interruptor, se añade una regla para cambiar esa propiedad. Esta es la configuración:

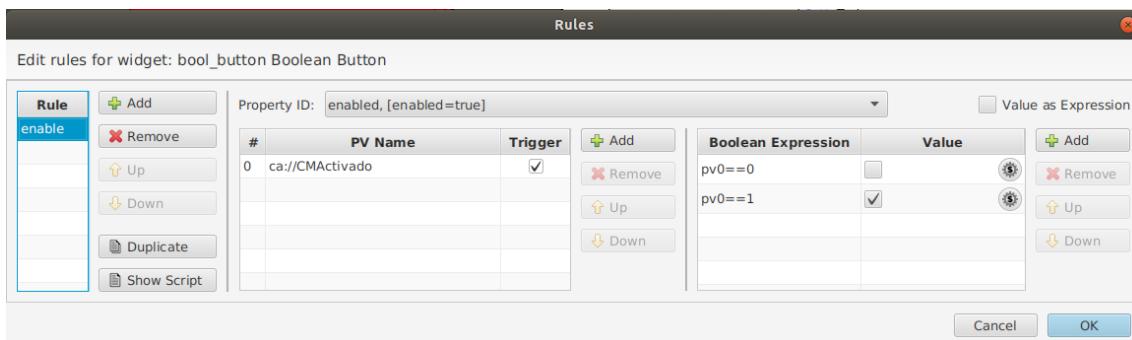


Ilustración 152 - Phoebus regla para *enabled*

En tiempo de ejecución y en modo automático se vería así:

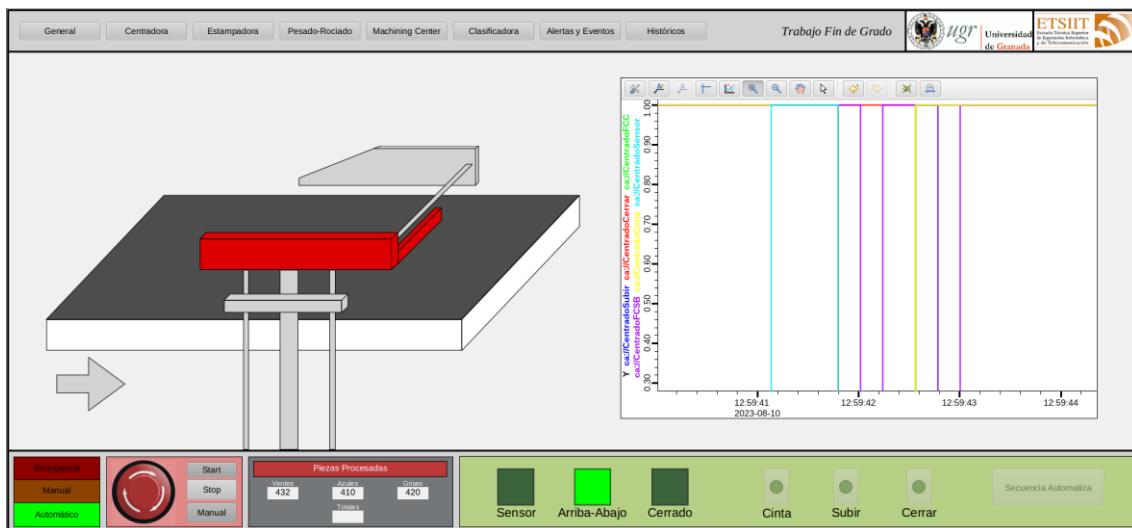


Ilustración 153 - Phoebus Centradora en ejecución automática

Y en modo manual, se hace zoom solo en la barra inferior, se vería:



Ilustración 154 - Phoebus Barra inferior en ejecución manual

### 5.10.6.1.2 WinCC OA

En este caso se va a usar un elemento nuevo, el *STD\_Switch\_1*. Su configuración para que accione un *DP* es igual que la vista hasta ahora, al pulsar aparece un cuadro de dialogo con un *DP Selector* y un recuadro para elegir si es del tipo interruptor o botón.

En este caso, para habilitarlo y deshabilitarlo ciertamente es mas complicado que en el caso anterior ya que hay que recurrir al script. Con otro tipo de objeto, como el *push button*, se realiza fácilmente con el asistente. Esto se ha hecho adrede para mostrar cómo sería el script, ya que éste puede ser **copiado y pegado sin cambios** al resto de objetos que tengan que reaccionar igual:

```
main()
{
    dyn_errClass err;

    if( !dpExists( "System1:CMActivado.:") ) //Esto es buena practica
    {
        setValue("", "color", "_dpdoesnotexist");
        return;
    }

    //Esta función conecta los cambios de estado del DP que se le pase y que ejecute la función que se le da por argumento
    dpConnect("activadoSiManual", "System1:CMActivado.");
    err = getLastErr();
    if (dynlen(err) > 0)
        setValue("", "color", "_dpdoesnotexist");
}

void activadoSiManual(string dpSource, bool boNewValue ){
    setValue("", "enabled", boNewValue); // Se habilita siempre que CMActivado lo esté
}
```

Entonces, en tiempo de ejecución se va a mostrar cuando se encuentra en estado automático:

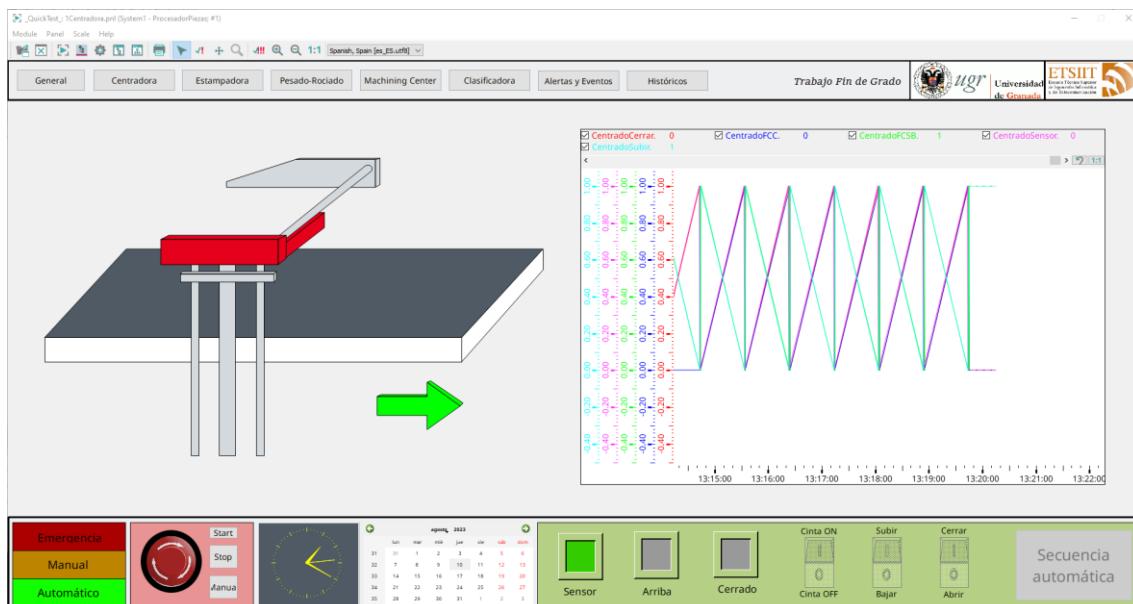


Ilustración 155 - WinCC OA Centradora en ejecución automática

Y cuando se encuentra en estado manual:

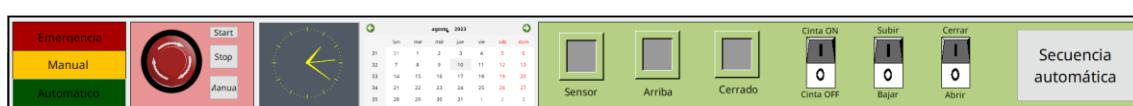


Ilustración 156 - WinCC OA Centradora en ejecución manual

### 5.10.6.2 Secuencia automática

En las estaciones *Centrador* y *Estampadora* tienen cierta secuencialidad en su funcionamiento. Pues para poder comprobar una configuración más complicada sobre las variables se ha optado por crear un botón que ejecute una secuencia de acciones para que replique los movimientos de las estaciones en el modo manual.

Esta secuencia automática al tener un nivel de complejidad superior se va a realizar mediante ejecución de scripts.

#### 5.10.6.2.1 EPICS

En Phoebus, se recomienda evitar este tipo de configuraciones y que se hagan en mayor medida mediante su configuración. Aun así permite la ejecución de scripts por si se necesitara realizar cualquier tarea de mayor complejidad.

Para comenzar con ello, se creará un *Action Button* y se le añadirá en sus acciones un script empotrado de Python.

CS-Studio ofrece una librería con diversos métodos para modificar las variables y elementos de Phoebus en scripts, pero no existe prácticamente documentación más que los comentarios del código fuente o la escasa ayuda que ofrece Phoebus con los ejemplos.

Aun así, un script muy básico que escribe sobre las variables de la estación *Centrador* quedaría así en Python:

```
# Embedded python script
from org.csstudio.display.builder.runtime.script import PVUtil, ScriptUtil
import time

widget.setPropertyValue('enabled', 0) //Se desactiva el elemento para que no se vuelva a pulsar

PVUtil.writePV("ca://CMCentradoSubir", 1,1)
PVUtil.writePV("ca://CMCentradoCerrar", 0,1)
PVUtil.writePV("ca://CMCentradoCinta", 0,1)

time.sleep(1) //Como no ofrece ningún método de espera al cambio de estado, simplemente se espera un tiempo

PVUtil.writePV("ca://CMCentradoSubir", 0,1)

time.sleep(1)

PVUtil.writePV("ca://CMCentradoCerrar", 1,1)

time.sleep(1)

PVUtil.writePV("ca://CMCentradoSubir", 1,1)
PVUtil.writePV("ca://CMCentradoCerrar", 0,1)
PVUtil.writePV("ca://CMCentradoCinta", 0,1)

time.sleep(1)

widget.setPropertyValue('enabled', 1)
```

#### 5.10.6.2.2 WinCC OA

En cambio, WinCC OA toda configuración que se realice con el asistente es traducido a código. De hecho, la gran virtud de WinCC OA es el abanico de posibilidades que se crea permitiendo la configuración por scripts propios. Éstos pueden tener la complejidad que se requiera o abrir cuadros de dialogo o alertas. Además, esta extensamente documentado.

A continuación, se muestra cómo queda la secuencia en el script:

```
#uses "classes/DialogFramework"
main()
{
    dyn_errClass err;
    bool lectura;
```

```

int iteraciones = 0;
//Deshabilitar elemento para evitar ejecución paralela de este script
this.enabled(0);

//Estado inicial en reposo
dpSetWait("System1:CMCentradoSubir._original._value", 1);
err = getLastErrorCode();
if (dynlen(err) > 0)
    errorDialog(err);

dpSetWait("System1:CMCentradoCerrar._original._value", 0);
err = getLastErrorCode();
if (dynlen(err) > 0)
    errorDialog(err);

//Espera
delay(1);

//Primera etapa BAJADA
dpSetWait("System1:CMCentradoSubir._original._value", 0);
err = getLastErrorCode();
if (dynlen(err) > 0)
    errorDialog(err);

//Espera hasta que llegue abajo
do{
    delay(1);
    iteraciones++;
    dpGet("System1:CentradoFCSB._online._value", lectura);
}while(iteraciones < 4 && !lectura);

if( iteraciones >= 4 && !lectura ) // Método para comprobar si ha ocurrido algún fallo en el entorno
físico
    DialogFramework::warning(makeMapping("text", "CENTRADORA - Cilindro vertical bloqueado",
                                         "buttonTextOk", "OK",
                                         "buttonTextNOK", NULL)); // make buttonTextNOK not visible

else{ //Solo si ha llegado abajo sin errores cerrará
    //Segunda etapa
    dpSetWait("System1:CMCentradoCerrar._original._value", 1);
    err = getLastErrorCode();
    if (dynlen(err) > 0)
        errorDialog(err);

    //Espera hasta que cierre pero no se comprueba porque en caso de que haya pieza no llega hasta el FCC
    delay(2);
}

//Vuelta a reposo, arriba y abierto
dpSetWait("System1:CMCentradoCerrar._original._value", 0);
err = getLastErrorCode();
if (dynlen(err) > 0)
    errorDialog(err);

dpSetWait("System1:CMCentradoSubir._original._value", 1);
err = getLastErrorCode();
if (dynlen(err) > 0)
    errorDialog(err);

this.enabled(1);
}

```

### 5.10.6.3 Diales

Los diales son generalmente esferas con números y marcas, que se utiliza para mostrar valores analógicos de medición. En este caso se han usado para reflejar el nivel del tanque de una forma diferente.

#### 5.10.6.3.1 EPICS

En el caso de Phoebus solo hay un elemento que pueda imitar un dial y es el *Meter*. Tiene escasa personalización y su configuración se limita a solo representar la variable, su formato y sus límites.

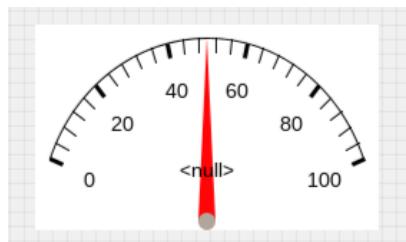


Ilustración 157 - Phoebus Dial

### 5.10.6.3.2 WinCC OA

En WinCC OA se dispone de dos opciones, una más parecida a la de Phoebus con una apariencia rudimentaria y otra mucho visual y personalizable pero que requiere que se modifique todo esto mediante script. Este objeto es **Dial Gauge EWO**, *EWO* es otra librería de objetos.

Mirando su documentación se puede ver como configurarlo detalladamente. Como se pretende representar el nivel del tanque, se le va a indicar cuales son los límites de éste (20% y 80%) para que los represente en colores.



Ilustración 158 - WinCC OA Dial Gauge EWO

Para obtener esa apariencia, este es el script:

```
main()
{
    // Para las franjas de color
    this.addMarking(0,2,"red");
    this.addMarking(2,8,"green");
    this.addMarking(8,10,"red");

    // Límite superior del DP
    this.maximum(10.0);

    // Animación de cristal no
    this.showGlass(0);

    // Ligar DP al dial
    dpConnect("setGauge","System1:PesadoRociadoNivelReal::_online.._value");
}

void setGauge(string dpSource, float newValue){
    setValue("DialGauge_ewo2",newValue);
}
```

### 5.10.6.4 Grupos

#### 5.10.6.4.1 EPICS

#### 5.10.6.4.2 WinCC OA

### 5.10.6.5 Variables de cálculo

En ocasiones, en el lado de los SCADA, se requiere un procesamiento de los datos para tratarlos o almacenarlos de forma más correcta. Para ello, se puede configurar en ambos SCADA una variable virtual, que no se comunique con el autómata, que adquiera valores de una o varias variables y las procese con alguna función matemática.

En este apartado se va a comprobar cómo se crearía con cada uno con el objetivo de calcular el número total de piezas procesadas.

#### 5.10.6.5.1 EPICS

Esta vez, se bajará a nivel de EPICS para configurar un nuevo tipo de *record*, el llamado *calc.d*

El registro de cálculo o "Calc" se utiliza para realizar operaciones algebraicas, relacionales y lógicas en valores obtenidos de otros registros. El resultado de sus operaciones puede luego ser accedido por otro registro para que pueda ser utilizado.

Este tipo de *record* puede reunir valores de hasta 12 variables, es decir, desde la A hasta la L. Estas valores deben venir de otros *records* y directamente desde OPC UA.

La expresión matemática es muy versátil ya que tiene operaciones algebraicas, trigonométricas, relacionales, lógicas y a nivel de bit. Además, tiene expresiones condicionales, las bien conocidas: *(condición)? A+B+C : A\*B\*C\*10*.

En este caso basta con la operación suma, así que el *record* se ve de la siguiente forma:

```
record(calc, "PiezasTotales") {
    field(INPA, "PiezasVerdes")
    field(INPB, "PiezasAzules")
    field(INPC, "PiezasGris")
    field(CALC, "A+B+C")
    field(SCAN, "1 second")
}
```

Una vez añadido el *record*, se tiene que recomilar el IOC con *make rebuild*.

#### 5.10.6.5.2 WinCC OA

En el caso de WinCC OA, hay que crear un nuevo DP del tipo de dato que se quiere conseguir después de la operación. En este caso, el que ya se creó con anterioridad para los valores numéricos. A ese nuevo DP, se le inserta una configuración llamada *DataPoint function* y se configura al tipo de función *DPE Connection*. Entonces, saldrá la configuración necesaria para añadir los DPs que se necesiten y la expresión matemática que se necesite.

