Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

ENTORNO PARA LA GENERACIÓN AUTOMÁTICA DE PROYECTOS DE AUTOMATIZACIÓN PARA MÁQUINAS MODULARES

Alumno *Terceño Lezcano, Javier*
Director *Orive Revillas, Darío*
Departamento *Ingeniería de Sistemas y Automática*
Curso académico *2017-2018*

Bilbao, 3 de septiembre de 2018

Resumen

En este documento se presenta un método para generar automáticamente proyectos de automatización en TIA Portal para máquinas modulares. Se ha utilizado principalmente la API denominada TIA Openness como herramienta para manipular los distintos proyectos involucrados en el proceso. También se han desarrollado dos aplicaciones para implementar la solución planteada. Estas incluyen desde la gestión de tipos de módulos hasta la generación de proyectos TIA Portal de máquinas pasando por la definición de nuevas máquinas. Finalmente se han realizado pruebas con distintas configuraciones de módulos para comprobar su funcionamiento sobre modelos digitales en simulación. El trabajo se enmarca dentro del paradigma de los sistemas reconfigurables, que busca conseguir una industria más flexible y adaptable a las necesidades del momento actual.

Palabras clave: Sistemas de Fabricación Reconfigurables, TIA Portal, TIA Openness, eXist-db, máquina modular, industria 4.0, AutomationML

Abstract

This document describes a methodology to generate modular machines' automation project in TIA Portal automatically. The API called TIA Openness has been used to manipulate de different projects involved in the process. The solution proposed has been implemented with two aplications. They allow to manage the module types, define new modular machines and generate the TIA Portal project. Then, several tests have been made for different configurations of modules with digital models simulated on a computer to validate its behaviour. This work fits into the paradigm of reconfigurable systems, looking forward to achieving more flexible industry adapted to the current context.

Keywords: Reconfigurable Manufacturing Systems, TIA Portal, TIA Openness, eXist-db, Industy 4.0, AutomationML

Laburpena

Dokumentu honetan TIA Portal-en makina modularrentzako automatizazio proiektuak automatikoki sortzeko metodo bat aurkezten da. Prozesuan parte hartzen duten proiektuak manipulatzeko TIA Openness API-a erabili da batez ere. Aldi berean, aurkezten den soluzioa inplementatzeko bi aplikazio garatu dira. Hauek modulu moten kudeaketatik TIA Portal automatizazio proiektu finalaren sortze operaziorarteko prozesua barne hartzen dute, makina berrien definiziotik igaroz. Azkenik modulu konfigurazio ezberdinetarako frogak egin dira, funtzionamendua eredu digitaletan simulatzeko. Lan hau sistema birkonfiguragarrien paradigman sartzen da, industria malgu eta moldakor bat lortzeko gaur egungo beharrei dagokionez.

Gako-hitzak: Fabrikazio Sistema Birkonfiguragarriak, TIA Portal, TIA Openness, eXist-db, makina modularra, industria 4.0, AutomationML.

Tabla de contenidos

LISTA DE ABREVIATURAS

AML: AutomationML

API: Application Programming Interface

DB: Data Block. Módulo de almacenamiento de datos. Es el equivalente en STEP7 al concepto “Estructura de datos” en la norma IEC 61131.

DMS: Dedicated Manufacturing Systems

FB: Function Block. Módulo con memoria programable por el usuario y que representa una función de control especializada.

FC: Function. Módulo sin memoria programable por el usuario y que representa una función de control especializada.

FMS: Flexible Manufacturing Systems

HMI: Human Machine Intrerface

MCD: Mechatronic Concept Designer. Herramienta de ingeniería de Siemens para modelar y simular componentes mecatrónicos.

OB: Organization Block. Es el equivalente en STEP7 al concepto “Tarea” en la norma IEC 61131. Constituyen el interfaz entre el sistema operativo y el programa de usuario.

PLC: Programmable Logic Controller

RMS: Reconfigurable Manufacturing Systems

TIA Portal: Totally Integrated Automation Portal. Herramienta de ingeniería de Siemens para la configuración, programación y puesta en marcha de PLCs, sistemas de supervisión y accionamientos.

UDT: User Data Type. Tipo de Dato definido por el usuario en TIA Portal.

XML: eXtensible Markup Language

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

En este documento se presenta el diseño e implementación de un método para la generación automática de proyectos de automatización de máquinas modulares. La propuesta surge del Grupo de Control e Integración de Sistemas (GCIS), vinculado al Departamento de Ingeniería de Sistemas y Automática de la UPV/EHU en colaboración con empresas interesadas en el proyecto.

En los apartados presentados a continuación se detalla la problemática vinculada a las máquinas modulares y se desarrolla una solución para la generación automática del proyecto de control utilizando TIA Portal. El objetivo final es facilitar y agilizar el proceso de creación y reconfiguración de máquinas modulares.

En una primera parte se comparan las características de distintos tipos de sistemas de fabricación, para centrarse posteriormente en los sistemas modulares. Se explican los beneficios que pueden aportar a la situación actual de la industria, tanto técnicamente como a nivel económico.

El proyecto de automatización de la máquina modular se generará partiendo de proyectos independientes de cada módulo. Por lo tanto se hace un análisis previo de la estructura que debe seguir un proyecto de un módulo para poder ser integrado en el proyecto final de la máquina.

Una vez hecho este análisis se procede a desarrollar la arquitectura software necesaria para gestionar la información de los proyectos, almacenarla y utilizarla para la generación. Se utilizarán tecnologías relacionadas con los formatos XML y AML, aprovechando que son los formatos que utiliza la API de Siemens TIA Portal Openness para exportar e importar información de los proyectos.

A continuación se presentan las pruebas de validación realizadas con la célula FMS 200 del laboratorio del Departamento de Ingeniería de Sistemas y Automática. Se ha comprobado de forma práctica el correcto funcionamiento de la arquitectura software desarrollada usando modelos en simulación sobre la herramienta NX-MCD.

Finalmente se muestra la descripción de las tareas realizadas y el análisis de costes, para cerrar con las conclusiones finales del trabajo.

2 Contexto

El presente Trabajo de Fin de Máster surge a propuesta del grupo de investigación Grupo de Control e Integración de Sistemas (GCIS), vinculado al Departamento de Ingeniería de Sistemas y Automática de la UPV/EHU. Este grupo de investigación trabaja en distintas áreas relacionadas con conceptos que se empiezan a vislumbrar como las bases de la industria para los próximos años; Industria 4.0 o Fábrica Inteligente entre otros.

Una de las líneas de investigación actuales está orientada a la automatización de sistemas de producción haciéndolos más flexibles e inteligentes. En este sentido, se pretende establecer metodologías y desarrollar nuevas herramientas con el objetivo de automatizar, controlar y optimizar los procesos de producción con el objetivo de conseguir mayor flexibilidad y capacidad de adaptación.

Este trabajo se enmarca dentro de esta línea de investigación. Con él se pretende avanzar en el desarrollo de herramientas para facilitar la automatización y control de sistemas de fabricación modulares, ámbito en el que algunas empresas ya han mostrado su interés y en sintonía con los objetivos de la Industria 4.0.

2.1 Tipos de sistemas de fabricación

Existe una gran variedad de sistemas de fabricación, los cuales suelen ser clasificados en distintos tipos: dedicados (DMS), flexibles (FMS) y reconfigurables (RMS) [1].

- **DMS** (*Dedicated Manufacturing Systems*) o líneas transfer. Son sistemas diseñados específicamente para la fabricación de una pieza. Están optimizados para la realización de ese proceso y su capacidad de adaptación es prácticamente nula. Se utilizan para realizar tiradas muy elevadas trabajando a su máxima capacidad.

- **FMS** (*Flexible Manufacturing System*). Son sistemas que buscan una mayor adaptabilidad a un amplio abanico de piezas. Para ello emplean máquinas de propósito general como las CNCs. Al no ser maquinaria específica para un tipo concreto de pieza, suelen disponer de elementos desaprovechados, lo cual supone un mayor coste. Por ejemplo, una CNC de seis ejes de la que solo se utilicen dos o tres ejes. Los FMS se utilizan para adaptarse a cambios en la producción que los DMS no pueden asumir. Su capacidad de producción es baja, por lo que solo se utilizan para tiradas cortas.

- **RMS** (*Reconfigurable Manufacturing System*). Surgen para cubrir el espacio que queda entre los FMS y los DMS. A diferencia de los FMS, se diseñan para unos requerimientos de producción concretos, pero buscan la capacidad de ser reconfigurados en caso de que se produzcan cambios en la demanda. La reconfiguración puede darse tanto a nivel de sistema (añadiendo o quitando máquinas) como a nivel de máquina. Para conseguir estas modificaciones a un bajo coste y en muy poco tiempo una de las características más importantes de estos sistemas es la modularidad. Tanto la estructura mecánica como el control

deben diseñarse en módulos que sean capaces de integrarse para abarcar las necesidades requeridas por los procesos de fabricación para los que están destinados.

En la Tabla 1 se comparan las principales características de estos sistemas.

	DMS	RMS	FMS
Estructura de la máquina	Fija	Ajustable	Fija
Centro del sistema	Pieza	Familia de piezas	Máquina
Escalabilidad	No	Sí	Sí
Flexibilidad	No	Personalizada	Sí

Tabla 1. Comparación entre tipos de sistemas de fabricación.

2.2 Modularidad

Un producto modular está compuesto por una serie de elementos denominados módulos. Estos módulos deben estar débilmente acoplados entre sí, de tal forma que se facilite la integración del módulo en distintos productos que compartan su funcionalidad con solo pequeñas modificaciones para adaptarlo.

Dentro del paradigma de los sistemas de fabricación reconfigurables se incluyen las máquinas modulares reconfigurables. Los módulos de dichas máquinas deben considerar tanto su parte mecánica y eléctrica como su software de control.

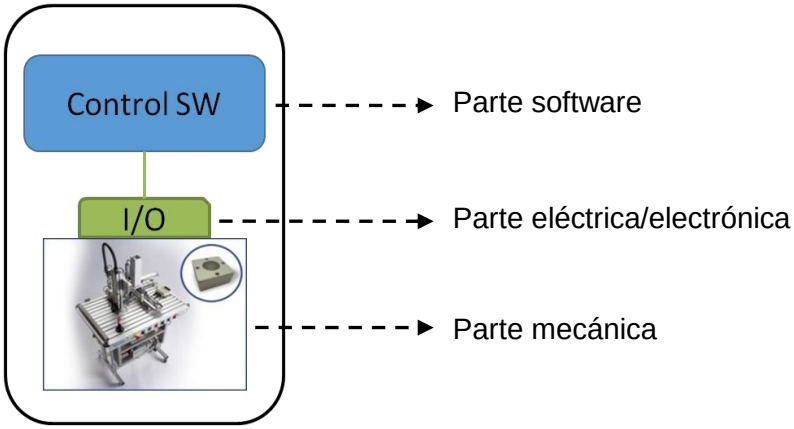


Figura 1. Partes de un módulo mecatrónico.

La Figura 2 muestra el meta-modelo de la vista funcional que define los componentes que puede contener la máquina siguiendo una estructura jerárquica.

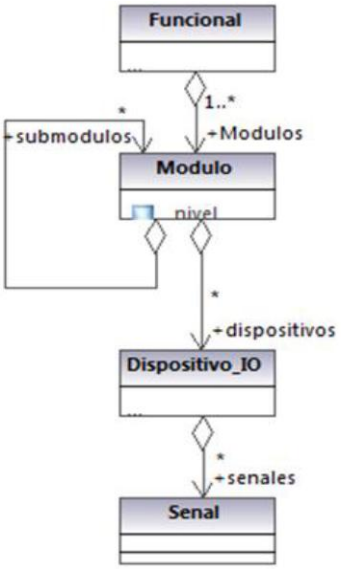


Figura 2. Meta-Modelo de vista funcional de una máquina modular.

De forma general, una máquina estará compuesta por varios módulos del mismo nivel. Un módulo puede estar compuesto a su vez por varios submódulos de nivel inferior, de forma que se pueden conseguir distintos grados de granularidad en función de las necesidades.

La granularidad de una máquina modular puede entenderse dentro de un rango amplio. Se puede pensar en módulos pequeños que realicen una única función como el movimiento de un simple cilindro hasta módulos más grandes que engloben un conjunto de funciones relacionadas.

Este trabajo se centra en la parte de control, desarrollando una metodología y herramientas capaces de integrar de forma rápida y eficiente el control de varios módulos mecatrónicos, consiguiendo una implementación práctica del concepto de modularidad en máquinas.

3 Objetivos y alcance

El objetivo de este proyecto es desarrollar una solución para generar automáticamente proyectos de automatización de máquinas modulares para un entrono Siemens, con la herramienta de ingeniería TIA Portal.

Una máquina modular está compuesta por varios componentes mecatrónicos (módulos). Cada módulo se comunica con sensores y actuadores a traves de señales de entrada/salida (E/S) de campo. Por otro lado es necesario que se comparta información de un módulo a otro, por ejemplo su estado o la finalización de determinada operación.

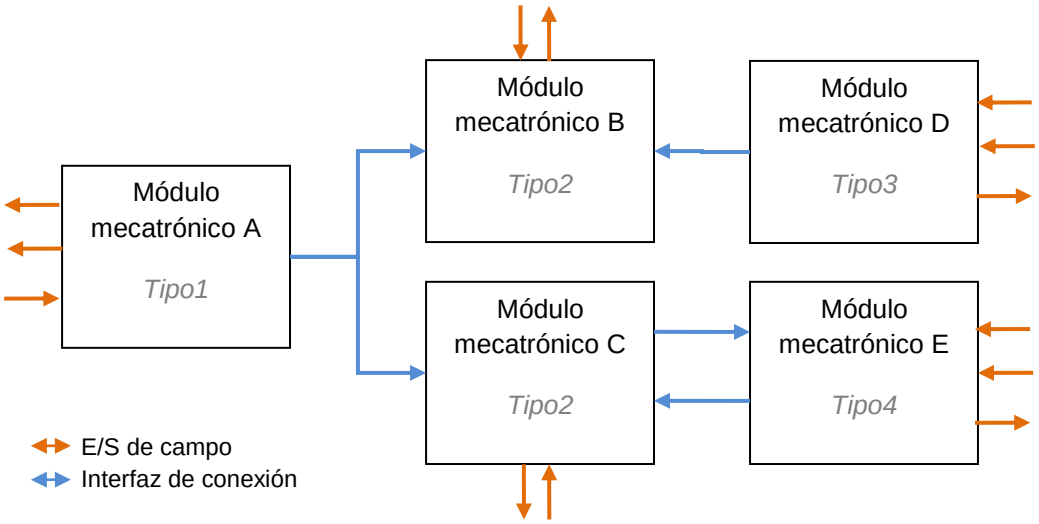


Figura 3. Esquema de máquina modular.

TIA Portal incluye desde su versión V13 SP1 el API llamado TIA Openness, el cual permite actuar sobre los proyectos de automatización desde aplicaciones externas. Entre sus funciones incluye la importación y exportación de la información de proyectos en formato XML y AML. Haciendo uso de estas funciones entre otras lo que se pretende conseguir es la automatización de la generación del proyecto de control en TIA Portal de máquinas modulares a partir de los proyectos de los tipos de módulos mecatrónicos que la componen.

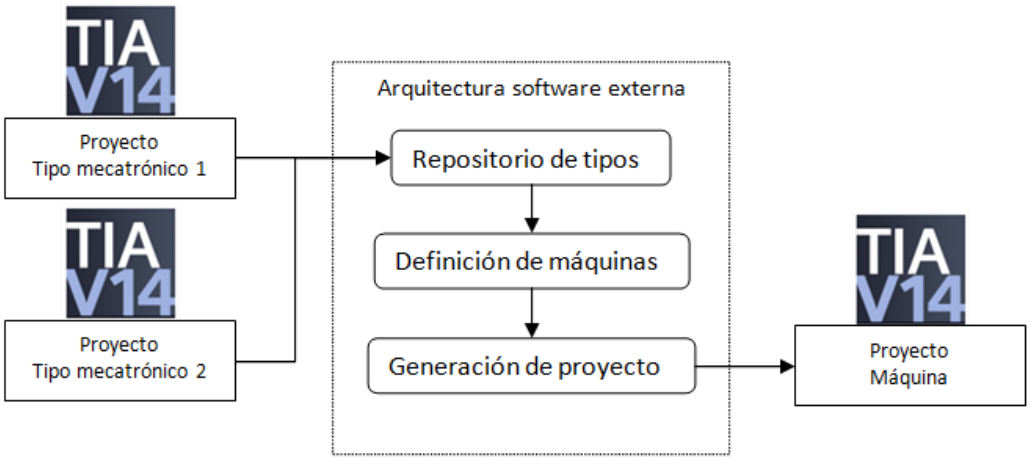


Figura 4. Requerimientos de la solución.

El alcance de este trabajo engloba el diseño y desarrollo de la arquitectura software necesaria para permitir al usuario la definición de nuevas máquinas y la generación automática de su correspondiente proyecto de automatización partiendo de los proyectos tipo de cada módulo. Para ello es necesario también establecer la estructura que deben seguir los proyectos tipo en TIA Portal, tanto a nivel de hardware como de software para hacer posible su integración posterior en un proyecto máquina.

Por lo tanto, este trabajo no se enfoca en el diseño de módulos mecatrónicos concretos, sino en el proceso de generación de proyectos de nuevas máquinas siguiendo una metodología concreta.

Las condiciones a tener en cuenta para poder llevar a cabo el trabajo de forma adecuada son:

- Los dispositivos de cada módulo mecatrónico se comunicarán con la CPU en la máquina final a través de una red Profinet IO. Los dispositivos involucrados en una red Profinet IO se agrupan en Controladores IO encargados de la comunicación con los Dispositivos IO, que son los dispositivos de campo.
- Los módulos mecatrónicos deben disponer de los Dispositivos IO que necesiten para gestionar sus señales de E/S.
- Un tipo de módulo mecatrónico puede instanciarse varias veces en la misma máquina, es decir, una máquina puede tener varios módulos del mismo tipo.
- Se busca la mínima intervención por parte del usuario para evitar errores en la definición y generación de nuevas máquinas.
- Se busca automatizar al máximo el proceso de generación del proyecto de automatización para reducir el tiempo dedicado a la programación y puesta en marcha de las máquinas.
- Aunque TIA Openness también permite manipular información relacionada con el HMI de los proyectos se ha decidido no incluirlos en este trabajo para centrarse en desarrollar la solución para proyectos con Dispositivos IO y un solo PLC como Controlador IO.

4 Beneficios

En este apartado se detallan los beneficios que puede aportar la modularidad en sistemas de fabricación y la generación automática de código en el desarrollo de proyectos de automatización.

4.1 Beneficios técnicos

A continuación se listan los principales beneficios técnicos de este trabajo.

- División de las labores de desarrollo. Pueden llevarse a cabo actualizaciones y mejoras de módulos sin afectar al resto del sistema, así como trabajar en paralelo en el desarrollo de nuevos módulos.
- Reducción de tiempo de diseño de nuevos sistemas de control. Una vez se dispone de los módulos mecatrónicos ya desarrollados, su integración para generar el programa de control de nuevas máquinas se convierte en una labor relativamente rápida.
- Reutilización de los módulos mecatrónicos en distintas máquinas. La capacidad de reutilizar el trabajo realizado previamente en otros proyectos permite un aumento de la eficiencia.
- Capacidad de cambiar la funcionalidad de los sistemas de producción y máquinas existentes para adaptarse a nuevos requerimientos de producción.
- Posibilidad de aplicar este paradigma a distintos niveles del sistema de producción. En este proyecto se aplicará a la generación de máquinas modulares. Pero la misma filosofía puede aplicarse a líneas de fabricación completas, por ejemplo, interpretando cada máquina como un módulo.
- La automatización del proceso de generación de proyectos de automatización elimina los errores propios de la programación manual. De esta forma se produce un aumento de la fiabilidad del código generado.

4.2 Beneficios económicos

Los beneficios técnicos explicados previamente pueden aportar a las empresas un aumento significativo de competitividad originada desde distintas perspectivas.

La rapidez en la adaptación y actualización de sistemas mejora la respuesta ante las variaciones en la demanda de los clientes. Pueden afrontarse cambios en los requisitos del proceso reutilizando el trabajo ya realizado para otros sistemas o hacer frente a un aumento en el volumen de la demanda duplicando la maquinaria de forma sencilla. Esto supone una gran ventaja frente a los sistemas DMS y FMS, los cuales ofrecen unas prestaciones fijas. Esta comparación se puede observar en la Figura 5, en la que un sistema RMS se reconfigura para hacer frente a un aumento en la

capacidad de producción de un producto A (de config. 1 a config. 2) y a continuación se reconfigura variando su funcionalidad para producir otras combinaciones de productos (config. 3 y config. 4).

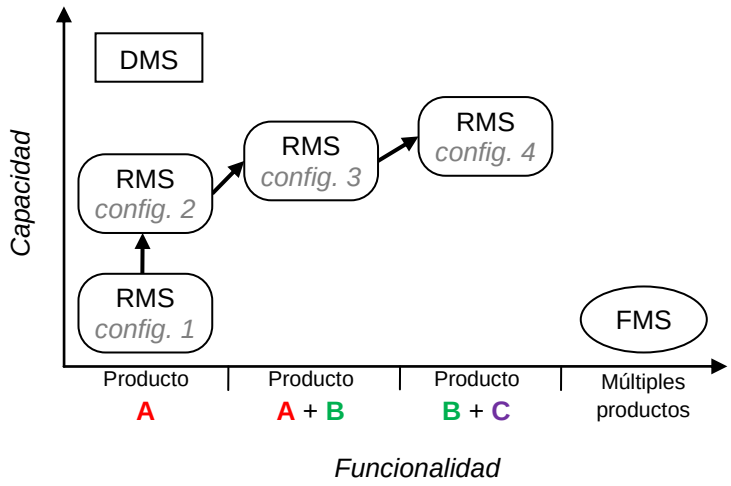


Figura 5. Sistemas de fabricación: Funcionalidad - Capacidad

Además se reduce el tiempo de salida al mercado de nueva maquinaria y nuevos productos. También se reduce el coste de diseño y desarrollo de nuevas máquinas al poder reutilizar módulos desarrollados previamente. Esto da lugar a un retorno de la inversión con menor riesgo, debido al aumento de la capacidad de oferta de una mayor variedad de productos [2].

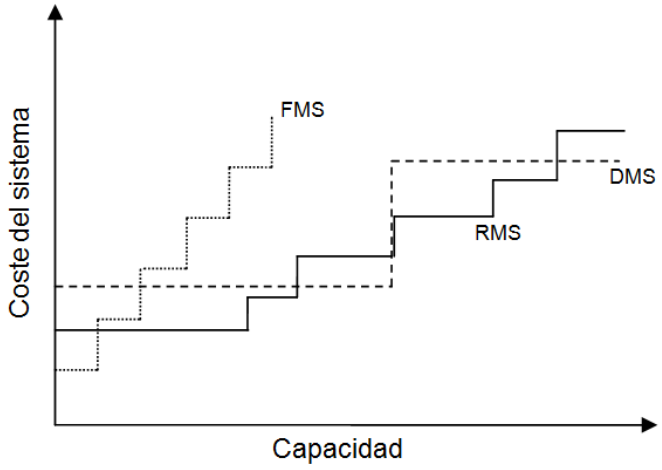


Figura 6. Sistemas de fabricación: Capacidad - Coste

La introducción de este paradigma, de igual forma que aporta a las empresas el dinamismo necesario en la actualidad, abre una puerta a la dinamización del mercado laboral en torno a él, pudiendo generar nuevos puestos de trabajo de calidad orientados al desarrollo y mantenimiento de estos sistemas. En consecuencia, también las instituciones públicas con el apoyo de la sociedad se están viendo interesadas en apoyar estas actividades para impulsar la economía en sus correspondientes ámbitos

de actuación. Ejemplos de ello son las iniciativas Basque Industry 4.0 y Industria Digitala [3] puestas en marcha recientemente por el Gobierno Vasco con un presupuesto de 4,76 millones de euros para apoyar el desarrollo de proyectos orientados a la modernización de la industria del País Vasco.

4.3 Beneficios científicos

Con el surgimiento de nuevas formas de enfocar la industria se está viendo incentivado el desarrollo de herramientas, estándares, etc. que puedan dar viabilidad a soluciones para las necesidades emergentes.

En este trabajo se profundiza en el conocimiento de algunas de estas herramientas, principalmente TIA Openness y AML para la generación de proyectos de automatización.

Fruto del trabajo del grupo GCIS en esta línea de investigación se ha publicado el artículo “Generación automática del proyecto de automatización TIA Portal para máquinas modulares” para las XXXVIII Jornadas de Automática, celebradas en Gijón en 2017.

También se ha participado en la International Conference on Industrial Informatics (INDIN 2018) en Oporto, con “A Tool Suit for Automatic Generation of Modular Machine Automation Projects”.

Además, junto con los proyectos de modelado y simulación por ordenador de componentes mecatrónicos realizados desde el grupo GCIS, se ha ganado el Premio Siemens a la mejor propuesta de trabajo presentada por estudiantes de últimos cursos de estudios universitarios sobre la temática: “Automatización y digitalización. Industria 4.0”, de las XXXIX Jornadas de Automática, celebradas en Badajoz en 2018.

5 Riesgos

Para poder tener éxito en la implantación de sistemas modulares hay que tener en cuenta algunos factores importantes ya que de lo contrario se pueden producir resultados no deseados.

Si el diseño de los módulos mecatrónicos no asegura su reutilización el trabajo invertido en ellos no será aprovechado de forma óptima. Por ello deben estar diseñados desde un inicio dentro de un ámbito de actuación en el cual puedan integrarse, teniendo claro qué funciones van a desarrollar y qué relación van a tener con su entorno.

Al no diseñarse el sistema como un todo sino por partes, puede que no se optimice al máximo en cuestiones como espacio utilizado, cantidad de dispositivos hardware, etc. Por ejemplo, si un módulo utiliza 5 entradas digitales de una tarjeta de 8 entradas y otro utiliza 3 entradas digitales de una tarjeta de 8 entradas. En un sistema dedicado podría haberse utilizado un único dispositivo para el total de las 8 entradas. En el sistema modular cada módulo debe tener su propio dispositivo de entradas.

Se están produciendo cambios y evolución en los estándares y herramientas usadas en los procesos de ingeniería. El tener establecidas ciertas metodologías para los sistemas modulares puede suponer una menor capacidad de adaptación a estos cambios o un aumento de la inversión requerida para actualizarlas. Es importante mantenerse informado de las actualizaciones que se produzcan para poder actuar con rapidez.

Además, el tener que mantener ciertos criterios a la hora de diseñar módulos puede suponer un problema para introducir cambios más drásticos, perdiendo flexibilidad estratégica.

La modularidad de máquinas es un paradigma complejo. Implementarlo puede requerir el rediseño de los métodos de trabajo de una empresa. Contar con personal cualificado y dar formación a los empleados puede facilitar esta transición.

6 Estado del arte

Con la evolución de la economía a gran escala y los sistemas de producción instalados por todo el mundo es difícil encontrar en la actualidad un sector en el que la demanda supere a la capacidad de producción. Las empresas no pueden diferenciarse por producir cada vez más, por lo que buscan competir a través de otros factores como la personalización de productos, rapidez y capacidad de respuesta frente a cambios en la demanda.

Para conseguir estos requerimientos los procesos se vuelven cada vez más complejos y los métodos de trabajo convencionales no son capaces de abarcar todas las necesidades actuales. Ante esta situación, se están abriendo paso nuevas formas de proceder en la industria que buscan dar respuesta a los nuevos retos. La flexibilidad en la fabricación de lotes pequeños de productos más personalizados es básica para satisfacer a una variedad de clientes cada vez más exigentes. Además estas adaptaciones deben conseguirse de forma rápida y fiable, intentando mantener la mayor productividad posible.

Con estos nuevos objetivos se pone en marcha la conocida como cuarta revolución industrial o Industry 4.0. La idea general es digitalizar la industria, aprovechar la información de la planta para anticiparse a errores y realizar mejoras consiguiendo unos procesos más inteligentes. Hasta ahora las diferentes herramientas de ingeniería que conformaban los sistemas PLM (*Product Lifecycle Managment*) o MES (*Manufacturing Execution Systems*) de la empresa se gestionaban de forma independiente. Sin embargo, la tendencia actual se orienta a relacionar e integrar estos sistemas para conseguir mayor control de los procesos. Por ejemplo, Siemens ofrece la plataforma Teamcenter para relacionar sus herramientas PLM como NX o Tecnomatix con el sistema MES y con TIA Portal.

Un ejemplo de la potencialidad de estas herramientas es la realización de pruebas con Software-In-the-Loop. Consiste en realizar un modelo del proceso denominado “gemelo digital” sobre el que poder realizar pruebas en simulación pudiendo observar de forma bastante precisa cómo se comportaría en la realidad. De esta forma se pueden depurar errores y probar cambios en el diseño antes de llegar al proceso de desarrollo de la planta real. Se consigue con ello un ahorro en costes y en tiempo. Desde el grupo de investigación GCIS se están desarrollando otros proyectos orientados a la simulación con Software-In-The-Loop. Entre las herramientas utilizadas destacan NX-MCD (Mechatronic Concept Designer) para modelizar y simular componentes mecatrónicos y PLCSIM Advance para simular la CPU que ejecute el programa de control sobre ese modelo, todo ello en un entorno Siemens.

Junto con otras estrategias como la generación automática de proyectos de automatización que se persigue en este trabajo se consigue actuar sobre diferentes fases del ciclo de vida del producto. Tanto el control de los módulos de forma individual como de las máquinas puede ser validado en una primera etapa sobre el modelo en simulación, sin la necesidad de disponer de la máquina real.

Se aprecia cómo las estrategias de competitividad empresarial se distancian de lo que son capaces de ofrecer los sistemas dedicados y métodos clásicos, acercándose más

a las características de los sistemas reconfigurables con nuevas metodologías de trabajo. Es por ello que existe un alto interés por este tipo de sistemas y se prevé que a medida que se vayan desarrollando en el futuro se consoliden como uno de los pilares de la industria.

Los sistemas de fabricación reconfigurables llevan siendo estudiados desde la década de los 90. En 1999, [1] preveía las necesidades de la industria del siglo XIX y establecía características necesarias en los RMS como la modularidad, integrabilidad y capacidad de diagnóstico entre otras. Varios trabajos han planteado métodos para integrar módulos mecánicamente. Por ejemplo, [4] analiza distintas formas de unión de estructuras de hexágonos. En el apartado del control, [5] estudia el nivel de granularidad requerido para optimizar la reutilización de los módulos.

Respecto a la generación del código del programa de control Siemens ofrece para descargar desde su web una aplicación a modo de demostración de uso de TIA Openness [6] capaz de generar el proyecto de una máquina modular. No obstante, presenta la limitación de que está pensada exclusivamente para un caso particular de máquina con unas determinadas características. En este trabajo se desarrolla un método más general con la intención de abarcar cualquier máquina modular, sin restringirse a casos concretos.

6.1 AutomationML (AML)

AutomationML es un formato de intercambio de datos basado en XML. Está pensado para integrar distintas herramientas a lo largo del proceso de ingeniería siguiendo el modelo de Industria 4.0. Actualmente AML está estandarizado en la norma IEC 62714.

Se ha desarrollado para cumplir requisitos como la adaptabilidad a distintas aplicaciones, eficiencia, ser legible por las personas y basarse en estándares internacionales.

Sigue el paradigma de orientación a objetos. Un archivo AML contiene una estructura jerárquica de objetos que están a su vez caracterizados por una serie de atributos. De esta forma se pueden modelar tanto objetos físicos como lógicos de una planta. Un objeto puede contener información de su topología, geometría, cinemática y control.