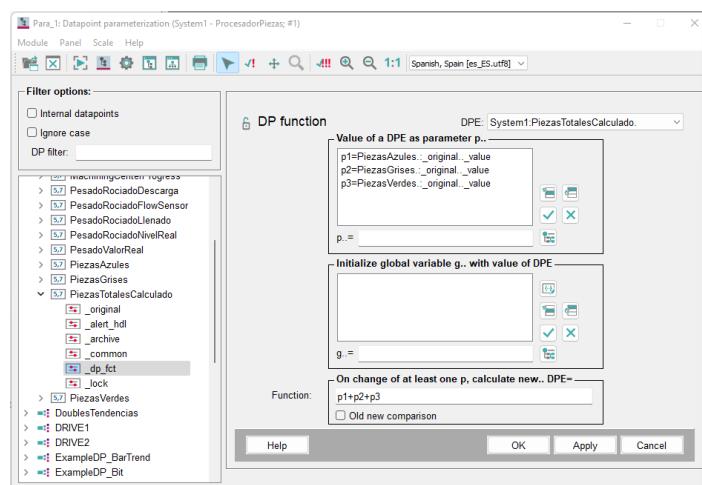


Ilustración 159 - WinCC OA Data Point de calculo

## 5.11 Gestión de alarmas

### 5.11.1 EPICS

Existen diversos modos de gestionar alarmas en EPICS ya que existe un módulo independiente con su propia interfaz gráfica. Además, Phoebus permite la gestión de alarmas con herramientas adicionales.

En el segundo caso, las herramientas necesarias son **Kafka** (Distribuido por Apache, que almacena la configuración de las alarmas y hace de intermediario entre la interfaz y el servidor de alarmas ) y el **servidor** que monitoriza los estados de alarma de los *records*.

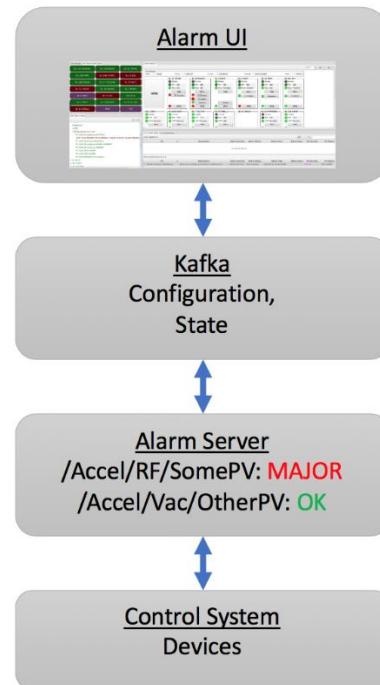


Ilustración 160 - Phoebus Componentes del sistema de alarmas

Según el archivo anterior, hay que instalar Kafka para su uso en producción en */opt/Kafka*. Para seguir preparando el sistema para producción, es conveniente que se ejecute como servicio del sistema. En *Phoebus/app/alarm* se encuentran los archivos *.service* que facilitan esto. Para ello se ejecutan los siguientes comandos:

1. `sudo cp *.service /etc/systemd/System`
2. `# Start manually`
3. `sudo systemctl start zookeeper.service`
4. `sudo systemctl start kafka.service`
5. `sudo systemctl start alarm_server.service`
6. `# Enable startup when host boots`
7. `sudo systemctl enable zookeeper.service`
8. `sudo systemctl enable kafka.service`
9. `sudo systemctl enable alarm_server.service`

Una vez con los servicios en funcionamiento, hay que crear un *Topic*. En la carpeta anterior ya se facilita un script *create\_alarm\_topics.sh* el cual se usa de la siguiente forma:

```
sh create_alarm_topics.sh Accelerator
```

Entonces ya se puede abrir Phoebus y sus aplicaciones de *Alarm* y *Alarm Tree* que leen del servidor anterior los datos.

Para que el servidor sea realmente efectivo, falta por configurar a los *records* que lo requieran sus estados de alarma. En el caso de los valores analógicos se tienen dos rangos de alarma, el superior e

inferior y para cada extremo hay dos etiquetas para definirlos: *HIHI*, *HIGH*, *LOW*, *LOLO*. También hay etiquetas para asignar severidad a esos límites.

Como ejemplo, así quedaría el *record* del nivel del tanque:

```
record(ai, "PesadoRociadoNivelReal") {  
    field(DTYP, "OPCUA")  
    field(INP, "@SUB1 ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoNivelReal")  
    field(HIHI, "10")  
    field(HIGH, "9")  
    field(LOW, "1")  
    field(LOLO, "0")  
    field(LSV, "MAJOR")  
    field(LLSV, "MAJOR")  
    field(HSV, "MINOR")  
    field(HHSV, "MAJOR")  
    field(SCAN, "I/O Intr")  
}
```

### 5.11.2 WinCC OA

En WinCC OA todo lo necesario para gestionar alarmas ya está incluido en el programa. Para que un *DP* pueda levantar una alarma, hay que insertarle una configuración adicional llamada *Alert Handling* del mismo modo que se han insertado otras configuraciones. Para recordarlo, desde el módulo *PARA* pulsar con el botón derecho sobre el *DP*, pulsar sobre el desplegable *Insert config* y saldrá una ventana con las posibles configuraciones y elegir la segunda:

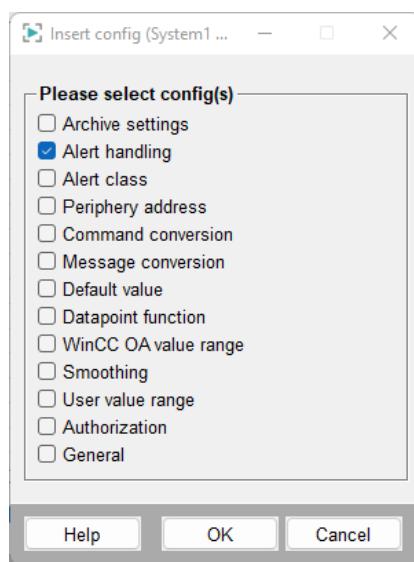


Ilustración 161 - WinCC OA Insert Config Alert

Entonces ya se podrá editar dicha configuración. Se pueden añadir distintos rangos de alarma con sus respectivos valores, hasta 20. Además, se puede elegir si incluir el valor o no en el rango y un texto que acompañe a la alarma. Entonces para mostrar como quedaría configurado para el valor del tanque sería:

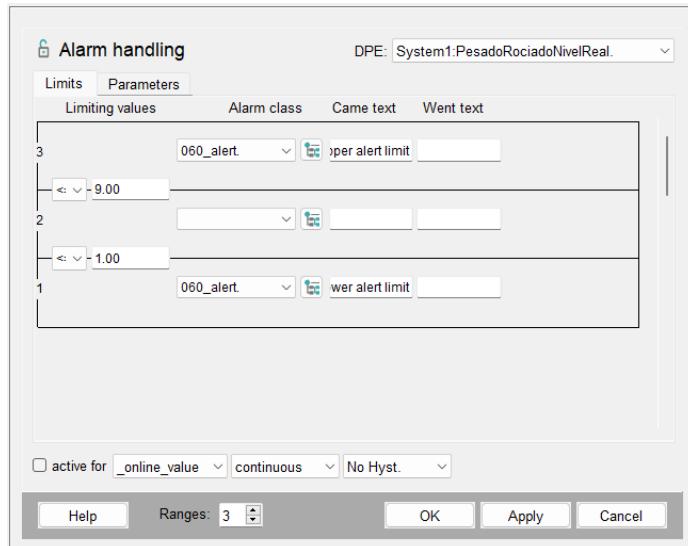


Ilustración 162 - WinCC OA Ventana Alarm Handling

Entonces, la mayor parte de los objetos reaccionarán sin configuración adicional alguna a los estados de alarma. Además, el programa ofrece una ventana ya desarrollada para manejar todas las alarmas y eventos. Esta ventana se puede acceder, en tiempo de ejecución, desde *System Management > Diagnosis > Alarm and Events Screen*. Pero para tener un acceso directo más cómodo en la aplicación final, se ha configurado un botón en la barra superior para que ejecute un comando en su script de clicado. Queda de la siguiente forma:

```
main()
{
    openAES("aes_default");
}
```

Al abrirla, hay que pulsar el botón *start* que está en la parte inferior. Entonces, tiene la siguiente apariencia:

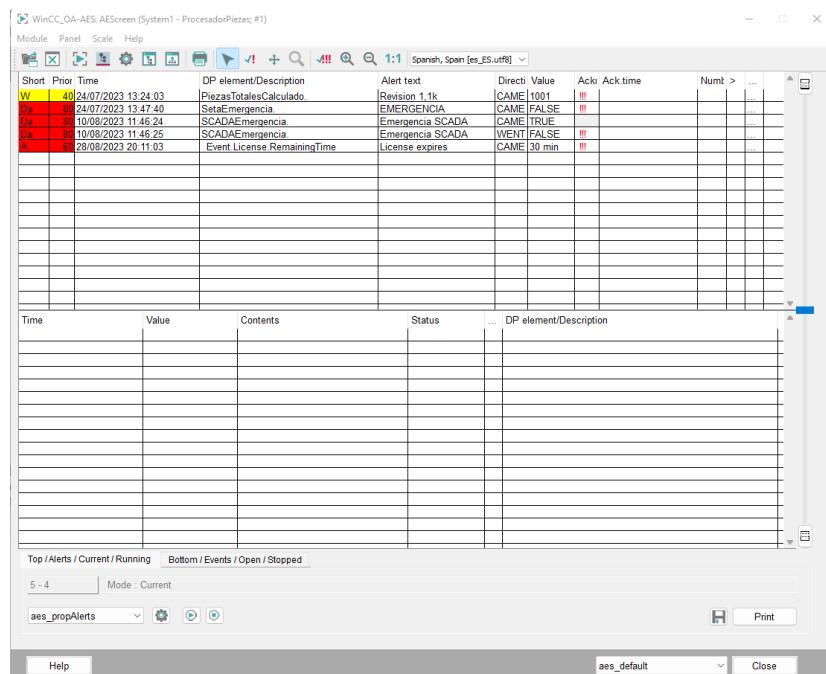


Ilustración 163 - WinCC OA Alarm and Events Screen

## 5.12 Gestión de remanencia de datos

### 5.12.1 EPICS

De nuevo, el sistema de archivado de datos o históricos no viene implementado de base. Phoebus en este caso necesita de un sistema RDB, es decir, una base de datos relacional. Esta base de datos se configura con una plantilla que el mismo Phoebus proporciona.

Entonces para configurar este sistema se comienza instalando *MySQL*, activando el servicio y seleccionando una contraseña para *root*:

1. `sudo apt install mariadb-server`
2. `sudo systemctl start mariadb`
3. `/usr/bin/mysql_secure_installation`

Y para iniciar la RDB con el sistema:

```
sudo systemctl enable mariadb.service
```

Es entonces cuando se crea las tablas de archivado. Phoebus proporciona los comandos necesarios para crearlas simplemente copiando y pegándolos como comandos. Para ello hay que conectarse a *MySQL* como *root*:

```
mysql -u root -p'$root'
```

El archivo plantilla con los comandos se encuentra en su GitHub en [services/archive-engine/dbd/MySQL.dbd](#).

Para que este servicio sepa que *records* monitorizar, hay que crear un fichero *.xml*. Para conocer la estructura de este archivo, se puede exportar una “demo” del propio servicio con el siguiente comando:

```
sh archive-engine.sh -engine Demo -export Demo.xml
```

Entonces, modificando ese archivo, se añaden las variables que se desean almacenar de la siguiente forma:

```
<channel>
  <name>ca://PesadoRociadoValorReal</name>
  <period>0.1</period></monitor>
</channel>
```

Entonces, habrá que importar el archivo y luego levantar el servicio con los siguientes comandos:

1. `archive-engine.sh -engine Demo -import Demo.xml -port 4812 -replace_engine`
2. `archive-engine.sh -engine Demo -port 4812 -settings my_settings.ini`

Phoebus tiene configurado por defecto leer directamente de la base de datos, por lo que no se requiere configuración adicional. Cuando acceda un objeto gráfico a una variable, accederá a su archivo.

### 5.12.2 WinCC OA

De igual forma que con las alarmas, la gestión de remanencia de datos ya viene incluida en el programa y basta con añadir una nueva configuración a los DPs, en este caso *Archive settings*. Una vez añadida, para activarla hay que marcar la casilla de *Active*. En este momento se puede escoger entre los distintos archivos donde guardar los DPs o incluso crear uno nuevo, esos archivos se pueden configurar en número máximo de DPs, tamaño del archivo, compresión, tiempo...

Además, hay otra casilla llamada *smoothing* la cual permite reducir el volumen de datos que se procesan en el sistema WinCC OA. Esto se dejará de lado, pero es bueno tener en cuenta su existencia.

Entonces, la configuración quedaría de la siguiente forma:



Ilustración 164 - WinCC OA Archiving

Con esta configuración realizada, cualquier objeto que acceda a un DP accede directamente al archivo histórico y recupera sus datos.

## 5.13 Aplicaciones finales

Se adjuntan las imágenes en el anexo 1 ([146](#)).

# 6 Planificación y presupuesto

---

Para asegurar que se abarcara adecuadamente las diferentes áreas del proyecto, se llevó a cabo una división en varias partes durante la fase de planificación.

Comenzamos con la necesidad de comparar dos herramientas software que en principio tienen el mismo objetivo, creación de SCADAS para el control y supervisión de un entorno. En este caso, el entorno donde se quieren testear es en uno industrial.

## 6.1 Fases

### 6.1.1 Fase inicial

Se parte con **ningún** tipo de **conocimiento** sobre ambos programas, por lo que esta fase puede considerar de **aprendizaje** además de la realización de pruebas para hacer una primera toma de contacto.

#### 6.1.1.1 Aprendizaje

Para adquirir conocimientos y competencias en el uso de los programas, se ha comenzado por una comprensión profunda de los conceptos fundamentales de control de procesos y sistemas distribuidos

Posteriormente se realizó un estudio exhaustivo de la documentación oficial de EPICS, comprendiendo su arquitectura y filosofía. Además, revisar los recursos relacionados como documentos introductorios, programas de ejemplo y revisión de los módulos existentes.

Seguidamente, se realizó otro estudio exhaustivo de la documentación de WinCC OA, captando desde primera hora las similitudes y diferencias de ambos software.

#### 6.1.1.2 Pruebas

##### 6.1.1.2.1 Instalación

Una vez adquiridas las bases necesarias, se instalaron los programas para realizar las pruebas. Desde este momento, con EPICS se descubrió algunos problemas como las numerables dependencias con librerías y sistemas operativos, fijando desde este momento la necesidad de una máquina virtual con un sistema operativo Linux.

##### 6.1.1.2.2 Estudio de simulación

Estudio de simulación del autómata – plcsim advanced VS plcsim + nettoplcsim

Para este momento se usó una maqueta simulada, ya desarrollada, de la asignatura *Informática Industrial* impartida por el tutor de este proyecto, Miguel Damas, con el motivo de realizar estas primeras pruebas de la forma más rápida y profunda.

Además, se fue estudiando qué autómata iba a ser el más adecuado, con que programa se iba a simular (*PLCSim o PLCSim Advanced*) ya que de ellos dependería el modelo y la forma de comunicación con los SCADAS.

##### 6.1.1.2.3 Estudio de comunicaciones

Estudio de comunicaciones – opcua o s7nodave

Aunque se tuvo desde primera hora cierta inclinación hacia la comunicación mediante OPC UA, se estuvo indagando en el uso del módulo de EPICS **s7nodeave**. Con él se comenzaron las pruebas de instalación de EPICS en Windows, pero por problemas de dependencias de librerías, se tuvo que trasladar a Ubuntu en una máquina virtual.

En la máquina virtual, también se estudió la comunicación OPC UA la cual fue la definitiva.

### 6.1.2 Fase de análisis de requisitos

Terminando las etapas de los estudios de la simulación y comunicaciones, se comenzó a recabar los requisitos que debían cumplir las aplicaciones SCADA como son la homogeneidad entre pantallas, dimensiones de los objetos y texto, gestión de las alarmas y de históricos.

Además, se estableció una jerarquía en las ventanas de las aplicaciones.

No solo se reunieron los requisitos de los SCADA, sino que se llegó, en mutuo acuerdo con el tutor, en cómo debería ser el proceso industrial final. Debía ser un proceso secuencial, como en un acelerador de partículas, debía tener tanto elementos digitales como analógicos para posteriormente, en el desarrollo de los SCADA ,se pudiera hacer uso del mayor tipo de elementos disponibles.

### 6.1.3 Fase de desarrollo

#### 6.1.3.1 Desarrollo Factory IO

Antes de terminar la fase de análisis de requisitos, se desarrollaron distintos prototipos de simulaciones que se fueran adecuando a estos requisitos y poder verlos de una forma más visual, ayudando así a llenar los que faltara.

Una vez los requisitos estaban establecidos, se diseñó el proceso final el cual ha perdurado casi intacto hasta el final del proyecto. Solo ciertos elementos o el orden fueron cambiados.

#### 6.1.3.2 Programación PLC

Con el proceso casi acabado, se comenzó la programación de éste. Gracias a que se diseñó modularmente, siendo estos módulos independientes, se pudo hacer la programación del PLC paralelamente y así modificar ciertos aspectos del proceso para aligerar esta programación.

#### 6.1.3.3 EPICS y WinCC OA

En esta etapa se intercalaron los desarrollos de EPICS y WinCC OA para que ambos SCADAS progresaran casi al mismo tiempo y ver de forma objetiva las ventajas e inconvenientes de cada uno.

Esta etapa se puede considerar que es la que más tiempo y dedicación a requerido.

### 6.1.4 Fase de prueba y corrección

Enlaza con la anterior, en la finalización de cada pequeño hito se iba comprobando su funcionamiento y corrigiéndose en el caso de que no fuera el esperado.

### 6.1.5 Redacción de la memoria

Casi desde el comienzo de este proyecto, la información con la que se iba indagando se trasladó a la memoria de forma rudimentaria y poco estructurada para su posterior formateo y cribado. Desde el

momento que el proceso con Factory IO ya tenía forma, se comenzó la redacción para ir dejando bien documentado el inicio de esta memoria.

## 6.2 Diagrama de Gantt

El siguiente diagrama consiste en una representación gráfica en forma de barras que muestra las actividades del proyecto en el eje horizontal y las fases de éste en el eje vertical. Cada barra representa una actividad, su longitud indica su duración estimada y su posición en el tiempo muestra cuándo se llevó a cabo. Además, las barras se superponen para mostrar dependencias entre tareas y la secuencia en las que se han realizado.

Aunque no es común en este tipo de diagramas, se ha modificado ligeramente el grosor de las barras para apreciar rápidamente donde se encontraron más conflictos y/o la dedicación fue más intensa.

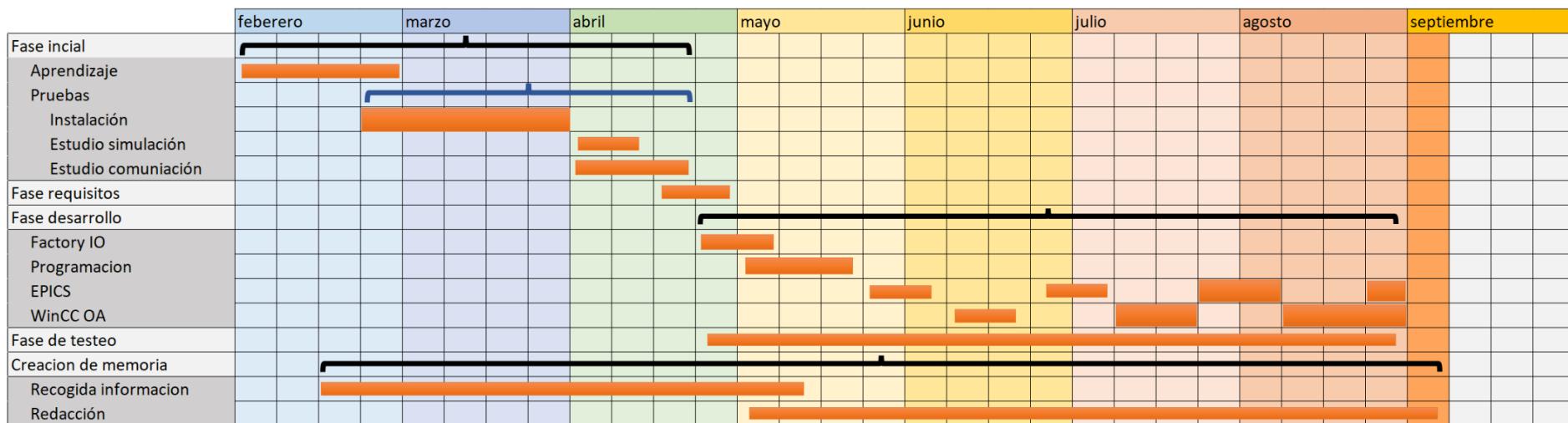


Ilustración 165 - Diagrama de Gantt

## 6.3 Presupuesto

### 6.3.1 Recursos

En este punto se va a desglosar los recursos necesitados, tanto hardware como software.

#### 6.3.1.1 Hardware

- Ordenador portátil MSI GF62 8re.

#### 6.3.1.2 Software

- **Sistemas operativos (64 bits):**
  - Windows 10 pro
  - Ubuntu 18
- **Dependencias de EPICS:**
  - Git
  - Make
  - G++
  - Libreadline
- **Dependencias de OPCUA:**
  - CMake
  - OpenSSL
  - LibXML2
  - *Unified Automation C++ SDK*
- **Dependencias de Phoebus:**
  - JDK11
  - Maven
- **WinCC OA**

### 6.3.2 Costes

#### 6.3.2.1 Licencias

Solo interviene la licencia de WinCC OA, el precio se ha estimado a raíz de un presupuesto real para un proyecto de gran envergadura. Se ha cogido las licencias acordes a este caso de uso: WinCC OA Server Basic, alarmas, tendencias, archivado y OPC UA por **1307,12€**; WinCC OA V3.18 para el desarrollo con el editor gráfico, catálogo de iconos, entorno de scripts y framework de ensayo **5227,30€**; WinCC OA 100 Power Tags (estimado al 0,40cts por tag) **40€** y WinCC OA Basic Training **2801,25€**.

El costo del sistema operativo Windows no se tendrá en cuenta ya que ambos programas pueden usarse en sistemas operativos alternativos sin coste.

El coste total de licencias asciende a **9375,67€**.

#### 6.3.2.2 Recursos materiales

En este caso solo se tendrá en cuenta el ordenador donde se ha realizado el proyecto, aunque en una aplicación final no tiene por qué ser este equipo.

Se estipula que los materiales informáticos se amortizan entre los 4 años de forma acelerada hasta los 8 de manera lenta. Como se le ha dado un uso intensivo para este proyecto, se tendrá en cuenta el límite inferior.

Este costó en 2019 la cantidad de 1248,99€, por lo tanto, se amortizará 312,25€ cada año. O lo que es lo mismo, 26€ al mes. Este proyecto se ha desarrollado durante 7 meses por lo que los costos materiales ascienden a **182€**.

#### 6.3.2.3 Recursos humanos

El coste de personal se tendrá en cuenta un único empleado, el cual tendría el puesto de Ingeniero en Sistemas Informáticos.

Este puesto en España, según datos publicados en *indeed*, cobra de media la hora a 15,93€. Con este dato, se adapta el presupuesto de este proyecto.

En los siete meses de desarrollo del proyecto, traducido a días es 212 desde uno de febrero hasta 1 de septiembre. Eliminando las cuatro semanas dedicadas a Factory IO y programación son 184 días. A cinco horas de media por día, 920 horas. Por lo que el gasto en recursos humanos asciende a **14655,6€**.

#### 6.3.2.4 Otros

Aquí se tendrá en cuenta las necesidades del personal, como escritorio, gasto en luz o internet:

- **Escritorio 300€**
- **Fibra óptica 182€.** Teniendo en cuenta que suele costar las más baratas a 26€ al mes.
- **Luz 60,72€.** Sabiendo que este ordenador consume 600W y que el precio medio de la luz, según la OCU en estos siete meses, es de 0,11 €/kW se obtiene:  $0,6\text{ kW} * 920 \text{ horas} * 0,11 \text{ €/kW} = 60,72\text{ €}$ .

Ascendiendo este costo a **550,72€**.

#### 6.3.2.5 Total

A continuación, se resumen los gastos del estudio completo del proyecto:

Descripción	Gasto
Licencias	9375,67
Recursos Humanos	14665,6
Otros	550,72
<b>Total</b>	<b>24591,99</b>

Para un desglose más adecuado, se muestran los gastos divididos por cada programa. Teniendo en cuenta que EPICS ha requerido un 60% del tiempo total y WinCC OA un 40%, se divide acordemente los recursos humanos y dividiendo licencias:

Descripción	EPICS	WinCC OA
Licencias	0	9375,67
Recursos Humanos	8799,36	5866,24
Otros	550,72	550,72
<b>Total</b>	<b>9350,08</b>	<b>15792,63</b>

Tabla 2 - Resumen de gastos por programa

# 7 Análisis y valoraciones

---

Este apartado se va a dividir en diferentes subsecciones claras para diferenciar claramente lo que son primeras impresiones de valoraciones finales. Esos apartados son:

**Discusión:** donde se recogen las características principales de cada programa y se comparan de forma paralela para recoger una idea general.

**Impresiones:** Este apartado puede parecer engañoso ya que se recogen las **primera impresiones** de cada programa. Estas impresiones pueden llamarse también **barreras de entrada**. Por lo que se recoge aquí no perdura en el tiempo en su mayoría, pero es importante darlo a conocer.

**Tabla comparativa:** Se coloca este apartado en este orden **deliberadamente** porque se quiere estacionar la idea que se está formando el lector y que revise una tabla comparativa que resume todo el proyecto.

**Curvas de esfuerzo:** Con la misma intención que el apartado anterior, se quiere volver a moldear la idea del lector con una gráfica que muestra de forma visual esas barreras de entrada.

**Conclusión:** Para terminar, las conclusiones que se ha forjado un servidor durante el desarrollo del proyecto.

## 7.1 Discusión en base a los resultados

WinCC OA y EPICS son dos sistemas de software ampliamente utilizados en la industria para la supervisión y el control de procesos. Aunque ambos tienen sus ventajas y desventajas, la elección entre ellos dependerá de las necesidades y requisitos específicos de cada proyecto.

En primer lugar, la interfaz de usuario y la capacidad de visualización de WinCC OA son altamente reconocidas por su diseño intuitivo (a partir del corto-medio plazo) y capacidades de personalización. Esta plataforma ofrece una amplia variedad de herramientas para crear pantallas de visualización de procesos, gráficos y paneles de control, lo que facilita la representación visual de datos complejos.

Por otro lado, EPICS tiene capacidades de visualización, pero su enfoque se centra más en el control de procesos y la comunicación de dispositivos. El papel de creación de interfaces la delega en módulos, los cuales tienen sus diferencias. Phoebe en este caso es altamente intuitivo, en el corto plazo, pero a medio-largo plazo se aprecia carencias en las herramientas para crear sistemas más complejos.

En segundo lugar, WinCC OA a menudo se elige en entornos donde la integración con otros sistemas y dispositivos es esencial. Su capacidad para comunicarse con una amplia gama de protocolos industriales y dispositivos de campo es una ventaja significativa.

EPICS también tiene capacidades de comunicación, pero su enfoque original estaba en la física experimental y, aunque se ha adaptado para su uso en la industria, depende completamente de que un tercero haya implementado ese driver. Aunque, siendo ventaja del software libre, se tiene la capacidad de crear su propio driver completamente personalizado o una alta probabilidad de que un tercero ya lo haya hecho.

Otra consideración importante es el soporte. WinCC OA cuenta con un respaldo sólido de soporte técnico y documentación extensa.

EPICS es una solución de código abierto con una comunidad activa, pero su implementación y soporte podrían depender más de la capacidad interna de la organización para resolver problemas y mantener el sistema.

En resumen, la elección entre WinCC OA y EPICS dependerá de factores como la interfaz de usuario, las necesidades de comunicación, la escalabilidad, capacidad monetaria, garantías y el soporte requerido en un proyecto específico.

Si se valora en el momento del desarrollo una interfaz de usuario intuitiva, una fuerte integración con otros sistemas y un soporte robusto, WinCC OA podría ser la elección preferida en comparación con EPICS.

Sin embargo, cada sistema tiene sus propias fortalezas y debilidades, por lo que es importante evaluar cuidadosamente las necesidades del proyecto antes de tomar una decisión.

## 7.2 Impresiones

La filosofía con la que se ha creado EPICS hace que la primera toma de contacto sea tediosa y requiera mucho tiempo para poder implantarlo por completo. La instalación de dependencias, tanto del software base, módulos y drivers junto a la instalación de módulos y su configuración hace que la experiencia se vea mermada desde primer momento.

Algo favorecedor de EPICS, hablando del software base, es que está ampliamente documentado de cara a su funcionamiento. Aunque en la parte del proceso de instalación escasean ciertas aclaraciones, se aporta ejemplos que lo completan.

En contraposición, la documentación de los módulos y drivers dependen plenamente de quien los haya creado. En el caso del driver OPC UA, se presupone demasiado que se conoce y se tiene buena habilidad con EPICS, lo que hace su instalación y configuración más complicada. Phoebus, descendiente de CS-Studio, la documentación es extremadamente limitada y pese a ofrecer gran variedad de ejemplos, éstos se quedan cortos y no se puede profundizar en el uso de cada objeto. Además, tiene documentación relativamente escondida, es decir, no se destaca en la documentación oficial ni en otros sitios que existe más información sobre los servicios que ofrece dentro de las carpetas profundas del programa.

Después de haberse familiarizado con todo lo anterior, es relativamente sencillo poner en marcha un IOC.

Un punto controvertido son los cambios en el IOC, ya que debe recompilarse y relanzarse. Esto puede ser contraproducente, pero puede hacerse en cualquier momento sin cerrar la interfaz de Phoebus.

La creación de interfaces es bastante sencilla y versátil. Las herramientas que ofrece Phoebus son bastante intuitivas y cómodas de usar, pero rápidamente se puede percibir que no se pueden crear aplicaciones con mucha complejidad. Es una aplicación para crear interfaces sencillas.

Phoebus ofrece la posibilidad de añadir funcionalidades con el scripting, un punto a favor es que puede ser en Python o Java Script, pero no recomiendan su uso ya que no está diseñado para ejecutar scripts. Eso limita en gran medida la personalización, que queda ligada a lo que se puede hacer con scripts simples, malabares con las expresiones booleanas entre las distintas configuraciones. Puede considerarse suficiente en muchos casos, pero no es lo suficiente profundo y para una personalización exhaustiva llega a ser muy tedioso o casi inabarcable.

En la gestión de alarmas vuelve a hacerse visible la escasez y lo mal enlazada que se encuentra la documentación de Phoebus. Solo se explica cómo funciona el sistema de alarmas y no qué herramientas son realmente necesarias o como se implantan en la aplicación, ya que no vienen de base. No es hasta el último punto de su documentación donde indica que para más detalles se visualice el archivo *Readme.md* en el directorio *Phoebus/app/alarm/*. Que es donde se encuentra un pequeño tutorial para hacer funcionar un sistema de alarmas ejemplo.

Para almacenar los datos de cada *record* vuelve a ser necesario indagar dentro de los archivos de Phoebus hasta encontrar el servicio de archivado y su documentación. Además, no viene configurado de base. Hay que crear una base de datos relacional por cuenta propia y adaptarla a las necesidades de Phoebus. Al menos aporta una plantilla que agiliza esa configuración.

En el caso de WinCC OA, la instalación es de lo más sencillo. En un solo ejecutable se incluye todas las dependencias, drivers y módulos necesarios. Esto genera una pequeña falsa creencia que el resto de programa se usará así de fácil.

El punto más controvertido de WinCC OA puede ser su documentación ya que es realmente extensa, pero se puede encontrar ayuda sobre cualquier tipo de objeto. El problema es que al ser tan extensa está relativamente mal enlazada. Si se busca como conseguir algo, como puede ser la configuración de un elemento para conectarlo con un DP o modificarlo, no se encontrará toda la información necesaria en un mismo apartado, además que los ejemplos son escuetos.

La interfaz queda lejos de ser intuitiva, en los primeros intentos se hace complicado acceder a todas las configuraciones y tener claro para qué sirve cada ventana. Requiere de cierto tiempo adaptarse, pero una vez conseguido se vuelve muy amigable, fácil de usar y rápida de gestionar. Eso permite profundizar en gran medida en la configuración de cada objeto.

El scripting en WinCC OA es algo que cuesta trabajar en los primeros intentos, pero una vez se consigue entender, se abre un inmenso abanico de posibilidades. Permite crear aplicaciones muy complejas de forma bastante sencilla y rápida ya que este lenguaje lleva la misma estructura que el lenguaje C. Además, incluyen una extraordinaria cantidad de funciones y métodos para cada objeto. También tiene gran variedad de funciones que actúan sobre DPs, haciendo que la personalización sea casi ilimitada.

Establecer alarmas para los DPs es realmente sencillo, basta con establecer cuáles son los límites en sus posibles valores y prioridades. La mayoría de los objetos de WinCC OA ya reaccionan al estado de alarma de un DP por lo que no se requiere demasiada configuración adicional. Además, incluye una ventana que muestra todas las alarmas y eventos activos.

Mantener un histórico de los valores también es sencillo de activar, basta con modificar individualmente los DPs o configurando un *Master DP*. Entonces cualquier objeto que muestre gráficas lee directamente todo los datos históricos, haciéndolos accesibles.

La gestión de usuarios ya no es tan simple ni se puede configurar de forma tan unificada como los elementos anteriores, requiere de especificar usuarios por un lado y configurar como se ejecuta la aplicación por otro pasando por establecer más drivers y ventanas de inicio de sesión.

Aunque estas impresiones pueden dar a pensar que finalmente se tiene preferencia por uno de los dos software, no es así. Sería injusto comparar EPICS junto a Phoebus con WinCC OA, tan a la ligera.