6.2 TIA Openness

TIA Openness es el API ofrecido por Siemens para manipular proyectos de TIA Portal desde aplicaciones externas. De esta forma se posibilita su integración en cualquier entorno de desarrollo para automatizar las labores de ingeniería.

Permite abrir nuevas instancias de TIA Portal a las que poder conectarse y trabajar con ellas desde programas propios. Existen dos modos principales de operación: con interfaz de usuario o sin interfaz de usuario. En el primer caso, iniciará la instancia abriendo el interfaz habitual de TIA Portal, así se podrá visualizar en cada momento las operaciones que se lleven a cabo sobre él. En caso de utilizar el modo sin interfaz,

TIA Openness abrirá un proceso de TIA Portal en la memoria del PC pero no será visible para el usuario. No obstante, se podrá de igual forma actuar sobre él haciendo uso de las funciones del API.

Entre las posibilidades que ofrece TIA Openness se encuentran la importación y exportación de información de proyectos de automatización en archivos XML y AML.

6.2.1 Importación/Exportación de hardware (AML)

La configuración hardware de un proyecto se puede exportar en un único archivo con formato AML.

En este archivo se encuentra una estructura jerárquica de objetos que representan el hardware del proyecto con sus características. En general, habrá un elemento AutomationProject que hace referencia al proyecto y dentro de este se encuentran tres categorías principales: las subredes (subnets), los dispositivos (Devices) y grupos de dispositivos (DeviceUserFolders). Entre la información incluida para cada objeto se encuentra el nombre, tipo, direcciones IP, etc. El modelo de objetos completo puede consultarse en el Anexo I.

En la Figura 7 se muestra a modo de ejemplo y de forma simplificada la estructura de un AML resultante de la exportación del hardware de un proyecto.

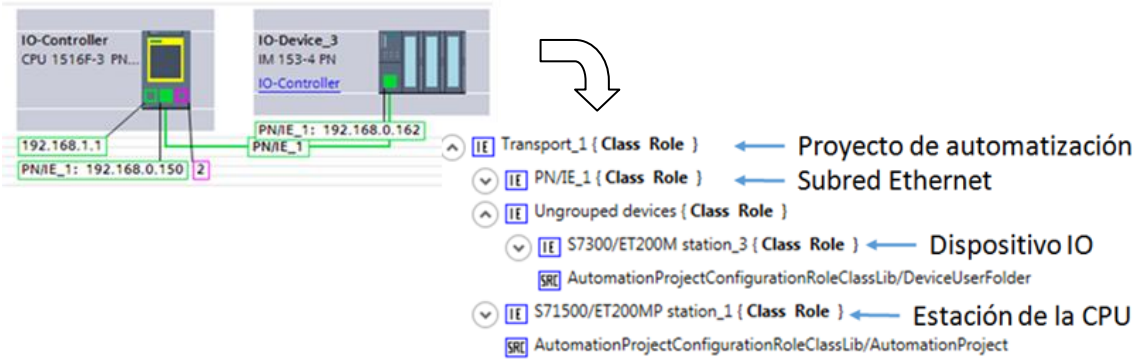


Figura 7. Ejemplo de exportación en AML de un proyecto TIA Portal.

En este archivo AML también se definen los enlaces entre distintos elemento, por ejemplo entre los dispositivos y la subred o entre Controlador IO y Dispositivos IO.

6.2.2 Importación/Exportación de software (XML)

Cada bloque de software de la CPU de un proyecto de automatización puede ser exportado en un archivo XML.

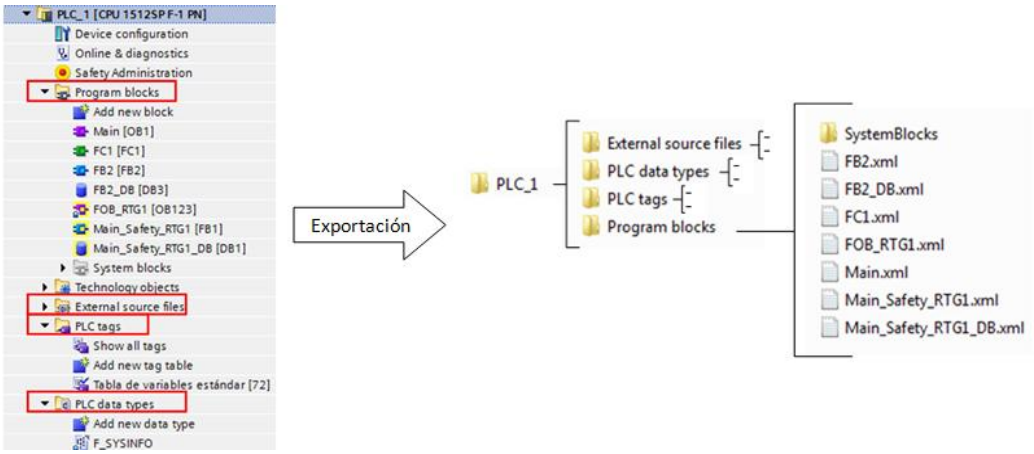


Figura 8. Ejemplo de exportación del software de un proyecto TIA Portal.

TIA Openness también permite el proceso inverso, es decir, un archivo con el formato adecuado puede ser importado a un proyecto existente en TIA Portal.

Por lo tanto no solo se puede obtener información de los proyectos para analizarla externamente, sino que también se pueden modificar o generar archivos XML para importarlos de forma automatizada a un nuevo proyecto.

7 Análisis del problema

Se desea generar el proyecto de automatización de máquinas modulares automáticamente partiendo de los proyectos tipo de los módulos mecatrónicos de los que está compuesta.

El proyecto de automatización de la máquina estará compuesto por los Dispositivos IO y el software de control de cada módulo y una CPU como Controlador IO. La información de los dispositivos IO y del software se obtiene del proyecto tipo de cada módulo. La información de la CPU se obtendrá de proyectos que incluyen una configuración base compuesta por la CPU conectada a una red Profinet IO. Por lo tanto se puede decir que el punto de partida son los proyectos tipo de los módulos y los proyectos CPU.

En todos estos proyectos en TIA Portal se pueden encontrar los elementos e interacciones que se muestran en la Figura 9.

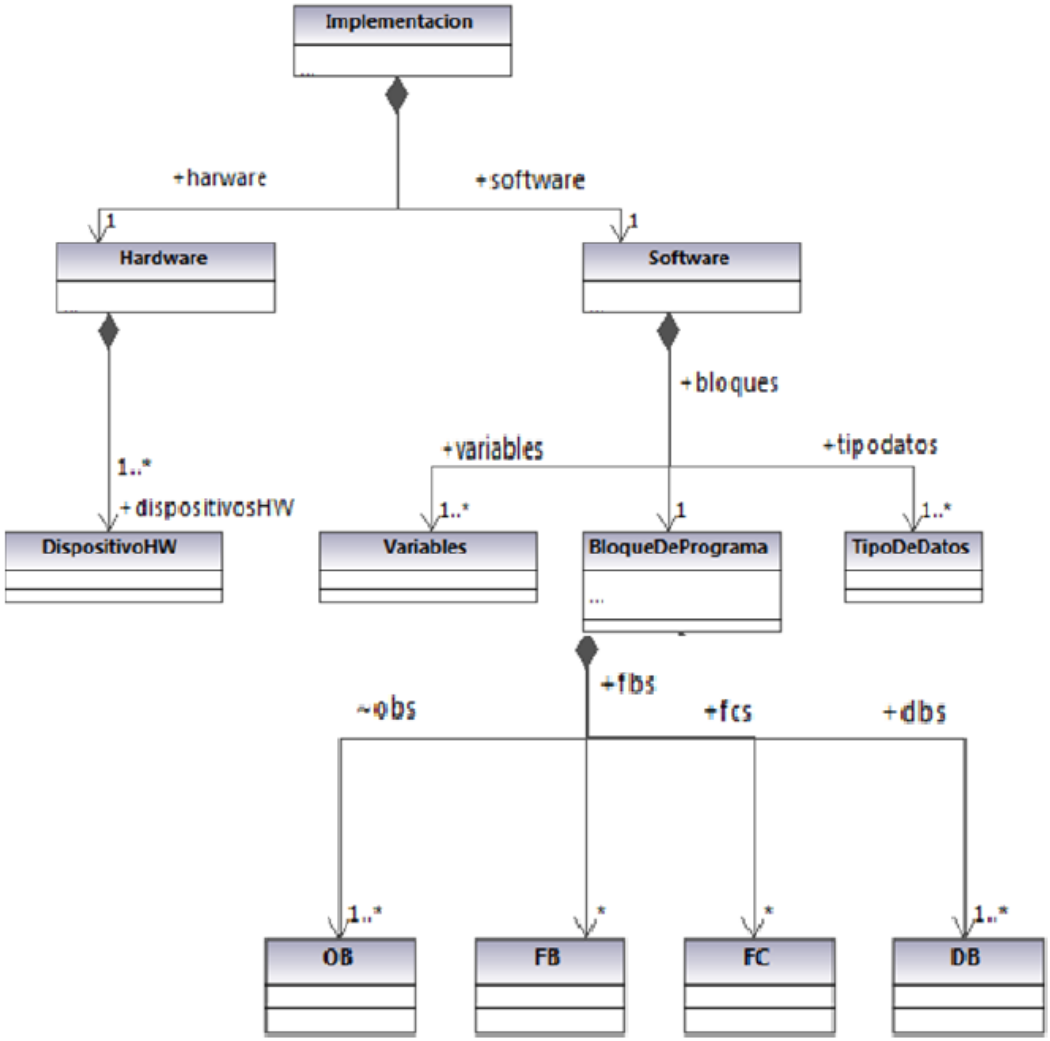


Figura 9. Meta-Modelo vista de implementación de un proyecto TIA Portal.

En la figura puede observarse como un proyecto se divide principalmente en Hardware y Software. El Hardware incluye la información de los dispositivos (Controlador IO y Dispositivos IO) y las redes. En cuanto al software, se divide en Tablas de Variables, Bloques de Programa y Tipos de Datos de Usuario. Los Bloques de programa pueden ser de cuatro tipos diferentes. Los OBs (*Organization Block*) constituyen el interfaz entre el sistema operativo y el programa de usuario, los FBs y FCs son las funciones del programa de usuario y los DBs (*Data Block*) bloques de datos. Los DBs pueden ser GlobalDBs para uso general, o InstanceDBs para guardar información de los FBs de una ejecución a otra.

En el proyecto a generar de la máquina existirán diferentes elementos que, al provenir de distintos proyectos, pueden producir solapamiento de nombres, direcciones y otros conflictos que imposibiliten su integración directa en un único proyecto final. Esto implica que para cada módulo que se desee incluir en el proyecto máquina se deben realizar una serie de adaptaciones para garantizar su unicidad. Estos elementos son:

- Direcciones IP de dispositivos
- Nombre de los dispositivos
- Direcciones de E/S de los Dispositivos IO
- Nombres y direcciones de variables
- Nombres de Tablas de Variables
- Nombres de Tipos de Datos de Usuario
- Nombres y números de Bloques de Programa

Las modificaciones que se realicen afectarán a su vez a otras partes del proyecto máquina, como las comunicaciones, los parámetros actuales usados en las llamadas a funciones, etc.

Además se debe establecer la forma en la que los módulos puedan compartir información en el proyecto máquina.

Se exponen a continuación los criterios adoptados, analizando la estructura necesaria en los proyectos TIA Portal para conseguir dirigirse hacia una solución viable.

7.1 Proyectos tipo en TIA Portal

El proyecto en TIA Portal de cada módulo mecatrónico debe estar realizado de acuerdo con los siguientes criterios.

7.1.1 Hardware

Deben incorporar los Dispositivos IO para gestionar sus señales de E/S de campo. Para poder desarrollar el software del proyecto es necesario que disponga además de

una CPU conectada como Controlador IO a través de una red Profinet a los Dispositivos IO.

Cada Dispositivo IO se caracteriza por:

- Nombre
- Dirección IP
- Dirección Inicial de Entradas
- Dirección Inicial de Salidas

7.1.2 Software

Cada proyecto de un módulo mecatrónico dispondrá de una serie de funciones (FCs y FBs) en las que se implementará el programa de control de ese módulo. Para seguir una estructura concreta que facilite el proceso de generación posterior, se ha decidido que las llamadas a estas funciones se realicen desde una o varias FCs (Principales) que a su vez serán llamadas desde los OBs.

Se diferenciarán las FC Principales de las FC genéricas porque las FC Principales no tendrán parámetros de entrada ni salida, ya que solo se utilizarán para realizar las llamadas al resto de funciones.

En los siguientes apartados se exponen los detalles a tener en cuenta en relación a las necesidades del software de los proyectos tipo.

7.1.2.1 Variables globales (Marcas vs DBs)

Para manejar información dentro del propio módulo pueden utilizarse tanto marcas de la memoria (M) como Bloques de Datos (DB).

La llamada a las variables de DBs se realiza siempre a través del nombre de este, es decir, #NombreDB.NombreVariable. Con el objetivo de evitar duplicidades de nombres y direcciones a la hora de juntar varios módulos, el uso de DBs supone una opción más sencilla y estructurada, ya que será suficiente con adaptar el nombre del DB para mantener las variables internas de cada módulo perfectamente identificadas de forma biunívoca. Sin embargo con uso de marcas independientes habría que tener en cuenta el nombre de cada variable y su dirección. Por lo tanto, todas las variables globales que se utilicen en un módulo se declararán en DBs globales, quedando en las Tablas de Variables sólo las variables de E/S relacionadas con los Dispositivos IO.

7.1.2.2 Uso de contadores y temporizadores (S7 vs IEC)

Para el uso de contadores y temporizadores TIA Portal cuenta con dos opciones: contadores y temporizadores que siguen el estándar IEC 61131 y los propios de Siemens S7. Los del estándar IEC 61131 están incluidos en funciones software SFB, mientras que los de S7 utilizan registros hardware.

Si se utilizan los del IEC 61131, en el proceso de generación pueden tratarse como al resto de bloques de programa (FB y DB), mientras que si se utilizaran los de S7 habría que tener en cuenta las direcciones de memoria de las variables tipo Timer y de tipo Counter.

Siguiendo el mismo criterio que con el caso de las variables globales, se ha decidido que se utilicen exclusivamente contadores y temporizadores IEC, ya que facilitan el tratamiento de la información en el proceso de generación del proyecto de la máquina.

7.1.2.3 Interacción entre módulos mecatrónicos

Para garantizar la posibilidad de usar un módulo en diferentes máquinas este debe ser autocontenido. Por lo tanto un proyecto tipo no puede compartir ni hacer referencia directa a variables de otros módulos. Sin embargo es muy posible que al realizar una máquina los módulos necesiten enviarse información unos a otros. Un ejemplo de esto puede ser una máquina que tenga dos módulos de transporte, y el primero necesite saber si el siguiente está libre para poder continuar con su movimiento.

Realizar esta comunicación con entradas y salidas físicas supone un aumento en la complejidad del componente mecatrónico y mayores costes, ya que se está incluyendo hardware cuando en realidad se puede realizar a través del software.

En cuanto a las soluciones mediante software, puede pensarse en diseñar los módulos utilizando el mismo nombre en los distintos proyectos tipo para las variables que se compartan entre módulos. Sin embargo esto es un problema porque el diseño de un módulo está condicionado a las variables utilizadas en otro. Por lo tanto los módulos estarían fuertemente acoplados y se perdería la flexibilidad que se busca a través de la modularidad.

Para dar solución a este problema se ha optado por declarar en cada proyecto tipo las variables de conexión que se necesiten, representando el interfaz de relaciones con otros módulos. Posteriormente es el usuario, a la hora de definir una nueva máquina, el que decide de esas variables de un módulo con cuales se corresponde en otro. De esta manera se evita la dependencia directa entre módulos, ya que las relaciones las decide posteriormente el usuario.

Estas variables de conexión se han de declarar en dos DBs de conexión. El *DB_CONNECTION_IN* representa la información que puede recibir de otros módulos y *DB_CONNECTION_OUT* la que puede salir hacia otros módulos.

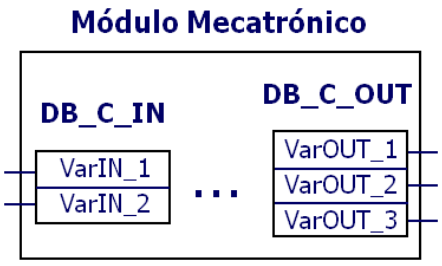


Figura 10. DBs de Conexión.

A pesar de la solución aportada, que se explicará después con más detalle, hay que tener en cuenta que un diseño adecuado de los módulos y sus interfaces sigue siendo fundamental. Módulos con interfaces de conexión sencillos y estandarizados serán más fáciles de integrar en diferentes máquinas. Por otro lado un mal diseño puede no ser capaz de aprovechar todo el potencial del concepto de modularidad, hasta el punto de ser contraproducente.

7.1.2.4 Estructura general del software

En la Figura 11 se muestra la estructura resultante de las decisiones previas respecto al software de un proyecto tipo de un módulo mecatrónico.

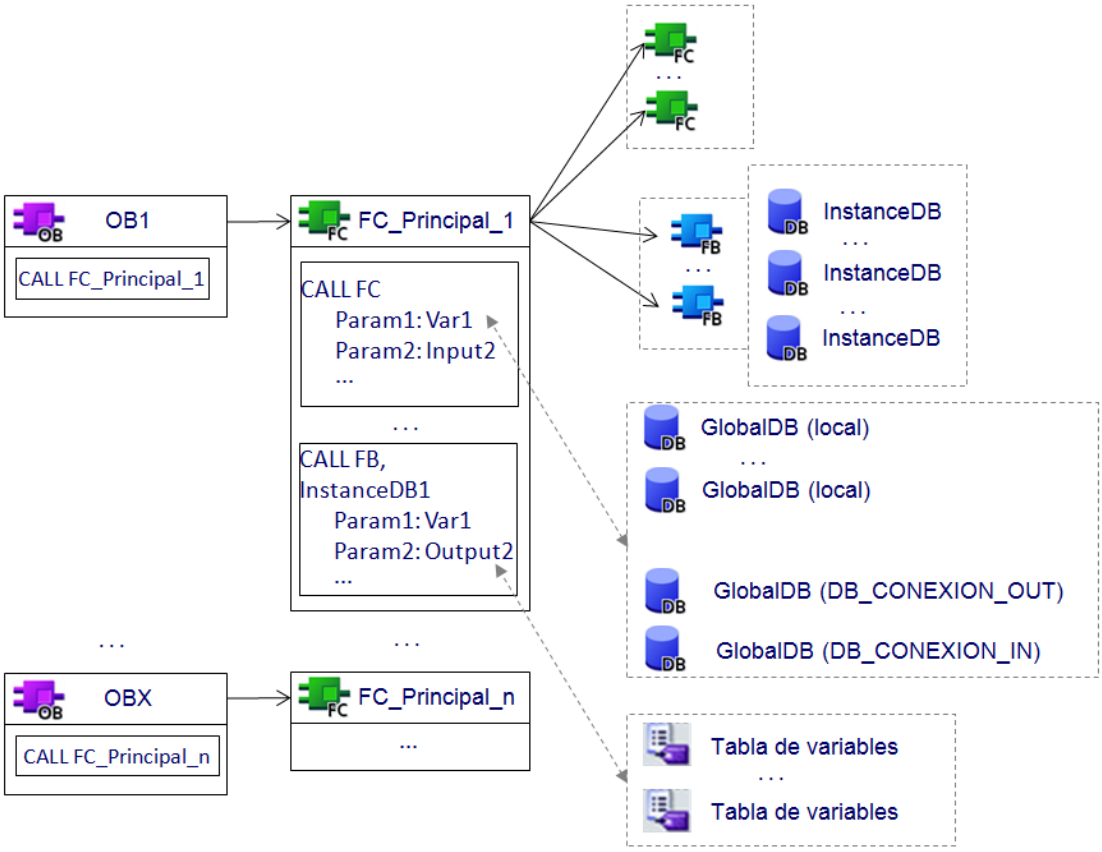


Figura 11. Estructura del software de un proyecto tipo.

Se observa cómo solo en las FC Principales se hace uso de variables declaradas o en DBs o en Tablas de Variables.

Los GlobalDBs se dividen en DBs de conexión y DBs (locales) de uso interno por el módulo. La única forma de identificar posteriormente cuáles son los DBs de conexión es por su nombre, ya que para TIA Portal tanto estos como los locales son del mismo tipo (GlobalDB).

Las Tablas de Variables solo contendrán las variables relacionadas con las direcciones de E/S de los dispositivos IO puesto que, como se ha explicado previamente, no se usarán marcas, contadores ni temporizadores de la memoria.

Conviene recordar que además de los elementos mostrados en la Figura 11 el proyecto debe incluir los Tipos de Datos de Usuario (UDT) que se necesiten en el software de ese componente mecatrónico.

7.2 Proyecto CPU en TIA Portal

Cada proyecto tipo incorpora obligatoriamente una CPU, ya que de lo contrario no se podría añadir el software necesario para el control del mismo. Sin embargo, en el proyecto máquina solo habrá una CPU a la que se conecten todos los Dispositivos IO de todos los módulos.

Entre las opciones posibles para la selección de la CPU en el proyecto máquina está la de elegir un módulo de la máquina como base sobre la que incorporar el resto de elementos de los demás módulos. Pero esto pondría un módulo a distinto nivel que el resto.

Para mantener todos los módulos al mismo nivel se ha optado por realizar proyectos que tengan solo CPU y red Profinet, y utilizar estos como base sobre la que incorporar el hardware y el software de todos los módulos.

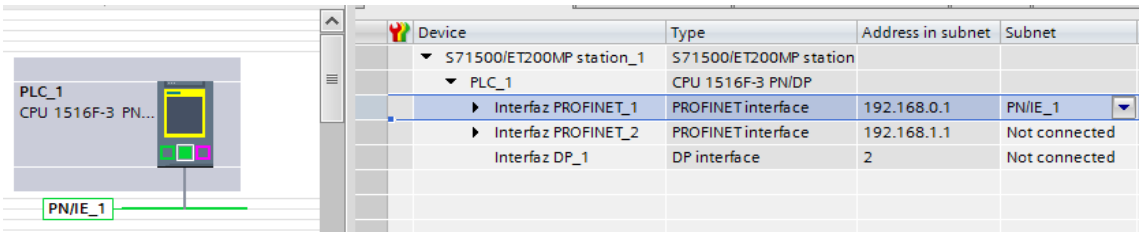


Figura 12. Ejemplo de hardware de proyecto CPU en TIA Portal.

Se ha decidido que la CPU pueda incluir tarjetas de entradas y salidas, en el caso de que se considere necesario, y por lo tanto también el software necesario para controlarlas. Al permitir esto se hace necesario tratar este proyecto como si fuese un módulo más, es decir, requerirá las adaptaciones de nombres y direcciones igual que los módulos mecatrónicos. La diferencia es que, en el proyecto máquina, del hardware de este proyecto se conservará la CPU y la red Profinet mientras que de los módulos mecatrónicos solo lo relacionado con los Dispositivos IO. Como ventaja está que se da más versatilidad a estas configuraciones base, como desventaja un aumento en la complejidad. Pero al ser las modificaciones muy similares a las que ya se van a realizar en los módulos se ha decidido tenerlo en cuenta.

7.3 Proyecto máquina en TIA Portal

7.3.1.1 Hardware

El proyecto máquina estará compuesto por una sola CPU conectada a través de una red Profinet a todos los dispositivos IO.

Se ha decidido que el proyecto de automatización final de la máquina incluya una sola CPU, ya que abarca buena parte de la problemática presente en la industria y simplifica la solución al no tener que diferenciar qué CPU controla a qué Dispositivos IO.

7.3.1.2 Software

Algunos bloques de software pueden repetirse de forma innecesaria si están presentes en varios módulos de la máquina. Es el caso de Tipos de Datos, FBs y FCs genéricas (con parámetros de entrada y salida). Está la posibilidad de mantener un bloque de este tipo por cada módulo, independientemente de que sea igual que el existente en otro módulo. Así se mantiene cada instancia de módulo perfectamente identificada en el proyecto máquina. Sin embargo, se estaría realizando un uso innecesario de la memoria del PLC.

Para aprovechar de forma eficiente el espacio en memoria se ha optado por no repetir los bloques que sean utilizados por varios módulos. Estos bloques se incluirán en el proyecto máquina una sola vez en una carpeta denominada “Common Elements”. Lo que si es necesario duplicar para cada módulo es la llamada a los FBs y sus DBs de instancia.

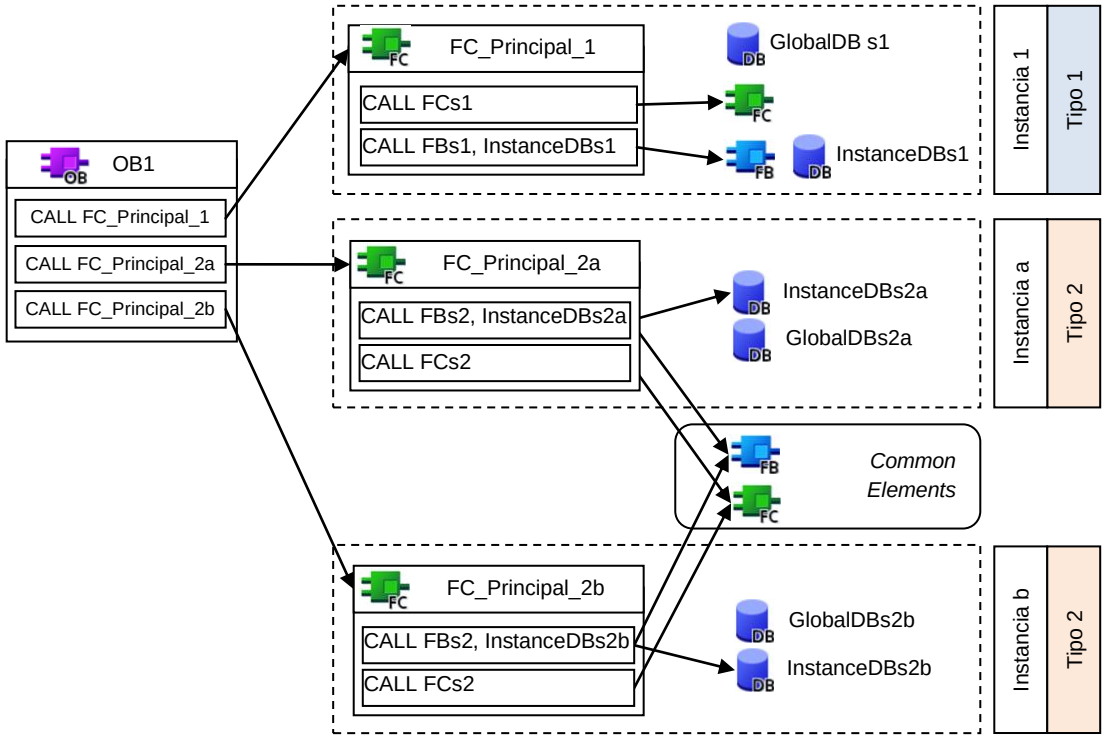


Figura 13. Bloques de Programa de proyecto máquina en TIA Portal. Ejemplo con un módulo tipo 1 y dos módulos tipo 2.

8 Descripción de la solución

A continuación se procede a detallar el diseño de la arquitectura software necesaria para todo el proceso, desde la exportación de proyectos tipo hasta obtener el proyecto final. La Figura 14 muestra una visión global de la misma.

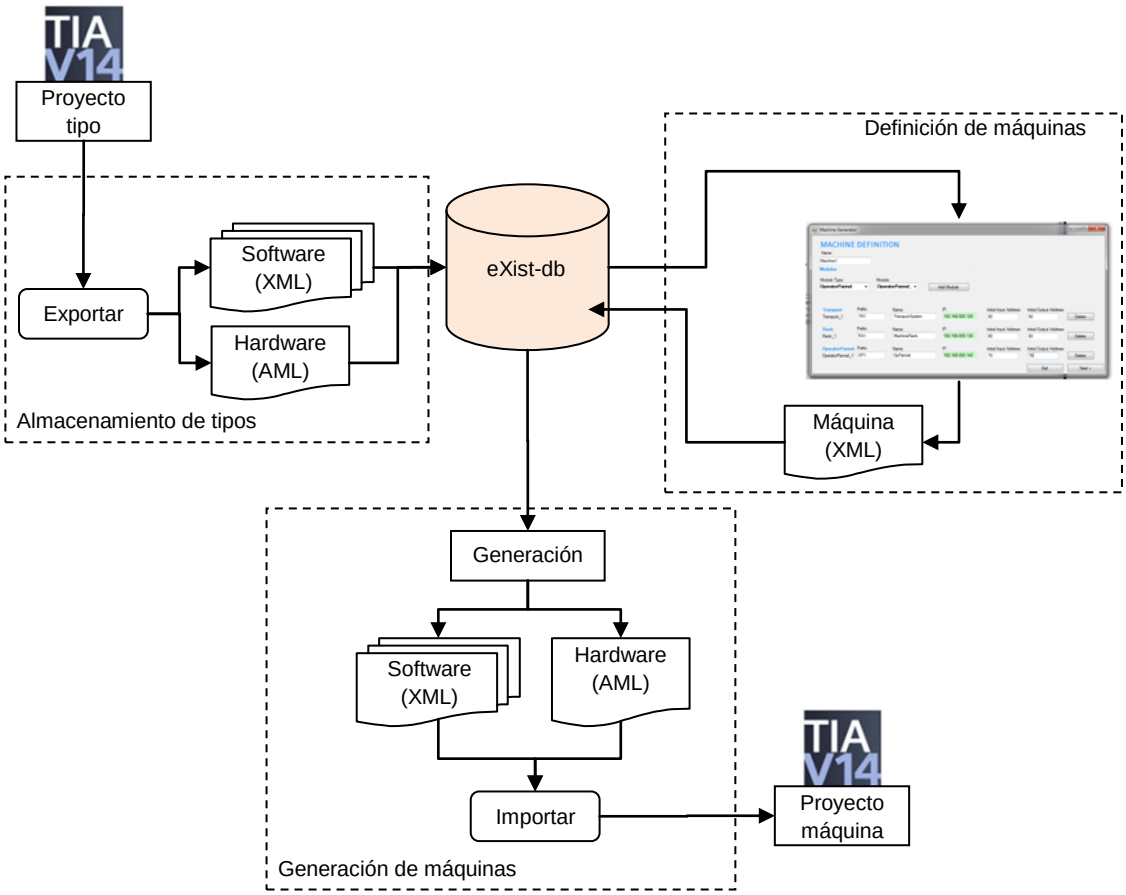


Figura 14. Esquema de arquitectura software completa.

Se divide en tres grandes fases: gestión del repositorio de módulos mecatrónicos, definición de nuevas máquinas y generación del proyecto de las máquinas definidas.

En la primera fase, los proyectos tipo de los módulos mecatrónicos se exportan utilizando TIA Openness. Esta acción genera un archivo XML por cada bloque de software y un archivo AML con el hardware del proyecto.

Para disponer de esta información en el proceso de generación, estos archivos se almacenan en una base de datos eXist-db que se usa como repositorio de tipos de componentes mecatrónicos.

Posteriormente, el usuario puede definir una nueva máquina seleccionando los módulos que necesite de los presentes en el repositorio de tipos.

Finalmente, siguiendo la definición de la máquina hecha por el usuario, se generan los archivos XML y AML que serán importados a TIA Portal para obtener el proyecto de automatización de la máquina.