EPICS contiene absolutamente todas las ventajas e inconvenientes del software libre y de código abierto. Los inconvenientes se han hecho evidentes muy fácilmente:

- La complejidad, de cara a la implementación por la resolución de dependencias y la documentación heterogénea y mal relacionada.
- Falta de soporte formal, aunque tiene una comunidad debidamente activa éste es menos rápido y estructurado que el que puede proporcionar una empresa. No se ha comentado anteriormente, pero se contactó mediante el portal *GitHub* con *Ralph Lange* para pedir ayuda sobre el módulo OPC UA y se hizo evidente este hecho.
- Variedad de opciones en los módulos, esto es un arma de doble filo y la razón por la que se dice que no es justo comparar EPICS junto a Phoebus. Puede que Phoebus no haya sido la elección con más versatilidad y robustez, pero hasta que no se estudia en profundidad no se puede conocer sus carencias.
- Requerimiento de habitualidad en entorno EPICS y preferiblemente en Linux. Realmente no tiene por qué ser un inconveniente, ya que un experto informático puede resolver la implementación de este software en menos tiempo. Pero cierto es que se ha necesitado muchísimo menos tiempo con WinCC OA que con EPICS.
- Garantía, por desgracia uno de los puntos más graves de cara a una empresa es que el software no viene ligado a ninguna garantía. Los proveedores no se hacen cargo de los posibles fallos que puedan causar. Por lo que se debe invertir tiempo y dinero a parte para respaldar o evitar sus posibles fallos.

Puede que las ventajas no se hayan hecho tan visibles, pero realmente se tiene que:

- Es transparente, para un informático experimentado puede indagar en profundidad cómo funciona el programa. Así detectar posibles problemas, evitar posibles vulnerabilidades o crear mejores soluciones que se adapten más a sus necesidades.
- Seguridad, ya que la amplia comunidad de desarrolladores siempre buscar problemas en el código para solucionarlos con rapidez.
- Personalización, ligado a la transparencias y a la variedad de módulos. Algo que no se puede olvidar es que es completamente flexible en cuanto a su modificación e implantación, puede adaptarse fácilmente a cualquier requerimiento independientemente de su escala o complejidad.
- Extensión, a riesgo de verse redundante es importante mencionarlo. Si el software base juntos sus módulos se queda corto, se tiene la libertad de crear propias extensiones que añadan funcionalidad o faciliten el uso de éste.
- El costo, algo obvio en este tipo de software es la cantidad de ahorro que supone respecto a un software comercial.
- Longevidad, dado que el código es accesible y éste no depende de un único proveedor. Si el proyecto original se detiene, puede seguir desarrollándose por otros si lo desean. Ya ha ocurrido que en el mercado se haya cerrado el soporte para un software privativo y que la comunidad le tenga cierto aprecio o utilidad que le pidan a la compañía la liberación de código para proseguir el proyecto.

## 7.3 Tabla resumen

Descripción	EPICS con Phoebus	WinCC OA
<b>Documentación</b>	Dependiente del proveedor	Unificada y completa
<b>Instalación</b>	Se debe cumplir demasiadas dependencias	Un solo archivo ejecutable
<b>Creación de proyecto</b>	Script sencillo y rápido	Asistente lento, extenso y poco flexible
<b>Configuración drivers</b>	Mal documentado y los errores aportan poca ayuda	Mal documentado
<b>Creación de variables</b>	Asistente no incluido e interfaz anticuada	Asistente incluido
<b>Interfaz</b>	Muy intuitiva y ligera	Poco intuitiva
<b>Scripting</b>	Uso reducido y auxiliar	Integración total
<b>Complejidad alcanzable</b>	Simple	Compleja
<b>Alarms</b>	Herramienta externa con poca integración	Integrado y rapido de activar
<b>Históricos</b>	Herramienta externa con poca integración	Integrado y rapido de activar
<b>Usuarios</b>	No existe de manera nativa	Tedioso de configurar
<b>Aplicación final</b>	Phoebus es la propia herramienta de desarrollo y visualizador de aplicación final	Diversos métodos: propio PC, web...
<b>Soporte</b>	Depende de la comunidad	Formal
<b>Coste</b>	Software libre	Alto
<b>Escalabilidad</b>	Escalable	Escalable

Código de colores		
Punto a favor		Punto en contra
	Punto medio	

## 7.4 Curvas de esfuerzo

Las curvas de esfuerzo que se van a mostrar están basadas en el tiempo y esfuerzo requerido durante todas las fases del proyecto. Las impresiones personales tambien se tienen en cuenta, por lo que se ha aportado subjetividad a estas gráficas.

Cabe destacar que no se muestran valores numéricos para el eje vertical al haber introducido esta subjetividad. Lo que se trata de mostrar es la diferencia de esfuerzo y tiempo requerido para el desarrollo de cada etapa en cada programa.

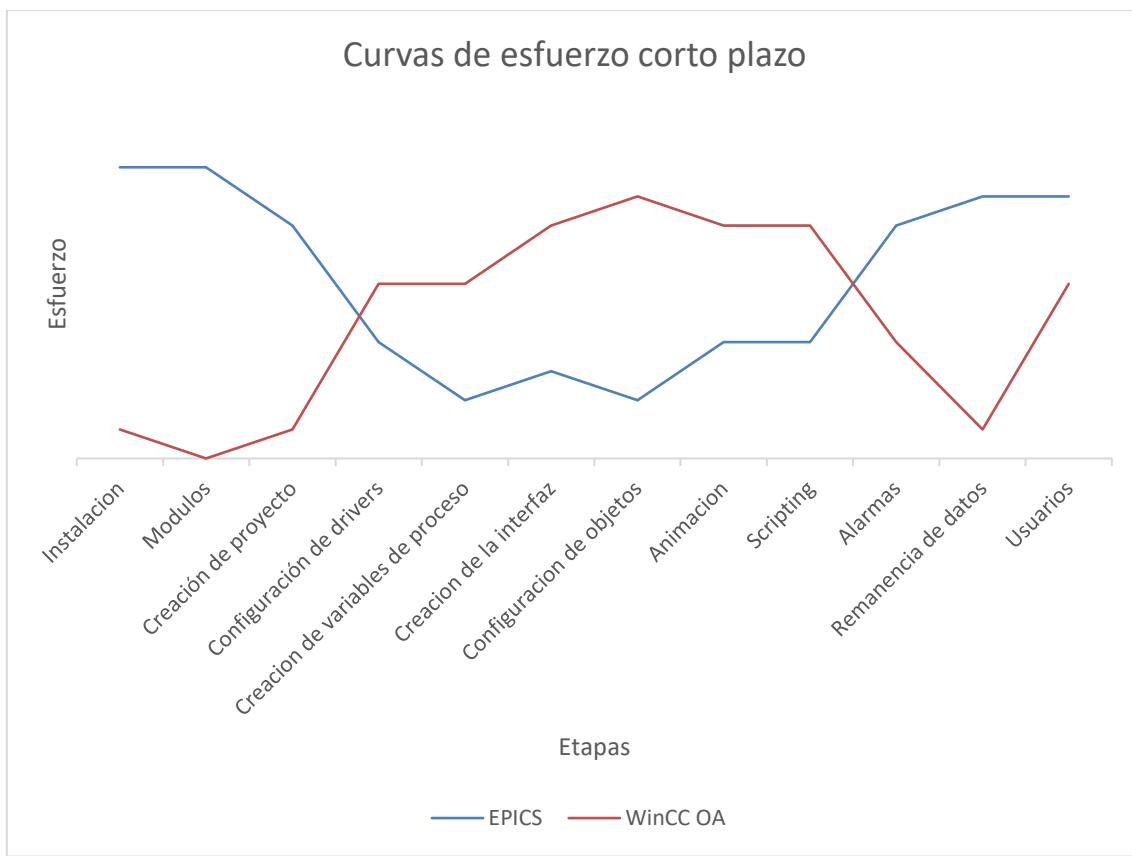


Ilustración 166 - Curvas de esfuerzo a corto plazo

# 8 Conclusiones y trabajos futuros

---

## 8.1 Conclusiones

Durante este proyecto se ha podido comprobar, tal como se ha indicado anteriormente, que no se puede comparar tal cual los dos frameworks de control analizados, ya que su concepción y características son significativamente diferentes.

El software base de EPICS es ampliamente usado en el sector de la investigación científica como en aceleradores de partículas, en éstos se invierte grandes cantidades de dinero y los experimentos son realmente costosos como para usar un software no fiable. Por lo que sería **insensato negarle su cabida en la industria**. Una vez habituado a la filosofía de funcionamiento, es una herramienta relativamente sencilla de implantar y muy potente. Tiene la versatilidad de que puede haber distintos IOCs en funcionamiento en una misma máquina, e incluso de estar repartidos por la red en distintas máquinas, pese a ello ser relativamente sencillo su acceso.

Algo negativo es la gestión de cada IOC, hablando de cara a los *records*, ya que a gran escala se hace impracticable. No obstante, se puede subdividir en IOCs más reducidos para facilitarlo o crear herramientas propias que ayuden a su gestión, lo que conlleva un incremento en el tiempo total de desarrollo.

Por otra parte, WinCC OA es una opción completamente factible, favorecedora y versátil para usar en la industria ya que es su principal objetivo, incluso tiene más áreas de acción. Con esta herramienta es irrefutable que el desarrollo de un SCADA se puede realizar en menos tiempo, más personalizada, con mayor complejidad y seguridad, con plenas garantías y certificados industriales. Además, la gestión de los *Data Points* es un elemento clave, ya que se pueden crear, eliminar o modificar en masa y lo hace especialmente útil para aplicaciones de gran escala.

Pero el ámbito monetario es algo que no se puede dejar de lado. El presupuesto que se ha consultado para hacer una estimación de las licencias para este proyecto es de un tamaño considerable, ya que incluye un sistema con redundancias, gran cantidad de datos, comunicaciones con diversos dispositivos, etcétera; ascendiendo el costo de licencias a *doscientos cincuenta mil euros*. Es una cantidad considerable, la cual no existe con EPICS ni si quiera ampliando las horas requeridas por los recursos humanos en mayor proporción.

Por tanto, en la primera comparación la opción que se elegiría es EPICS, si se tratara de un proyecto de pequeña a media-gran escala.

En cuanto a la creación de interfaces es cierto que hay una decantación clara, como desarrollador, por uno de los software: WinCC OA. Al principio ha costado habituarse a su entorno, quizás unas barreras de entrada ligeramente altas pero justificables. Una vez pasadas es realmente cómodo trabajar con este software, la imaginación no se ve limitada por casi nada. El desarrollo de la aplicación fue bastante ágil gracias a la gran cantidad de herramientas disponibles y la gran cantidad de configuraciones aplicables a cada objeto.

Aunque Phoebus es una herramienta versátil, no se puede profundizar a la escala que lo hace WinCC OA. Con Phoebus se puede crear aplicaciones, tanto de pequeña como gran escala, mostrar y manejar distintos datos sin demasiada complejidad, hacer interfaces profesionales y de calidad, pero no con mucha complejidad ni demasiada personalización. Ciertamente es usado junto con EPICS en

estas instalaciones científicas, pero en las imágenes que muestran se aprecian aplicaciones muy estáticas y simples (no confundir con escasas, de hecho, suelen estar recargadas de muchos indicadores booleanos en forma de matriz o distribuidos en forma de lo que se quiera retratar).

Por tanto, de cara a la creación de interfaces se puede concluir que WinCC OA es un claro ganador.

No obstante, como desarrollador de la aplicación me decantaría indudablemente por **EPICS** para proyectos de **pequeño o mediano tamaño**. Con esta escasa experiencia y teniendo buen manejo en entornos Linux, es más que suficiente para que sea una tarea fácil y rápida.

Sin embargo, si me pusieran al mando del desarrollo de SCADAS para un **gran proyecto**, y tuviera libertad de elección, me decantaría por **WinCC OA**. No solo por sus ventajas en la gestión masiva de datos y la facilidad en crear las interfaces, sino también por las garantías y certificados que este programa conlleva.

Finalmente, después de estos meses y a pesar de todos los contratiempos, considero que el aprendizaje de ambos frameworks de control ha sido una experiencia enriquecedora para mi formación, que además me ha capacitado para detectar cuándo es más adecuado utilizar uno u otro.

## 8.2 Líneas futuras de trabajo

Aunque las aplicaciones se han desarrollado de forma completa, un análisis más profundo de los frameworks de control podría incluir los siguientes aspectos:

- Programación a bajo nivel de módulos y drivers para EPICS, managers para WinCC OA. Ya que ambos programas permiten que desarrollen de terceros, una buena práctica sería descubrir los requerimientos de cada uno para implementar estos extras y ver las dificultades en cada uno.
- Algo ciertamente interesante es investigar la posibilidad de la creación de las ventanas autodimensionables y comprobar esta funcionalidad en ambos programas.
- Abrir las aplicaciones a una red abierta. Es decir, todo esto se ha realizado pensando en una red local cerrada y segura, pero ambos software permiten que se puedan acceder a ellos desde redes externas.
- Seguridad en las comunicaciones. Relacionado con lo anterior, al abrir una comunicación a una red abierta lo más sensato es encriptar y autenticar las comunicaciones.
- Control de accesos. Ambos programas permiten bloquear ciertas características o accesos a sus configuraciones, pues profundizar en ello para una aplicación real es interesante.
- Podría tratarse de comparar los distintos módulos disponibles para creación de interfaces, aunque su extensión daría para un proyecto entero, sería interesante saber cual de ellos es que mayores herramientas ofrece.

## 9 Incidencias

---

En este apartado se van a recoger las incidencias a las que se han enfrentado durante el desarrollo de este proyecto. Es importante darlas a conocer ya que han consumido un tiempo razonable, además de poder servir de ayuda a quien quiera inmiserirse en estos programas en un futuro.

Durante las pruebas, en la fase inicial, se estuvo estudiando el uso de la librería s7nodave la cual se comunica directamente con el PLC sin ningún intermediario o configuración adicional (más que permitir en el autómata la lectura de datos). Este driver requiere además del *Asyn Driver*, que es una librería genérica para comunicaciones asíncronas y sirve a muchos módulos. El problema llega en la compilación, donde solo permite su instalación en sistemas basados en Linux y no en todos. Primeramente, se intentó en la última versión de Ubuntu, la 22.02, pero los errores que mostraba eran ciertamente incongruentes como que no encontraba sus propios códigos fuente. Después de un tiempo tratando de solucionarlos y probando otras versiones del driver, los errores que mostraba estaban relacionados librerías anticuadas. Pues se comenzó a probar en versiones inferiores de Ubuntu, como la 20 o la 19 sin éxito. Solo funcionó en la 18.

Una vez instalado tanto Asyn como s7nodave, el siguiente problema que se encontró fue que no se establecía conexión con el autómata. Tras muchos intentos se dedujo que el problema residía en las interconexiones entre el autómata con la máquina virtual, por lo que esta vía de comunicación quedó apartada.

La anterior librería no solo quedó descartada por esos problemas, sino que se tenía una mejor opción como el módulo OPC UA, el cual no queda exento de incidencias. Entre sus dependencias se haya la necesidad de usar el cliente SDK de *Unified Automation*, donde hubo problemas con que *cmake* de base no encontraba *openssl* o *libxml2*. La solución a esto queda recogida en el proceso de instalación comentado en apartados anteriores.

Una vez instalado OPC UA, hay que configurar el IOC para que pueda acceder a este módulo y usarlo. En este punto es donde se necesita una descripción detallada de cómo hacerlo, pero la documentación queda escasa por presuponer que se tiene habilidad con todo el entorno EPICS. Esto llevo un tiempo hasta conseguirlo.

Hasta aquí se han comentado las incidencias con el entorno EPICS, pero tambien ha habido con WinCC OA. A la hora de configurar la comunicación mediante OPC UA es donde se hace presente la ya comentada mal enlazada documentación. Por un lado, se explica detalladamente como configurar el cliente, pero para que ese cliente funcione se debe añadir un manager adicional. Ese hecho no está bien detallado y se tardó un tiempo razonable en encontrar en la documentación el apartado que indica como incluirlo. Además, al manager hay que asignarle un número que no es arbitrario y en la documentación no lo explica adecuadamente. Se tuvo que recurrir a la ayuda de otro estudiante que estaba usando esta misma aplicación para su trabajo fin de master.



# Bibliografía

---

- Bee, L. (2022). *PLC and HMI development with Siemens TIA Portal : develop PLC and HMI programs using standard methods and structured approaches with TIA Portal V17*. Birmingham, UK: Packt Publishing.
- Berger, H. (2016). *Automating with SIMATIC: Hardware and Software, Configuration and Programming, Data Communication, Operator Control and Process Monitoriong*. Publicis Publishing.
- Bishop, B. (13 de Diciembre de 2022). *Lawrence Livermore National Laboratory achieves fusion ignition*. Obtenido de <https://www.llnl.gov/archive/news/lawrence-livermore-national-laboratory-achieves-fusion-ignition>
- Cappelli, M., Bagnasco, A., Diaz, J., Sousa, J., Ambi, F., Campedrer, A., . . . Ibarra, A. (Septiembre de 2021). Status of the engineering design of the IFMIF-DONES Central Instrumentation and Control Systems. *Fusion Engineering and Desing*, Volumen 170. Obtenido de <https://www.sciencedirect.com/science/article/abs/pii/S0920379621004506>
- Ciemat. (s.f.). *IFMIF-DONES*. Obtenido de <http://www.fusion.ciemat.es/dones-2/>
- E.Mandado, J.Marcos, C.Fernández, & I.Armesto. (2018). *Sistemas de automatización y autómatas programables*. Marcombo S.A.
- EPICS Control. (2019). *EPICS Documentation*. Obtenido de Overview: [https://docs.epics-controls.org/en/latest/guides/EPICS\\_Intro.html](https://docs.epics-controls.org/en/latest/guides/EPICS_Intro.html)
- EPICS Control. (2019). *EPICS Extensions*. Obtenido de <https://epics-controls.org/resources-and-support/extensions/>
- EPICS Control Web. (s.f.). *Home*. Obtenido de <https://epics-controls.org/>
- EPICS Controls. (s.f.). *Guia para desarrolladores IOCs*.
- ETM Professional Control GmbH. (2023). *SIMATIC WinCC Open Architecture Documentation*. Obtenido de [https://www.winccoa.com/documentation/WinCCOA/latest/en\\_US/index.html](https://www.winccoa.com/documentation/WinCCOA/latest/en_US/index.html)
- FactoryIO - Docs. (s.f.). Obtenido de <https://docs.factoryio.com/>
- Guerrero, J. (2019). *Programación Estructurada de Autómatas Programables con GRAFCET*. Paraninfo.
- IFMIF-DONES-España. (2022). *Programa DONES*. Obtenido de <https://ifmif-dones.es/es/programa-dones/>
- Indeed. (s.f.). *Sueldo medio en España para Ingeniero en sistemas*. Obtenido de <https://es.indeed.com/career/ingeniero-en-sistemas/salaries>
- ITER Organization. (s.f.). *ITER Page*. Obtenido de <https://www.iter.org/>
- ITER Organization. (s.f.). *The ITER Tokamak*. Obtenido de <https://www.iter.org/mach>
- J.A.Mercado. (2019). *Sistemas programables avanzados*. Paraninfo.

KepWare. (s.f.). *KepServer Info*. Obtenido de <https://www.kepserverexopc.com/kepware-kepserverex-features/>

L.Peciña. (2019). *Programación de controladores avanzados SIMATIC S7 1500 con TIA Portal, AWL/KOP y SCL*. Marcombo S.A.

Marsching, S. (2013). *s7nodave Documentation*. Obtenido de <https://oss.aquenos.com/epics/s7nodave/>

OPC-Foundation. (s.f.). *OPC Unified Architecture*. Obtenido de <https://opcfoundation.org/about/opc-technologies/opc-ua/>

OPC-Foundation. (s.f.). *What is OPC*. Obtenido de <https://opcfoundation.org/about/what-is-opc/>

Peciña, L. (2018). *Comunicaciones industriales y WinCC*. Marcombo S.A.

SIMATIC WinCC OA. (2023). *What is WinCC OA*. Obtenido de [https://www.winccoa.com/documentation/WinCCOA/3.18/en\\_US/GettingStarted/GettingStarted-02.html](https://www.winccoa.com/documentation/WinCCOA/3.18/en_US/GettingStarted/GettingStarted-02.html)

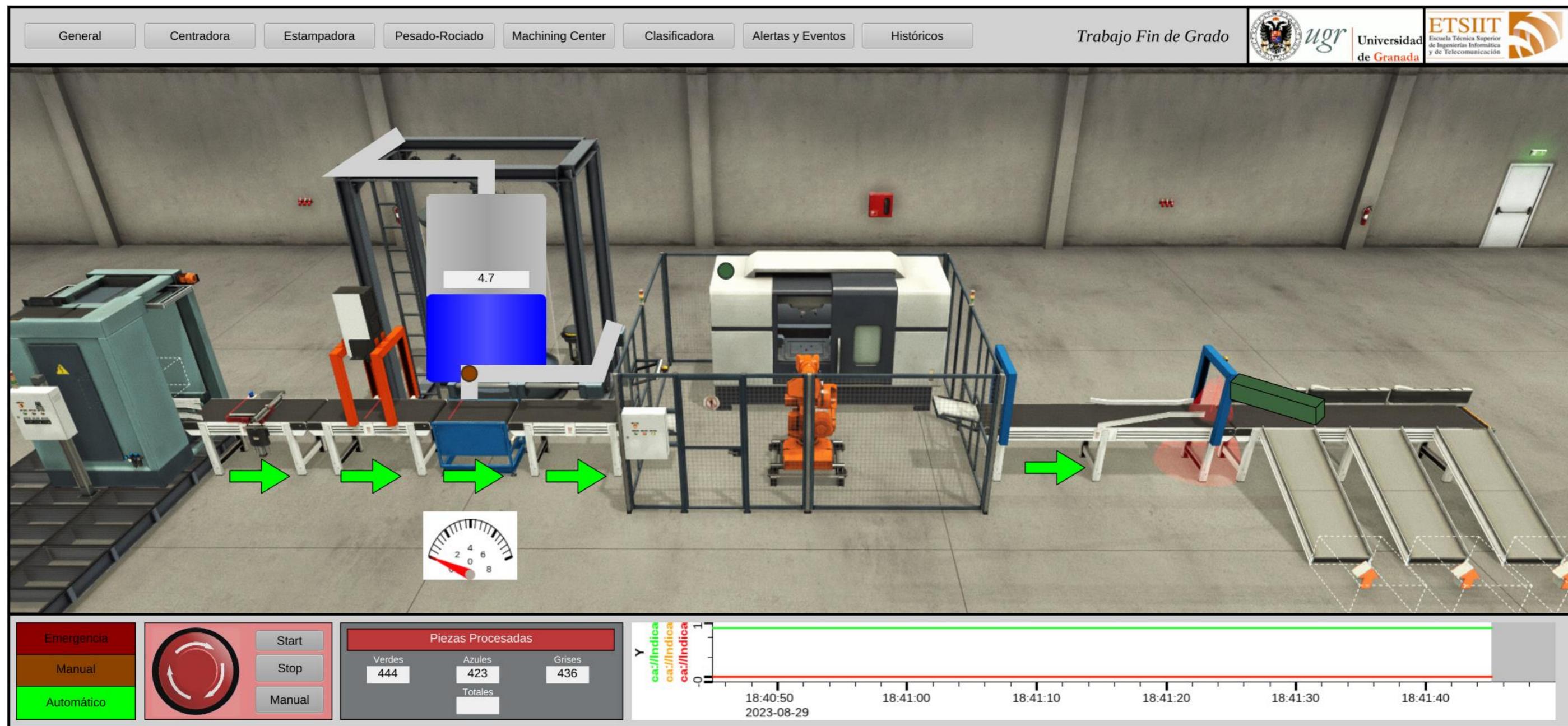
Valenzuela, E., Cano-Delgado, A., Cruz-Miranda, J., Rouret, M., Miccichè, G., Ros, E., ... Diaz, J. (2022). *The IFMIF-DONES remote handling control system: Experimental setup for OPC UA integration*. Obtenido de <https://www.sciencedirect.com/science/article/pii/S0920379623003587?via%3Dihub>