8.1 Base de datos eXist-db

En todas las fases se hace uso de la base de datos eXist-db orientada a modelos. Es una base de datos de tipo NoSQl y XML nativa, lo cual facilita el almacenamiento y manejo de archivos en formato XML, que es el usado por TIA Openness y por lo tanto el que se usa para almacenar la información de los componentes mecatrónicos.

Una de las características de esta base de datos es que no se necesita especificar un XML schema de la información que se almacena, lo cual da mucha flexibilidad para introducir diferentes tipos de documentos, y para futuras ampliaciones. Esto no significa que no se pueda hacer validación de los documentos XML que se introduzcan.

En la Figura 15 se muestra la estructura de colecciones elegida para la base de datos de acuerdo con las necesidades de la solución.

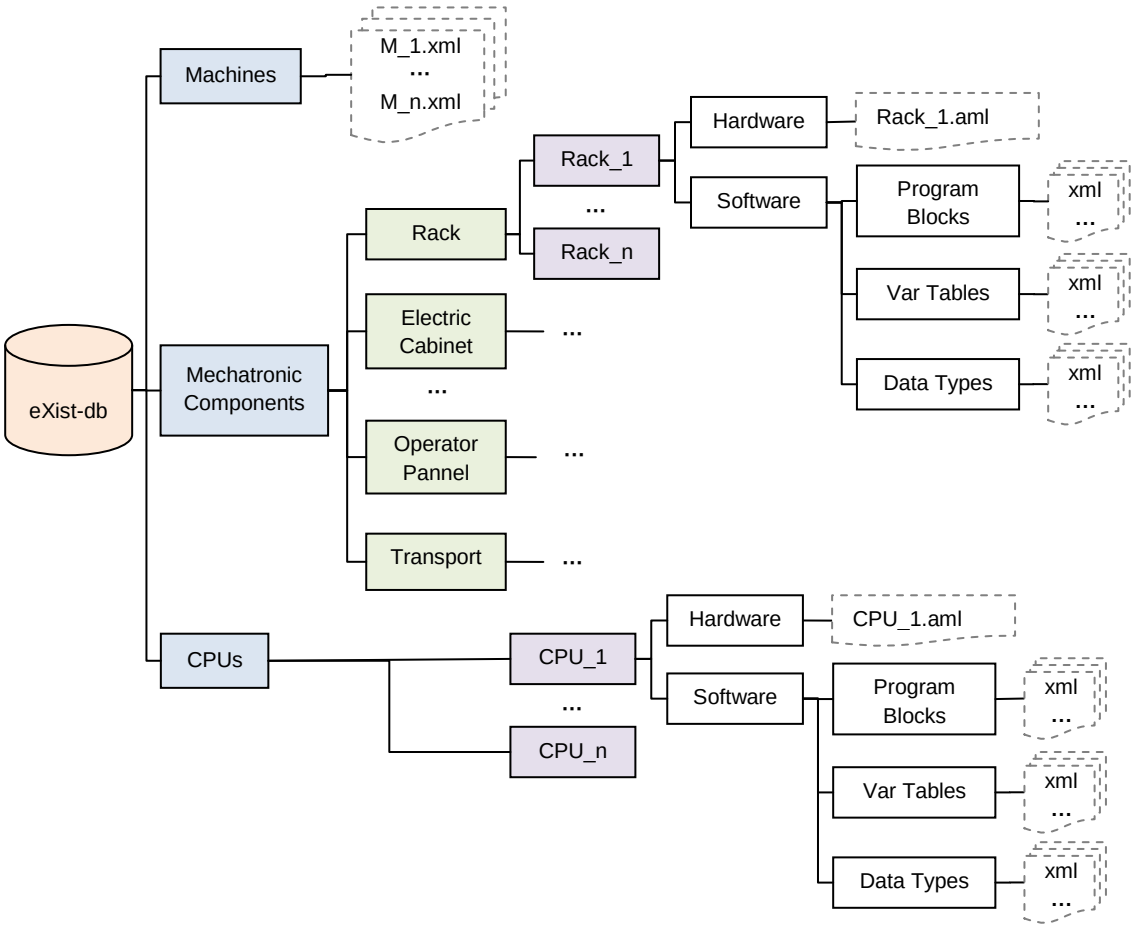


Figura 15. Estructura de colecciones planteada para la base de datos eXist-db.

Los componentes mecatrónicos se subdividen en diferentes grupos. Se ha hecho una primera clasificación fruto de las necesidades analizadas en colaboración con una empresa de una de sus líneas de fabricación. Los principales grupos que surgen de este análisis son: Bastidor (*Rack*), Armario Electrico (*Electric Cabinet*), Botonera (*Operator Pannel*) y Transporte (*Transport*). No obstante, esta clasificación dependerá de las necesidades de la empresa que lo implemente, por lo que se debe poder borrar y crear nuevos grupos.

8.2 Definición de máquinas

El usuario debe ser capaz de aportar los detalles propios de la máquina que desee definir. Una vez lo haga se guarda un archivo XML en la base de datos con la información introducida.

8.2.1 Primer paso: Elección de módulos.

En primer lugar se deben seleccionar los módulos mecatrónicos de los que está formada. Cada uno requiere una parametrización que permita su integración en el proyecto máquina, por lo tanto a cada uno hay que asignarle:

- Un prefijo. Se utilizará para modificar los nombres de funciones y variables que puedan producir repeticiones con otros módulos. También ayudará a localizar visualmente el módulo del que procede cada elemento del proyecto máquina.
- Nombre. Identifica al módulo dentro de la máquina.
- IP inicial. Un módulo puede tener varios dispositivos conectados a la red. Eligiendo la dirección IP más baja de ese módulo las del resto de dispositivos se asignarán de forma consecutiva.
- Dirección inicial de entradas de los dispositivos. De igual forma que con las IPs, se elige la más baja y se asignarán las demás de forma consecutiva.
- Dirección inicial de salidas de los dispositivos. La dirección inicial de salidas puede ser distinta a la dirección inicial de entradas.

Además, se han de tener en cuenta las siguientes restricciones:

- Los prefijos no pueden repetirse, ya que en ese caso no se garantizaría la diferencia de los nombres de dispositivos y de variables entre distintos módulos.
- Las IPs no pueden repetirse. Si a un módulo se le asigna la dirección IP inicial 192.168.0.100 y tiene tres dispositivos IO conectados a la red, ocuparán las direcciones 100, 101 y 102. El resto de módulos no pueden utilizar estas IPs en ninguno de sus dispositivos
- Las direcciones de entrada no pueden repetirse. Un módulo ocupará una serie de direcciones de entrada consecutivas según los dispositivos de los que

disponga. El resto de módulos no pueden utilizar direcciones dentro de ese rango. De igual forma sucede con las direcciones de salida.

A continuación se debe elegir la configuración base de CPU que controlará la máquina. Los parámetros a asignarle son el prefijo y la IP inicial. Las direcciones iniciales de E/S solo serán necesarias si esa configuración base de CPU incorpora tarjetas de E/S.

8.2.2 Segundo paso: Relaciones entre módulos mecatrónicos

Para terminar la definición de la máquina hay que especificar las relaciones existentes entre distintos módulos, es decir, qué información comparte cada módulo con el resto.

Para saber qué variables puede enviar un módulo a otros y cuáles puede recibir se utiliza la información almacenada en los DBs de Conexión. Cada variable del DB_CONNECTION_IN puede ser vinculada con una variable del mismo tipo procedente del DB_ CONNECTION_OUT de otro módulo.

Una variable de conexión de entrada puede tener asociada un máximo de una variable de conexión de salida de otro módulo. Sin embargo, una variable de salida puede enviarse a varias de entrada distintas.

8.2.3 Archivo XML de definición de máquina

Una vez se termina con la definición de la máquina se genera un archivo XML que recoge toda la información introducida. El elemento principal de este archivo es el elemento máquina. Dentro incluye cada módulo y dentro de cada módulo la información de sus dispositivos y las relaciones de sus variables de conexión de entrada.

En cada módulo se define una relación por cada variable de conexión de entrada que reciba una de salida de otro módulo. Puesto que una variable de entrada solo puede tener una variable de salida relacionada, el formato de una relación es siempre el mismo: “Variable Entrada ← (Módulo Origen - Variable salida)”.

8.3 Generación del código del proyecto máquina

Partiendo de la definición de una máquina y de los proyectos guardados en la base de datos se puede proceder con la generación de los archivos XML y AML correspondientes al software y al hardware del proyecto máquina. A continuación se describen los pasos a dar para poder integrar los elementos de todos los módulos en un único proyecto.

8.3.1 Hardware (AML)

El objetivo es obtener un archivo AML con los elementos del hardware de todos los módulos que componen la máquina definida. Se parte del archivo AML del proyecto CPU. Dispone de la subred Ethernet y la CPU. Además también se encuentra el

enlace entre ellas. Sobre este archivo se realizan en primer lugar las modificaciones propias de la CPU y después se añade el hardware del resto de módulos. Los pasos a seguir son los siguientes:

1º - En primer lugar se realizan las siguientes modificaciones sobre el AML del proyecto CPU:

- Establecer la IP del nodo del primer interfaz Ethernet de la CPU de acuerdo con la definición de la máquina. Este es el interfaz Profinet con el que se realiza la conexión a la red y posteriormente al resto de dispositivos.

```
<InternalElement ID="41f78e02-aeaa-4b3b-834e-2ef6a06252c4" Name="Interfaz PROFINET_1">
  <Attribute Name="Label" AttributeDataType="xs:string"><Value>X1</Value></Attribute>
  <Attribute Name="PositionNumber" AttributeDataType="xs:int"><Value>32768</Value></Attribute>
  <Attribute Name="BuiltIn" AttributeDataType="xs:boolean"><Value>true</Value></Attribute>
  <InternalElement ID="2e354682-4d5c-49c4-afd6-f0a08f39ef28" Name="E1">
    <Attribute Name="Type" AttributeDataType="xs:string">
      <Value>Ethernet</Value>
    </Attribute>
    <Attribute Name="NetworkAddress" AttributeDataType="xs:string">
      <Value>192.168.0.1</Value>
    </Attribute>
```

Figura 16. Sección de AML de interfaz Profinet.

Una CPU puede tener dos interfaces Profinet. Las direcciones IP de los distintos interfaces deben estar en subredes distintas. Si se elimina el valor de la IP en el archivo AML cuando se importe a TIA Portal se asigna una por defecto de forma automática. Se debe eliminar la IP del segundo interfaz si lo hubiese, para que en la importación se asigne automáticamente una IP que no cree conflicto con el resto. Además se sabe que TIA Portal asigna al segundo interfaz una IP en la subred 192.168.1.XXX, por lo que la IP definida para el primer interfaz debe estar en la subred 192.168.0.XXX.

- Establecer las direcciones de las tarjetas de E/S si las tuviese, de acuerdo con la definición de la máquina.

```
<Attribute Name="StartAddress" AttributeDataType="xs:int">
  <Value>0</Value>
</Attribute>
<Attribute Name="Length" AttributeDataType="xs:int">
  <Value>16</Value>
</Attribute>
<Attribute Name="IoType" AttributeDataType="xs:string">
  <Value>Input</Value>
</Attribute>
</Attribute>
```

Figura 17. Sección de AML con dirección inicial de tarjeta de entradas.

- Añadir el prefijo de la CPU a los nombres de los InternalElement de la estación de la CPU. Con esto se evita que el nombre de un dispositivo sea el mismo que el presente en otros módulos.

```
<InternalElement ID="01b84861-e58b-4c40-89f9-ef0568a574a9" Name="C1_ET 200SP station_1">
  <Attribute Name="TypeIdentifier" AttributeDataType="xs:string">
    <Value>System:Device.ET200SP</Value>
  </Attribute>
  <InternalElement ID="52e1325c-504f-40b9-8d34-0bfbbbbac2dd" Name="C1_Rack_0">
    <Attribute Name="TypeName" AttributeDataType="xs:string">
      <Value>Rack</Value>
```

Figura 18. Sección de AML con el prefijo añadido a los nombres de los InternalElements.

- El elemento CPU incluye un objeto de clase TagTable por cada tabla de variables que tenga el proyecto TIA Portal de origen. En cada TagTable se encuentran las variables de E/S. Esto es así porque después en el archivo AML se establece el enlace entre esas variables y las señales de E/S de los dispositivos hardware según sus direcciones. Las tablas de variables también se exportan completas con los XMLs del software. Puesto que la tabla de variables también se exporta en la parte del software y se trabajará desde allí, basta con eliminar todos los objetos TagTable y los enlaces de esas variables en el AML de la CPU.

- Añadir un elemento IoSystem al interfaz Profinet. Dado que en los proyecto CPU no hay Dispositivos IO conectados a la CPU, el interfaz Profinet de esta no se exporta con ningún elemento IoSystem. Se requiere para poder establecer posteriormente el enlace entre Dispositivos IO y Controlador IO.

```
<InternalElement ID="c6de3af9-49dd-48df-921d-23e5ef7bb1b3" Name="PROFINET IO-System">
  <Attribute Name="Number" AttributeDataType="xs:int">
    <Value>100</Value>
  </Attribute>
  <ExternalInterface ID="f9f9edf0-fc9a-41e8-bd36-c5be558b0c69" Name="LogicalEndPoint_IoSystem"
                     RefBaseClassPath="CommunicationInterfaceClassLib/LogicalEndPoint" />
  <SupportedRoleClass RefRoleClassPath="AutomationProjectConfigurationRoleClassLib/IoSystem" />
</InternalElement>
```

Figura 19. Sección AML del elemento IoSystem.

2º - En este punto el AML aun no tiene el elemento de clase DeviceUserFolder "*Ungrouped Devices*" porque el proyecto CPU no tenía Dispositivos IO. Por lo tanto es necesario añadirlo para incluir después en él los Dispositivos IO de los módulos.

Una vez se hayan realizado estos cambios es el momento de añadir los Dispositivos IO de los componentes mecatrónicos.

3º - Para el archivo AML de cada módulo, primero se realizan los siguientes cambios a los Dispositivos IO.

- Añadir el prefijo a los nombres de todos sus elementos
- Establecer la IP al interfaz Profinet.
- Establecer las direcciones de E/S.

Después se añade el Dispositivo IO al elemento *Ungrouped Devices* del AML final.

Como puede observarse en las figuras con secciones de AML, cada InternalElement tiene un atributo ID cuyo valor es un número de 16 bytes que debe ser único. Pese a las bajas probabilidades de que un número con este formato se repita, en esta

aplicación puede haber elementos que procedan del mismo proyecto tipo, por lo que es indispensable tener esta posibilidad en cuenta. Antes de añadir el dispositivo a los *Ungrouped Devices* se debe comprobar que no se repita ningún ID con los que ya están en el AML final, y si coinciden modificarlo manteniendo ese formato de 16 bytes.

4º - Por último hay que crear los enlaces siguientes.

- Dispositivos IO y subred Ethernet. Se enlaza el ExternalInterface del nodo del interfaz Profinet con el de la subred.
- Dispositivos IO y Controlador IO (CPU). Se enlaza el ExternalInterface del Interfaz Profinet del dispositivo con el del IoSystem de la CPU.

```
<InternalLink Name="Link To Subnet_1"
              RefPartnerSideA="2e354682-4d5c-49c4-afd6-f0a08f39ef28:LogicalEndPoint_Node"
              RefPartnerSideB="0b69cdea-7dd6-4aab-8ecc-366c7c2c1023:LogicalEndPoint_Subnet" />
<InternalLink Name="Link To Subnet_2"
              RefPartnerSideA="6d1923bb-529f-4291-bea1-760541e4b835:LogicalEndPoint_Node"
              RefPartnerSideB="0b69cdea-7dd6-4aab-8ecc-366c7c2c1023:LogicalEndPoint_Subnet" />
<InternalLink Name="Link To IoSystem_1"
              RefPartnerSideA="05baf024-3229-4177-9d08-701f721e17be:LogicalEndPoint_Interface"
              RefPartnerSideB="c6de3af9-49dd-48df-921d-23e5ef7bb1b3:LogicalEndPoint_IoSystem" />
```

Figura 20. Sección de AML con los enlaces necesarios.

Para identificar los elementos de cada enlace se utiliza el formato "*IDdelElemento:NombreDeSuExternalInterface*". Por ejemplo, el enlace destacado en azul en la Figura 20 es el enlace de la CPU con el Dispositivo IO. El *RefPartnerSideB* representa el elemento IoSystem de la CPU, por lo tanto su ID y el nombre de su ExternalInterface son los que se ven en la Figura 19.

Al terminar con estos pasos se obtiene un AML con todos los elemento del hardware de la máquina.

8.3.2 Software

El software del proyecto máquina se va a estructurar por carpetas según los módulos de la máquina, tanto los Bloques de Programa como las Tablas de Variables y Tipos de Datos de Usuario. Para los Tipos de Datos de Usuario y las FBs y FCs genéricas, en caso de que se repita en varios módulos el bloque de software solo se generará una vez y se incluirá en la carpeta de "Common Elements". Este caso se dará, por ejemplo, cuando dos módulos sean del mismo tipo.

Se debe conseguir la estructura de carpetas y archivos XML correspondiente a la máquina en memoria del PC antes de pasar al proceso de importación. Para ello se describe a continuación el tratamiento a realizar con los archivos XML, disponibles en la base de datos, de cada módulo.

8.3.2.1 Tipos de Datos de Usuario

Los Tipos de Datos de Usuario no requieren modificaciones. Simplemente el XML de un Tipo de Dato se añade a la carpeta de su módulo correspondiente, y si ya se ha añadido ese Tipo de Dato con otro módulo se mueve el archivo a la carpeta de Tipos de Datos comunes.

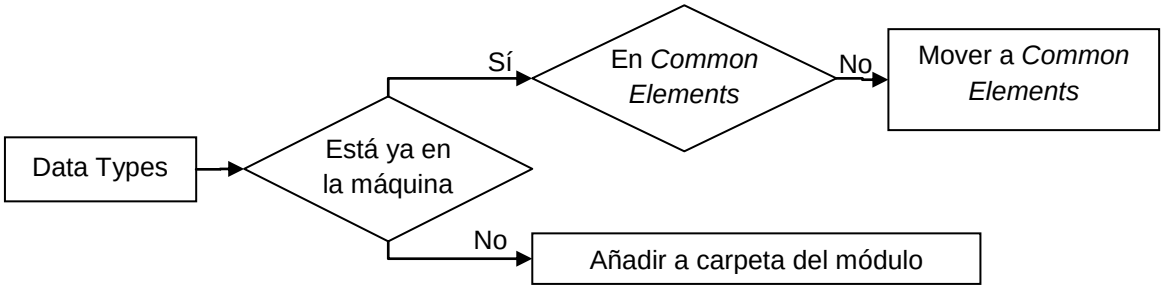
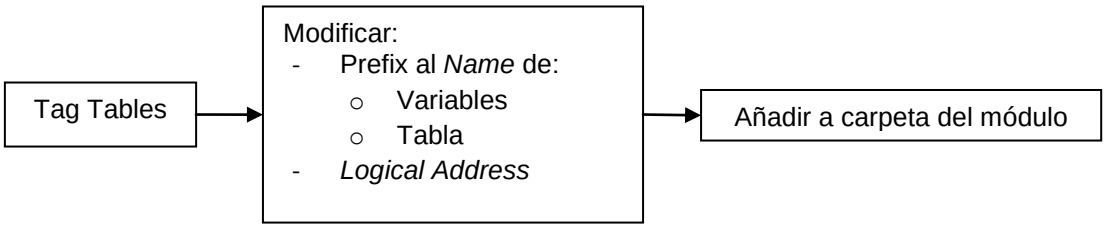


Figura 21. Tratamiento de los Tipos de Datos de Usuario.

8.3.2.2 Tablas de variables

Cada módulo tendrá que tener sus tablas de variables relacionadas con las direcciones de los Dispositivos IO. Se debe añadir el prefijo al nombre de la tabla y al nombre de las variables. Las direcciones deben ser modificadas según las establecidas para los Dispositivos IO de ese módulo, de tal forma que se mantenga la relación entre variable y señal del dispositivo.




```
<AttributeList>
  <Name>M2_Tabla_de_Variables1</Name>
</AttributeList>
```

```
<SW.Tags.PlcTag ID="1" CompositionName="Tags">
  <AttributeList>
    <DataTypeName>Bool</DataTypeName>
    <ExternalAccessible>true</ExternalAccessible>
    <ExternalVisible>true</ExternalVisible>
    <ExternalWritable>true</ExternalWritable>
    <LogicalAddress>%I0.0</LogicalAddress>
    <Name>M2_PU_MA_SE_PU_PI</Name>
```

Figura 22. Tratamiento de Tablas de Variables. Secciones de XML de los elementos a modificar.

8.3.2.3 Bloques de Programa

Los bloques de programa en TIA Portal tienen asignado un número de bloque. En los OBs este número sí diferencia OBs con distintos modos de ejecución. Para el resto de bloques solo es la dirección de referencia para realizar las llamadas e instanciaciones, que en general pueden hacerse por nombre o por dirección (por el número). Las

llamadas a los distintos bloques y variables en los archivos XML se representan utilizando el nombre. Por lo tanto el número en este caso no es relevante. Debido a esto, para evitar que se solapen los números de bloques, se ha decidido eliminar la información del número en el archivo XML de cada bloque. De esta forma es TIA Portal en el momento de la importación el que le asigna automáticamente un número que este libre y se garantiza que no haya solapes.

Además de esto, otros cambios a realizar son los siguientes.

- DBs

Los *DB_CONNECTION_IN* solo se han utilizado para que el usuario pueda definir las relaciones de conexión entre módulos, pero no se van a incluir en el proyecto final, puesto que esas variables se sustituirán por las de los *DB_CONNECTION_OUT* relacionadas.

En todos los demás DBs, se debe añadir el prefijo al nombre para que en caso de que se repita algún tipo de módulo cada instancia de módulo utilice su propio DB.

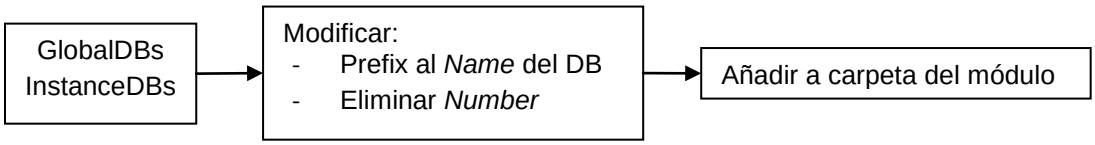


Figura 23. Tratamiento de DBs.

- FBs

En el caso de que un FB se repita en varios módulos solo se generará una vez en la carpeta de *Common Elements*, por lo tanto no necesita diferenciarse con el prefijo. Su código no debe usar variables globales sino parámetros locales para garantizar su reusabilidad.

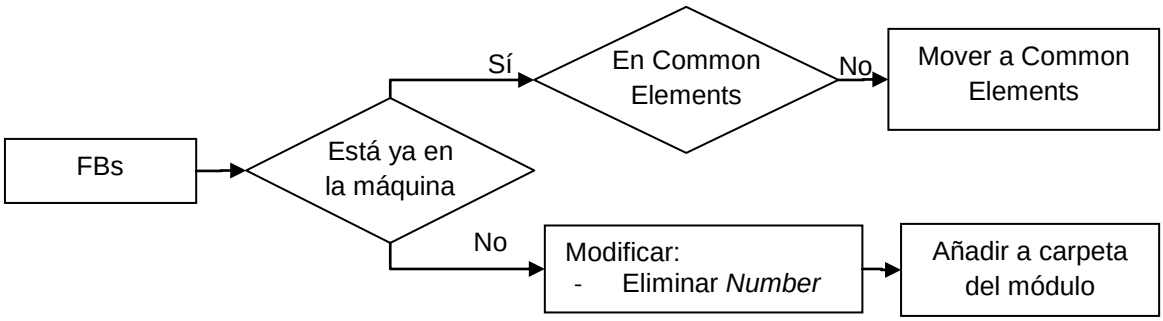


Figura 24. Tratamiento de FBs.

- FCs

Como ya se ha comentado, las FCs se dividen en genéricas y Principales. El tratamiento de las FCs genéricas es el mismo que el de un FB. Sin embargo, una FC Principal sí va a usar variables globales en su código para pasarlas como parámetros actuales a las llamadas de FBs y FCs genéricas. Esos parámetros actuales pueden ser variables de DBs ó variables de E/S de las Tablas de Variables.

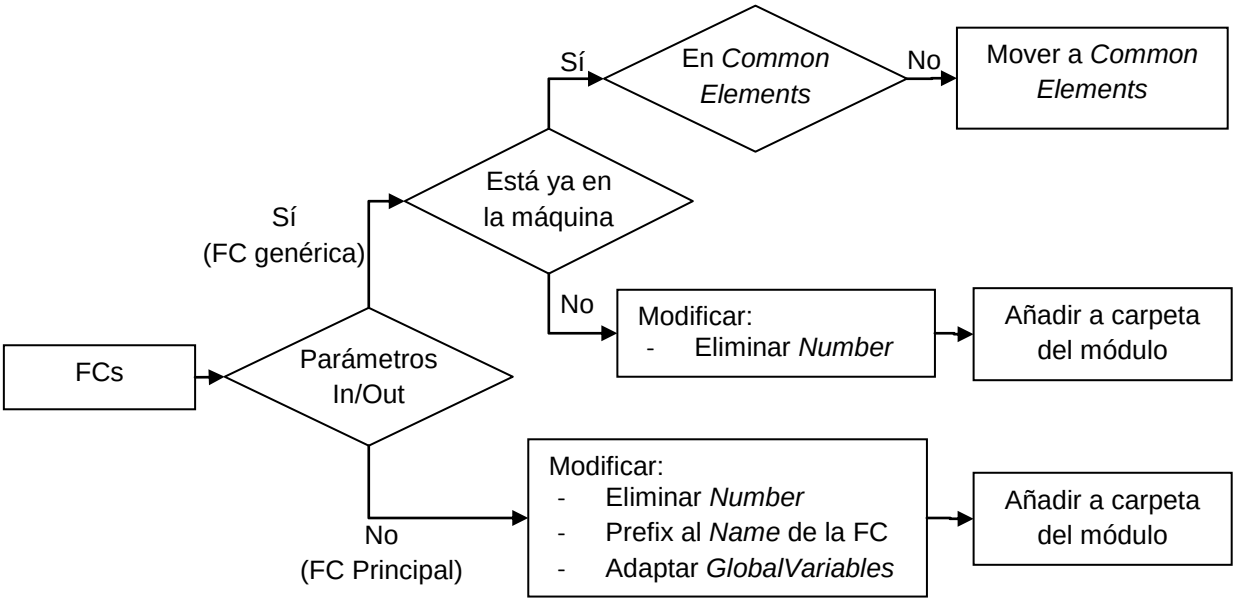


Figura 25. Tratamiento de FCs.

En los archivos XML se identifica como *GlobalVariable* todas las variables globales utilizadas. En este caso las *GlobalVariables* son las variables globales utilizadas como parámetros actuales en las llamadas a funciones y el DB de instancia de las llamadas a FBs. Dependiendo de la variable que sea se trata de forma distinta:

- Variable del *DB_CONNECTION_IN*: se busca en las relaciones de conexión con qué variable y de qué módulo se relaciona y se sustituye por ella.

Sección de XML de un parámetro en la llamada a una función (de un proyecto tipo).

```
<Parameter Name="PI_BAS" Section="Input" Type="Bool">
  <Access Scope="GlobalVariable">
    <Symbol>
      <Component Name="DB_CONNECTION_IN" />
      <Component Name="IP_MM_RACK" />
    </Symbol>
  </Access>
</Parameter>
```

Relación en el XML de definición de la máquina

```
<Relation variableIn="IP_MM_RACK" sourceComponent="bR" variableOut="IPos_O" />
```

Resultado del XML modificado (para el proyecto máquina).

```
<Parameter Name="PI_BAS" Section="Input" Type="Bool">
  <Access Scope="GlobalVariable">
    <Symbol>
      <Component Name="bR_DB_CONNECTION_OUT" />
      <Component Name="IPos_O" />
    </Symbol>
  </Access>
</Parameter>
```

Figura 26. Sección de XML de variable de conexión IN como parámetro actual.

- Resto de variables: se añade el prefijo del módulo al nombre del parámetro actual para que se mantenga el vínculo con los nombres de variables y DBs a los que se les ha cambiado el nombre previamente.

- OBs

Las llamadas a funciones presentes en los OBs de cada módulo deben modificarse con los prefijos para que hagan referencia correctamente a los nombres de las FCs Principales que han sido modificados. Después deben juntarse en un único OB por cada número de OB el código de todos los módulos, puesto que cada número de OB tiene su modo de ejecución particular y no se puede duplicar como se hace con otros bloques.

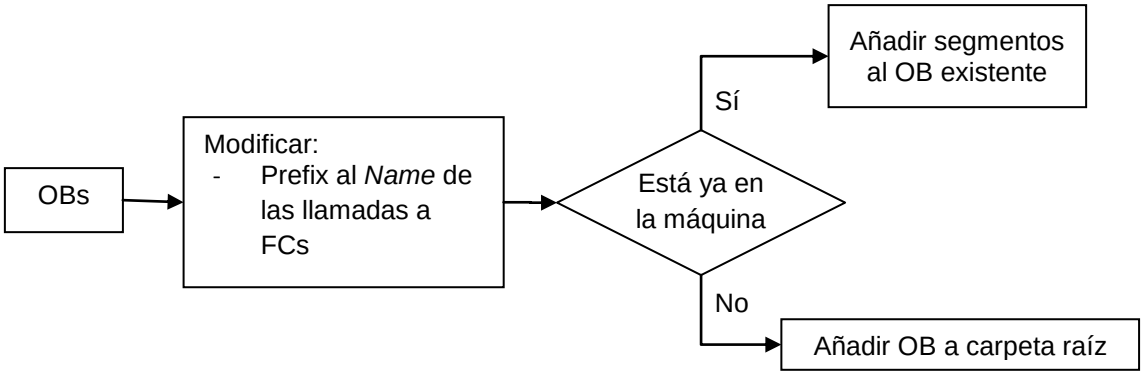


Figura 27. Tratamiento de los OBs.

8.4 Importación a TIA Portal del proyecto máquina

Una vez se tienen todos los archivos se pasa a la importación de forma ordenada a un proyecto TIA Portal, haciendo uso de las funciones que ofrece TIA Openness. Se debe seguir el mismo orden que se seguiría en el caso de realizar el proyecto manualmente.

1) En primer lugar se crea un nuevo proyecto TIA Portal vacío.

2) A continuación se activan los lenguajes que se usen en todos los módulos. TIA Portal permite añadir comentarios en el proyecto en diferentes idiomas. Esta información se exporta en los archivos XML. Si al importar no estuviesen todos los idiomas utilizados activados en el proyecto se lanzaría una excepción.

3) Después se importa el Hardware, indicando en las funciones de TIA Opennes la ruta del archivo AML generado.

4) Una vez se termina con el Hardware se procede con el software, empezando por los Tipos de Datos. Si se intentase importar una función que utiliza un Tipo de Dato que aun no está en el proyecto se detendría la importación y se lanzaría una excepción. De igual forma si un Tipo de Dato utiliza otro Tipo de Dato, este debe estar ya presente en el proyecto TIA Portal. Por lo tanto se empieza importando los Tipos de Datos que utilizan tipos básicos (BOOL, WORD, INT…) y se continúa con el resto según se vaya pudiendo.

5) Siguiendo el mismo criterio, se importan Tablas de Variables y Bloques de Programa según un orden adecuado, indicado en la Figura 28.

6) Por último se compila el proyecto para comprobar que se ha generado correctamente y se guarda en el directorio elegido por el usuario.