## Anexos

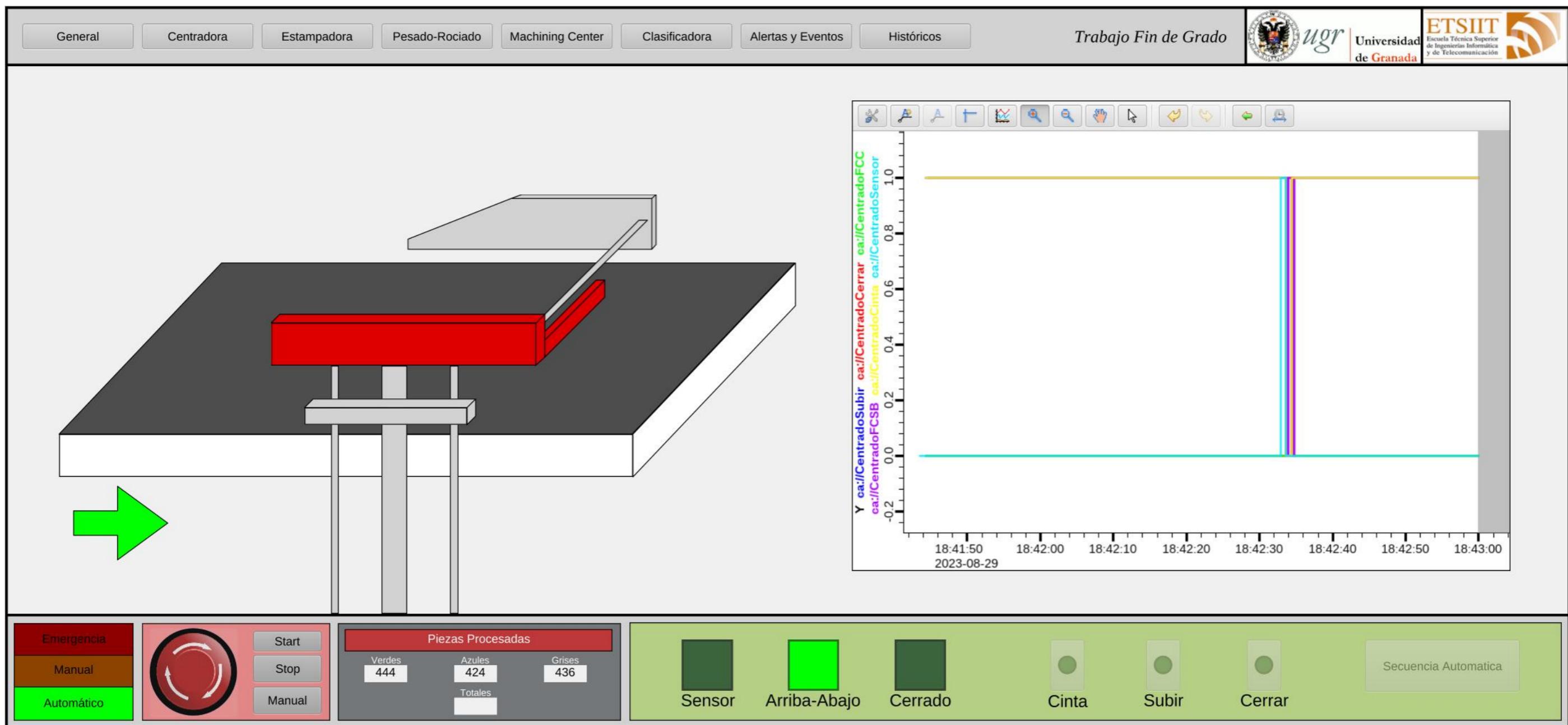
### Paneles finales

EPICS

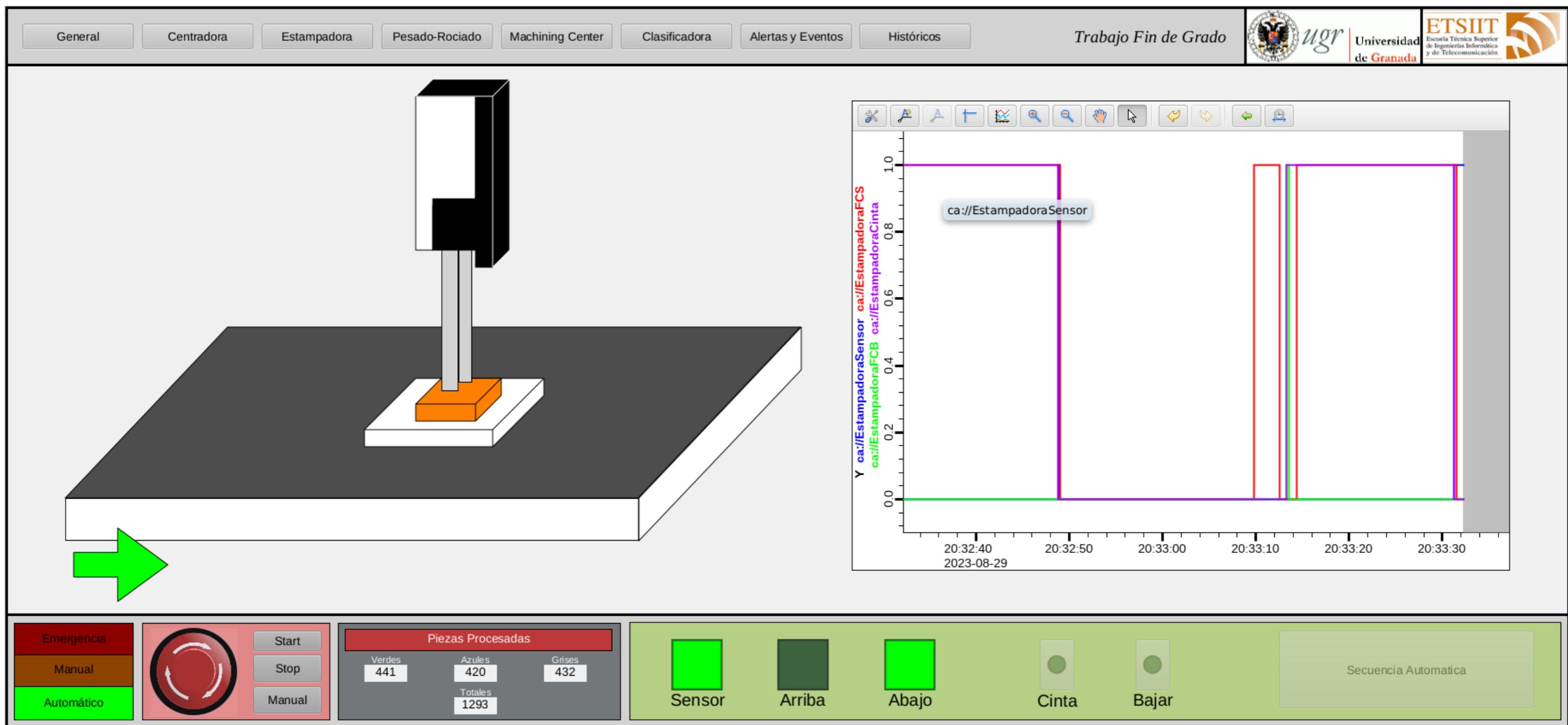
Main



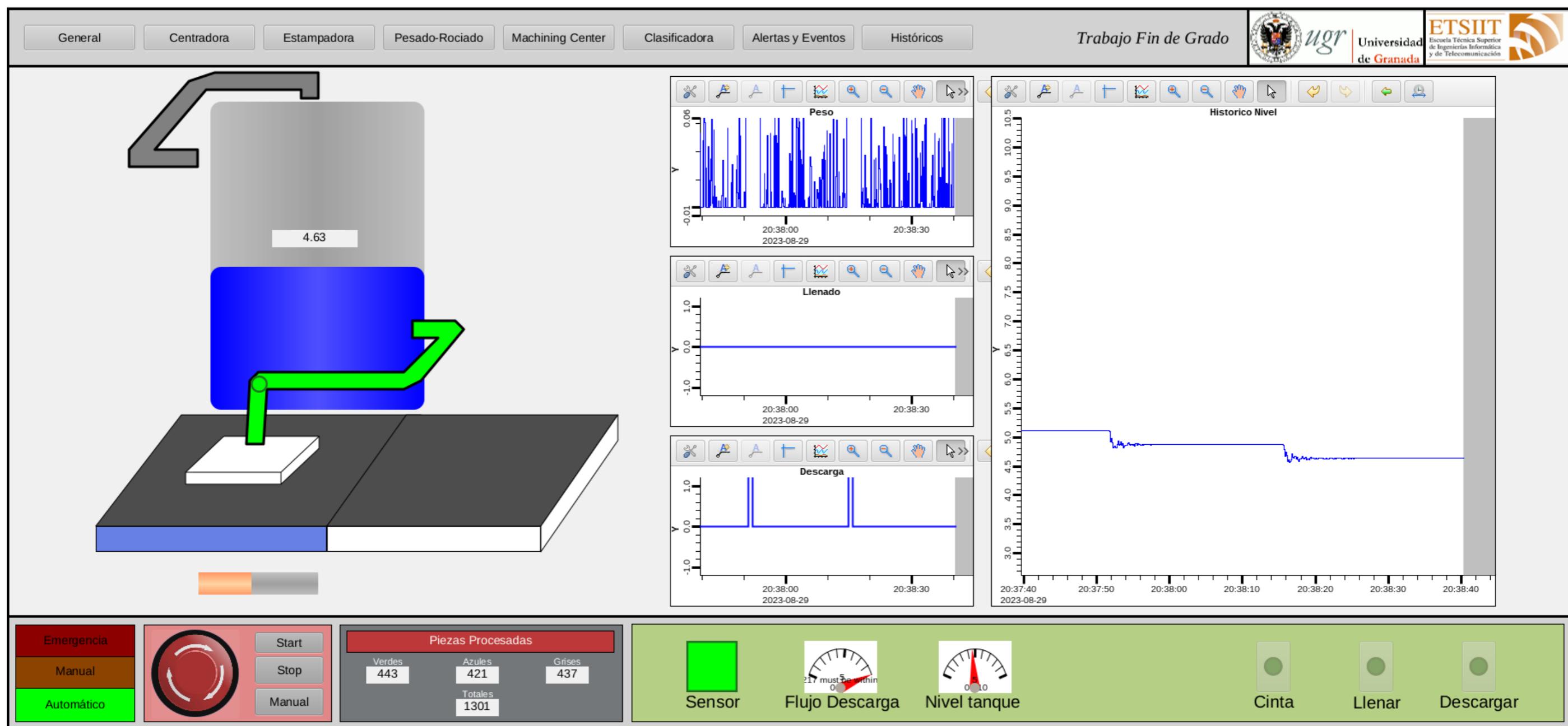
## Centrador



## Estampadora



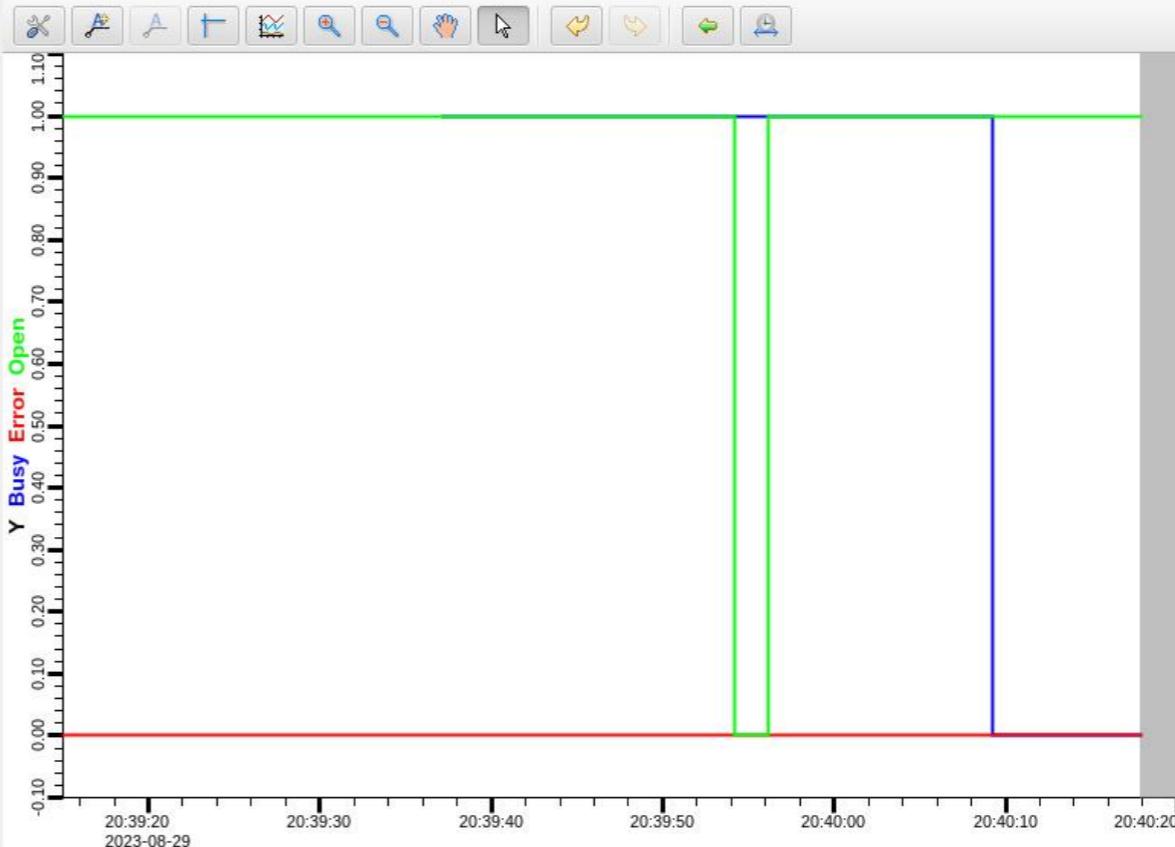
## Pesado Rociado



## Machining Center

General Centradora Estampadora Pesado-Rociado Machining Center Clasificadora Alertas y Eventos Históricos Trabajo Fin de Grado

UGR ETSIIT Universidad de Granada



Piezas Procesadas		
Verdes	Azules	Grises
445	421	437
Totales		
1303		

Emergency  
Manual  
Automático

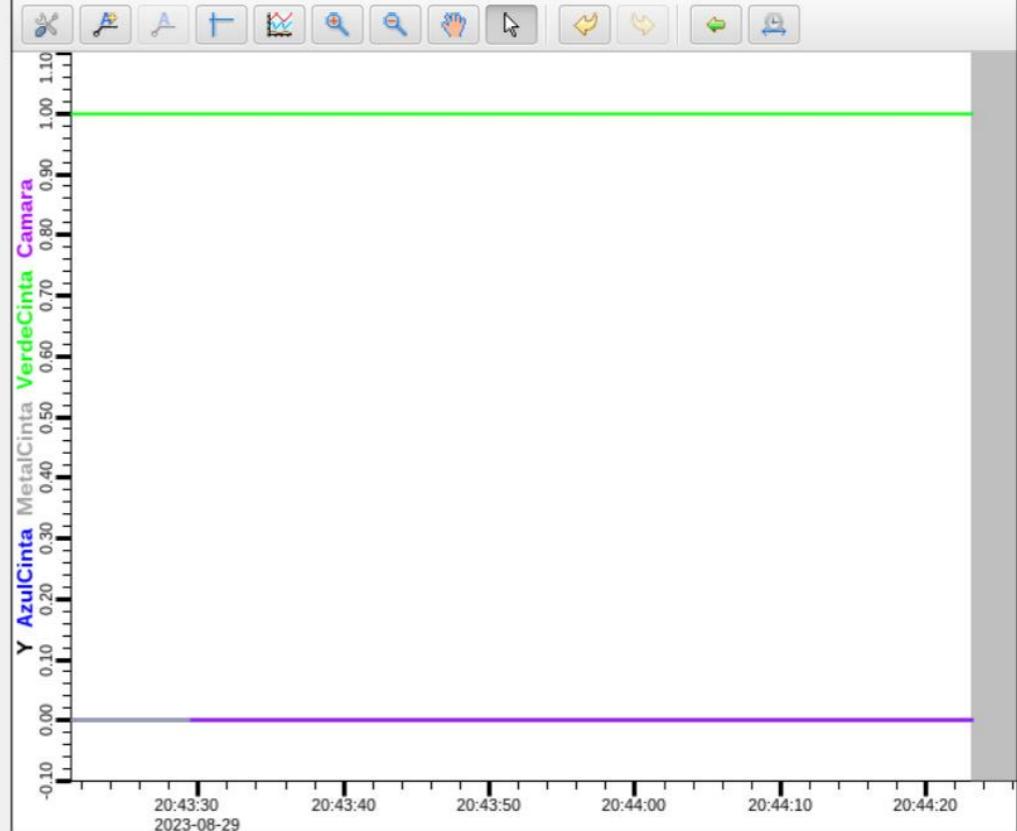
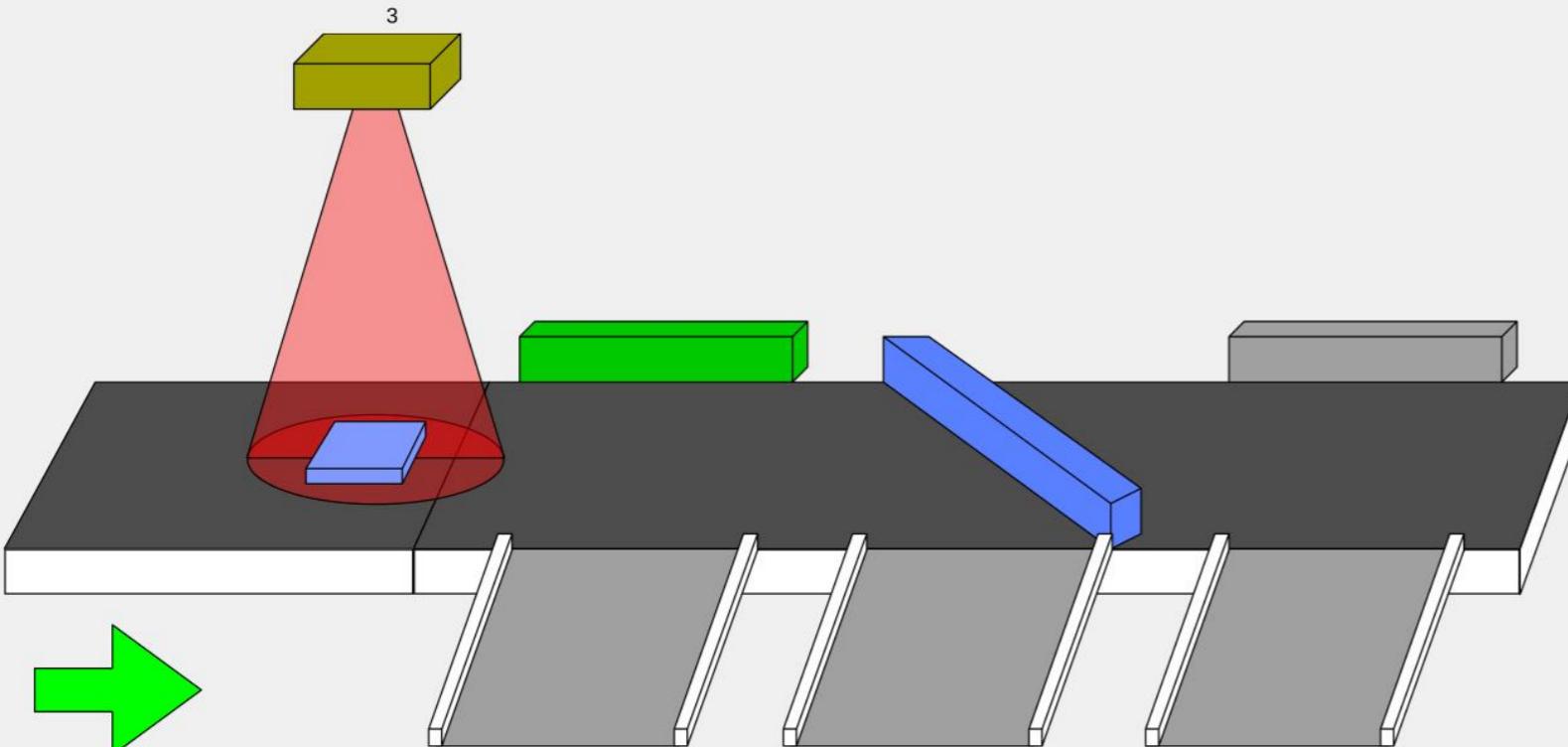
Start  
Stop  
Manual

Ocupado Abierto Error

Progreso delineado

## Clasificadora

General Centrador Estampadora Pesado-Rociado Machining Center Clasificadora Alertas y Eventos Históricos Trabajo Fin de Grado  