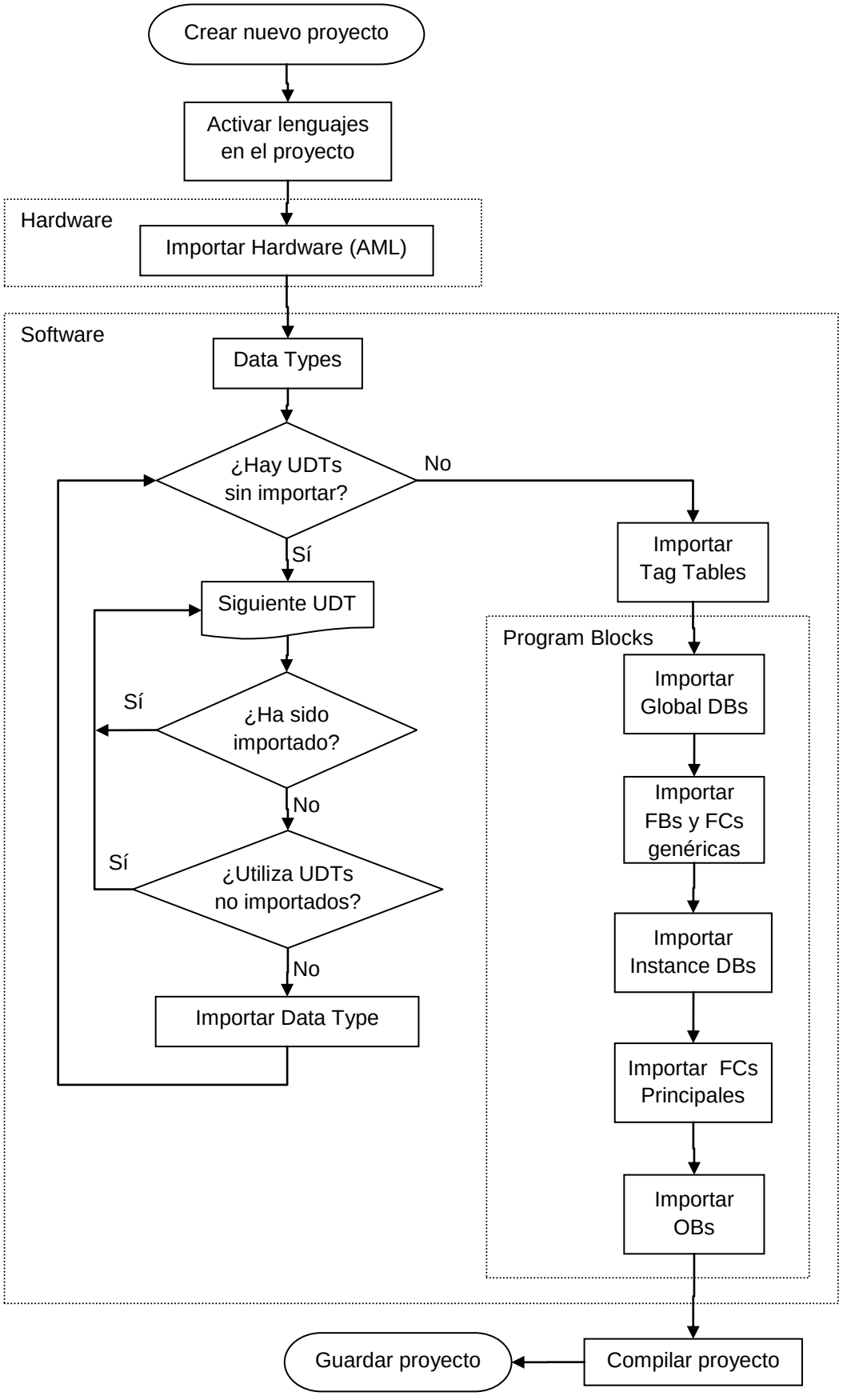


Figura 28. Diagrama del proceso de importación.

9 Metodología

En este apartado se explica cómo la solución descrita se ha implementado en dos aplicaciones separadas, los medios utilizados a tal efecto y las pruebas de validación realizadas.

Todo el trabajo se ha llevado a cabo con los medios disponibles en el laboratorio del Departamento de Ingeniería de Sistemas y Automática de la UPV/EHU. Para las tareas de programación se ha empleado un PC con Visual Studio 2015 y TIA Portal V14 SP1. TIA Openness se incluye con esta versión de TIA Portal. La base de datos eXist-db se ejecuta desde el mismo PC por sencillez durante el desarrollo, aunque podría estar en un host distinto siempre y cuando se tenga acceso a ella.

Para las pruebas finales se ha utilizado la célula FMS 200 del laboratorio. Utilizando el la herramienta NX-MCD para el modelado y la simulación de componentes mecatrónicos se ha comprobado el funcionamiento de proyectos TIA Portal generados para distintas configuraciones de módulos.

9.1 Implementación de la solución

Se han desarrollado dos aplicaciones pensando en los dos perfiles distintos de usuarios que pueden participar en el proceso. Por un lado se encuentra el perfil del desarrollador de componentes mecatrónicos, que realiza los proyectos tipo y los sube a la base de datos. Por otro lado está el que usa la información almacenada para definir y generar nuevas máquinas.

Para realizar ambas aplicaciones se ha utilizado el lenguaje C#, ya que es con el que trabaja la API TIA Openness. Como entorno de desarrollo se ha usado Visual Studio.

9.1.1 Aplicación de desarrollador

Con esta aplicación el usuario es capaz de realizar la gestión de la base de datos. Puesto que la información almacenada de los proyectos CPU difiere de la almacenada para los módulos mecatrónicos, se han separado en dos pestañas distintas cada caso.

Las funciones que incorpora para las CPUs son añadir y eliminar tipos de CPUs de la base de datos. Mientras que para los componentes mecatrónicos también incorpora la opción de crear o eliminar grupos (Operator Pannel, Transport…).

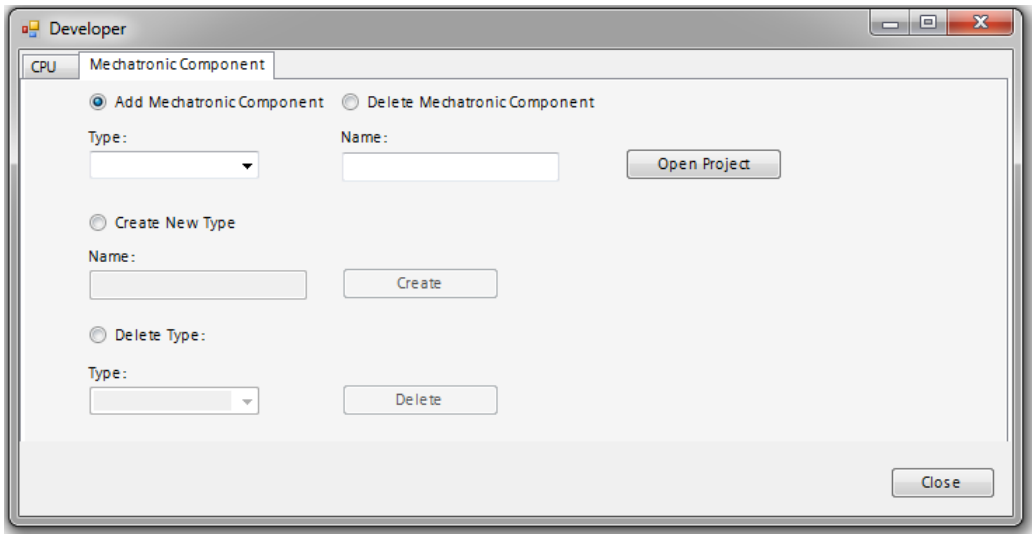


Figura 29. Aplicación de desarrollador.

Al hacer click en Añadir el programa permite al usuario seleccionar el proyecto de TIA Portal con la información que se desea almacenar. Una vez seleccionado se utilizan las funciones de TIA Openness para exportar la información en archivos AML y XML al disco duro. A continuación esos archivos se suben a la base de datos.

El siguiente código abre un proyecto TIA Portal:

```
using (TiaPortal tiaPortal = new TiaPortal(TiaPortalMode.WithoutUserInterface))
{
       ProjectComposition projects = tiaPortal.Projects;
       project = projects.Open(new FileInfo(projectPath));
       //... Uso de project. Exportar ...
}
```

Para abrir TIA Portal se ha utilizado la opción *WithoutUserInterface*, para simplificar la información visual con la que interactúa el usuario. La aplicación muestra un mensaje para informar cuando la exportación termina y otro cuando la subida a la base de datos se completa.

TIA Openness tiene funciones diferentes para exportar el hardware y el software. Para exportar el hardware se utiliza el siguiente código:

```
CaxProvider caxProvider = project.GetService<CaxProvider>();
if (caxProvider != null)
       caxProvider.Export(project, new FileInfo(amlFilePath),
                                   new FileInfo(logFilePath));
```

En este caso, además del AML, se genera un archivo .log con información sobre el resultado del proceso de exportación. Este archivo solo se utiliza para comprobar si ha habido errores en la exportación y avisar al usuario. Si no ha habido ningún error este archivo se elimina y se sube a la base de datos solo el AML. Para las CPUs se sube el AML completo, pero para los componentes mecatrónicos se sube solo la parte de los Dispositivos IO.

Para exportar el software TIA Openness tiene funciones para exportar bloques de forma individual. Pero también se puede exportar toda la estructura del software, que

es lo que interesa en este caso, accediendo al software del PLC del proyecto (*softwareContainer.Software*)

```
OpennessHelper.ExportStructure(softwareContainer.Software,
                               ExportOptions.None, Path);
```

En los archivos XML que se generan en la exportación puede haber algunos elementos de tipo *ReadOnly* y de tipo *Informative* que son ignorados por TIA Openness en el proceso de importación. Con el parámetro *ExportOptions.None* se indica que no exporten estos tipos de elementos. Así se simplifican los archivos XML almacenando solo la información necesaria para el uso posterior en el proceso de generación. Por ejemplo, de una llamada a un FB exporta tanto el nombre del FB como su número de bloque. Sin embargo, en la importación solo tiene en cuenta el nombre del FB e ignora el número. Por lo tanto utilizando el parámetro *ExportOptions.None* en el archivo XML no se exporta el número del FB de la llamada.

9.1.2 Aplicación de Definición y Generación de Máquinas

Esta aplicación ofrece tres funciones principales desde el menú inicial: definir nueva máquina, abrir máquina definida y generar proyecto.

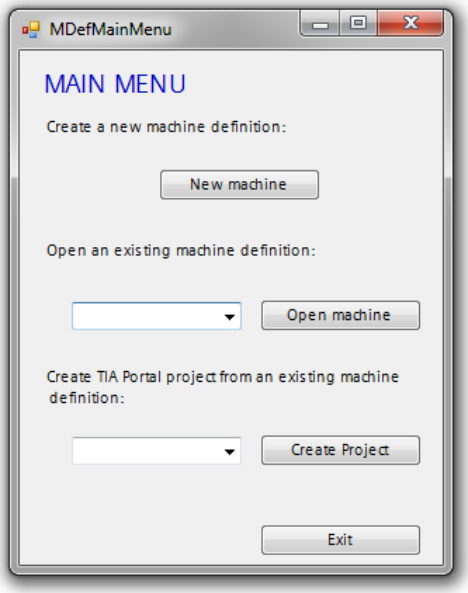


Figura 30. Aplicación de Definición y Generación de Máquinas: Menú Principal.

- **Definir nueva máquina**

Al hacer click en el botón "*New Machine*" se abre una ventana, en la que en primer lugar se pide un nombre para la máquina.

La aplicación se conecta a la base de datos para detectar los módulos disponibles y mostrárselos al usuario. Se seleccionan los módulos de los que esté compuesta, los cuales irán apareciendo debajo para poder parametrizarlos siguiendo las condiciones establecidas en el apartado 8.2.1 de este documento.

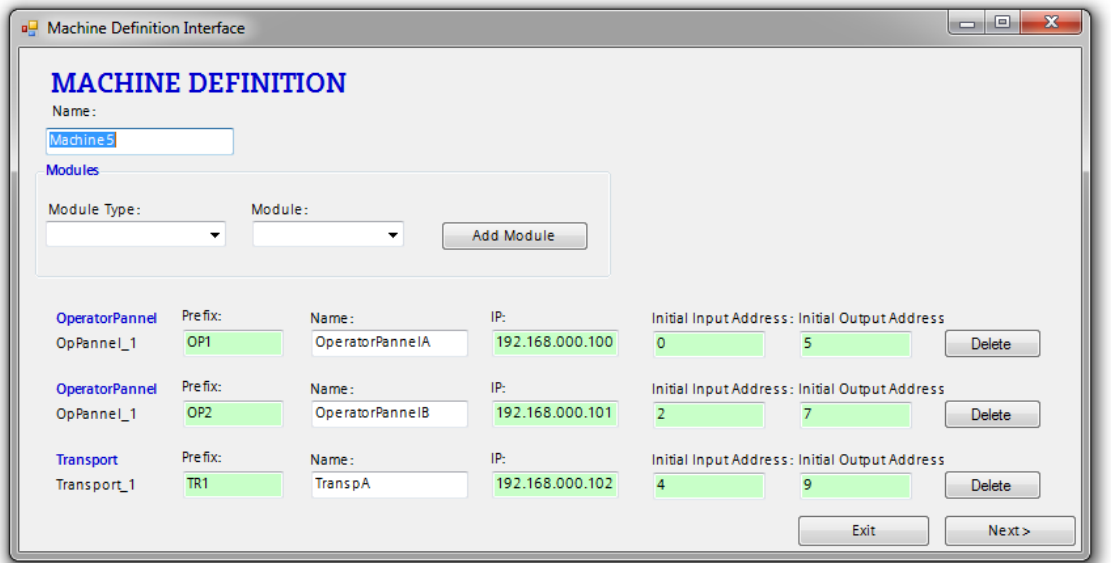


Figura 31. Interfaz de selección de módulos.

Según se añade un módulo la aplicación completa automáticamente sus campos con valores que no supongan un conflicto con el resto de módulos a modo de sugerencia. A continuación el usuario puede modificarlos si lo desea.

Si se rellenase un campo incumpliendo alguna condición, por ejemplo con una IP ya utilizada, se destaca el fondo en color rojo y no se permite al usuario avanzar al siguiente paso.

Al terminar este paso y hacer click en "*Next*" aparece la ventana de selección de CPU de las disponibles en la base de datos.

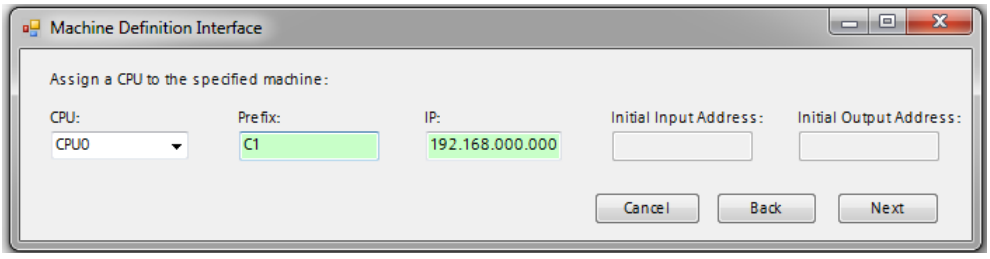


Figura 32. Interfaz de selección de CPU.

Los campos de las direcciones iniciales de E/S solo se habilitan cuando la configuración de la CPU elegida las incorpora. De lo contrario aparecen deshabilitados.

A continuación se muestra la ventana para definir las relaciones entre las variables de conexión entre módulos.

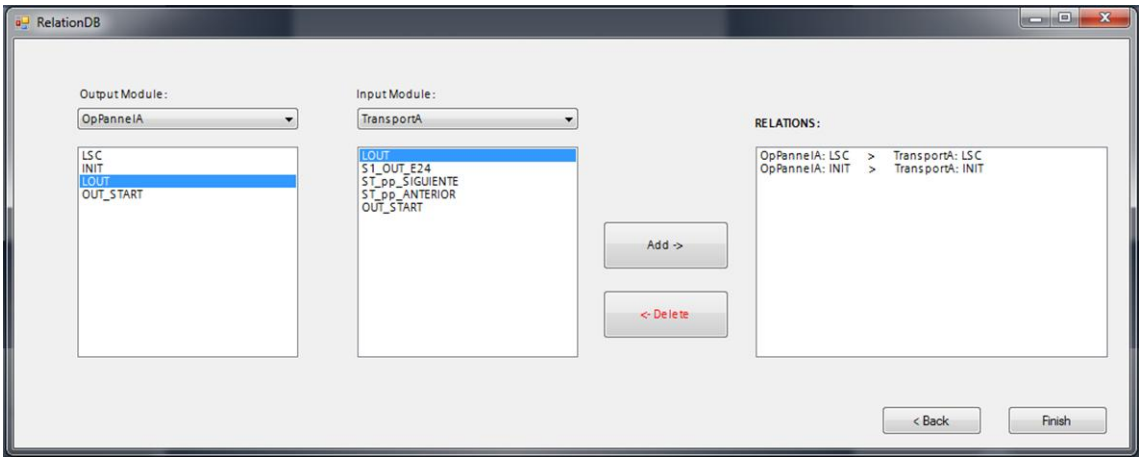


Figura 33. Interfaz de selección de relaciones entre módulos.

A la izquierda, tras seleccionar uno de los módulos de la máquina, la aplicación se conecta a la base de datos para obtener la información de su DB de conexión de salida para mostrar sus variables. Tras marcar una de ellas, en la derecha aparecen las variables del DB de conexión de entradas del resto de módulos. Solo se muestran las variables de entrada que sean del mismo tipo que la variable de salida seleccionada. Marcando la salida con la entrada a la que se conecta y haciendo click en "*Add*" se añade a la lista de relaciones. La entrada se elimina de la lista de entradas, puesto que ya se ha establecido su origen, sin embargo la salida puede seguir enviándose a otras entradas.

Se ha considerado más intuitivo para el usuario que el interfaz muestra las relaciones en el sentido inverso al requerido en el XML de definición de la máquina, es decir, "cada salida a qué entradas se conecta" y no "para cada entrada de qué salida procede". Esta información se procesa para guardarla acorde a las necesidades del XML.

Una vez se hayan establecido todas las relaciones y el usuario haga click en "*Finish*" el archivo XML de definición de la máquina se almacena en la base de datos con toda la información introducida.

- **Abrir máquina definida**

La aplicación permite actualizar la información de máquinas o definir nuevas partiendo de los datos de una ya definida. Para ello, desde el menú principal hay que seleccionar la máquina de las disponibles en la base de datos y hacer click en "*Open Machine*". Se abrirán las ventanas de definición de máquina pero en este caso ya rellenadas con la información de la máquina seleccionada. El usuario puede modificar los campos que considere oportunos y guardar un nuevo XML con la nueva definición.

- **Generar proyecto**

Por último la aplicación permite generar el proyecto TIA Portal de una máquina, simplemente seleccionando la definición disponible en la base de datos de una máquina y haciendo click en "*Create Proyect*". Se muestra entonces un cuadro de diálogo en el que el usuario elige el directorio del PC en el que desea guardar el

proyecto y acto seguido se inicia automáticamente el proceso descrito en los apartados 8.3 y 8.4 de este documento (Generación de código e Importación a TIA Portal).

En cuanto a la generación de código, la forma de proceder utilizada consiste en acceder a la base de datos de forma recursiva para descargarse los archivos XML y AML de un módulo y aplicarles las modificaciones correspondientes (apartado 8.3) para a continuación hacer lo mismo con el siguiente módulo. En primer lugar se actúa sobre el proyecto CPU, ya que la generación del archivo AML así lo requiere. A continuación se procede con los componentes mecatrónicos.

Una vez terminada la generación de los archivos AML y XML de la máquina se abre el TIA Portal, esta vez haciendo uso de su interfaz de usuario con la opción *WithUserInterface*, y se crea un nuevo proyecto sobre el que se importan los archivos generados.

```
using (TiaPortal tiaPortal = new TiaPortal(TiaPortalMode.WithUserInterface))
{
       ProjectComposition projects = tiaPortal.Projects;
       projects.Create(new DirectoryInfo(ProjectDirectory), proyectName);
       Project newProject = projects[0];
       //... Uso de newProject. Importar ...
}
```

Para importar el hardware se utiliza la siguiente función:

```
CaxProvider caxProvider = project.GetService<CaxProvider>();
if (caxProvider != null)
{
    caxProvider.Import(new FileInfo(hardwarepath), new FileInfo(logpathcomplete),
                                                   CaxImportOptions.MoveToParkingLot);
}
```

La importación del software se realiza bloque a bloque. Cada bloque de software se importa en TIA Portal en el grupo (carpeta en TIA Portal) de su módulo correspondiente. Para ello en primer lugar se selecciona el grupo correspondiente en el software del proyecto TIA Portal y si no existe se crea. A continuación se importa el archivo xml a ese grupo con una función similar a la del hardware.

Finalizada la importación se compila el proyecto automáticamente. En primer lugar se compila el software del proyecto y acto seguido el hardware. TIA Openness no soporta la compilación de hardware si este tiene configuración de seguridad. Si así fuese la aplicación muestra un mensaje advirtiendo de que no ha podido compilar el Hardware. El usuario podrá en cualquier caso realizar la compilación completa de forma manual desde el interfaz de TIA Portal.

9.2 Pruebas realizadas. Análisis de los resultados

La validación de la solución desarrollada se ha realizado en la célula FMS 200 disponible en el laboratorio del Departamento de Ingeniería de Sistemas y Automática.

Esta célula no estaba diseñada de forma modular originalmente. Sin embargo, se ha conseguido dividir el control de las diferentes estaciones en módulos. Así que se disponen de los elementos necesarios para realizar las pruebas que confirmen el buen funcionamiento del entorno software desarrollado.

Además de sobre la célula real, también se han realizado pruebas de validación sobre modelos en simulación modelados con NX-MCD. La simulación ha permitido analizar el funcionamiento de configuraciones no disponibles en la célula real, con el fin de hacer comprobaciones más exhaustivas de la potencialidad de este trabajo.

La célula FMS 200 consta de cinco estaciones en las que se ensambla un conjunto de piezas. El diseño en módulos mecatrónicos utilizado es la siguiente:

- Grupo OperatorPannel:
  - Botonera_1: Manejo de las formas de mando en automático y manual.
- Grupo Rack:
  - Bastidor_1: Procesamiento de la base.
  - Bastidor_2: Colocación de rodamiento y bulón.
  - Bastidor_3: Colocación de tapa.
  - Bastidor_4: Colocación de pieza en almacén.
- Grupo Transport:
  - Transporte_1: Cinta sencilla de avance con operación en estación.
  - Transporte_2: Cinta sencilla de avance sin operación en estación.
  - Transport_3: Cinta de avance con posicionamiento y con operaciones en estación.

La configuración de CPU utilizada ha sido en todo momento una CPU S7 1516-3 PN/DP sin E/S.

De los tipos de módulos mecatrónicos identificados previamente se dispone tanto de el proyecto en TIA Portal como del modelo en simulación de Botonera_1, Bastidor_1, Bastidor_3, Transporte_1 y Transporte_2. Con ellos se tienen todos los componentes que forman la Estación 1 y la Estación 3.

Estación	OperatorPannel	Transporte	Rack
1	Botonera_1	Transporte_1	Bastidor_1
3	Botonera_1	Transporte_1	Bastidor_3

Tabla 2. Componentes mecatrónicos de Estación 1 y Estación 3.

En los apartados siguientes se detalla el caso de la estación 3, con la explicación de sus elementos, la generación de su proyecto y el resultado final. Por último se comentan otras pruebas con distintas configuraciones.

9.2.1 Descripción de Estación 3

La estación 3 es la encargada de colocar la tapa del conjunto. En primer lugar recibe el pallet en la cinta de transporte con la pieza base. El transporte identifica el código del pallet, el cual indica qué tipo de tapa se requiere para ese montaje. El alimentador de tapas vertical dispensa tapas que son posicionadas sobre el plato divisor con la pinza 1. A continuación el plato divisor rota una posición. Con diferentes sensores se

identifica qué tipo de tapa es. Si no se corresponde con el tipo necesario para el montaje requerido el extractor retira la tapa. Si es la tapa que se necesita la pieza se deja en el plato divisor hasta llegar a la posición en la que la pinza 2 la coge y la coloca sobre la pieza base.

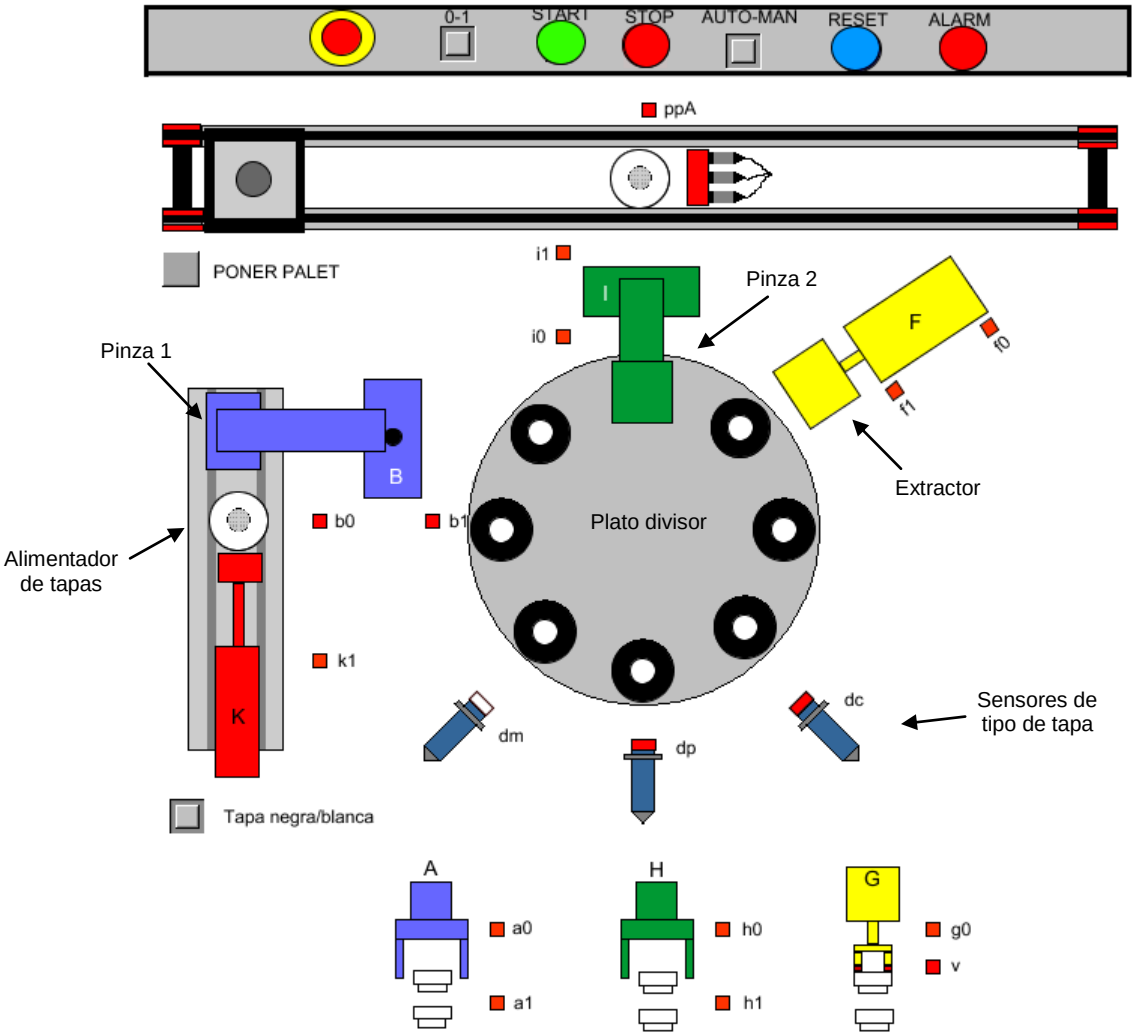


Figura 34. Plano de la Estación 3.

9.2.1.1 Componente mecatrónico Botonera_1

Es el encargado de transmitir las señales de mando dadas desde sus pulsadores y conmutadores.

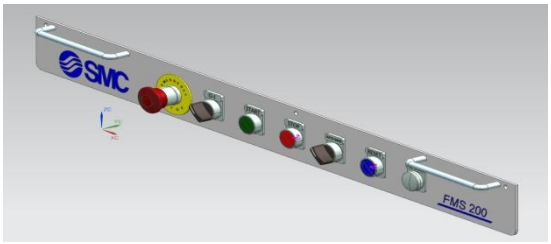


Figura 35. Botonera_1. Modelo en NX-MCD.

Su proyecto tipo en TIA Portal está compuesto de los siguientes elementos.

- **Hardware**
  - CPU (necesaria para desarrollar el software)
  - Rail de tipo ET200M con:
    - Dispositivo IO
      - Tarjeta DI 16x24VDC (2 bytes)
      - Tarjeta DO 16x24VDC/0.5A (2 bytes)

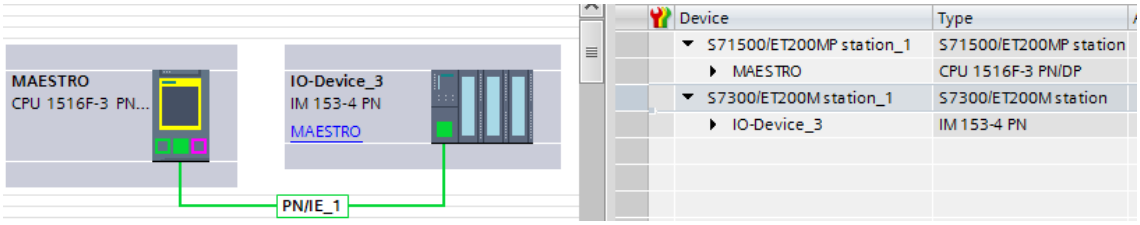


Figura 36. Hardware del proyecto tipo Botonera_1.

- **Software**

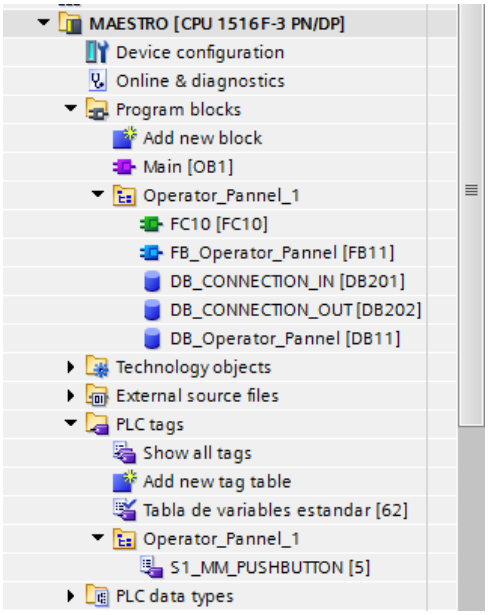


Figura 37. Software del proyecto tipo Botonera_1

  - Tabla de variables *S1_MM_PUSHBUTTON*.
    - *ALARM*: Salida que activa el piloto luminoso en caso de alarma
    - *RESET*: Pulsador para resetear las alarmas.
    - *START*: Pulsador de marcha.
    - *STOP*: Pulsador de paro.
    - *AUTO_MAN*: Selector de modo automático o manual.

S1_MM_PUSHBUTTON

	Name	Data type	Address
1	ALARM	Bool	%Q84.0
2	RESET	Bool	%I84.3
3	START	Bool	%I84.0
4	STOP	Bool	%I84.1
5	AUTO_MAN	Bool	%I84.2
6	<Add new>		

Tabla 3. Tabla de Variables de Botonera_1.

- Bloques de programa
  - FB_Operator_Pannel (FB11). FB de control de las formas de mando.
  - DB_Operator_Pannel (DB11). DB de instancia de FB_Operator_Pannel.
  - DB_CONNECTION_IN. Señales procedentes de otros módulos. Indican si el resto de módulos están en su posición inicial, para permitir iniciar la secuencia de movimientos.

DB_CONNECTION_IN

	Name	Data type
1	Static	
2	IP_MM_RACK	Bool
3	IP_MM_TRANSPORT	Bool

Tabla 4. DB_CONNECTION_IN de Botonera_1.

  - DB_CONNECTION_OUT. Señales que dan las órdenes al resto de módulos.

DB_CONNECTION_OUT

	Name	Data type
1	Static	
2	LSC	Bool
3	INIT	Bool
4	LOUT	Bool
5	OUT_START	Bool

Tabla 5. DB_CONNECTION_OUT de Botonera_1.

  - FC10 (FC10). FC Principal con el Call al FB_Operator_Pannel.

```
1   CALL  "FB_Operator_Pannel", "DB_Operator_Pannel"          %FB11, %DB11
2      PI_BAS         :="DB_CONNECTION_IN".IP_MM_RACK        %DB201.DBX0.0
3      PI_TRA         :="DB_CONNECTION_IN".IP_MM_TRANSPORT   %DB201.DBX0.1
4      AUTO_MAN       :="AUTO_MAN"                             %I84.2
5      START          :="START"                                %I84.0
6      STOP           :="STOP"                                 %I84.1
7      EMERGENCY_STOP :=TRUE                                    TRUE
8      RESET          :="RESET"                                %I84.3
9      ONDA_CUA       :=FALSE                                   FALSE
10     LSC            :="DB_CONNECTION_OUT".LSC              %DB202.DBX0.0
11     INIT           :="DB_CONNECTION_OUT".INIT             %DB202.DBX0.1
12     LOUT           :="DB_CONNECTION_OUT".LOUT             %DB202.DBX0.2
13     OUT_START      :="DB_CONNECTION_OUT".OUT_START        %DB202.DBX0.3
14     ALARM          :="ALARM"                                %Q84.0
```

Figura 38. FC Principal del proyecto tipo Botonera_1.

Se puede observar la utilización de variables locales al módulo (por ejemplo *Auto_Man*), variables que se sustituirán por las variables de conexión de salida de otros módulos (por ejemplo *DB_CONNECTION_IN.IP_MM_RACK*), y variables que podrán leer otros módulos (por ejemplo *DB_CONNECTION_OUT.LSC*).

9.2.1.2 Componente mecatrónico Transporte_1

El transporte es una cinta que recibe el pallet y lo mueve hasta el sensor del centro, posición en la que espera a recibir la operación de la estación.

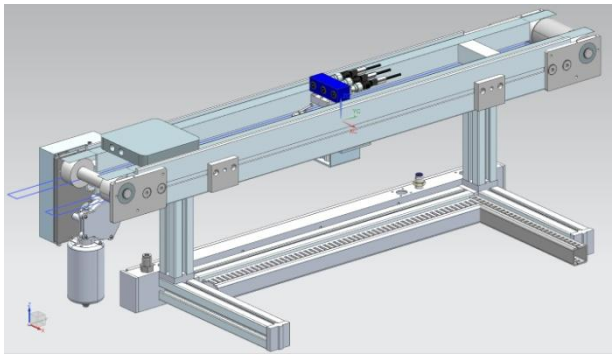


Figura 39. Transporte_1: modelo en NX-MCD.

Su proyecto en TIA Portal está compuesto de los siguientes elementos.

- **Hardware**
  - CPU (necesaria para desarrollar el software)
  - Rail de tipo ET200M con:
    - Dispositivo IO
      - Tarjeta DI 16x24VDC (2 bytes)
      - Tarjeta DO 16x24VDC/0.5A (2 bytes)

- **Software**. Repite la estructura de todos los proyectos tipo, mostrada con más detalle para el caso de Botonera_1. Con la diferencia de que en este caso hay varios FBs a los que se llama desde el FC Principal (*FC_Transport_1*).

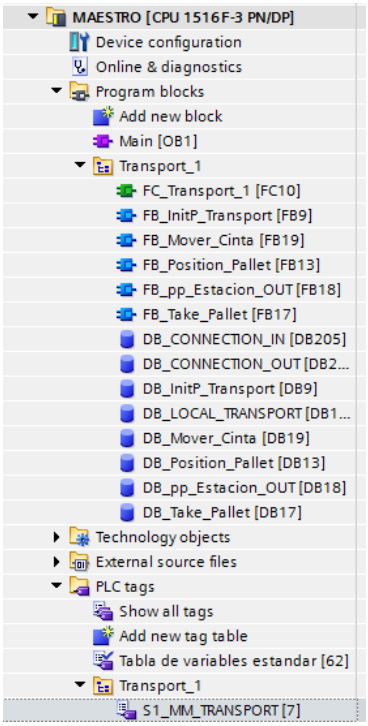


Figura 40. Software del proyecto tipo Transporte_1.

Entre sus señales de E/S en la tabla de variables *S1_MM_TRANSPORT* encontramos las salidas que controlan el motor que mueve la cinta, el sensor que detecta presencia de pallet y tres sensores inductivos con los que se identifica el tipo de pallet.

- *DB_CONNECTION_IN*. Señales de mando que recibe (*LSC*, *LINIT*, *LOUT*, *OUT_START*), señal que indica la finalización de la operación (*S1_OUT_E24*) y señales de presencia de pallet en el transporte anterior y en el siguiente si los hubiese.

	Name	Data type
1	Static	
2	LSC	Bool
3	INIT	Bool
4	LOUT	Bool
5	S1_OUT_E24	Bool
6	ST_pp_SIGUIENTE	Bool
7	ST_pp_ANTERIOR	Bool
8	OUT_START	Bool

Tabla 6. DB_CONNECTION_IN de Transporte_1.

- *DB_CONNECTION_OUT*. Variable de posición inicial (*IP_MM_TRANSPORT*), presencia de pallet (*ST_pp_Estación*), byte que indica el tipo de pallet presente (*Sensores_Palet*), y variables de finalización de funciones.

DB_CONNECTION_OUT		
	Name	Data type
1	Static	
2	IP_MM_TRANSPORT	Bool
3	FIN_UBICAR_PALET	Bool
4	FIN_RETIRAR_PALET	Bool
5	ST_pp_ESTACION	Bool
6	Sensores_Palet	Byte

Tabla 7. DB_CONNECTION_OUT de Transporte_1.

La variable de presencia de pallet se detecta con un sensor (variable en la tabla de variables), pero para comunicarla a otros módulos hace falta que esté en el *DB_CONNECTION_OUT*. El *FB_pp_Estacion_OUT* se utiliza para copiar la entrada del sensor a la variable del *DB_CONNECTION_OUT*.

9.2.1.3 Componente mecatrónico Bastidor_3

Engloba todo el proceso necesario para la colocación de la tapa. Un sensor detecta la presencia de tapa a la salida del alimentador para colocarla después sobre el plato divisor. Para detectar el tipo de tapa se utiliza un sensor inductivo, otro capacitivo y otro fotoeléctrico. Con la combinación de la información de estos sensores se identifica el tipo de tapa.

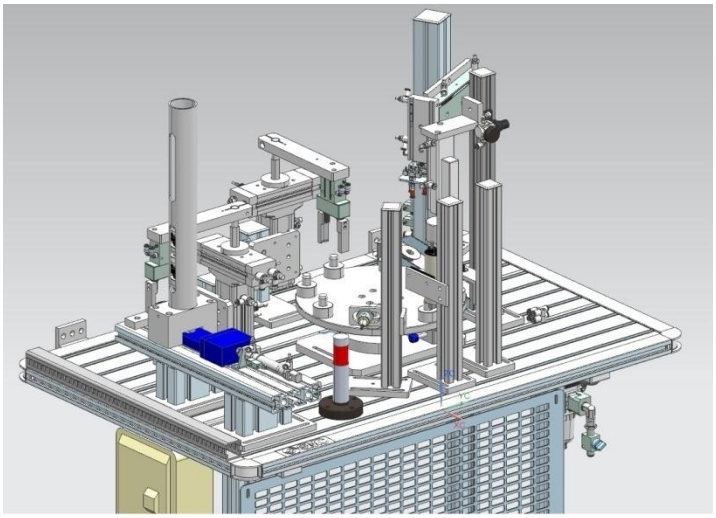


Figura 41. Bastidor_3. Modelo en NX-MCD.

Su proyecto en TIA Portal está compuesto de los siguientes elementos.

- **Hardware**
  - CPU (necesaria para desarrollar el software)
  - Rail de tipo ET200M con:
    - Dispositivo IO
      - 2x Tarjeta DI 16x24VDC (4 bytes)
      - Tarjeta DO 16x24VDC/0.5A (2 bytes)

- **Software.** Repite la estructura de todos los proyectos tipo. En su tabla de variables se encuentran las señales de control de los accionamientos (pinzas, cilindros…) y las entradas de los distintos sensores.
  - *DB_CONNECTION_IN*. Recibe las señales de mando, el código de identificación del tipo de pallet (*Sensores_Palet*) con el que podrá comprobar si la tapa es la adecuada y la señal de que el transporte ha terminado de ubicar el pallet.

DB_CONNECTION_IN		
	Name	Data type
1	Static	
2	LSC	Bool
3	INIT	Bool
4	LOUT	Bool
5	Sensores_Palet	Byte
6	FIN_UBICAR_PALET	Bool

Tabla 8. DB_CONNECTION_IN de Bastidor_3.

  - *DB_CONNECTION_OUT*. Indicación de estado en posición inicial (*S3_PIS3*) y señal de finalización de la operación (*S3_OUT_E310*).

DB_CONNECTION_OUT		
	Name	Data type
1	Static	
2	S3_PIS3	Bool
3	S3_OUT_E310	Bool

Tabla 9. DB_CONNECTION_OUT de Bastidor_3.

9.2.2 Almacenamiento de módulos en base de datos

Con la aplicación de desarrollador, presentada en el apartado 9.1.1, se ha subido a la base de datos la información de los proyectos de todos los módulos mecatrónicos desarrollados.

Algunas observaciones al respecto:

- El tiempo que tarda la aplicación en realizar este proceso no es fundamental, ya que es solo el almacenamiento.

- El tiempo total empleado para la exportación de cada módulo es del orden de un minuto, dependiendo del componente mecatrónico y de las caracteristicas del PC utilizado.

- La parte que más tiempo consume es la utilización de TIA Portal, tanto para abrirlo como para realizar la exportación. Una vez termina esto, el tiempo de subida de los archivos XML y AML a la base de datos es prácticamente despreciable.

9.2.3 **Definición de máquina: Estación 3**

Con la aplicación presentada en el apartado 9.1.2 se ha definido la Estación 3.

Tipo	Prefijo	Nombre	IP inicial	Dirección inicial de E	Dirección inicial de S
Botonera_1	O3	Botonera	192.168.0.140	0	0
Bastidor_3	R3	Bastidor	192.168.0.141	2	2
Transport_1	T3	Transporte	192.168.0.142	6	6

Tabla 10. Definición de la Estación 3: selección de módulos.

A la configuración base de CPU se le ha asignado el prefijo C1 y la IP 192.168.0.120. No tiene E/S por lo que no hay que especificar esas direcciones.

Las relaciones a definir entre módulos se muestran en la Tabla 12.

Módulo	Variable de Conexión Out		Módulo	Variable de Conexión In
Botonera	*LSC*	→	Bastidor	*LSC*
		→	Transporte	*LSC*
	INIT	→	Bastidor	*INIT*
		→	Transporte	*INIT*
	LOUT	→	Bastidor	*LOUT*
		→	Transporte	*LOUT*
	OUT_START	→	Transporte	*OUT_START*
Bastidor	*S3_PIS3*	→	Botonera	*IP_MM_RACK*
	S3_OUT_E310	→	Transporte	*S1_OUT_E24*
Transporte	*IP_MM_TRANSPORT*	→	Botonera	*IP_MM_TRANSPORT*
	FIN_UBICAR_PALLET	→	Bastidor	*FIN_UBICAR_PALLET*
	Sensores_Palet	→	Bastidor	*Sensores_Palet*

Tabla 11. Definición de la Estación 3: relaciones entre módulos.

Resumiendo estas relaciones, puede decirse que la botonera recibe el estado (posición inicial) del resto de módulos y envía al resto de módulos las señales de mando. Además el transporte envía la indicación de que hay un pallet preparado y el código de identificación de ese pallet al bastidor. El bastidor también avisa de la finalización de su operación al transporte. Las entradas a Transporte *ST_pp_SIGUIENTE* y *ST_pp_Anterior* quedan sin relación porque no hay transporte anterior y siguiente en este caso.

En la Figura 41 se muestra el XML resultado de la definición. Cada módulo contiene su Dispositivo IO con sus direcciones, dos relaciones con variables de conexión de entrada en Botonera, cinco en Transporte y otras cinco en Bastidor.

```
<?xml version="1.0" encoding="UTF-8"?>
- <Machine id="Estacion3">
  - <OperatorPannel id="Botonera" type="Operator_Pannel_1" prefix="O3">
      <IO_Device startOutputAddress="0" startInputAddress="0" ipAddress="192.168.000.140" Name="O3_IO-Device_3"/>
      <Relation variableOut="IP_MM_TRANSPORT" sourceComponent="T3" variableIn="IP_MM_TRANSPORT"/>
      <Relation variableOut="S3_PIS3" sourceComponent="R3" variableIn="IP_MM_RACK"/>
    </OperatorPannel>
  - <Transport id="Transporte" type="Transport_1" prefix="T3">
      <IO_Device startOutputAddress="6" startInputAddress="6" ipAddress="192.168.000.142" Name="T3_IO-Device_2"/>
      <Relation variableOut="LSC" sourceComponent="O3" variableIn="LSC"/>
      <Relation variableOut="INIT" sourceComponent="O3" variableIn="INIT"/>
      <Relation variableOut="LOUT" sourceComponent="O3" variableIn="LOUT"/>
      <Relation variableOut="OUT_START" sourceComponent="O3" variableIn="OUT_START"/>
      <Relation variableOut="S3_OUT_E310" sourceComponent="R3" variableIn="S1_OUT_E24"/>
    </Transport>
  - <Rack id="Bastidor" type="Bastidor_3" prefix="R3">
      <IO_Device startOutputAddress="2" startInputAddress="2" ipAddress="192.168.000.141" Name="R3_IO-Device_5"/>
      <Relation variableOut="LSC" sourceComponent="O3" variableIn="LSC"/>
      <Relation variableOut="INIT" sourceComponent="O3" variableIn="INIT"/>
      <Relation variableOut="LOUT" sourceComponent="O3" variableIn="LOUT"/>
      <Relation variableOut="Sensores_Palet" sourceComponent="T3" variableIn="Sensores_Palet"/>
      <Relation variableOut="FIN_UBICAR_PALET" sourceComponent="T3" variableIn="FIN_UBICAR_PALET"/>
    </Rack>
    <CPU cpuIP="192.168.000.120" cpuPrefix="C1" cpuType="CPU1"/>
</Machine>
```

Figura 42. Archivo XML de definición de la Estación 3.

9.2.4 Resultado: proyecto en TIA Portal de Estación 3

Todo el proceso de generación del proyecto e importación, mostrado en la Figura 28, ha tardado alrededor de dos minutos. Igual que sucede con la exportación, este tiempo depende de las prestaciones del PC y del tamaño del proyecto a generar.

A continuación se muestra el resultado del proyecto máquina generado.

- **Hardware**

Se encuentran la CPU conectada como Controlador IO a los Dispositivos IO procedentes de cada módulo. Se puede observar En la Figura 42 el prefijo en el nombre de cada dispositivo y las direcciones IP según la definición de la máquina. También se ve cómo las direcciones de interfaces de comunicación que en el AML se han eliminado las ha generado automáticamente TIA Portal en la importación (Interfaz Profinet 2: 192.168.1.1). Además se ven las direcciones de las señales de E/S del dispositivo de Botonera.

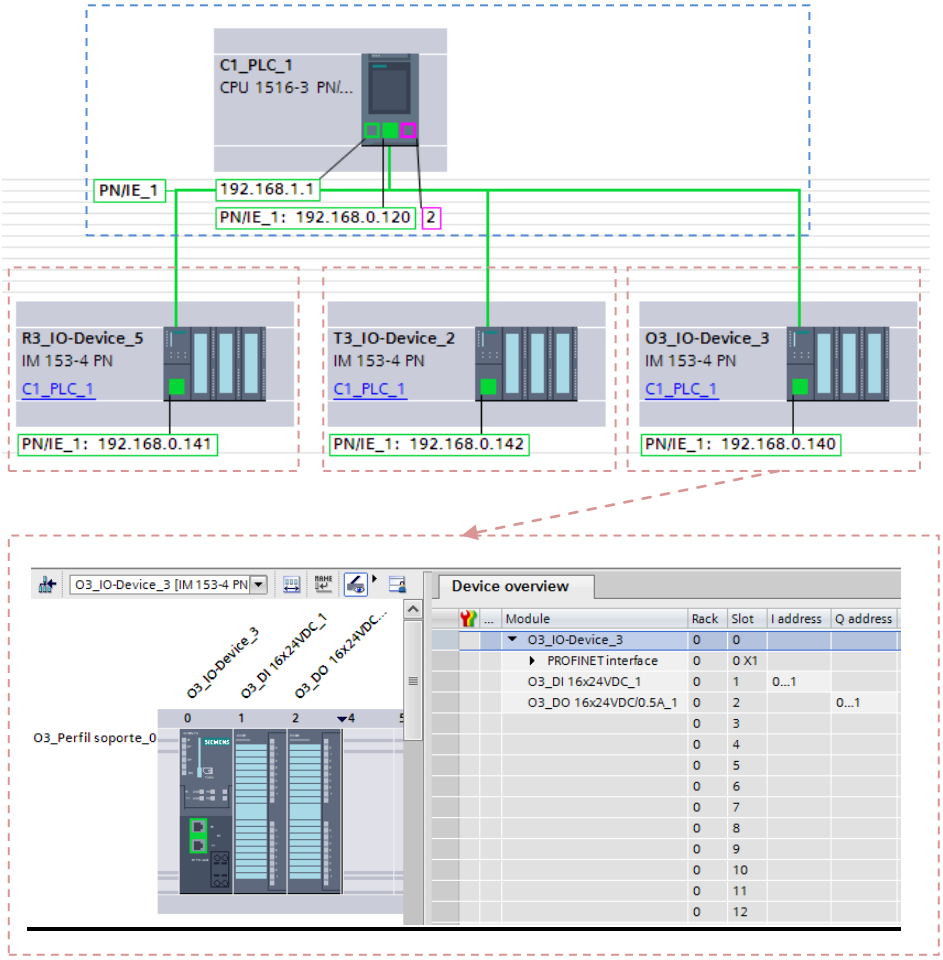


Figura 43. Hardware del proyecto máquina de la Estación 3.

- **Software**

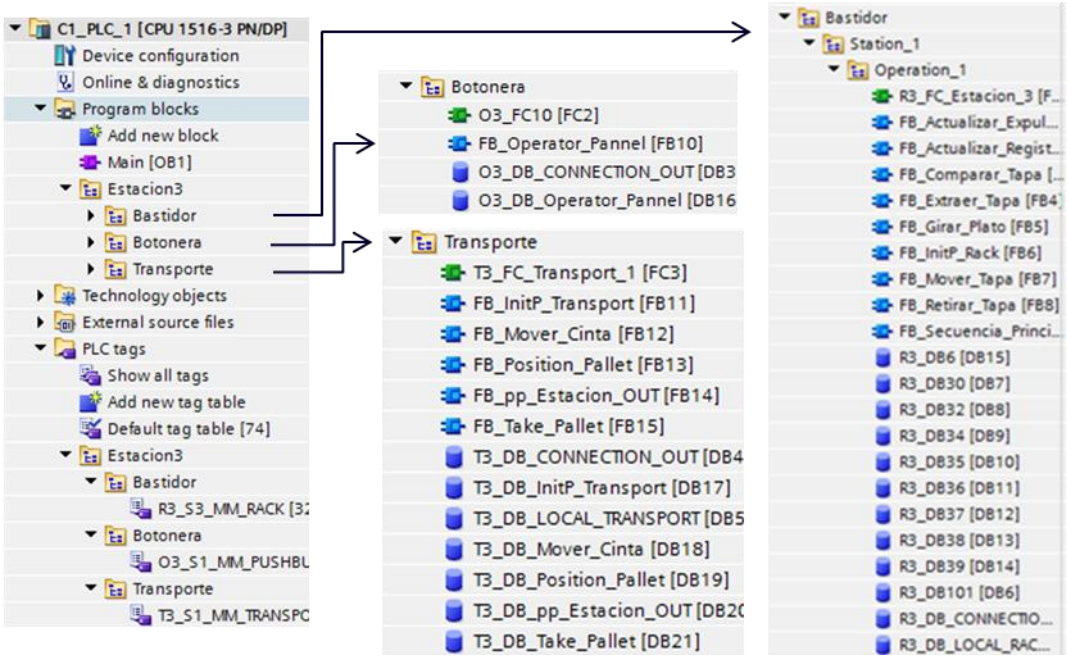


Figura 44. Software del proyecto máquina de la Estación 3.

La Figura 43 muestra la estructura de carpetas del software del proyecto máquina, ordenada por componentes mecatrónicos. Cada bloque tiene el prefijo añadido a su nombre, excepto las funciones genéricas, que en este caso son todas FBs. De los DBs de conexión se conservan solo los *DB_CONNECTION_OUT*.

En las tablas de variables puede verse el prefijo en el nombre, y el cambio de dirección. En la Tabla 13 se muestra el ejemplo del módulo Botonera, con prefijo O3 y direcciones E/S empezando en el byte 0 igual que su Dispositivo IO.

O3_S1_MM_PUSHBUTTON

	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...
1	O3_ALARM	Bool	%Q0.0	☐	☑	☑	☑
2	O3_RESET	Bool	%I0.3	☐	☑	☑	☑
3	O3_START	Bool	%I0.0	☐	☑	☑	☑
4	O3_STOP	Bool	%I0.1	☐	☑	☑	☑
5	O3_AUTO_MAN	Bool	%I0.2	☐	☑	☑	☑

Tabla 12. Tabla de Variables de Botonera_1 integrada en el proyecto máquina de la Estación 3.

En las funciones principales se han adaptado los parámetros actuales.

Proyecto tipo

```
CALL  "FB_Operator_Pannel", "DB_Operator_Pannel"
   PI_BAS          :="DB_CONNECTION_IN".IP_MM_RACK
   PI_TRA          :="DB_CONNECTION_IN".IP_MM_TRANSPORT
   AUTO_MAN        :="AUTO_MAN"
   START           :="START"
   STOP            :="STOP"
   EMERGENCY_STOP  :=TRUE
   RESET           :="RESET"
   ONDA_CUA        :=FALSE
   LSC             :="DB_CONNECTION_OUT".LSC
   INIT            :="DB_CONNECTION_OUT".INIT
   LOUT            :="DB_CONNECTION_OUT".LOUT
   OUT_START       :="DB_CONNECTION_OUT".OUT_START
   ALARM           :="ALARM"
```

Proyecto máquina

```
CALL  "FB_Operator_Pannel", "O3_DB_Operator_Pannel"
   PI_BAS          :="R3_DB_CONNECTION_OUT".S3_PIS3
   PI_TRA          :="T3_DB_CONNECTION_OUT".IP_MM_TRANSPORT
   AUTO_MAN        :="O3_AUTO_MAN"
   START           :="O3_START"
   STOP            :="O3_STOP"
   EMERGENCY_STOP  :=TRUE
   RESET           :="O3_RESET"
   ONDA_CUA        :=FALSE
   LSC             :="O3_DB_CONNECTION_OUT".LSC
   INIT            :="O3_DB_CONNECTION_OUT".INIT
   LOUT            :="O3_DB_CONNECTION_OUT".LOUT
   OUT_START       :="O3_DB_CONNECTION_OUT".OUT_START
   ALARM           :="O3_ALARM"
```

Figura 45. Comparación de la FC Principal de Botonera_1 en el proyecto tipo y el resultado en el proyecto máquina.

En la Figura 44 puede verse la comparación entre la FC principal del proyecto tipo Botonera_1 y como queda esa FC en el proyecto máquina. Se observa como todos los parámetros actuales han sido modificadas con el prefijo. Nótese que los dos primeros parámetros en la llamada al FB en el proyecto tipo son del *DB_CONNECTION_IN* (entradas de otros módulos) y por lo tanto en el proyecto máquina se han sustituido por la variable de salida de otro módulo según las relaciones que se definieron. Por esto los *DB_CONNECTION_IN* no se añaden al proyecto final.

Respecto a los OBs, dado que los proyectos tipo solo utilizaban el OB1, en el proyecto final solo está el OB1 en el que se han juntado todas las llamadas a las FC Principales.

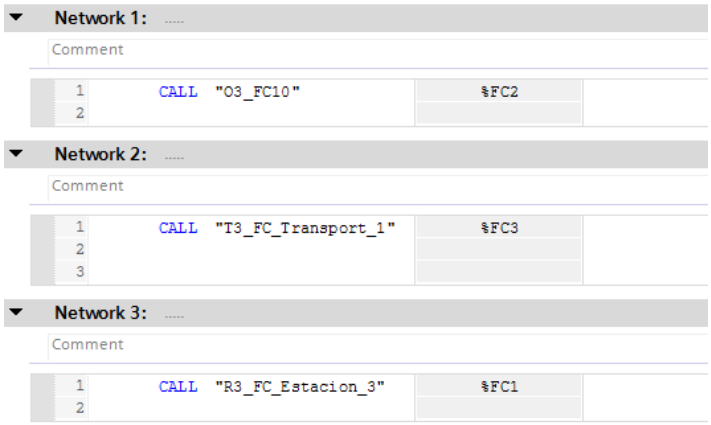


Figura 46. OB1 en el proyecto máquina de la Estación 3.

9.2.5 Validación del proyecto máquina generado

NX-MCD es el software de Siemens diseñado para facilitar el diseño de máquinas. Engloba desde el modelado de piezas en 3D con sus propiedades físicas hasta la simulación de su comportamiento según un control determinada. Permite el desarrollo de componentes de forma modular, pudiendo reutilizar los diseños realizados en nuevos proyectos. Pueden realizarse tanto diseños geométricamente simples y conceptuales como diseños con un elevado nivel de precisión y detalle. Se pueden generar sensores y actuadores para vincular los eventos que se produzcan con señales del modelo.

Con la Estación 3 se ha utilizado su modelo en NX-MCD para comprobar el funcionamiento del proyecto generado. Para ejecutar el programa de control se ha requerido el software PLCSIM Advanced.

PLCSIM Advanced posibilita crear en el propio PC instancias que simulen un PLC. De esta forma, conectándolo con otras aplicaciones se puede ejecutar el proyecto TIA Portal y simular su funcionamiento sin la necesidad de utilizar el hardware real. Se puede crear una instancia de PLC sin más que eligiendo un tipo de CPU y una IP. Una vez esté creada la instancia TIA Portal la detecta en la IP correspondiente. A continuación se carga en ella el proyecto de TIA Portal generado. En TIA Portal, antes de cargar el proyecto en PLCSIM, hay que habilitar la opción de “Permitir simulación al compilar bloques” en las propiedades del proyecto. Es necesario realizar este paso porque en la simulación que se va a realizar no se simula la red, y activando esa opción se indica a TIA Portal que realice la compilación teniendo eso en cuenta.

Con el proyecto de automatización cargado en el PLCSIM, desde NX-MCD se realiza el mapeado entre las señales del modelo y las señales de la instancia del PLC. Las señales de salida del programa de control se asignan a las señales de entrada correspondientes en el modelo y viceversa.

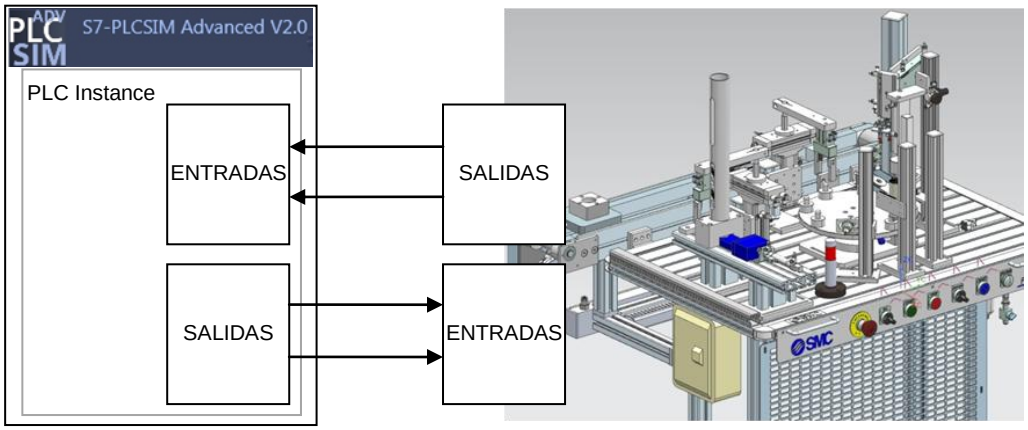


Figura 47. Interacción entre PLCSIM Advanced y NX-MCD.

Tras realizar la simulación se ha podido comprobar que el modelo responde con el comportamiento esperado, confirmando que el programa generado en TIA Portal es correcto.

A continuación, dado que se dispone en el laboratorio de la estación real, se ha realizado la comprobación también sobre ella. Por lo tanto en este caso hay que tener en cuenta que sí se utiliza el Hardware real. Tras cargar en el PLC el proyecto TIA Portal, el PLC busca los Dispositivos IO para cargarles su configuración. Los Dispositivos IO se identifican por su nombre Profinet. El PLC en primer lugar busca los Dispositivos IO conectados a la red según los nombres establecidos en el proyecto para después asignarles su configuración. Por lo tanto, para que el PLC pueda detectarlos, el nombre en Profinet de los Dispositivos IO en el proyecto tiene que coincidir con el de los dispositivos reales. Al importar el Hardware al proyecto desde el AML TIA Portal asigna el nombre Profinet de forma automática. El usuario debe modificarlo con el del dispositivo real antes de cargar el proyecto al PLC.

9.2.6 Otras configuraciones

Se han realizado más pruebas con distintas configuraciones de módulos.

- Estación 1. Es muy similar a la Estación 3. También está compuesta por una Botonera_1 y un Transporte_1, con la diferencia de que en este caso monta un Bastidor_1. En el Bastidor_1 se recibe la pieza base, comprueba si es correcta y si es así la pone en el pallet del transporte para que continúe a la siguiente estación. También se ha probado en simulación y sobre la estación real.

- Dos Estaciones 1. Esta configuración consta de dos Estaciones 1 trabajando en paralelo, independiente la una de la otra pero controladas por una misma CPU. Por lo tanto, en el proyecto a generar hay dos módulos iguales de cada tipo de los usados por la Estación 1. Las relaciones de las variables de los DBs de conexión determinan qué módulos trabajan en conjunto y por lo tanto a qué estación pertenece cada uno.

En la Figura 47 se observa la carpeta generada de elementos comunes (*Common Elements*), puesto que hay módulos que se repiten y por lo tanto usan los mismo FBs.