Emergencia  
Manual  
Automático

Start  
Stop  
Manual

Piezas Procesadas

Verdes	Azules	Grises
448	423	438
Totales		
1309		

Ultima pieza

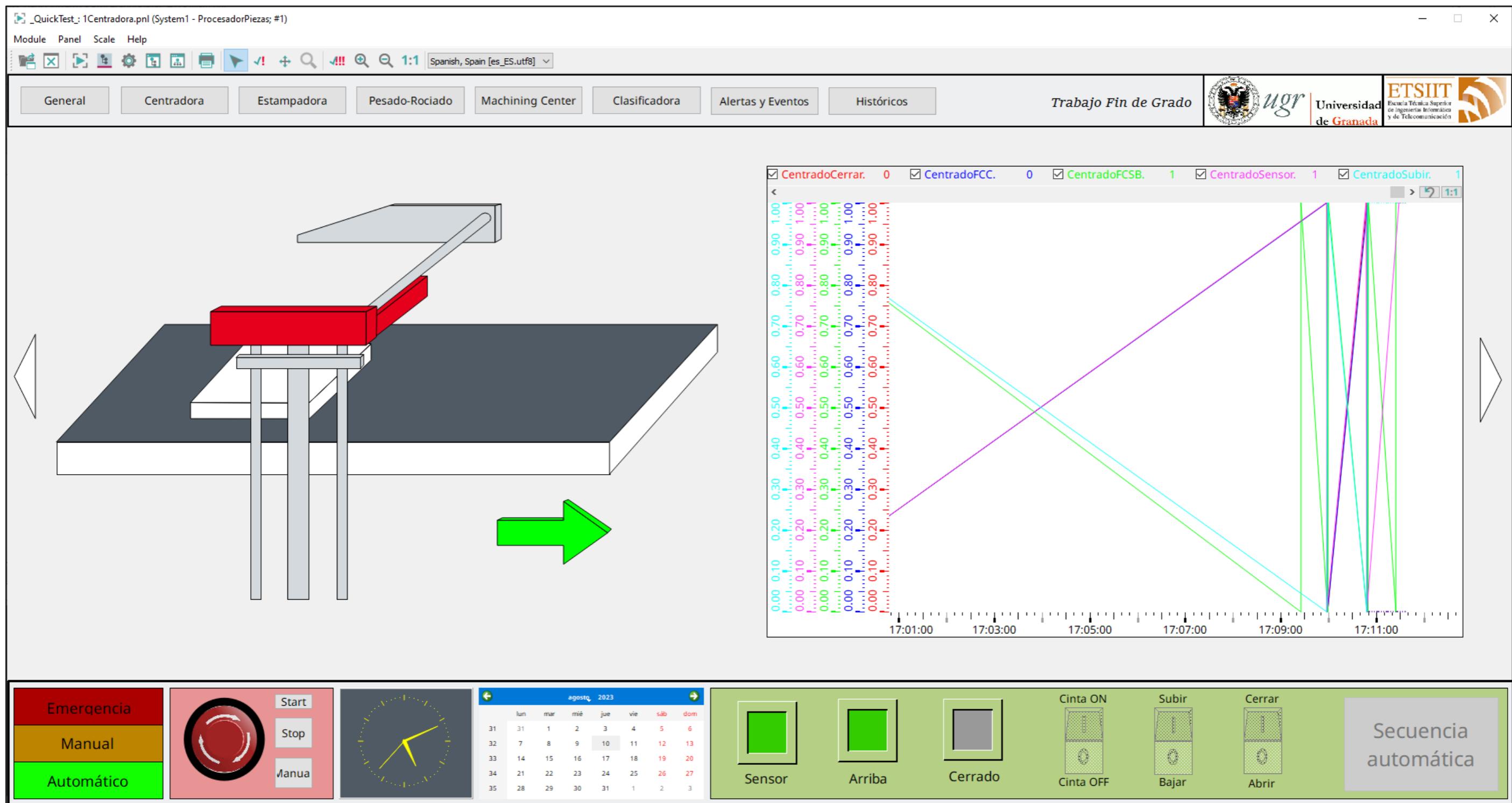
Verdes Azules Grises Cintas

## WinCC OA

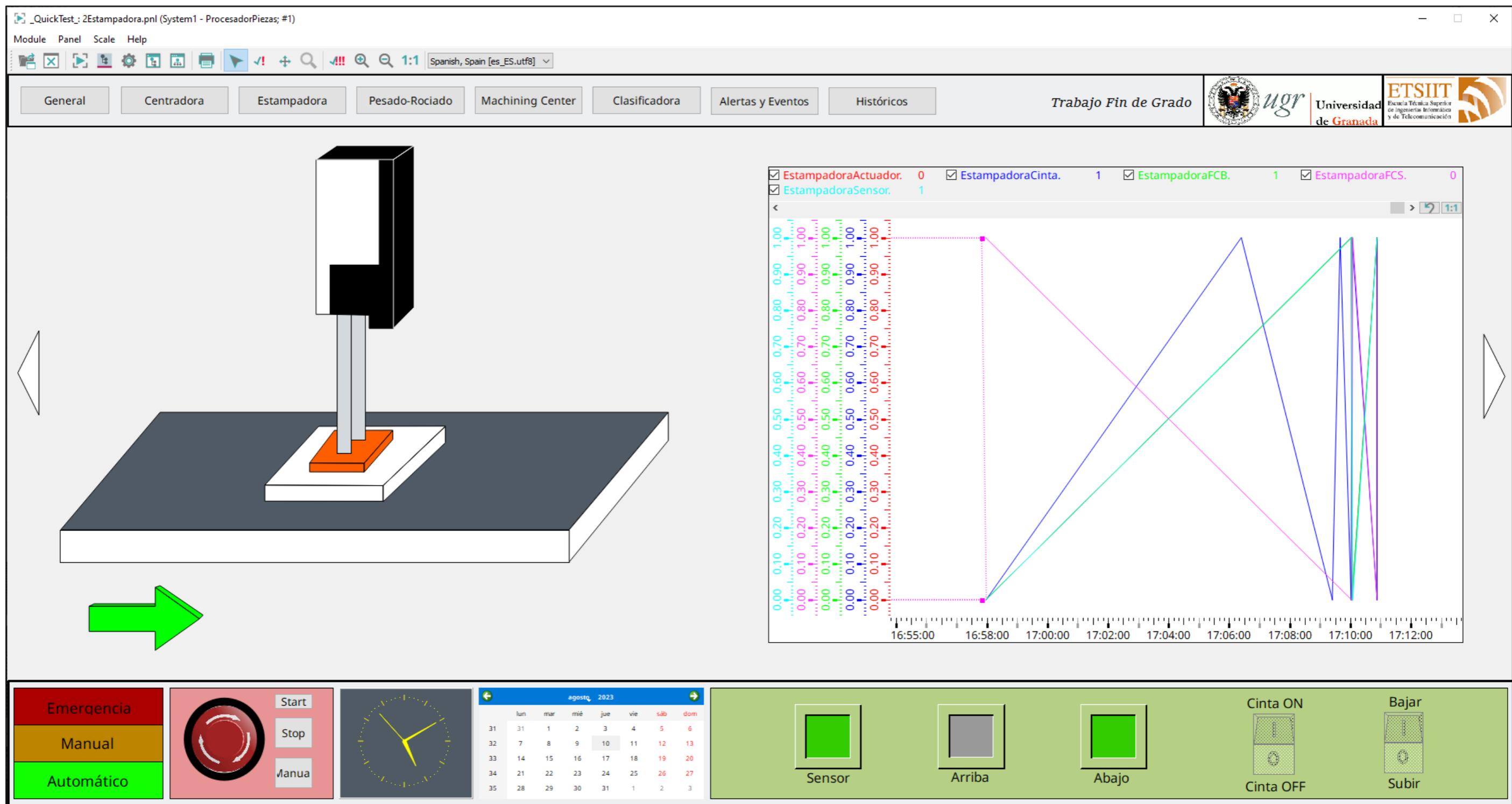
### Main



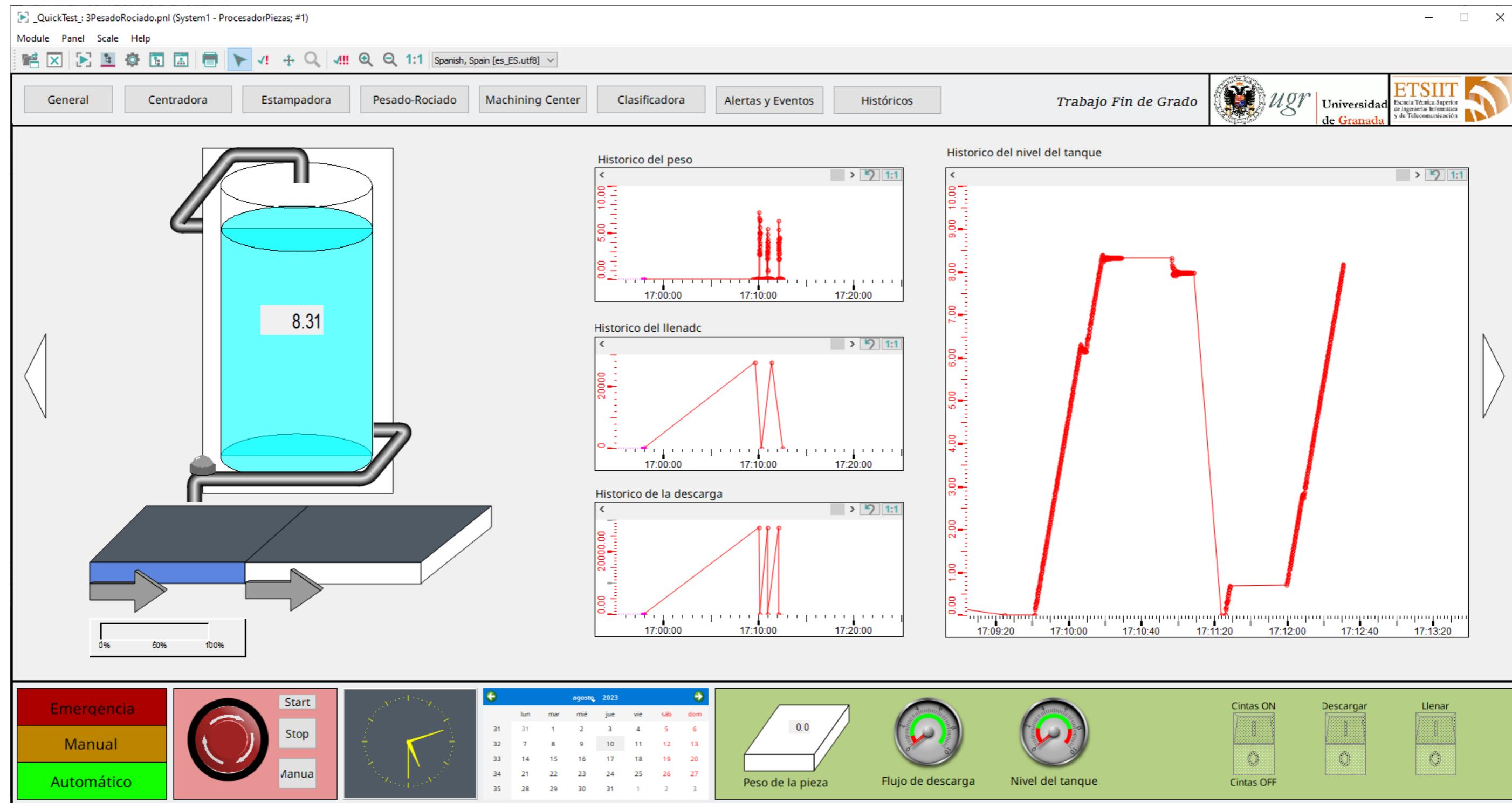
## Centrador



## Estampadora



## Pesado Rociado



## Machining Center

QuickTest\_4MachiningCenter.pnl (System1 - ProcesadorPiezas; #1)

Module Panel Scale Help

Spanish, Spain [es\_ES.utf8] 1:1

General Centradora Estampadora Pesado-Rociado Machining Center Clasificadora Alertas y Eventos Históricos Trabajo Fin de Grado

ETSIIT UGR Universidad de Granada

MachiningCenterError. 0  MachiningCenterBusy. 1

MachiningCenterOpen. 1

17:04:00 17:06:00 17:08:00 17:10:00 17:12:00

Emergency  
Manual  
Automático

Start  
Stop  
Manua

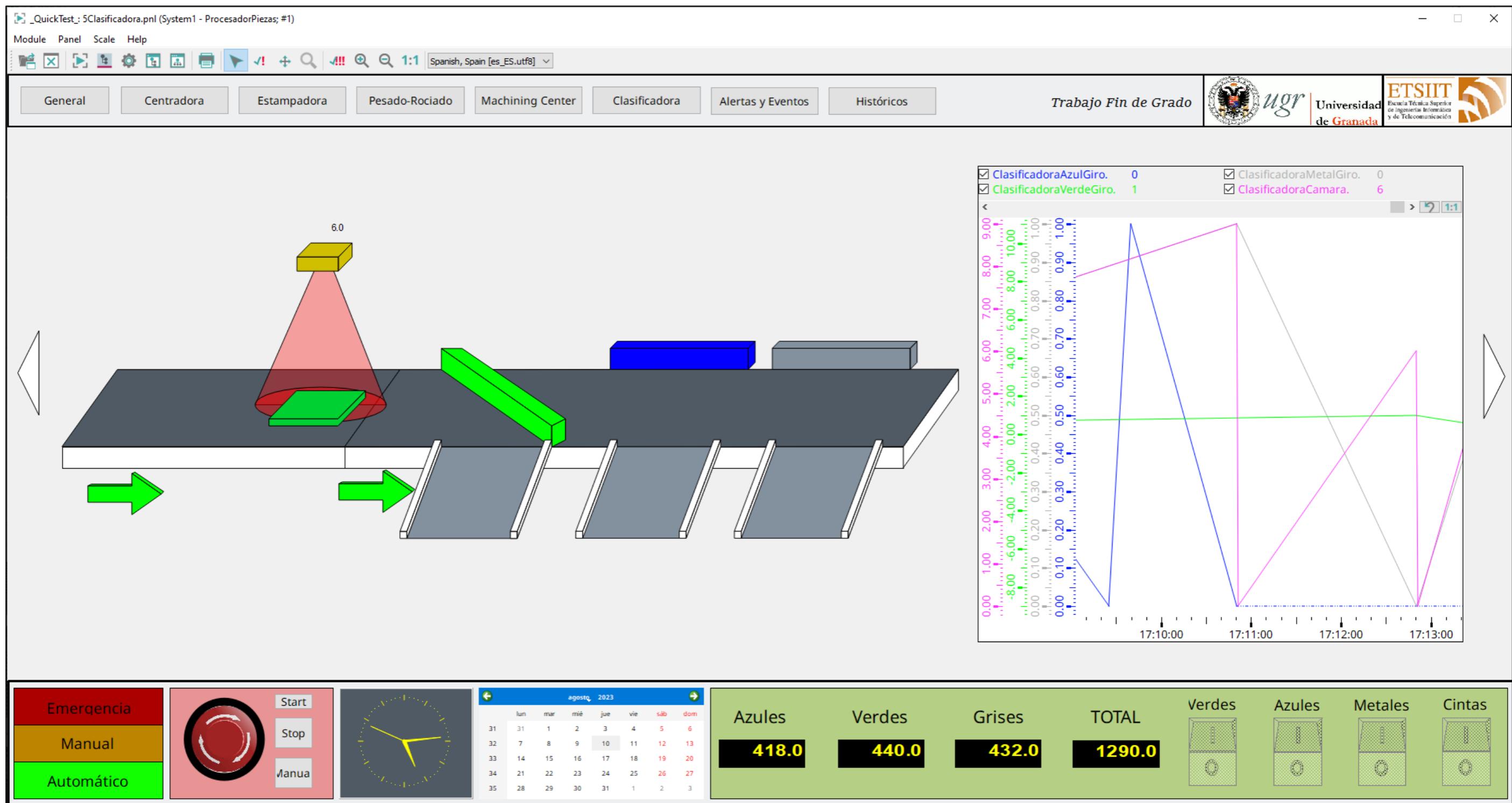
Ocupado  
Abierto  
Error

agosto 2023

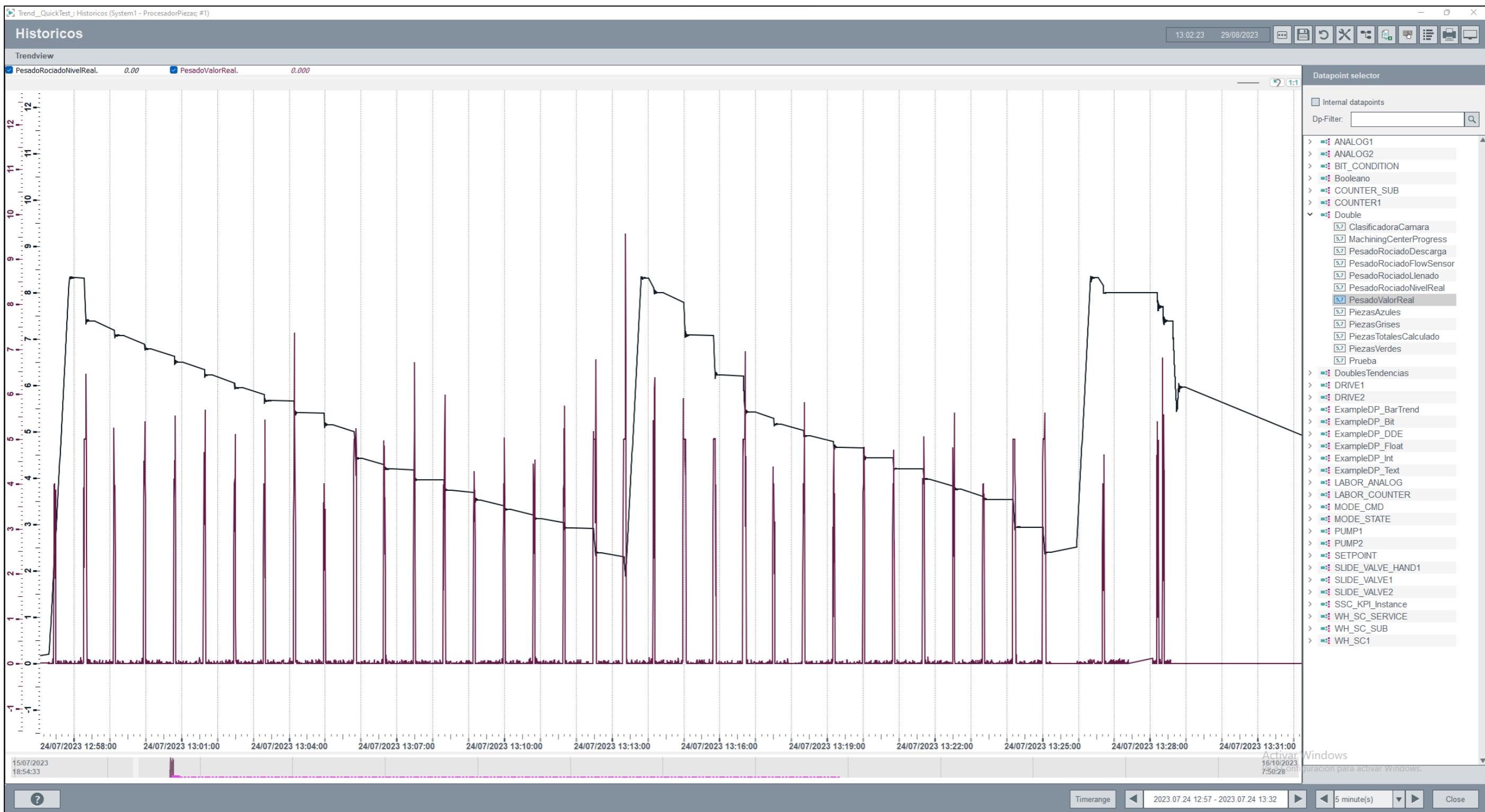
lun	mar	mié	jue	vie	sáb	dom
31	31	1	2	3	4	5
32	7	8	9	10	11	12
33	14	15	16	17	18	19
34	21	22	23	24	25	26
35	28	29	30	31	1	2