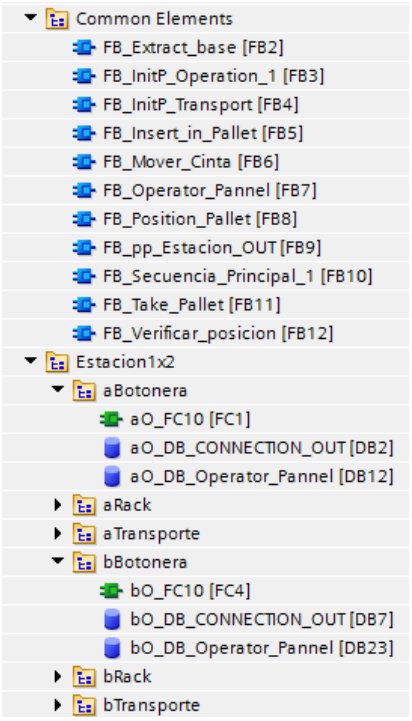


Figura 48. Software del proyecto máquina de dos Estaciones 1.

Este caso solo se ha probado sobre el modelo en simulación obteniendo un funcionamiento adecuado.

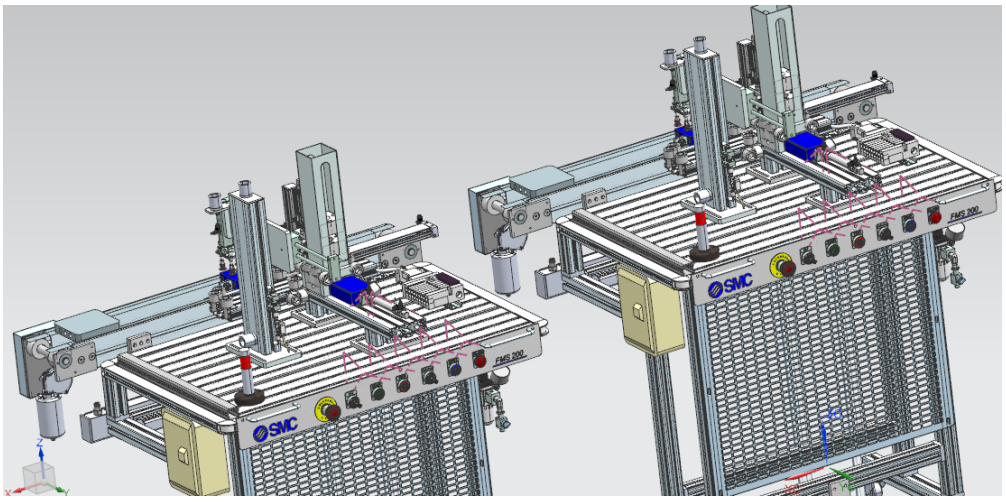


Figura 49. Dos Estaciones 1. Modelo en NX-MCD.

- Estación 1 y Estación 3 unidas por un Transporte_2. El Transporte_2 es una cinta como el Transporte_1 pero programada para que no se detenga el pallet a recibir una operación. Este caso también ha sido probado solo en simulación obteniendo un funcionamiento correcto.

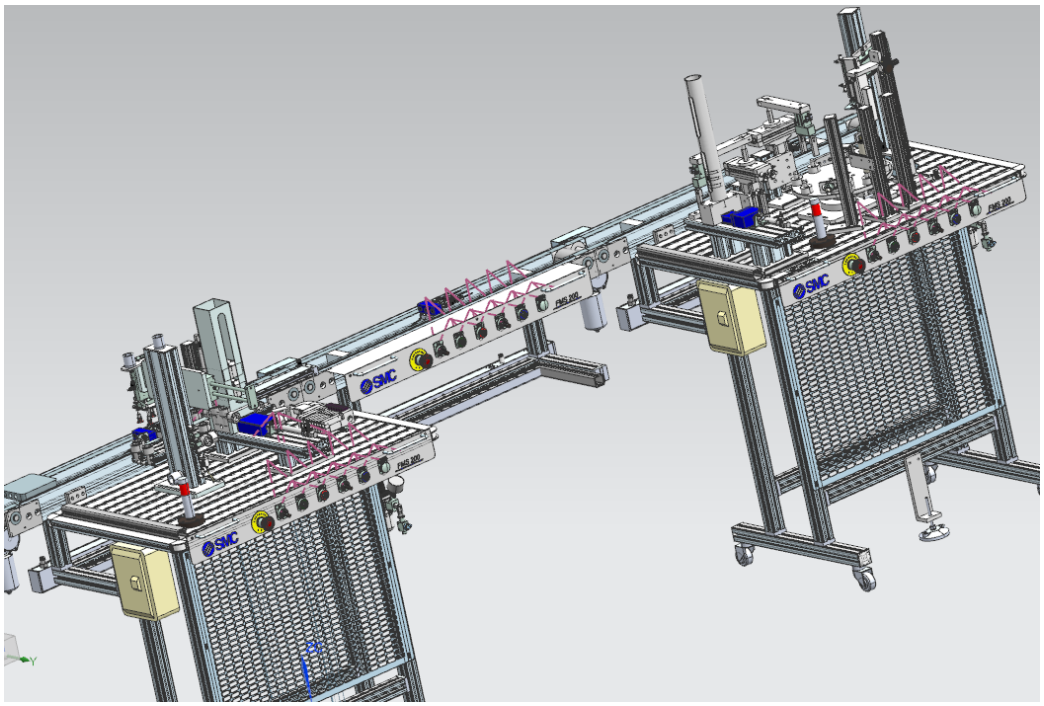


Figura 50. Estación 1 y Estación 3 unidas por un Transporte_2. Modelo en NX-MCD.

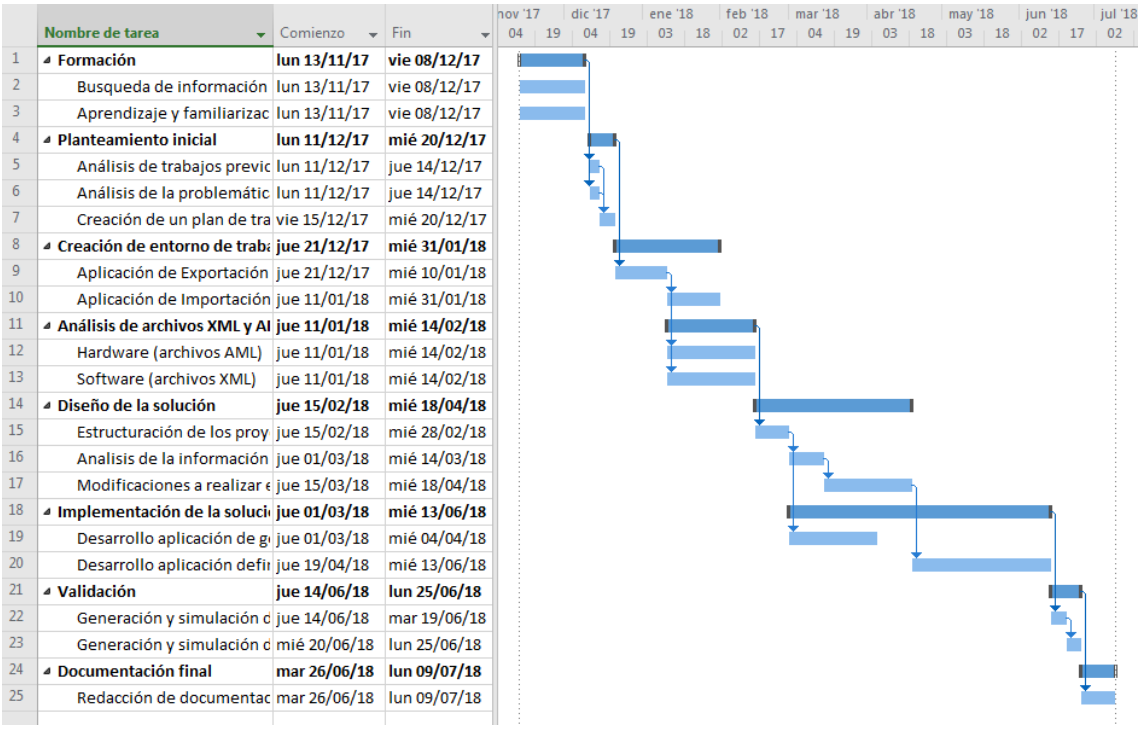
10 Descripción de tareas

A continuación se muestran las tareas realizadas a lo largo de este trabajo.

	TAREA	DESCRIPCIÓN
1	Formación *(20 días)*	Busqueda de información y lectura de manuales sobre TIA Openness, eXist-db, Visual Studio, XML y AML. Familiarización con las herramientas.
2	Planteamiento inicial *(8 días)*	- Análisis de trabajos previos sobre máquinas modulares. - Análisis de la problemática a resolver. - Creación de un plan de trabajo.
3	Creación de entorno de trabajo *(25 días)*	Con TIA Openness, desarrollar aplicaciones para: - Trabajar sobre proyectos TIA Portal. - Exportar proyectos en archivos XML y AML para su posterior análisis. - Importación de archivos XML y AML.
4	Análisis de archivos XML y AML *(25 días)*	- Analizar para cada tipo de elemento del software y del hardware de un proyecto TIA Portal qué información y con qué estructura se exporta. - Analizar qué cambios permite realizar y cómo reacciona TIA Portal al importarlos.
5	Diseño de la solución *(40 días)*	- Establecer la estructura de los proyectos tipo y de los proyectos máquina. - Análisis de información requerida en la definición de nuevas máquinas. - Modificaciones a realizar en los XML y AML para la generación de los proyectos máquina.
6	Implementación de la solución *(70 días)*	- Desarrollo de aplicación de gestión del repositorio de componentes mecatrónicos (base de datos eXist-db). - Desarrollo de aplicación de definición y generación de máquinas.
7	Validación *(8 días)*	- Generación de proyectos TIA Portal para distintas configuraciones de máquinas. - Análisis de los resultados. - Simulación sobre modelos en NX-MCD.
8	Documentación *(10 días)*	Redacción de documentación final.

Tabla 13. Descripción de tareas.

La duración de este proyecto asciende a 161 días, desde noviembre de 2017 hasta julio de 2018.



	Nombre de tarea	Comienzo	Fin
1	**Formación**	**lun 13/11/17**	**vie 08/12/17**
2	Busqueda de información	lun 13/11/17	vie 08/12/17
3	Aprendizaje y familiarizac	lun 13/11/17	vie 08/12/17
4	**Planteamiento inicial**	**lun 11/12/17**	**mié 20/12/17**
5	Análisis de trabajos previc	lun 11/12/17	jue 14/12/17
6	Análisis de la problemátic	lun 11/12/17	jue 14/12/17
7	Creación de un plan de tra	vie 15/12/17	mié 20/12/17
8	**Creación de entorno de traba**	**jue 21/12/17**	**mié 31/01/18**
9	Aplicación de Exportación	jue 21/12/17	mié 10/01/18
10	Aplicación de Importación	jue 11/01/18	mié 31/01/18
11	**Análisis de archivos XML y Al**	**jue 11/01/18**	**mié 14/02/18**
12	Hardware (archivos AML)	jue 11/01/18	mié 14/02/18
13	Software (archivos XML)	jue 11/01/18	mié 14/02/18
14	**Diseño de la solución**	**jue 15/02/18**	**mié 18/04/18**
15	Estructuración de los proy	jue 15/02/18	mié 28/02/18
16	Analisis de la información	jue 01/03/18	mié 14/03/18
17	Modificaciones a realizar e	jue 15/03/18	mié 18/04/18
18	**Implementación de la soluci**	**jue 01/03/18**	**mié 13/06/18**
19	Desarrollo aplicación de g	jue 01/03/18	mié 04/04/18
20	Desarrollo aplicación defi	jue 19/04/18	mié 13/06/18
21	**Validación**	**jue 14/06/18**	**lun 25/06/18**
22	Generación y simulación d	jue 14/06/18	mar 19/06/18
23	Generación y simulación d	mié 20/06/18	lun 25/06/18
24	**Documentación final**	**mar 26/06/18**	**lun 09/07/18**
25	Redacción de documentac	mar 26/06/18	lun 09/07/18

Figura 51. Diagrama de Gantt. Tareas realizadas.

11 Análisis de costes

Este trabajo se desarrolla en paralelo con el curso académico 2017/2018. El tiempo invertido a lo largo del proyecto no ha sido constante ya que se ha visto afectado por la carga de trabajo de actividades ajenas a él. Sin embargo, sí se puede hacer una estimación bastante aproximada de unas 700 horas de proyecto, algo más de 4 horas al día de media.

La base de datos eXist-db es Open Source por lo que no supone un coste añadido. En cuanto a Visual Studio 2015, se ha utilizado la versión gratuita de este entorno de desarrollo.

Análisis de costes			
Amortización	Tasa Horaria	Horas	
Ordenador	0,11€	700	77,78 €
TIA Portal V14 SP1	0,83 €	650	541,67 €
NX-MCD	0,42 €	35	14,58 €
PLCSIM Advanced V2.0	0,49 €	35	17,01 €
Microsoft Office 365	0,11 €	700	73,89 €
PLC + Dispositivos Hardware	0,20 €	25	4,86 €
			729,79 €
Horas internas	Tasa Horaria	Horas	
Director	40 €	175	7.000,00 €
Ingeniero Junior	20 €	700	14.000,00 €
			21.000,00 €
Gastos		Horas	
Consumo eléctrico ordenador	0,2 kW	700	20 €
Material de oficina y otros			80 €
			100 €
Subtotal			21.829,79 €
Indirectos	7%		1.528,09 €
TOTAL			**23.357,88 €**

Tabla 14. Análisis de costes.

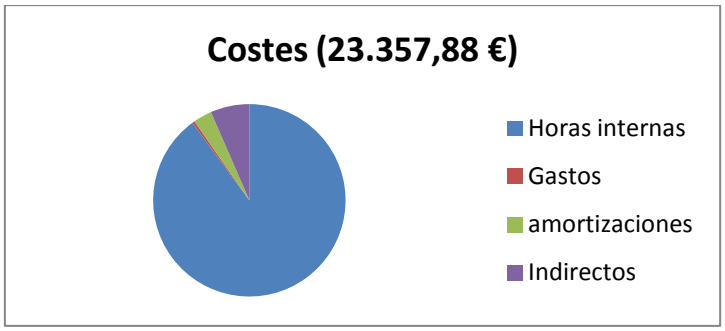


Figura 52. Costes.

Debido a la naturaleza del proyecto no es posible analizar el retorno de la inversión, ya que depende de quién aplique una solución de este tipo y bajo qué circunstancias concretas lo haga.

12 Conclusiones

El paradigma de los sistemas reconfigurables aporta a la industria unas características en consonancia con la flexibilidad buscada en la economía actual. Sin embargo su aplicación no es sencilla ya que afecta no solo al control sino también a la mecánica o la eléctrica, por lo que aun no están sólidamente asentados en la industria. En este trabajo se ha diseñado una metodología que da solución a la generación automática de proyectos de automatización para máquinas modulares con TIA Portal.

Para conseguir el proyecto de automatización de la máquina modular se parte de los proyectos de cada módulo mecatrónico y un proyecto con la configuración base de CPU. Se ha usado una base de datos eXist-db para almacenar la información de los proyectos en archivos exportados con TIA Openness en formato XML y AML.

Con las aplicaciones desarrolladas se ha conseguido implementar la solución, desde la gestión de la base de datos hasta la generación automática del proyecto en TIA Portal pasando por la definición de nuevas máquinas. De esta forma se ha reducido la intervención del usuario en el proceso, eliminando así errores propios del factor humano y permitiendo generar los proyectos de máquinas en pocos minutos.

Para validar los resultados se ha utilizado la célula FMS 200 del laboratorio del Departamento de Ingeniería de Sistemas y Automática. Con los módulos que la forman se han generado los proyectos para distintas configuraciones de máquinas: una estación con módulos distintos, dos estaciones iguales trabajando en paralelo y varias estaciones distintas con módulos del mismo tipo repetidos entre ellas. Los proyectos generados se han probado sobre modelos en simulación, realizados con la herramienta software NX-MCD.

A pesar de que la célula FMS 200 no estaba diseñada en origen de forma modular, se ha conseguido dividir su control en módulos y utilizarlos para validar el buen funcionamiento de la solución desarrollada. Para sacar el máximo potencial a los sistemas reconfigurables hay que tener en cuenta el concepto de modularidad desde sus primeras fases del diseño para conseguir una reusabilidad óptima de los módulos. Por lo tanto las necesidades pueden variar según cada caso.

Este trabajo se ha desarrollado para una CPU conectada por Profinet a los Dispositivos IO. TIA Openness también permite trabajar con otro tipo de redes como Profibus o ASI, además de con interfaces HMI. Esto puede abrir nuevos caminos para trabajos futuros que complementen la solución planteada, buscando ofrecer un mayor abanico de recursos en la programación de máquinas modulares.

13 Fuentes de información

[1] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, H. Van Brussel. (1999). “Reconfigurable Manufacturing Systems”, *CIRP Annals*, vol 48, Issue 2, pp. 527-540.

[2] Y. Koren, M. Shpitalni (2010). “Design of reconfigurable manufacturing systems”, *Journal of Manufacturing Systems,* vol 29, Issue 4, pp. 130-141.

[3] SPRI, (17 mayo, 2018) “El Gobierno Vasco pone en marcha dos programas de ayuda a la Industria 4.0. con 4,7 millones de euros”.

http://www.spri.eus/es/basque-industry-comunicacion/gobierno-vasco-pone-marcha-dos-programas-ayuda-la-industria-4-0-47-millones-euros/

[4] Eckart Uhlmann, Mihir Saoji, Bernd Peukert. (2016). “Principles for Interconnection of Modular Machine Tool Frames”, *Procedia CIRP,* vol 40, pp. 413-418.

[5] C. Maga, N. Jazdi, and P. Göhner. (2011). “Reusable models in industrial automation: experiences in defining appropriate levels of granularity”, IFAC Proceeding. Volumes., vol. 44, no. 1, pp. 9145–9150.

[6] Siemens. (2018). “TIA Portal Openness: Generating a Modular Machine with S7-1500”.

https://support.industry.siemens.com/cs/document/109739678/tia-portal-openness%3A-generating-a-modular-machine-with-s7-1500?dti=0&lc=en-WW

[7] Siemens. (2017). “SIMATIC Automating projects with scripts”.

https://support.industry.siemens.com/cs/document/109477163/simatic-automating-projects-with-scripts?dti=0&lc=en-WW

14 Anexo I. Sinopsis de objetos y parámetros de importación y exportación de CAx

Esquema obtenido del manual de uso de TIA Openness [7].

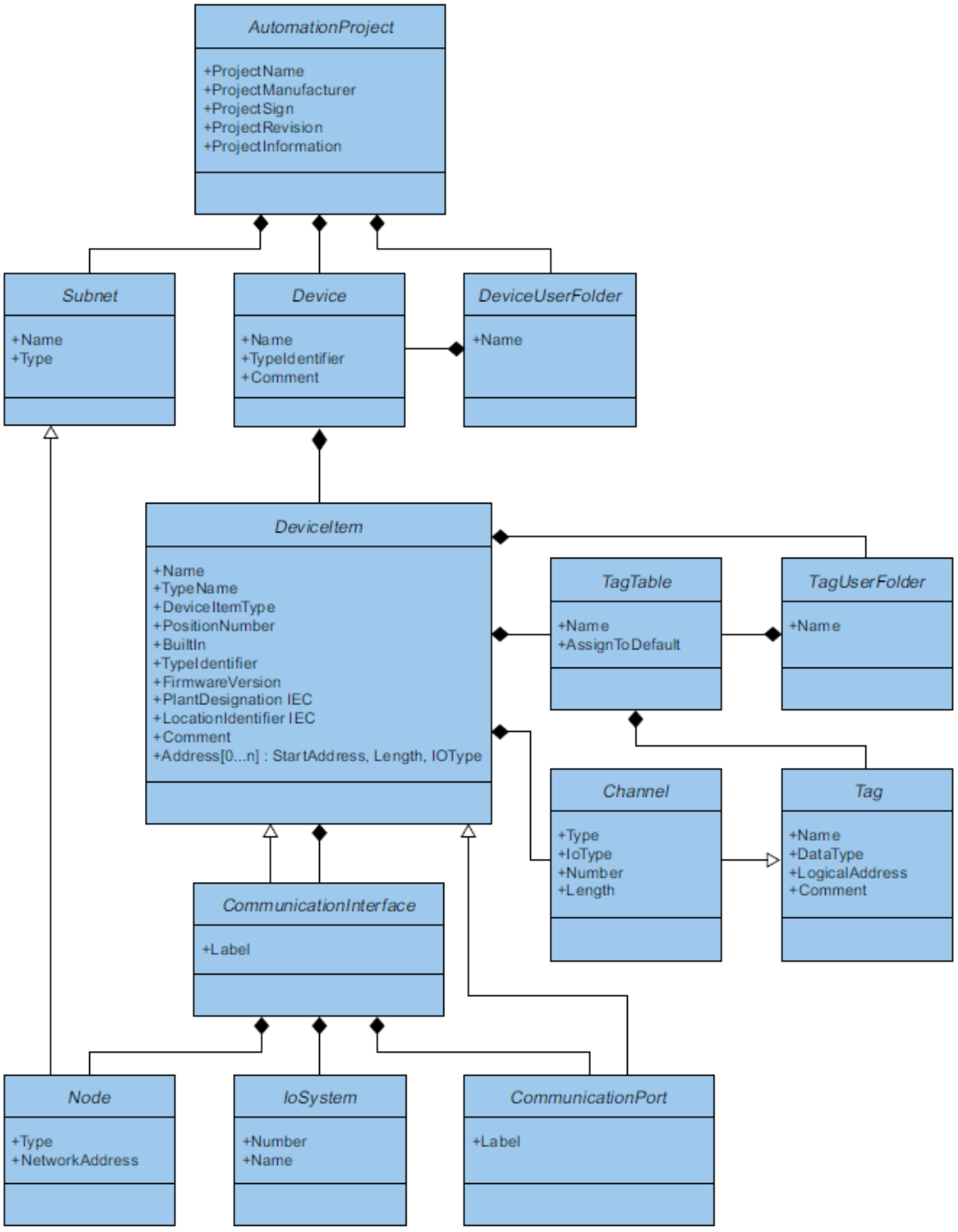


Figura 53. Sinopsis de objetos y parámetros de importación/exportación de CAx.

15 Anexo II. Código de la aplicación para la generación de proyectos de máquinas

A continuación se muestra el código del programa desarrollado para la generación de proyectos de automatización a partir de la información de la base de datos.

En primer lugar se muestra la función *bProject_Click* que se ejecuta al hacer click en el botón "*Create Project*" de la aplicación mostrada en el apartado 9.1.2. En ella se hace uso de las siguientes clases:

- SoftwareFiles.cs, ProgramBlocksFiles.cs, GlobalDBs.cs
  Se utilizan para estructurar los diferentes archivos XML del software de un proyecto exportado y poder recorrerlos de forma ordenada.

- AMLMethods.cs
  Métodos utilizados para realizar las modificaciones en los archivos AML.

- XMLMethods.cs
  Métodos utilizados para realizar las modificaciones en los archivos XML.

- TIAPortalImportMethods.cs
  Métodos utilizados para importar los archivos generados a proyectos de TIA Portal.

- Others.cs
  Otros métodos.