0% 50% 100%

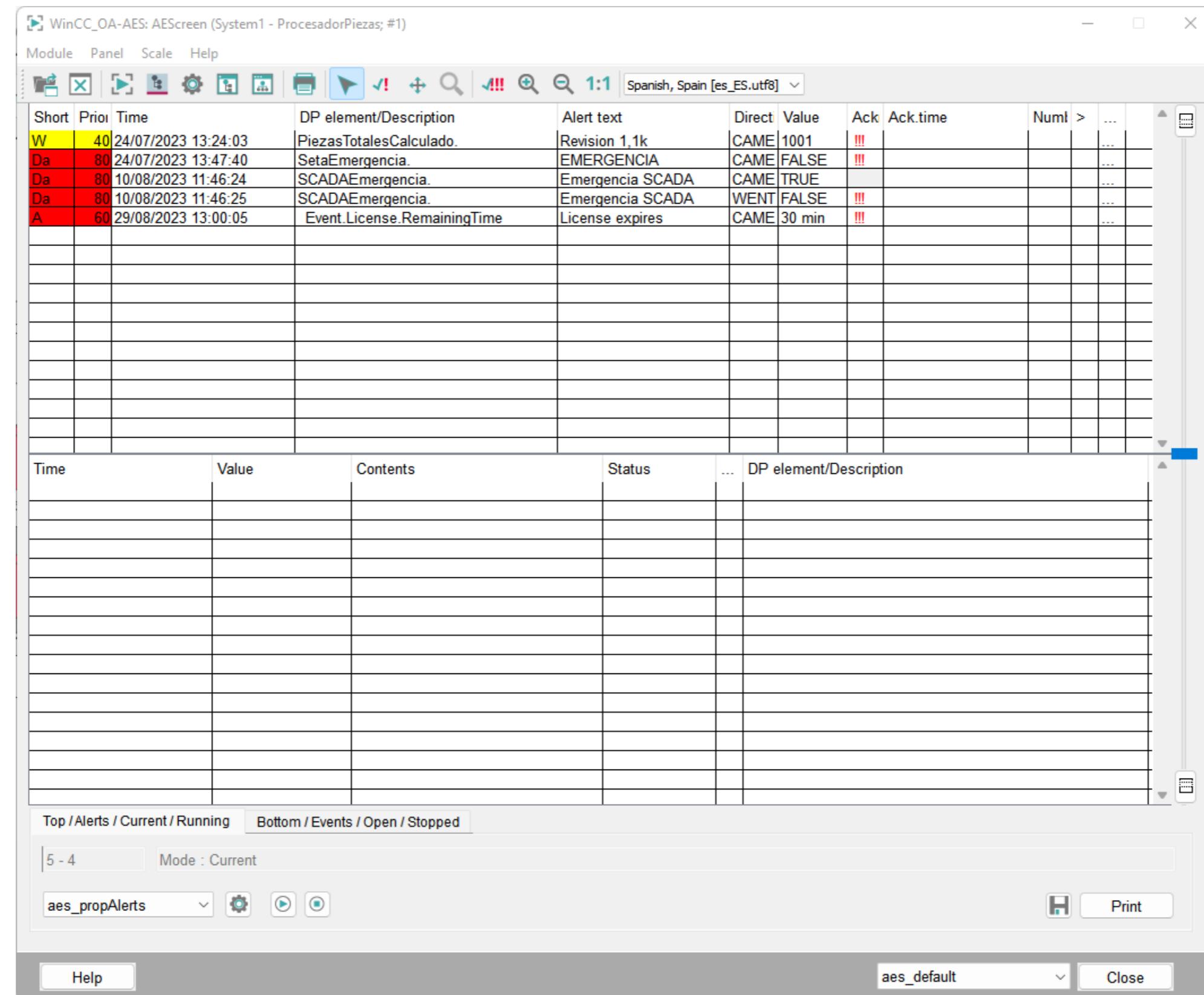
## Clasificadora



## Históricos



## *Alarmas*



## Records

En el presente anexo se muestran los records tal cual quedaron en el apartado [5.9](#), es decir, no contiene las configuraciones de alarmas ni archivado.

```

record(bi, "BotonAuto") {
    field(DTYP, "OPCUA")
    field(INP, "@SUB1 ns=2;s=ProcesadorPiezas.s7300.BotonAuto")
    field(SCAN, "I/O Intr")
}

record(bi, "BotonManual") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.BotonManual"
    field(SCAN, "I/O Intr")
}

record(bi, "BotonReset") {
    field(DTYP, "OPCUA")
    field(INP, "@SUB1 ns=2;s=ProcesadorPiezas.s7300.BotonReset")
    field(SCAN, "I/O Intr")
}

record(bi, "BotonStart") {
    field(DTYP, "OPCUA")
    field(INP, "@SUB1 ns=2;s=ProcesadorPiezas.s7300.BotonStart")
    field(SCAN, "I/O Intr")
}

record(bi, "BotonStop") {
    field(DTYP, "OPCUA")
    field(INP, "@SUB1 ns=2;s=ProcesadorPiezas.s7300.BotonStop")
    field(SCAN, "I/O Intr")
}

record(bi, "CentradoCerrar") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.CentradoCerrar"
    field(SCAN, "I/O Intr")
}

record(bi, "CentradoCinta") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.CentradoCinta"
    field(SCAN, "I/O Intr")
}

record(bi, "CentradoFCC") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.CentradoFCC"
    field(SCAN, "I/O Intr")
}

record(bi, "CentradoFCSB") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.CentradoFCSB"
    field(SCAN, "I/O Intr")
}

record(bi, "CentradoSensor") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.CentradoSensor"
    field(SCAN, "I/O Intr")
}

record(bi, "CentradoSubir") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.CentradoSubir"
    field(SCAN, "I/O Intr")
}

record(bi, "ClasificadoraAzulCinta") {
    field(DTYP, "OPCUA")
}

field(INP,
    ns=2;s=ProcesadorPiezas.s7300.ClasificadoraAzulCinta")
field(SCAN, "I/O Intr")
}

record(bi, "ClasificadoraAzulGiro") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.ClasificadoraAzulGiro"
    field(SCAN, "I/O Intr")
}

record(bi, "ClasificadoraCinta") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.ClasificadoraCinta"
    field(SCAN, "I/O Intr")
}

record(bi, "ClasificadoraCinta1") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.ClasificadoraCinta1"
    field(SCAN, "I/O Intr")
}

record(bi, "ClasificadoraMetalCinta") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.ClasificadoraMetalCinta"
    field(SCAN, "I/O Intr")
}

record(bi, "ClasificadoraMetalGiro") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.ClasificadoraMetalGiro"
    field(SCAN, "I/O Intr")
}

record(bi, "ClasificadoraVerdeCinta") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.ClasificadoraVerdeCinta"
    field(SCAN, "I/O Intr")
}

record(bi, "ClasificadoraVerdeGiro") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.ClasificadoraVerdeGiro"
    field(SCAN, "I/O Intr")
}

record(bi, "EstampadoraActuador") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.EstampadoraActuador"
    field(SCAN, "I/O Intr")
}

record(bi, "EstampadoraCinta") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.EstampadoraCinta")
    field(SCAN, "I/O Intr")
}

record(bi, "EstampadoraFCB") {
    field(DTYP, "OPCUA")
    field(INP,
        ns=2;s=ProcesadorPiezas.s7300.EstampadoraFCB")
    field(SCAN, "I/O Intr")
}

```

```

record(bi, "EstampadoraFCS") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.EstampadoraFCS")
    field(SCAN, "I/O Intr")
}

record(bi, "EstampadoraSensor") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.EstampadoraSensor")
    field(SCAN, "I/O Intr")
}

record(bi, "IndicadorAuto") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.IndicadorAuto")
    field(SCAN, "I/O Intr")
}

record(bi, "IndicadorEmergencia") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.IndicadorEmergencia")
    field(SCAN, "I/O Intr")
}

record(bi, "IndicadorManual") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.IndicadorManual")
    field(SCAN, "I/O Intr")
}

record(bi, "MachiningCenterBusy") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.MachiningCenterBusy")
    field(SCAN, "I/O Intr")
}

record(bi, "MachiningCenterError") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.MachiningCenterError")
    field(SCAN, "I/O Intr")
}

record(bi, "MachiningCenterOpen") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.MachiningCenterOpen")
    field(SCAN, "I/O Intr")
}

record(bi, "PesadoCinta") {
    field(DTYP, "OPCUA")
    field(INP, "@SUB1 ns=2;s=ProcesadorPiezas.s7300.PesadoCinta")
    field(SCAN, "I/O Intr")
}

record(bi, "PesadoCinta2") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.PesadoCinta2")
    field(SCAN, "I/O Intr")
}

record(bi, "PesadoRociadoDescargaLed") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoDescargaLed")
    field(SCAN, "I/O Intr")
}

record(bi, "PesadoSensor") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.PesadoSensor")
    field(SCAN, "I/O Intr")
}

record(bi, "SetaEmergencia") {
    field(DTYP, "OPCUA")
    field(INP,
    ns=2;s=ProcesadorPiezas.s7300.SetaEmergencia")
    field(SCAN, "I/O Intr")
}

record(bo, "CMActivado") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMActivado")
}

record(bo, "CMCentradoCerrar") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMCentradoCerrar")
}

record(bo, "CMCentradoCinta") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMCentradoCinta")
}

record(bo, "CMCentradoSubir") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMCentradoSubir")
}

record(bo, "CMClasificadoraAzulCinta") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMClasificadoraAzulCinta")
}

record(bo, "CMClasificadoraCinta") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMClasificadoraCinta")
}

record(bo, "CMClasificadoraMetalCinta") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMClasificadoraMetalCinta")
}

record(bo, "CMClasificadoraVerdeCinta") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMClasificadoraVerdeCinta")
}

record(bo, "CMEstampadoraActuador") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMEstampadoraActuador")
}

record(bo, "CMEstampadoraCinta") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMEstampadoraCinta")
}

record(bo, "CMPesadoCinta") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMPesadoCinta")
}

record(bo, "CMPesadoRociadoDescarga") {
    field(DTYP, "OPCUA")
    field(OUT,
    ns=2;s=ProcesadorPiezas.s7300.CMPesadoRociadoDescarga")
}

record(bo, "CMPesadoRociadoLlenado") {
    field(DTYP, "OPCUA")
}

```

```

        field(OUT,
ns=2;s=ProcesadorPiezas.s7300.CMPesadoRociadoLlenado")
    }

record(bo, "SCADEEmergencia" {
    field(DTYP, "OPCUA")
    field(OUT,
ns=2;s=ProcesadorPiezas.s7300.SCADAEmergencia")
}

record(bo, "SCADAManual" {
    field(DTYP, "OPCUA")
    field(OUT,
ns=2;s=ProcesadorPiezas.s7300.SCADAManual")
}

record(bo, "SCADAStart" {
    field(DTYP, "OPCUA")
    field(OUT,
ns=2;s=ProcesadorPiezas.s7300.SCADAStart")
}

record(bo, "SCADASTop" {
    field(DTYP, "OPCUA")
    field(OUT, "@SUB1 ns=2;s=ProcesadorPiezas.s7300.SCADASTop")
}

record(mbbi, "PesadoRociadoFlowSensor" {
    field(DTYP, "OPCUA")
    field(INP,
ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoFlowSensor")
    field(SCAN, "I/O Intr")
}

record(mbbi, "PesadoRociadoDescarga" {
    field(DTYP, "OPCUA")
    field(INP,
ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoDescarga")
    field(SCAN, "I/O Intr")
}

record(ai, "PesadoValorReal" {
    field(DTYP, "OPCUA")
    field(INP,
ns=2;s=ProcesadorPiezas.s7300.PesadoValorReal")
    field(SCAN, "I/O Intr")
}

    "@SUB1

    record(mbbi, "PiezasAzules" {
        field(DTYP, "OPCUA")
        field(INP,
ns=2;s=ProcesadorPiezas.s7300.PiezasAzules")
        field(SCAN, "I/O Intr")
    }

    record(mbbi, "PiezasGrises" {
        field(DTYP, "OPCUA")
        field(INP, "@SUB1 ns=2;s=ProcesadorPiezas.s7300.PiezasGrises")
        field(SCAN, "I/O Intr")
    }

    record(mbbi, "PiezasVerdes" {
        field(DTYP, "OPCUA")
        field(INP,
ns=2;s=ProcesadorPiezas.s7300.PiezasVerdes")
        field(SCAN, "I/O Intr")
    }

    record(mbbi, "ClasificadoraCamara" {
        field(DTYP, "OPCUA")
        field(INP,
ns=2;s=ProcesadorPiezas.s7300.ClasificadoraCamara")
        field(SCAN, "I/O Intr")
    }

    record(ai, "PesadoRociadoNivelReal" {
        field(DTYP, "OPCUA")
        field(INP,
ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoNivelReal")
        field(SCAN, "I/O Intr")
    }

    record(mbbi, "MachiningCenterProgress" {
        field(DTYP, "OPCUA")
        field(INP,
ns=2;s=ProcesadorPiezas.s7300.MachiningCenterProgress")
        field(SCAN, "I/O Intr")
    }

    record(mbbi, "PesadoRociadoLlenado" {
        field(DTYP, "OPCUA")
        field(INP,
ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoLlenado")
        field(SCAN, "I/O Intr")
    }

```

## Datos para importar variables a WinCC OA

ElementName	TypeName
BotonAuto	Booleano
BotonManual	Booleano
BotonReset	Booleano
BotonStart	Booleano
BotonStop	Booleano
CentradoCerrar	Booleano
CentradoCinta	Booleano
CentradoFCC	Booleano
CentradoFCSB	Booleano
CentradoSensor	Booleano
CentradoSubir	Booleano
ClasificadoraAzulCinta	Booleano
ClasificadoraAzulGiro	Booleano
ClasificadoraCinta	Booleano
ClasificadoraCinta1	Booleano
ClasificadoraMetalCinta	Booleano
ClasificadoraMetalGiro	Booleano
ClasificadoraVerdeCinta	Booleano
ClasificadoraVerdeGiro	Booleano
CMActivado	Booleano
CMCentradoCerrar	Booleano
CMCentradoCinta	Booleano
CMCentradoSubir	Booleano
CMClasificadoraAzulCintaTrue	Booleano
CMClasificadoraCinta	Booleano
CMClasificadoraMetalCintaTrue	Booleano
CMClasificadoraVerdeCintaTrue	Booleano
CMEstampadoraActuador	Booleano
CMEstampadoraCinta	Booleano
CMPesadoCinta	Booleano
CMPesadoRociadoDescarga	Booleano
CMPesadoRociadoLlenado	Booleano
EstampadoraActuador	Booleano
EstampadoraCinta	Booleano
EstampadoraFCB	Booleano
EstampadoraSensor	Booleano
IndicadorAuto	Booleano
IndicadorEmergencia	Booleano
IndicadorManual	Booleano
MachiningCenterBusy	Booleano
MachiningCenterError	Booleano
MachinningCenterOpen	Booleano
PesadoCinta	Booleano
PesadoCinta2	Booleano

```

PesadoRociadoDescargaLed Booleano
PesadoRociadoNivel Booleano
PesadoSensor Booleano
PesadoValor Booleano
SetaEmergencia Booleano
PesadoValorReal Double
PesadoRociadoNivelReal Double
PesadoRociadoFlowSensor Double
ClasificadoraCamara Double
MachiningCenterProgress Double
PesadoRociadoDescarga Double
PesadoRociadoLlenado Double
PiezasAzules Double
PiezasGrises Double
PiezasVerdes Double

# PeriphAddrMain
Manager/User ElementName TypeName _address.._type _address.._reference _address.._poll_group _address.._connection _address.._offset _address.._subindex _address.._direction _address.._datatype
 _address.._drv_ident
UI (1)/0 BotonAuto. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.BotonAuto" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 BotonManual. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.BotonManual" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 BotonReset. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.BotonReset" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 BotonStart. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.BotonStart" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 BotonStop. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.BotonStop" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CentradoCerrar. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CentradoCerrar" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CentradoCinta. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CentradoCinta" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CentradoFCC. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CentradoFCC" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CentradoFCSB. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CentradoFCSB" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CentradoSensor. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CentradoSensor" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CentradoSubir. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CentradoSubir" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 ClasificadoraAzulCinta. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.ClasificadoraAzulCinta" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 ClasificadoraAzulGiro. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.ClasificadoraAzulGiro" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 ClasificadoraCinta. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.ClasificadoraCinta" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 ClasificadoraCinta1. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.ClasificadoraCinta1" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 ClasificadoraMetalCinta. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.ClasificadoraMetalCinta" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 ClasificadoraMetalGiro. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.ClasificadoraMetalGiro" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 ClasificadoraVerdeCinta. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.ClasificadoraVerdeCinta" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 ClasificadoraVerdeGiro. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.ClasificadoraVerdeGiro" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CMActivado. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CMActivado" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CMCentradoCerrar. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CMCentradoCerrar" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CMCentradoCinta. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CMCentradoCinta" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CMCentradoSubir. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CMCentradoSubir" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CMClasificadoraAzulCintaTrue. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CMClasificadoraAzulCintaTrue" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CMClasificadoraCinta. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CMClasificadoraCinta" 0 0 0 0 1 750 "OPCUA"
UI (1)/0 CMClasificadoraMetalCintaTrue. Booleano 16 "KepServer$Datos$1$1$ns=2;s=ProcesadorPiezas.s7300.CMClasificadoraMetalCintaTrue" 0 0 0 0 1 750 "OPCUA"

```

UI (1)/0 CMClasificadoraVerdeCintaTrue.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.CMClasificadoraVerdeCintaTrue"		0	0	0	0	1	750	"OPCUA"
UI (1)/0 CMEstampadoraActuador.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.CMEstampadoraActuador"		0	0	0	0	1	750	"OPCUA"
UI (1)/0 CMEstampadoraCinta.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.CMEstampadoraCinta"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 CMPesadoCinta.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.CMPesadoCinta"	0	0	0	1	750	"OPCUA"		
UI (1)/0 CMPesadoRociadoDescarga.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.CMPesadoRociadoDescarga"		0	0	0	0	1	750	"OPCUA"
UI (1)/0 CMPesadoRociadoLlenado.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.CMPesadoRociadoLlenado"		0	0	0	0	1	750	"OPCUA"
UI (1)/0 EstampadoraActuador.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.EstampadoraActuador"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 EstampadoraCinta.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.EstampadoraCinta"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 EstampadoraFCB.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.EstampadoraFCB"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 EstampadoraSensor.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.EstampadoraSensor"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 IndicadorAuto.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.IndicadorAuto"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 IndicadorEmergencia.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.IndicadorEmergencia"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 IndicadorManual.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.IndicadorManual"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 MachiningCenterBusy.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.MachiningCenterBusy"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 MachiningCenterError.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.MachiningCenterError"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 MachiningCenterOpen.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.MachiningCenterOpen"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PesadoCinta.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoCinta"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PesadoCinta2.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoCinta2"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PesadoRociadoDescargaLed.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoDescargaLed"		0	0	0	0	1	750	"OPCUA"
UI (1)/0 PesadoRociadoNivel.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoNivel"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PesadoSensor.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoSensor"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PesadoValor.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoValor"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 SetaEmergencia.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.SetaEmergencia"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PesadoValorReal.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoValorReal"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PesadoRociadoNivelReal.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoNivelReal"		0	0	0	0	1	750	"OPCUA"
UI (1)/0 PesadoRociadoFlowSensor.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoFlowSensor"		0	0	0	0	1	750	"OPCUA"
UI (1)/0 ClasificadoraCamara.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.ClasificadoraCamara"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 MachiningCenterProgress.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.MachiningCenterProgress"		0	0	0	0	1	750	"OPCUA"
UI (1)/0 PesadoRociadoDescarga.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoDescarga"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PesadoRociadoLlenado.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PesadoRociadoLlenado"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PiezasAzules.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PiezasAzules"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PiezasGrises.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PiezasGrises"	0	0	0	0	1	750	"OPCUA"	
UI (1)/0 PiezasVerdes.	Booleano	16	"KepServer\$Datos\$1\$1\$ns=2;s=ProcesadorPiezas.s7300.PiezasVerdes"	0	0	0	0	1	750	"OPCUA"	