15.1 Función *bProject_Click*

```
private async void bProject_Click(object sender, EventArgs e)
{

    string tempPath = Path.GetTempPath() + @"MachineDef\";
    if (Directory.Exists(tempPath)) Directory.Delete(tempPath, true);
    Directory.CreateDirectory(tempPath);

    string ExistDownloadedPath = tempPath + @"ExistDownloaded\"; // Where the
files will be downloaded from eXistDB
    if (Directory.Exists(ExistDownloadedPath))
Directory.Delete(ExistDownloadedPath, true);
    Directory.CreateDirectory(ExistDownloadedPath);

    string finalDirectory = tempPath + @"Definida\"; // Where the files of the
new proyect will be created
    if (Directory.Exists(finalDirectory)) Directory.Delete(finalDirectory, true);
    Directory.CreateDirectory(finalDirectory + @"Hardware\");
    Directory.CreateDirectory(finalDirectory + @"Software\");

    string ProjectDirectory = ""; // Where the machine's Tia Portal project will
be saved
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
```

```
        if (!Directory.Exists(folderBrowserDialog1.SelectedPath + @"\" +
comboProject.Text.Substring(0, comboProject.Text.Length - 4) + @"\"))
ProjectDirectory = folderBrowserDialog1.SelectedPath;
        else
        {
            MessageBox.Show("That directory already contains a folder named \"" +
comboProject.Text.Substring(0, comboProject.Text.Length - 4) + "\". Select
another directory. ");
            return;
        }
    }
    else return;

    string importHardwareLogPath = tempPath + @"log";
    if (Directory.Exists(importHardwareLogPath))
Directory.Delete(importHardwareLogPath, true);
    Directory.CreateDirectory(importHardwareLogPath);

    string MachDefFolder = tempPath + @"MachineDownloaded\";
    if (Directory.Exists(MachDefFolder)) Directory.Delete(MachDefFolder, true);
    Directory.CreateDirectory(MachDefFolder);

    XmlDocument doc = new XmlDocument();
    doc.Load(String.Concat("http://localhost:8080/exist/rest/db/Machine/",
comboProject.SelectedItem.ToString()));
    doc.Save(MachDefFolder + comboProject.SelectedItem.ToString() + ".xml");

    string MachDefPath = MachDefFolder + comboProject.SelectedItem.ToString() +
".xml";// The path of the machine definition XML

    List<string> DataTypesAdded = new List<string>();
    List<string> UsedLanguages = new List<string>();
    int nextM = 0; // To assign the direction to Marks, Timers and Counters.
    int nextT = 0; // This program asign de directions if the proyect has this
kind of variables,
    int nextC = 0; // but the mechatronic modoules will not use them.

    XmlDocument xmlMachDef = new XmlDocument();
    xmlMachDef.Load(MachDefPath);

    XmlNodeList machineparts =
xmlMachDef.DocumentElement.SelectSingleNode("/Machine").ChildNodes;
    XmlNode CPUnode = null;
    List<XmlNode> modules = new List<XmlNode>();

    #region Checking if the modules exist in the database
    foreach (XmlNode machinepart in machineparts)
    {
        if (machinepart.Name == "CPU")
        {
            CPUnode = machinepart;
        }
        else modules.Add(machinepart);

    }

    string results;

    try
    {
        results = await ExistDB.ExistGet(@"CPU/" +
CPUnode.Attributes["cpuType"].Value);
```

```
    }
    catch (HttpRequestException ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine("Project generation aborted");
        return;
    }

    foreach (XmlNode module in modules)
    {
        string componentType = module.Name;
        string moduleType = module.Attributes["type"].Value;
        try
        {
            results = await ExistDB.ExistGet(@"Types/" + componentType + @"/" +
moduleType);
        }
        catch (HttpRequestException ex)
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine("Project generation aborted");
            return;
        }
    }
    #endregion Checking if the modules exist in the database

    #region CPU
    string machineName =
xmlMachDef.DocumentElement.SelectSingleNode("/Machine").Attributes["id"].Value;
    string cpuType = CPUnode.Attributes["cpuType"].Value;

    string getresults = await ExistDB.ExistGetCollection(@"CPU\" + cpuType,
ExistDownloadedPath);
    string cpuPath = ExistDownloadedPath + @"CPU\" + cpuType;

    string cpuPrefix = CPUnode.Attributes["cpuPrefix"].Value + "_";
    string cpuIpAddress = CPUnode.Attributes["cpuIP"].Value;
    string cpuStartInputAddress = "";
    string cpuStartOutputAddress = "";
    foreach (XmlAttribute attribute in CPUnode.Attributes)
    {
        if (attribute.Name == "cpuStartInputAddress") cpuStartInputAddress =
attribute.Value;
        else if (attribute.Name == "cpuStartOutputAddress") cpuStartOutputAddress
= attribute.Value;
    }



    #region CPU Hardware
    XmlDocument amlCPUBase = new XmlDocument();
    amlCPUBase.Load(GetAMLPath(new DirectoryInfo(cpuPath)).FullName);

    List<string> oldInAddressesCPU = new List<string>(); // Save the previous
addresses
    List<string> newInAddressesCPU = new List<string>(); // and the new addresses
to know how to change the TagTables later.
    List<string> oldOutAddressesCPU = new List<string>(); // Save the previous
addresses
    List<string> newOutAddressesCPU = new List<string>(); // and the new
addresses to know how to change the TagTables later.
```

```
    oldInAddressesCPU = GetInAddresses(amlCPUBase);
    oldOutAddressesCPU = GetOutAddresses(amlCPUBase);
    amlCPUBase = TransformCPUBase(amlCPUBase, cpuIpAddress, cpuStartInputAddress,
cpuStartOutputAddress, cpuPrefix);
    newInAddressesCPU = GetInAddresses(amlCPUBase);
    newOutAddressesCPU = GetOutAddresses(amlCPUBase);
    var InAddressesCPU = oldInAddressesCPU.Zip(newInAddressesCPU, (k, v) => new {
k, v }).ToDictionary(x => x.k, x => x.v);
    var OutAddressesCPU = oldOutAddressesCPU.Zip(newOutAddressesCPU, (k, v) =>
new { k, v }).ToDictionary(x => x.k, x => x.v);

    #endregion CPU Hardware

    #region CPU Software
    SoftwareFiles softwareCPU = new SoftwareFiles(new DirectoryInfo(cpuPath));
    foreach (string language in softwareCPU.GetLanguages())
    {
        if (!UsedLanguages.Contains(language)) UsedLanguages.Add(language);
    }

    #region Data types
    foreach (FileInfo dataType in softwareCPU.userDataTypes)
    {
        bool repeated = false;
        SoftwareFiles machineFiles = new SoftwareFiles(new
DirectoryInfo(finalDirectory));
        foreach (FileInfo machineUDT in machineFiles.userDataTypes)
        {
            if (machineUDT.Name == dataType.Name) //Machine directory already
have that Data Type
            {
                repeated = true;
                if (!Others.GetSoftwareGroupPath(machineUDT,
false).Contains("Common Elements")) //If its not in "Common Elements" yet, move
it there
                {
                    Directory.CreateDirectory(finalDirectory + @"Software\" +
cpuType + @"\" + @"PLC data types\" + @"Common Elements\");
                    machineUDT.MoveTo(finalDirectory + @"Software\" + cpuType +
@"\" + @"PLC data types\" + @"Common Elements\" + machineUDT.Name);
                }
                break;
            }
        }

        if (repeated == false)
        {
            string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"PLC data types\" + GetSoftwareGroupPath(dataType, false);
            Directory.CreateDirectory(dirDef);
            dataType.CopyTo(dirDef + dataType.Name);
        }
    }
    #endregion

    #region TagTables
    foreach (FileInfo tagTableFile in softwareCPU.tagTables)
    {
        XmlDocument xmlTagTable = new XmlDocument();
        xmlTagTable.Load(tagTableFile.FullName);
        xmlTagTable = TransformTagTable(xmlTagTable, cpuPrefix, InAddressesCPU,
OutAddressesCPU, ref nextM, ref nextT, ref nextC);
```

```
        string dirDef = finalDirectory + @"Software\" + cpuType + @"\" + @"PLC
tags\" + GetSoftwareGroupPath(tagTableFile, false);
        Directory.CreateDirectory(dirDef);
        xmlTagTable.Save(dirDef + cpuPrefix + tagTableFile.Name);
    }
    #endregion

    #region PLC Blocks

    #region GlobalDBs

    foreach (FileInfo globalDBFile in
softwareCPU.programBlocks.GlobalDBs.NormalDBs.Concat(softwareCPU.programBlocks.Gl
obalDBs.ConnectionOutDBs))
    {
        XmlDocument xmlGlobalDB = new XmlDocument();
        xmlGlobalDB.Load(globalDBFile.FullName);
        xmlGlobalDB = TransformDB(xmlGlobalDB, cpuPrefix);
        string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + GetSoftwareGroupPath(globalDBFile, false);
        Directory.CreateDirectory(dirDef);
        xmlGlobalDB.Save(dirDef + cpuPrefix + globalDBFile.Name);
    }

    #endregion GlobalDBs

    #region FBs
    foreach (FileInfo FB in softwareCPU.programBlocks.FBs)
    {
        bool repeated = false;
        SoftwareFiles machineFiles = new SoftwareFiles(new
DirectoryInfo(finalDirectory));
        foreach (FileInfo machineFB in machineFiles.programBlocks.FBs)
        {
            if (machineFB.Name == FB.Name) //Machine directory already have that
FB
            {
                repeated = true;
                if (!Others.GetSoftwareGroupPath(machineFB,
false).Contains("Common Elements")) //If its not in "Common Elements" yet, move
it there
                {
                    Directory.CreateDirectory(finalDirectory + @"Software\" +
cpuType + @"\" + @"Program blocks\" + @"Common Elements\");
                    machineFB.MoveTo(finalDirectory + @"Software\" + cpuType +
@"\" + @"Program blocks\" + @"Common Elements\" + machineFB.Name);
                }
                break;
            }
        }

        if (repeated == false)
        {
            XmlDocument xmlFB = new XmlDocument();
            xmlFB.Load(FB.FullName);
            xmlFB = TransformFB(xmlFB);

            string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + GetSoftwareGroupPath(FB, false);
            Directory.CreateDirectory(dirDef);
            xmlFB.Save(dirDef + FB.Name);
        }
```

```
    }
    #endregion FBs

    #region InstanceDBs
    foreach (FileInfo instanceDBFile in softwareCPU.programBlocks.InstanceDBs)
    {
        XmlDocument xmlInstanceDB = new XmlDocument();
        xmlInstanceDB.Load(instanceDBFile.FullName);
        xmlInstanceDB = TransformDB(xmlInstanceDB, cpuPrefix);
        string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + GetSoftwareGroupPath(instanceDBFile, false);
        Directory.CreateDirectory(dirDef);
        xmlInstanceDB.Save(dirDef + cpuPrefix + instanceDBFile.Name);
    }
    #endregion InstanceDBs

    #region FCs
    foreach (FileInfo FCFile in softwareCPU.programBlocks.FCs)
    {
        XmlDocument xmlFC = new XmlDocument();
        xmlFC.Load(FCFile.FullName);
        xmlFC = TransformFC(xmlFC, cpuPrefix);

        XmlNodeList globalVariables =
xmlFC.SelectNodes("//*[@Scope='GlobalVariable']");
        if (globalVariables.Count == 0) //Generic Function. Only one is needed
for the whole project
        {
            bool repeated = false;
            SoftwareFiles machineFiles = new SoftwareFiles(new
DirectoryInfo(finalDirectory));
            foreach (FileInfo machineFC in machineFiles.programBlocks.FCs)
            {
                if (machineFC.Name == FCFile.Name) //Machine directory already
have that Data Type
                {
                    repeated = true;
                    if (!Others.GetSoftwareGroupPath(machineFC,
false).Contains("Common Elements")) //If its not in "Common Elements" yet, move
it there
                    {
                        Directory.CreateDirectory(finalDirectory + @"Software\" +
cpuType + @"\" + @"Program blocks\" + @"Common Elements\");
                        machineFC.MoveTo(finalDirectory + @"Software\" + cpuType
+ @"\" + @"Program blocks\" + @"Common Elements\" + machineFC.Name);
                    }
                    break;
                }
            }
            if (repeated == false)
            {
                string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + GetSoftwareGroupPath(FCFile, false);
                Directory.CreateDirectory(dirDef);
                xmlFC.Save(dirDef + FCFile.Name);
            }
        }
        else //FCFile uses global Variables (DBs or I/O). One FC of this type for
each module is needed.
        {
            string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + GetSoftwareGroupPath(FCFile, false);
```

```
            Directory.CreateDirectory(dirDef);
            xmlFC.Save(dirDef + cpuPrefix + FCFile.Name);
        }
    }
    #endregion FCs

    #region OBs
    foreach (FileInfo OB in softwareCPU.programBlocks.OBs)
    {
        bool isInMachine = false;
        SoftwareFiles machineFiles = new SoftwareFiles(new
DirectoryInfo(finalDirectory));
        foreach (FileInfo machineOB in machineFiles.programBlocks.OBs)
        {
            if (machineOB.Name == OB.Name) //Machine directory already have the
OB
            {
                isInMachine = true;
                if (machineOB.DirectoryName != finalDirectory + @"Software\" +
cpuType + @"\" + @"Program blocks\") //If its not in the main folder yet, move it
there
                {
                    Directory.CreateDirectory(finalDirectory + @"Software\" +
cpuType + @"\" + @"Program blocks\");
                    machineOB.MoveTo(finalDirectory + @"Software\" + cpuType +
@"\" + @"Program blocks\" + machineOB.Name);
                }
                XmlDocument xmlMachineOB = new XmlDocument();
                xmlMachineOB.Load(machineOB.FullName);
                XmlDocument xmlOB = new XmlDocument();
                xmlOB.Load(OB.FullName);
                xmlMachineOB = AddToOB(xmlMachineOB, xmlOB, cpuPrefix);
                xmlMachineOB.Save(finalDirectory + @"Software\" + cpuType + @"\"
+ @"Program blocks\" + machineOB.Name);
            }
        }

        if (isInMachine == false)
        {
            XmlDocument xmlOB = new XmlDocument();
            xmlOB.Load(OB.FullName);
            xmlOB = TransformOB(xmlOB, cpuPrefix);

            Directory.CreateDirectory(finalDirectory + @"Software\" + cpuType +
@"\" + @"Program blocks\");
            xmlOB.Save(finalDirectory + @"Software\" + cpuType + @"\" + @"Program
blocks\" + OB.Name);
        }
    }
    #endregion OBs

    #endregion PLC Blocks


    #endregion CPU Software

    Directory.Delete(ExistDownloadedPath + @"CPU\", true);
    #endregion CPU

    #region Mechatronic Modules

    foreach (XmlNode module in modules)
```

```
    {
        string componentType = module.Name;
        string moduleName = module.Attributes["id"].Value;
        string moduleType = module.Attributes["type"].Value;
        string firstIP =
module.SelectSingleNode(".//*[@ipAddress]").Attributes["ipAddress"].Value;
        string prefix = module.Attributes["prefix"].Value + "_";
        string startInputAddress =
module.SelectSingleNode(".//*[@startInputAddress]").Attributes["startInputAddress
"].Value;
        string startOutputAddress =
module.SelectSingleNode(".//*[@startOutputAddress]").Attributes["startOutputAddre
ss"].Value;
        XmlNodeList xmlRelations = module.SelectNodes(".//Relation");

        getresults = await ExistDB.ExistGetCollection(@"Types\" + componentType +
@"\" + moduleType, ExistDownloadedPath);
        string modulepath = ExistDownloadedPath + @"Types\" + componentType +
@"\" + moduleType;

        #region Mechatronic Module Hardware
        XmlDocument amlMod = new XmlDocument();
        amlMod.Load(GetAMLPath(new DirectoryInfo(modulepath)).FullName); //Load
the AML file of the module
        XmlDocument amlUnDev = new XmlDocument();

amlUnDev.LoadXml(amlMod.DocumentElement.SelectSingleNode("//SupportedRoleClass[@R
efRoleClassPath='AutomationProjectConfigurationRoleClassLib/DeviceUserFolder']").
ParentNode.OuterXml);

        List<string> oldInAddresses = new List<string>(); // Save the previous
addresses
        List<string> newInAddresses = new List<string>(); // and the new
addresses to know how to change the TagTables later.
        List<string> oldOutAddresses = new List<string>(); // Save the previous
addresses
        List<string> newOutAddresses = new List<string>(); // and the new
addresses to know how to change the TagTables later.

        oldInAddresses = GetInAddresses(amlUnDev);
        oldOutAddresses = GetOutAddresses(amlUnDev);

        amlUnDev = TransformUngroupedDevices(amlUnDev, firstIP,
startInputAddress, startOutputAddress, prefix);
        amlCPUBase = InsertUngroupedDevices(amlCPUBase, amlUnDev);
        newInAddresses = GetInAddresses(amlUnDev);
        newOutAddresses = GetOutAddresses(amlUnDev);

        var InAddresses = oldInAddresses.Zip(newInAddresses, (k, v) => new { k, v
}).ToDictionary(x => x.k, x => x.v);
        var OutAddresses = oldOutAddresses.Zip(newOutAddresses, (k, v) => new {
k, v }).ToDictionary(x => x.k, x => x.v);

        #endregion Mechatronic Module Hardware

        #region Mechatronic Module Software
        SoftwareFiles software = new SoftwareFiles(new
DirectoryInfo(modulepath));
        foreach (string language in software.GetLanguages())
        {
            if (!UsedLanguages.Contains(language)) UsedLanguages.Add(language);
        }
```

```
        #region Data types
        foreach (FileInfo dataType in software.userDataTypes)
        {
            bool repeated = false;
            SoftwareFiles machineFiles = new SoftwareFiles(new
DirectoryInfo(finalDirectory));
            foreach (FileInfo machineUDT in machineFiles.userDataTypes)
            {
                if (machineUDT.Name == dataType.Name) //Machine directory already
have that Data Type
                {
                    repeated = true;
                    if (!Others.GetSoftwareGroupPath(machineUDT,
false).Contains("Common Elements")) //If its not in "Common Elements" yet, move
it there
                    {
                        Directory.CreateDirectory(finalDirectory + @"Software\" +
cpuType + @"\" + @"PLC data types\" + @"Common Elements\");
                        machineUDT.MoveTo(finalDirectory + @"Software\" + cpuType
+ @"\" + @"PLC data types\" + @"Common Elements\" + machineUDT.Name);
                    }
                    break;
                }
            }

            if (repeated == false)
            {
                string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"PLC data types\" + machineName + @"\" + moduleName + @"\" +
GetSoftwareGroupPath(dataType, false);
                Directory.CreateDirectory(dirDef);
                dataType.CopyTo(dirDef + dataType.Name);
            }
        }
        #endregion

        #region TagTables
        foreach (FileInfo tagTableFile in software.tagTables)
        {
            XmlDocument xmlTagTable = new XmlDocument();
            xmlTagTable.Load(tagTableFile.FullName);
            xmlTagTable = TransformTagTable(xmlTagTable, prefix, InAddresses,
OutAddresses, ref nextM, ref nextT, ref nextC);
            string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"PLC tags\" + machineName + @"\" + moduleName + @"\" +
GetSoftwareGroupPath(tagTableFile, false);
            Directory.CreateDirectory(dirDef);
            xmlTagTable.Save(dirDef + prefix + tagTableFile.Name);
        }
        #endregion

        #region PLC Blocks

        #region GlobalDBs

        foreach (FileInfo globalDBFile in
software.programBlocks.GlobalDBs.NormalDBs.Concat(software.programBlocks.GlobalDB
s.ConnectionOutDBs))
        {
            // foreach normalDB and CONNECTIONOutDB. Ignore CONNECTIONInDB.
            XmlDocument xmlGlobalDB = new XmlDocument();
```

```
            xmlGlobalDB.Load(globalDBFile.FullName);
            xmlGlobalDB = TransformDB(xmlGlobalDB, prefix);
            string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + machineName + @"\" + moduleName + @"\" +
GetSoftwareGroupPath(globalDBFile, false);
            Directory.CreateDirectory(dirDef);
            xmlGlobalDB.Save(dirDef + prefix + globalDBFile.Name);
        }

        #endregion GlobalDBs

        #region FBs
        foreach (FileInfo FB in software.programBlocks.FBs)
        {
            bool repeated = false;
            SoftwareFiles machineFiles = new SoftwareFiles(new
DirectoryInfo(finalDirectory));
            foreach (FileInfo machineFB in machineFiles.programBlocks.FBs)
            {
                if (machineFB.Name == FB.Name) //Machine directory already have
that FB
                {
                    repeated = true;
                    if (!Others.GetSoftwareGroupPath(machineFB,
false).Contains("Common Elements")) //If its not in "Common Elements" yet, move
it there
                    {
                        Directory.CreateDirectory(finalDirectory + @"Software\" +
cpuType + @"\" + @"Program blocks\" + @"Common Elements\");
                        machineFB.MoveTo(finalDirectory + @"Software\" + cpuType
+ @"\" + @"Program blocks\" + @"Common Elements\" + machineFB.Name);
                    }
                    break;
                }
            }

            if (repeated == false)
            {
                XmlDocument xmlFB = new XmlDocument();
                xmlFB.Load(FB.FullName);
                xmlFB = TransformFB(xmlFB);

                string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + machineName + @"\" + moduleName + @"\" +
GetSoftwareGroupPath(FB, false);
                Directory.CreateDirectory(dirDef);
                xmlFB.Save(dirDef + FB.Name);
            }
        }
        #endregion FBs

        #region InstanceDBs
        foreach (FileInfo instanceDBFile in software.programBlocks.InstanceDBs)
        {
            XmlDocument xmlInstanceDB = new XmlDocument();
            xmlInstanceDB.Load(instanceDBFile.FullName);
            xmlInstanceDB = TransformDB(xmlInstanceDB, prefix);
            string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + machineName + @"\" + moduleName + @"\" +
GetSoftwareGroupPath(instanceDBFile, false);
            Directory.CreateDirectory(dirDef);
            xmlInstanceDB.Save(dirDef + prefix + instanceDBFile.Name);
```

```
        }
        #endregion InstanceDBs

        #region FCs Main

        foreach (FileInfo FCFile in software.programBlocks.FCMains)
        {
            XmlDocument xmlFC = new XmlDocument();
            xmlFC.Load(FCFile.FullName);
            xmlFC = TransformFC(xmlFC, prefix);
            xmlFC = TransformFCMainConnection(xmlFC, xmlRelations);

            string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + machineName + @"\" + moduleName + @"\" +
GetSoftwareGroupPath(FCFile, false);
            Directory.CreateDirectory(dirDef);
            xmlFC.Save(dirDef + prefix + FCFile.Name);
        }

        #endregion FCs Main

        #region FCs Generic

        foreach (FileInfo FCFile in software.programBlocks.FCs)
        {
            XmlDocument xmlFC = new XmlDocument();
            xmlFC.Load(FCFile.FullName);
            xmlFC = TransformFC(xmlFC, prefix);

            bool repeated = false;
            SoftwareFiles machineFiles = new SoftwareFiles(new
DirectoryInfo(finalDirectory));
            foreach (FileInfo machineFC in machineFiles.programBlocks.FCs)
            {
                if (machineFC.Name == FCFile.Name) //Machine directory already
have that generic FC
                {
                    repeated = true;
                    if (!Others.GetSoftwareGroupPath(machineFC,
false).Contains("Common Elements")) //If its not in "Common Elements" yet, move
it there
                    {
                        Directory.CreateDirectory(finalDirectory + @"Software\" +
cpuType + @"\" + @"Program blocks\" + @"Common Elements\");
                        machineFC.MoveTo(finalDirectory + @"Software\" + cpuType
+ @"\" + @"Program blocks\" + @"Common Elements\" + machineFC.Name);
                    }
                    break;
                }
            }

            if (repeated == false)
            {
                string dirDef = finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + machineName + @"\" + moduleName + @"\" +
GetSoftwareGroupPath(FCFile, false);
                Directory.CreateDirectory(dirDef);
                xmlFC.Save(dirDef + FCFile.Name);
            }
        }
        #endregion FCs Generic
```

```
        #region OBs
        foreach (FileInfo OB in software.programBlocks.OBs)
        {
            bool isInMachine = false;
            SoftwareFiles machineFiles = new SoftwareFiles(new
DirectoryInfo(finalDirectory));
            foreach (FileInfo machineOB in machineFiles.programBlocks.OBs)
            {
                if (machineOB.Name == OB.Name) //Machine directory already have
the OB with that number
                {
                    isInMachine = true;
                    if (machineOB.DirectoryName != finalDirectory + @"Software\"
+ cpuType + @"\" + @"Program blocks\") //If its not in the main folder yet, move
it there
                    {
                        Directory.CreateDirectory(finalDirectory + @"Software\" +
cpuType + @"\" + @"Program blocks\");
                        machineOB.MoveTo(finalDirectory + @"Software\" + cpuType
+ @"\" + @"Program blocks\" + machineOB.Name);
                    }
                    XmlDocument xmlMachineOB = new XmlDocument();
                    xmlMachineOB.Load(machineOB.FullName);
                    XmlDocument xmlOB = new XmlDocument();
                    xmlOB.Load(OB.FullName);
                    xmlMachineOB = AddToOB(xmlMachineOB, xmlOB, prefix);
                    xmlMachineOB.Save(finalDirectory + @"Software\" + cpuType +
@"\" + @"Program blocks\" + machineOB.Name);
                }
            }

            if (isInMachine == false)
            {
                XmlDocument xmlOB = new XmlDocument();
                xmlOB.Load(OB.FullName);
                xmlOB = TransformOB(xmlOB, prefix);

                Directory.CreateDirectory(finalDirectory + @"Software\" + cpuType
+ @"\" + @"Program blocks\");
                xmlOB.Save(finalDirectory + @"Software\" + cpuType + @"\" +
@"Program blocks\" + OB.Name);
            }
        }
        #endregion OBs

        #endregion PLC Blocks

        #endregion Mechatronic Module Software

        Directory.Delete(ExistDownloadedPath + @"Types\", true);
    }
    #endregion Mechatronic Modules

    #region Connections in AML and Save AML
    //amlCPUBase = ConnectCPUtoSubnet(amlCPUBase);
    amlCPUBase = ConnectUnDevstoSubnet(amlCPUBase);
    amlCPUBase = ConnectUnDevsToIoSystem(amlCPUBase);
    amlCPUBase.Save(finalDirectory + @"Hardware\" + machineName + ".aml");
    #endregion

    Console.WriteLine("Waiting for Tia Portal authorization...");
    using (TiaPortal tiaPortal = new TiaPortal(TiaPortalMode.WithUserInterface))
```

```
    {
        Console.WriteLine("Opening new project at " + ProjectDirectory +
machineName);
        ProjectComposition projects = tiaPortal.Projects;
        projects.Create(new DirectoryInfo(ProjectDirectory), machineName);
        Project newProject = projects[0];
        newProject.ShowHwEditor(Siemens.Engineering.HW.View.Network);
        Console.WriteLine("Activating project languages...");
        ActiveLanguages(newProject, UsedLanguages);

        #region Import Hardware
        Console.WriteLine("Importing Hardware...");
        ImportHardware(newProject, finalDirectory, importHardwareLogPath);

        //importlog Info (ConsoleWriteLine)
        LogStruct importlog = new LogStruct(importHardwareLogPath +
@"\import.log");

        Console.WriteLine("Errors: " + importlog.errors.Count);
        foreach (string error in importlog.errors) Console.WriteLine(" -> " +
error);

        Console.WriteLine("Warnings: " + importlog.warnings.Count);
        foreach (string warning in importlog.warnings) Console.WriteLine(" -> " +
warning);

        #endregion Import Hardware

        #region Import Software
        SoftwareFiles softwareDef = new SoftwareFiles(new
DirectoryInfo(finalDirectory));

        #region Import User Data Types
        Console.WriteLine("Importing User Data Types (" +
softwareDef.userDataTypes.Count.ToString() + ")... ");

        List<string> addedUDT = new List<string>();
        int index = 0;
        while (softwareDef.userDataTypes.Count > addedUDT.Count)
        {
            XmlDocument xmlUDT = new XmlDocument();
            xmlUDT.Load(softwareDef.userDataTypes[index].FullName);
            string nameUDT =
xmlUDT.DocumentElement.SelectSingleNode("/Document/SW.Types.PlcStruct/AttributeLi
st/Name").InnerText;
            XmlNodeList memberNodes =
xmlUDT.DocumentElement.SelectNodes("/Document//*[contains(name(), 'Member')]");
            List<string> usedUDT = new List<string>();
            foreach (XmlNode memberNode in memberNodes)
            {
                //XML file info:
                //- If the Datatype attribute is a User Data Type it is written
with "".
                //- The Name node of the UDT is without "".
                if (memberNode.Attributes["Datatype"].Value.StartsWith("\""))
usedUDT.Add(memberNode.Attributes["Datatype"].Value);
            }

            if (!usedUDT.Except(addedUDT).Any() && !addedUDT.Contains("\"" +
nameUDT + "\"")) //if (every usedUDT is in addedUDT && the UDT is not imported
yet)
            {
```

```
                Console.WriteLine("\"" + nameUDT + "\"" + "... ");
                ImportDataType(newProject,
softwareDef.userDataTypes[index].FullName, ImportOptions.None);
                addedUDT.Add("\"" + nameUDT + "\"");
            }
            index = (index + 1) % softwareDef.userDataTypes.Count;
        }

        Console.WriteLine("User Data Types imported");
        #endregion Import User Data Types

        #region Import TagTables
        Console.Write("Importing Tag Tables (" +
softwareDef.tagTables.Count.ToString() + ")... ");
        foreach (FileInfo tagTableFile in softwareDef.tagTables)
ImportTagTable(newProject, tagTableFile.FullName, ImportOptions.Override);
        Console.WriteLine("Tag Tables imported");
        #endregion Import Tagtables

        #region Import ProgramBlocks

        #region Import GlobalDBs
        Console.Write("Importing GlobalDBs (" +
softwareDef.programBlocks.GlobalDBs.NormalDBs.Count.ToString() + ")... ");
        foreach (FileInfo file in softwareDef.programBlocks.GlobalDBs.NormalDBs)
ImportSoftwareBlock(newProject, file.FullName, ImportOptions.None);
        Console.WriteLine("GlobalDBs imported");
        #endregion Import GlobalDBs

        #region Import Generic FBs and FCs
        //////Console.Write("Importing FBs... ");
        //////foreach (FileInfo file in softwareDef.programBlocks.FBs)
ImportSoftwareBlock(newProject, file.FullName, ImportOptions.None);
        //////Console.WriteLine("FBs imported");

        Console.WriteLine("Importing Generic FBs and FCs (" +
softwareDef.programBlocks.FBs.Count.ToString() + ") (" +
softwareDef.programBlocks.FCs.Count.ToString() + ")... ");

        List<FileInfo> GenericFunctions = new List<FileInfo>();
        GenericFunctions.AddRange(softwareDef.programBlocks.FBs);
        GenericFunctions.AddRange(softwareDef.programBlocks.FCs);
        List<string> addedFunc = new List<string>();
        int indexF = 0;
        while (GenericFunctions.Count > addedFunc.Count)
        {
            XmlDocument xmlFunc = new XmlDocument();
            xmlFunc.Load(GenericFunctions[indexF].FullName);
            string nameFunc =
xmlFunc.DocumentElement.SelectSingleNode("/Document/*[contains(name(),
'SW')]/AttributeList/Name").InnerText;
            XmlNodeList memberNodes =
xmlFunc.DocumentElement.SelectNodes("/Document//*[contains(name(), 'Member')]");
            List<string> usedFunc = new List<string>();
            foreach (XmlNode memberNode in memberNodes)
            {
                //XML file info:
                //- If the Datatype attribute is a User Data Type it is written
with "".
                //- The Name node of the UDT is without "".
                if (memberNode.Attributes["Datatype"].Value.StartsWith("\""))
usedFunc.Add(memberNode.Attributes["Datatype"].Value);
```

```
            }

            if (!usedFunc.Except(addedFunc).Any() && !addedFunc.Contains("\"" +
nameFunc + "\"")) //if (every usedUDT is in addedUDT && the UDT is not imported
yet)
            {
                Console.WriteLine("\"" + nameFunc + "\"" + "... ");
                ImportSoftwareBlock(newProject,
GenericFunctions[indexF].FullName, ImportOptions.None);
                addedFunc.Add("\"" + nameFunc + "\"");
            }
            indexF = (indexF + 1) % GenericFunctions.Count;
        }

        Console.WriteLine("Generic Functions imported");
        #endregion Import Generic Functions


        #region Import InstanceDBs
        Console.Write("Importing InstanceDBs (" +
softwareDef.programBlocks.InstanceDBs.Count.ToString() + ")... ");
        foreach (FileInfo file in softwareDef.programBlocks.InstanceDBs)
ImportSoftwareBlock(newProject, file.FullName, ImportOptions.None);
        Console.WriteLine("InstanceDBs imported");
        #endregion Import InstanceDBs

        #region Import FCs Main
        Console.Write("Importing Main FCs (" +
softwareDef.programBlocks.FCMains.Count.ToString() + ")... " +
softwareDef.programBlocks.FCMains.Count.ToString());
        foreach (FileInfo file in softwareDef.programBlocks.FCMains)
ImportSoftwareBlock(newProject, file.FullName, ImportOptions.None);
        Console.WriteLine("Main FCs imported");
        #endregion Import FCs Main

        #region Import OBs
        Console.Write("Importing OBs (" +
softwareDef.programBlocks.OBs.Count.ToString() + ")... ");
        foreach (FileInfo file in softwareDef.programBlocks.OBs)
ImportSoftwareBlock(newProject, file.FullName, ImportOptions.Override);
        Console.WriteLine("OBs imported");
        #endregion Import OBs

        #endregion Import Program Blocks

        #endregion

        #region Compile
        foreach (Device device in newProject.Devices)
        {
            Console.Write("Device: " + device.Name);
            Console.WriteLine(" - Compiling...");
            try //Hardware of a Device with failsafe CPU  cannot be compiled with
Openness
            {
                ICompilable compileService = device.GetService<ICompilable>();
                CompilerResult result = compileService.Compile();
                Console.WriteLine(result.Messages);
            }
            catch
            {
                Software softwareBase;
```

```
                PlcSoftware plcSoftware = null;

                DeviceItemComposition deviceItemComposition = device.DeviceItems;
                foreach (DeviceItem deviceItem in deviceItemComposition)
                {
                    SoftwareContainer softwareContainer =
deviceItem.GetService<SoftwareContainer>();
                    if (softwareContainer != null)
                    {
                        softwareBase = softwareContainer.Software;
                        plcSoftware = softwareBase as PlcSoftware;
                    }
                }
                try
                {
                    ICompilable compileService =
plcSoftware.GetService<ICompilable>();
                    CompilerResult result = compileService.Compile();
                    Console.WriteLine(" -> Software: " + result.ErrorCount + "
error(s)");
                    Console.WriteLine(" -> Hardware: Unable to compile
automatically. Try to compile it from the TIA Portal interface.");
                }
                catch
                {
                    Console.WriteLine(" -> Unable to compile Hardware and
Software. Try to compile Hardware and Software from the TIA Portal interface");
                }
            }
        }
        #endregion Compile

        Console.WriteLine("Saving...");
        newProject.Save();
        Console.WriteLine("Proyect saved at " + ProjectDirectory + machineName);
    }
}
```

15.2 *SoftwareFiles.cs*

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using static TIAPortalApplication.Utils.Generate.Others;

namespace TIAPortalApplication.Utils.Generate.Software_Structure
{
    class SoftwareFiles
    {
        public List<FileInfo> userDataTypes { get; set; }
        public List<FileInfo> tagTables { get; set; }
        public ProgramBlocksFiles programBlocks { get; set; }

        //These types of blocks cannot be imported
        public List<FileInfo> F_Files { get; set; }
        public List<FileInfo> SCLfiles { get; set; }
        //**************************************

        public SoftwareFiles(DirectoryInfo rootDirectory)
        {
            #region Get Software Folder
            DirectoryInfo[] subDirs = null;
            FileInfo[] softwareFiles = null;
            subDirs = rootDirectory.GetDirectories("Software");
            if (subDirs.Length != 1) Console.WriteLine("ERROR: The number of
\"Software\" folders found is not 1");
            else
            {
                softwareFiles =
WalkDirectoryTree(rootDirectory.GetDirectories("Software")[0]);
            }
            #endregion;

            #region XML Classification
            userDataTypes = new List<FileInfo>();
            tagTables = new List<FileInfo>();
            programBlocks = new ProgramBlocksFiles();
            F_Files = new List<FileInfo>();
            SCLfiles = new List<FileInfo>();

            foreach (FileInfo file in softwareFiles)
            {
                XmlDocument xmlFile = new XmlDocument();
                xmlFile.Load(file.FullName);
                if
(xmlFile.DocumentElement.SelectSingleNode("/Document//AttributeList/ProgrammingLa
nguage") != null &&

xmlFile.DocumentElement.SelectSingleNode("/Document//AttributeList/ProgrammingLan
guage").InnerText.StartsWith("F_"))
                {
                    F_Files.Add(file);
                }
                else if
(xmlFile.DocumentElement.SelectSingleNode("/Document//AttributeList/ProgrammingLa
nguage") != null &&
```

```
xmlFile.DocumentElement.SelectSingleNode("/Document//AttributeList/ProgrammingLan
guage").InnerText == "SCL")
                {
                    SCLfiles.Add(file);
                    // SCL blocks cannot be imported in Oppennes v14 SP1, so in
this program they are just ignored
                    // Posible update: They may be able to be imported in
Oppenness v15.
                }
                else
                {
                    XmlNodeList nodes =
xmlFile.DocumentElement.SelectNodes("/Document/*[contains(name(), 'SW')]");
                    if (nodes.Count == 1)
                    {
                        switch (nodes[0].Name)
                        {
                            case "SW.Tags.PlcTagTable":
                                XmlNode objectlist =
xmlFile.DocumentElement.SelectSingleNode("/Document/SW.Tags.PlcTagTable/ObjectLis
t");
                                if (objectlist != null) tagTables.Add(file); //
if the Tag Table doesn't have any tag it is ignored.
                                break;
                            case "SW.Blocks.GlobalDB":
                                string dbName =
xmlFile.DocumentElement.SelectSingleNode("/Document/SW.Blocks.GlobalDB/AttributeL
ist/Name").InnerText;
                                if
(dbName.ToUpper().StartsWith("DB_CONNECTION_IN"))
programBlocks.GlobalDBs.ConnectionInDBs.Add(file);
                                else if
(dbName.ToUpper().StartsWith("DB_CONNECTION_OUT"))
programBlocks.GlobalDBs.ConnectionOutDBs.Add(file);
                                else programBlocks.GlobalDBs.NormalDBs.Add(file);
                                break;
                            case "SW.Blocks.InstanceDB":
programBlocks.InstanceDBs.Add(file); break;
                            case "SW.Blocks.FB": programBlocks.FBs.Add(file);
break;
                            case "SW.Blocks.FC":

                                XmlNode interfaceNode =
xmlFile.DocumentElement.SelectSingleNode("/Document/*[contains(name(),
'SW')]/AttributeList/Interface");
                                if
(interfaceNode.SelectSingleNode("descendant::*[@Name='Input']").ChildNodes.Count
== 0 &&

interfaceNode.SelectSingleNode("descendant::*[@Name='Output']").ChildNodes.Count
== 0 &&

interfaceNode.SelectSingleNode("descendant::*[@Name='InOut']").ChildNodes.Count
== 0)
                                {
                                    programBlocks.FCMains.Add(file);
                                }
                                else programBlocks.FCs.Add(file);
                                break;
```

```
                            case "SW.Blocks.OB": programBlocks.OBs.Add(file);
break;
                            case "SW.Types.PlcStruct": userDataTypes.Add(file);
break;
                            default: Console.WriteLine("ERROR: File " + file.Name
+ "has not been able to be classified"); break;
                        }
                    }
                    else Console.WriteLine("ERROR: File " + file.Name + "has not
been able to be classified, " + nodes.Count);
                }
            }
            #endregion XML Classification
        }

        public List<string> GetLanguages()
        {
            List<string> languages = new List<string>();
            XmlDocument xmlLan = new XmlDocument();
            if (programBlocks.FCs.Count != 0)
            {
                xmlLan.Load(programBlocks.FCs.First().FullName);
                XmlNodeList cultures =
xmlLan.DocumentElement.SelectNodes("//Culture");
                foreach (XmlNode culture in cultures)
languages.Add(culture.InnerText);
            }
            else if (programBlocks.OBs.Count != 0)
            {
                xmlLan.Load(programBlocks.OBs.First().FullName);
                XmlNodeList cultures =
xmlLan.DocumentElement.SelectNodes("//Culture");
                foreach (XmlNode culture in cultures)
languages.Add(culture.InnerText);
            }
            return languages;
        }
    }
}
```

15.3 *ProgramBlocks.cs*

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TIAPortalApplication.Utils.Generate.Software_Structure
{
    class ProgramBlocksFiles
    {
        public GlobalDBs GlobalDBs = new GlobalDBs();
        public List<FileInfo> InstanceDBs = new List<FileInfo>();
        public List<FileInfo> FBs = new List<FileInfo>();
        public List<FileInfo> FCs = new List<FileInfo>();
        public List<FileInfo> FCMains = new List<FileInfo>();
        public List<FileInfo> OBs = new List<FileInfo>();
    }
}
```

15.4 *GlobalDBs.cs*

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TIAPortalApplication.Utils.Generate.Software_Structure
{
    class GlobalDBs
    {
        public List<FileInfo> NormalDBs = new List<FileInfo>(); //Every other
GlobalDB that is not a CONNECTION DB
        public List<FileInfo> ConnectionInDBs = new List<FileInfo>();
        public List<FileInfo> ConnectionOutDBs = new List<FileInfo>();
    }
}
```

15.5 *AMLMethods.cs*

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;

namespace TIAPortalApplication.Utils.Generate
{
    class AMLMethods
    {
        private static int maxNameChar = 24; // The name of the devices cannot be
longer than 24 characters

        #region AML UngroupedDevicesFolder
        public const string AMLUngroupedDevicesFolder =

"<InternalElement ID=\"360e9c10-b81f-4cac-838d-b403e9c403eb\" Name=\"Ungrouped
devices\">" +
    "<SupportedRoleClass RefRoleClassPath=
\"AutomationProjectConfigurationRoleClassLib/DeviceUserFolder\"/>" +
"</InternalElement>";
        #endregion

        #region AML IoSystem
        private const string AMLiosystem =
        "<InternalElement ID=\"73d4fb83-6eee-4b16-a02b-7ec40a3d3cbc\"
Name=\"PROFINET IO-System\" >" +
        "  <Attribute Name=\"Number\" AttributeDataType=\"xs:int\" >" +
        "    <Value>100</Value>" +
        "  </Attribute>" +
        "  <ExternalInterface ID=\"25c31780-fa4e-41f0-b077-f5a23b3b0fff\"
Name=\"LogicalEndPoint_IoSystem\"
RefBaseClassPath=\"CommunicationInterfaceClassLib/LogicalEndPoint\" />" +
        "  <SupportedRoleClass
RefRoleClassPath=\"AutomationProjectConfigurationRoleClassLib/IoSystem\" />" +
        "</InternalElement>";
        #endregion

        public static FileInfo GetAMLPath(DirectoryInfo root)
        {
            DirectoryInfo[] subDirs = null;

            subDirs = root.GetDirectories("Hardware");
            if (subDirs.Length != 1) Console.WriteLine("ERROR: The number of
\"Hardware\" folders found is not 1");
            else
            {
                FileInfo[] files = null;
                files = subDirs[0].GetFiles("*.aml");
                if (files.Length != 1) Console.WriteLine("ERROR: The number of
*.AML files found in \"Hardware\" folder is not 1");
                else return files[0];
            }
            Console.WriteLine("Error");
            return null;
        }
```

```
        public static XmlDocument TransformCPUBase(XmlDocument amlCPUBase, string
firstIP, string startInputAddress,string startOutputAddress, string prefix)
        {
            XmlNode rootCPU = amlCPUBase.DocumentElement;

            rootCPU = ChangeIPs(rootCPU, firstIP);

            int iStartInputAddress;
            Int32.TryParse(startInputAddress, out iStartInputAddress);
            int iStartOutputAddress;
            Int32.TryParse(startOutputAddress, out iStartOutputAddress);

            rootCPU = ChangeIOAddress(rootCPU, iStartInputAddress,
iStartOutputAddress);

            XmlNode cpuRackIENode =
rootCPU.SelectSingleNode("//SupportedRoleClass[@RefRoleClassPath='AutomationProje
ctConfigurationRoleClassLib/Device']").ParentNode; //There will only be one
Device IE in the AML. The one of the CPU Rack
            cpuRackIENode = ChangePrefix(cpuRackIENode, prefix);

            XmlNodeList tagtablesRoleClass =
rootCPU.SelectNodes("//SupportedRoleClass[@RefRoleClassPath='AutomationProjectCon
figurationRoleClassLib/TagTable']");
            foreach (XmlNode tagtableRoleClass in tagtablesRoleClass)
            {
                XmlNode tagtable = tagtableRoleClass.ParentNode;
                tagtable.ParentNode.RemoveChild(tagtable);
            }

            amlCPUBase = InsertIoSystemCPU(amlCPUBase);
            amlCPUBase = InsertUngroupedDevicesFolder(amlCPUBase);

            return amlCPUBase;
        }

        public static XmlDocument TransformUngroupedDevices(XmlDocument
amlUnDevs, string firstIP, string startInputAddress, string startOutputAddress,
string prefix)
        {
            XmlNode rootUnDevs = amlUnDevs.DocumentElement;

            rootUnDevs = ChangeIPs(rootUnDevs, firstIP);
            rootUnDevs = ChangePrefix(rootUnDevs, prefix);

            int istartInputAddress;
            Int32.TryParse(startInputAddress, out istartInputAddress);
            int istartOutputAddress;
            Int32.TryParse(startOutputAddress, out istartOutputAddress);
            rootUnDevs = ChangeIOAddress(rootUnDevs, istartInputAddress,
istartOutputAddress);

            return amlUnDevs;
        }

        public static XmlDocument InsertUngroupedDevices(XmlDocument amlDocBase,
XmlDocument AMLDocUnDevs)
        {
            XmlNode rootUnDevs = AMLDocUnDevs.DocumentElement;
            XmlNode rootBase = amlDocBase.DocumentElement;

            rootUnDevs = AdjustIDs(rootBase, rootUnDevs);
```

```
            #region Add UngroupedDevices
            XmlNodeList UnDevsNodes =
AMLDocUnDevs.SelectNodes("/InternalElement/child::InternalElement");//El primer
IE es el de Ungrouped Devices. Se cogen los IE que tiene dentro.
            foreach (XmlNode UnDev in UnDevsNodes)
            {
                XmlNode copyUnDev = amlDocBase.ImportNode(UnDev, true);
                XmlNode refNode =
rootBase.SelectSingleNode("//InternalElement[@Name='Ungrouped devices']");
                refNode.PrependChild(copyUnDev);
            }
            #endregion
            return amlDocBase;
        }

        private static int linkstoSubnet = 1; //Already have the CPU to Subnet
Link

        public static XmlDocument ConnectUnDevstoSubnet(XmlDocument amlDoc)
        {
            XmlNode root = amlDoc.DocumentElement;

            string netID =
root.SelectSingleNode("//SupportedRoleClass[@RefRoleClassPath='AutomationProjectC
onfigurationRoleClassLib/Subnet']").ParentNode.Attributes["ID"].Value;
            string netExternalInterfaceName =
root.SelectSingleNode("//SupportedRoleClass[@RefRoleClassPath='AutomationProjectC
onfigurationRoleClassLib/Subnet']").ParentNode.SelectSingleNode("ExternalInterfac
e").Attributes["Name"].Value;

            XmlNodeList SRCUnDevsNodes =
root.SelectNodes("//InternalElement[@Name='Ungrouped
devices']//SupportedRoleClass[@RefRoleClassPath='AutomationProjectConfigurationRo
leClassLib/Node']");
            foreach (XmlNode node in SRCUnDevsNodes)
            {
                if
(node.ParentNode.SelectSingleNode("Attribute[@Name='Type']/Value").InnerText ==
"Ethernet")
                {
                    string UnDevNodeID = node.ParentNode.Attributes["ID"].Value;
                    string UnDevNodeExternalInterfaceName =
node.ParentNode.SelectSingleNode("ExternalInterface").Attributes["Name"].Value;
                    linkstoSubnet = linkstoSubnet + 1;
                    XmlElement newInternalLink =
amlDoc.CreateElement("InternalLink");
                    newInternalLink.SetAttribute("Name", "Link to Subnet_" +
linkstoSubnet.ToString());
                    newInternalLink.SetAttribute("RefPartnerSideA", UnDevNodeID +
":" + UnDevNodeExternalInterfaceName);
                    newInternalLink.SetAttribute("RefPartnerSideB", netID + ":" +
netExternalInterfaceName);
                    XmlNode refNode =
root.SelectSingleNode("//SupportedRoleClass[@RefRoleClassPath='AutomationProjectC
onfigurationRoleClassLib/AutomationProject']");
                    amlDoc.ImportNode(newInternalLink, true);
                    refNode.ParentNode.InsertAfter(newInternalLink, refNode);
                }
            }
            return amlDoc;
        }
```

```
        private static int linksToIoSystem = 0;
        public static XmlDocument ConnectUnDevsToIoSystem (XmlDocument amlDoc)
        {
            XmlNode root = amlDoc.DocumentElement;
            string IoSystemID =
root.SelectSingleNode("//SupportedRoleClass[@RefRoleClassPath='AutomationProjectC
onfigurationRoleClassLib/IoSystem']").ParentNode.Attributes["ID"].Value;
            string IoSystemExternalInterfaceName =
root.SelectSingleNode("//SupportedRoleClass[@RefRoleClassPath='AutomationProjectC
onfigurationRoleClassLib/IoSystem']").ParentNode.SelectSingleNode("ExternalInterf
ace").Attributes["Name"].Value;
            XmlNodeList SRCUnDevsNodes =
root.SelectNodes("//InternalElement[@Name='Ungrouped
devices']//SupportedRoleClass[@RefRoleClassPath='AutomationProjectConfigurationRo
leClassLib/Node']");
            foreach (XmlNode node in SRCUnDevsNodes)
            {
                if
(node.ParentNode.SelectSingleNode("Attribute[@Name='Type']/Value").InnerText ==
"Ethernet")
                {
                    string UnDevNetInterfaceID =
node.ParentNode.ParentNode.Attributes["ID"].Value;
                    string UnDevExternalInterfaceName =
node.ParentNode.ParentNode.SelectSingleNode("ExternalInterface").Attributes["Name
"].Value;
                    linksToIoSystem = linksToIoSystem + 1;
                    XmlElement newInternalLink =
amlDoc.CreateElement("InternalLink");
                    newInternalLink.SetAttribute("Name", "Link to IoSystem_" +
linksToIoSystem.ToString());
                    newInternalLink.SetAttribute("RefPartnerSideA",
UnDevNetInterfaceID + ":" + UnDevExternalInterfaceName);
                    newInternalLink.SetAttribute("RefPartnerSideB", IoSystemID +
":" + IoSystemExternalInterfaceName);
                    XmlNode refNode =
root.SelectSingleNode("//SupportedRoleClass[@RefRoleClassPath='AutomationProjectC
onfigurationRoleClassLib/AutomationProject']");
                    amlDoc.ImportNode(newInternalLink, true);
                    refNode.ParentNode.InsertAfter(newInternalLink, refNode);
                }
            }
            return amlDoc;
        }

        private static XmlDocument InsertIoSystemCPU(XmlDocument amlCPU)
        {
            XmlNode root = amlCPU.DocumentElement;
            if
(root.SelectNodes("//SupportedRoleClass[@RefRoleClassPath='AutomationProjectConfi
gurationRoleClassLib/IoSystem']").Count == 0) //If there is not IoSystem yet.
            {
                XmlDocument DocIoSystem = new XmlDocument();
                DocIoSystem.LoadXml(AMLiosystem);
                XmlNode refCPUnode =
root.SelectSingleNode("//Attribute[@Name='DeviceItemType']/Value[text() =
'CPU']").ParentNode.ParentNode.SelectSingleNode("child::InternalElement/child::In
ternalElement/child::Attribute[@Name='Type']/Value[text() =
'Ethernet']").ParentNode.ParentNode;
                XmlNode copynode = amlCPU.ImportNode(DocIoSystem.DocumentElement,
true);
```

```
                refCPUnode.ParentNode.InsertBefore(copynode,
refCPUnode.ParentNode.LastChild);
            }
            return amlCPU;
        }

        private static XmlDocument InsertUngroupedDevicesFolder(XmlDocument
amlCPU)
        {
            XmlNode root = amlCPU.DocumentElement;
            XmlDocument DocUnDevFolder = new XmlDocument();
            DocUnDevFolder.LoadXml(AMLUngroupedDevicesFolder);
            XmlNode refCPUnode =
root.SelectSingleNode("//SupportedRoleClass[@RefRoleClassPath='AutomationProjectC
onfigurationRoleClassLib/AutomationProject']");
            XmlNode copynode = amlCPU.ImportNode(DocUnDevFolder.DocumentElement,
true);
            refCPUnode.ParentNode.InsertBefore(copynode, refCPUnode);
            return amlCPU;
        }

        public static List<string> GetInAddresses(XmlDocument amlDoc)
        {
            List<string> InList = new List<string>();
            XmlNode root = amlDoc.DocumentElement;
            XmlNodeList addresses =
root.SelectNodes("//Attribute[@Name='StartAddress']/Value");
            foreach (XmlNode address in addresses)
            {
                if
(address.ParentNode.ParentNode.SelectSingleNode("Attribute[@Name='IoType']/Value"
).InnerText == "Input")
                {
                    int iaddress;
                    Int32.TryParse(address.InnerText, out iaddress);
                    int length; // Bits

Int32.TryParse(address.ParentNode.ParentNode.SelectSingleNode("Attribute[@Name='L
ength']/Value").InnerText, out length);
                    for (int i = iaddress; i < (iaddress +
(int)Math.Ceiling((double)length / 8)); i++)
                    {
                        InList.Add(i.ToString());
                    }
                }
            }
            return InList;
        }

        public static List<string> GetOutAddresses(XmlDocument amlDoc)
        {
            List<string> OutList = new List<string>();
            XmlNode root = amlDoc.DocumentElement;
            XmlNodeList addresses =
root.SelectNodes("//Attribute[@Name='StartAddress']/Value");
            foreach (XmlNode address in addresses)
            {
                if
(address.ParentNode.ParentNode.SelectSingleNode("Attribute[@Name='IoType']/Value"
).InnerText == "Output")
                {
                    int iaddress;
```

```
                    Int32.TryParse(address.InnerText, out iaddress);
                    int length; // Bits

Int32.TryParse(address.ParentNode.ParentNode.SelectSingleNode("Attribute[@Name='L
ength']/Value").InnerText, out length);
                    for (int i = iaddress; i < (iaddress +
(int)Math.Ceiling((double)length / 8)); i++)
                    {
                        OutList.Add(i.ToString());
                    }
                }
            }
            return OutList;
        }

        private static XmlNode ChangeIOAddress(XmlNode root, int
startInputAddress, int startOutputAddress)
        {
            XmlNodeList addresses =
root.SelectNodes("//Attribute[@Name='StartAddress']");

            int inAddress = startInputAddress;
            int outAddress = startOutputAddress;

            foreach (XmlNode address in addresses)
            {
                string iotype =
address.ParentNode.SelectSingleNode("Attribute[@Name='IoType']/Value").InnerText;
// "Input" or "Output"
                int length; // Bits

Int32.TryParse(address.ParentNode.SelectSingleNode("Attribute[@Name='Length']/Val
ue").InnerText, out length);
                if (iotype == "Input")
                {
                    address.SelectSingleNode("Value").InnerText =
inAddress.ToString(); // Change StartAddress
                    inAddress = inAddress + (int)Math.Ceiling((double)length /
8); // Set the next StartAddress
                }
                if (iotype == "Output")
                {
                    address.SelectSingleNode("Value").InnerText =
outAddress.ToString(); // Change StartAddress
                    outAddress = outAddress + (int)Math.Ceiling((double)length /
8); // Set the next StartAddress
                }
            }
            return root;
        }

        private static XmlNode AdjustIDs(XmlNode root, XmlNode newroot)
        {
            XmlNodeList ElementsWithID = root.SelectNodes("//*[@ID]");
            List<string> IDs = new List<string>();
            foreach (XmlNode Element in ElementsWithID)
            {
                IDs.Add(Element.Attributes["ID"].Value);
            }

            XmlNodeList newElementsWithID = newroot.SelectNodes("//*[@ID]");
            foreach (XmlNode newElement in newElementsWithID)
```

```
            {
                if (IDs.Contains(newElement.Attributes["ID"].Value))
                {
                    newroot = ChangeID(newroot,
newElement.Attributes["ID"].Value);
                }
            }
            return newroot;
        }

        private static XmlNode ChangeID(XmlNode root, string duplicatedID)
        {
            XmlNodeList ElementsWithID = root.SelectNodes("//*[@ID]");
            foreach (XmlNode element in ElementsWithID)
            {
                if (element.Attributes["ID"].Value == duplicatedID)
                {
                    element.Attributes["ID"].Value = Guid.NewGuid().ToString();
                }
            }
            return root;
        }

        private static XmlNode ChangeIPs(XmlNode rootNode, string firstIP)
        {
            XmlNodeList DevItemsWithNodes =
rootNode.SelectNodes("//InternalElement[InternalElement/InternalElement/Attribute
[@Name='NetworkAddress']]");
            int i = Int32.Parse(firstIP.Substring((firstIP).LastIndexOf('.') +
1)); // Take de last IP number as integer
            foreach (XmlNode DevItem in DevItemsWithNodes)
            {
                XmlNodeList IPValues =
DevItem.SelectNodes("//Attribute[@Name='NetworkAddress']/Value");
                bool firstInterfaceChanged = false;
                foreach (XmlNode IPvalue in IPValues)
                {
                    if
(IPvalue.ParentNode.ParentNode.SelectSingleNode("Attribute[@Name='Type']/Value").
InnerText == "Ethernet") // The first Ethernet Interface.
                    {
                        if (!firstInterfaceChanged) // Change the first Interface
IP
                        {
                            string IP = firstIP.Remove(firstIP.LastIndexOf('.') +
1);

                            IP = IP + i.ToString();
                            IPvalue.InnerText = IP;
                            i = (i + 1) % 256;
                            firstInterfaceChanged = true;
                        }
                        else IPvalue.InnerText = ""; // TIA Portal will set the
rest of NetworkAdresses by default.
                    }
                }
            }
            return rootNode;
        }

        private static bool IsNameRepeated(XmlNode root, string name)
        {
            int ocurrences = 0;
```

```
            //check the root node
            if (root.Attributes["Name"].Value == name) ocurrences++;
            //check the descendants
            XmlNodeList InternalElements =
root.SelectNodes("descendant::InternalElement");
            foreach (XmlNode InternalElement in InternalElements)
            {
                if (InternalElement.Attributes["Name"].Value == name)
ocurrences++;
            }

            if (ocurrences > 1) return true;
            else return false;
        }

        private static XmlNode ChangePrefix(XmlNode root, string prefix)
        {
            //to the root node
            if ((prefix + root.Attributes["Name"].Value).Length <= maxNameChar)
            {
                root.Attributes["Name"].Value = prefix +
root.Attributes["Name"].Value;
            }
            else
            {
                root.Attributes["Name"].Value = (prefix +
root.Attributes["Name"].Value).Substring(0, maxNameChar);//this changes the
original name, so it can be repeated with other names of the same module
                int i = 0;
                while (IsNameRepeated(root, root.Attributes["Name"].Value) && i <
100)// if the new name is repeated, try from 0 to 99 sufix until the name is
unique.
                {
                    root.Attributes["Name"].Value =
root.Attributes["Name"].Value.Substring(0, maxNameChar - 2) + i.ToString();
                    i++;
                }
            }

            //to the descendant nodes
            XmlNodeList InternalElements =
root.SelectNodes("descendant::InternalElement");
            foreach (XmlNode InternalElement in InternalElements)
            {
                if ((prefix + InternalElement.Attributes["Name"].Value).Length <=
maxNameChar)
                {
                    InternalElement.Attributes["Name"].Value = prefix +
InternalElement.Attributes["Name"].Value;
                }
                else
                {
                    InternalElement.Attributes["Name"].Value = (prefix +
InternalElement.Attributes["Name"].Value).Substring(0, maxNameChar);//this
changes the original name, so it can be repeated with other names of the same
module
                    int i = 0;
                    while (IsNameRepeated(root,
InternalElement.Attributes["Name"].Value) && i < 100)// if the new name is
repeated, try up to 100 sufix until the name is unique.
                    {
```

```
                        InternalElement.Attributes["Name"].Value =
InternalElement.Attributes["Name"].Value.Substring(0, maxNameChar - 2) +
i.ToString();
                        i++;
                    }
                }
            }
            return root;
        }
    }
}
```

15.6 **XMLMethods.cs**

```
using System.Collections.Generic;
using System.Xml;

namespace TIAPortalApplication.Utils.Generate
{
    class XMLMethods
    {
        public static XmlDocument TransformTagTable(XmlDocument xmlTagTable,
string prefix, Dictionary<string, string> InChanges, Dictionary<string, string>
OutChanges, ref int nextM, ref int nextT, ref int nextC)
        {
            XmlNode root = xmlTagTable.DocumentElement;
            root = ChangeNamesPrefix(root, prefix);
            root = ChangeLogicalAddresses(root, InChanges, OutChanges, ref nextM,
ref nextT, ref nextC);
            return xmlTagTable;
        }

        private static XmlNode ChangeNamesPrefix(XmlNode root, string prefix)
        {
            XmlNodeList names = root.SelectNodes("//Name");
            foreach (XmlNode name in names)
            {
                name.InnerText = prefix + name.InnerText;
            }
            return root;
        }

        private static XmlNode ChangeLogicalAddresses(XmlNode root,
Dictionary<string, string> InChanges, Dictionary<string, string> OutChanges, ref
int nextM, ref int nextT, ref int nextC)
        {
            XmlNodeList logicalAddresses = root.SelectNodes("//LogicalAddress");
            foreach (XmlNode logicalAddress in logicalAddresses)
            {
                int FirstIndex =
logicalAddress.InnerText.IndexOfAny("0123456789".ToCharArray());
                int LastIndex;
                if (logicalAddress.InnerText.IndexOf('.') == -1) LastIndex =
logicalAddress.InnerText.Length;
                else LastIndex = logicalAddress.InnerText.IndexOf('.');
                string byteAddress =
logicalAddress.InnerText.Substring(FirstIndex, LastIndex - FirstIndex);
                string newByteAddress = "";

                switch (logicalAddress.InnerText[1])
                {
                    case 'I':
                        InChanges.TryGetValue(byteAddress, out newByteAddress);
                        break;
                    case 'Q':
                        OutChanges.TryGetValue(byteAddress, out newByteAddress);
                        break;
                    case 'M':
                        newByteAddress = nextM.ToString();
                        nextM += 1;
                        break;
                    case 'T':
                        newByteAddress = nextT.ToString();
                        nextT += 1;
```

```
                    break;
                case 'C':
                    newByteAddress = nextC.ToString();
                    nextC += 1;
                    break;
            }
            logicalAddress.InnerText = logicalAddress.InnerText.Substring(0,
FirstIndex) + newByteAddress + logicalAddress.InnerText.Substring(LastIndex);
        }
        return root;
    }

    public static XmlDocument TransformDataType(XmlDocument xmlDataType,
string prefix)
    {
        XmlNode root = xmlDataType.DocumentElement;
        XmlNode nameNode =
root.SelectSingleNode("//SW.Types.PlcStruct/AttributeList/Name");
        nameNode.InnerText = prefix + nameNode.InnerText;
        return xmlDataType;
    }

    public static XmlDocument TransformDB(XmlDocument xmlGlobalDB, string
prefix)
    {
        XmlNode name =
xmlGlobalDB.DocumentElement.SelectSingleNode("/Document//AttributeList/Name");
        name.InnerText = prefix + name.InnerText;

        XmlNode number =
xmlGlobalDB.DocumentElement.SelectSingleNode("/Document//AttributeList/Number");
        number.ParentNode.RemoveChild(number);

        return xmlGlobalDB;
    }

    public static XmlDocument TransformFB(XmlDocument xmlFB)
    {
        XmlNode number =
xmlFB.DocumentElement.SelectSingleNode("/Document//AttributeList/Number");
        number.ParentNode.RemoveChild(number);

        return xmlFB;
    }

    public static XmlDocument TransformFC(XmlDocument xmlFC, string prefix)
    {
        XmlNode number =
xmlFC.DocumentElement.SelectSingleNode("/Document//AttributeList/Number");
        number.ParentNode.RemoveChild(number);

        // If the FC doesn't have IO parameters it is a FC that makes the
calls to generic FCs and FBs.
        XmlNode interfaceNode =
xmlFC.DocumentElement.SelectSingleNode("//SW.Blocks.FC/AttributeList/Interface");
        if
(interfaceNode.SelectSingleNode("descendant::*[@Name='Input']").ChildNodes.Count
== 0 &&

interfaceNode.SelectSingleNode("descendant::*[@Name='Output']").ChildNodes.Count
== 0 &&
```

```
interfaceNode.SelectSingleNode("descendant::*[@Name='InOut']").ChildNodes.Count
== 0)
            {
                // Specific for each module, so the <Name> node have to be
changed with the prefix.
                XmlNode name =
xmlFC.DocumentElement.SelectSingleNode("/Document//AttributeList/Name");
                name.InnerText = prefix + name.InnerText;

                // It may use GlobalVariables, so the name of those
GlobalVariables have to be changed with the prefix.
                XmlNodeList GVnodes =
xmlFC.DocumentElement.SelectNodes("//*[@Scope='GlobalVariable']");
                foreach (XmlNode node in GVnodes)
                {
                    XmlNode component =
node.SelectSingleNode("descendant::*[name(.)='Component']"); //Only the first
Component node. If the parameter is DB.Something1.Something2 there will be 3
components. Change only the first (the one of the DB)

                    // The transformation to "DB_CONNECTION_IN" is made later
with the information of the relations.
                    if (component.Attributes["Name"].Value.ToUpper() !=
"DB_CONNECTION_IN")
                    component.Attributes["Name"].Value = prefix +
component.Attributes["Name"].Value;
                }
            }

            return xmlFC;
        }

        public static XmlDocument AddToOB(XmlDocument xmlBaseOB, XmlDocument
xmlNewOB, string prefix)
        {
            xmlNewOB = TransformOB(xmlNewOB, prefix);

            XmlNodeList compileUnits =
xmlNewOB.SelectNodes("//SW.Blocks.CompileUnit");
            foreach (XmlNode compileUnit in compileUnits)
            {
                XmlNode refNode =
xmlBaseOB.SelectSingleNode("//SW.Blocks.OB/ObjectList").LastChild;
                XmlNode copyCompileUnit = xmlBaseOB.ImportNode(compileUnit,
true);
                refNode.ParentNode.InsertBefore(copyCompileUnit, refNode);
            }

            xmlBaseOB = ChangeIDsOB(xmlBaseOB);

            return xmlBaseOB;
        }

        public static XmlDocument TransformOB(XmlDocument xmlOB, string prefix)
        {
            XmlNodeList callInfoList =
xmlOB.DocumentElement.SelectNodes("//*[name(.)='CallInfo']");

            foreach (XmlNode callInfo in callInfoList)
            {
```

```
                callInfo.Attributes["Name"].Value = prefix +
callInfo.Attributes["Name"].Value;
            }
            return xmlOB;
        }

        private static XmlDocument ChangeIDsOB(XmlDocument xmlOB)
        {
            XmlNodeList nodesWithID =
xmlOB.DocumentElement.SelectNodes("//*[@ID]");
            int i = 1;
            foreach (XmlNode nodewithID in nodesWithID)
            {
                nodewithID.Attributes["ID"].Value = i.ToString("X"); //i value in
hexadecimal as a string
                i++;
            }

            return xmlOB;
        }

        public static XmlDocument TransformFCMainConnection(XmlDocument
xmlFCMain, XmlNodeList xmlRelations)
        {
            XmlNodeList components =
xmlFCMain.DocumentElement.SelectNodes("//*[name(.)='Component']");
            List < XmlNode > inputsWithoutRelation = new List<XmlNode>();
            foreach (XmlElement component in components)
            {
                if (component.Attributes["Name"].Value.ToUpper() ==
"DB_CONNECTION_IN")
                {
                    string previousVarIn =
component.NextSibling.Attributes["Name"].Value;
                    //***** Opt. 2 *****
                    bool relationFound = false;
                    //***** Opt. 2 *****
                    foreach (XmlNode relation in xmlRelations)
                    {
                        if (relation.Attributes["variableIn"].Value ==
previousVarIn) // if (relation detected in the list)
                        {
                            component.NextSibling.Attributes["Name"].Value =
relation.Attributes["variableOut"].Value;
                            component.Attributes["Name"].Value =
relation.Attributes["sourceComponent"].Value + "_DB_CONNECTION_OUT"; //prefix +
"_DB_CONNECTION_Out"
                            //***** Opt. 2 *****
                            relationFound = true;
                            //***** Opt. 2 *****
                            break;
                        }
                    }

                    // If there is no relation for this In component
                    //
                    // (Now) Option 1: Don't do anything.
                    //                 The Call will have the parameter
DB_CONNECTION_In,
                    //                 so compilation error,
                    //                 so the user decides.
                    //       Option 2: Empty parameter.
```

```
                    //                The project will use de default value
defined for that IN parameter in the formal parameter of the function.

                    //*****Opt. 2 * ****
                    if (!relationFound) inputsWithoutRelation.Add(component);
                    //*****Opt. 2 * ****

                    //       Option 3: Every IN need (and have) a relation with
an OUT.
                    //                Allow to select not only OUTs but also
TRUE or FALSE for bools,
                    //                numbers for integers and reals. In the
XML.
                }
            }
            //***** Opt. 2 *****
            foreach (XmlNode inputWoRelation in inputsWithoutRelation)
            {
                //Deleting all the childs of the parameter node is necessary.
Only deleting the value of the
                //component would throw an exception when importing
                //Thats why first we iterate through the complete list of
components and now we already can
                //delete the CONNECTION In components whitout relation.
                XmlElement parameter =
inputWoRelation.ParentNode.ParentNode.ParentNode as XmlElement;
                parameter.IsEmpty = true;
            }
            //***** Opt. 2 *****

            return xmlFCMain;
        }
    }
}
```

15.7 *TIAPortalImportMethods.cs*

```
using Siemens.Engineering;
using Siemens.Engineering.Cax;
using Siemens.Engineering.Compiler;
using Siemens.Engineering.HW;
using Siemens.Engineering.HW.Features;
using Siemens.Engineering.SW;
using Siemens.Engineering.SW.Blocks;
using Siemens.Engineering.SW.Tags;
using Siemens.Engineering.SW.Types;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;

using static TIAPortalApplication.Utils.Generate.AMLMethods;
using static TIAPortalApplication.Utils.Generate.Others;

namespace TIAPortalApplication.Utils.Generate
{
    static class TiaPortalImportMethods
    {
        static public void ChangeProfinetName(Project project, string nodeIP,
string nodeName)
        {
            if (project.Subnets.Count != 0)
            {
                foreach (Node node in project.Subnets[0].Nodes)
                {
                    if (nodeIP == node.GetAttribute("Address").ToString())
                    {
                        foreach (EngineeringAttributeInfo attinfo in
node.GetAttributeInfos()) //Not every node has 'PnDeviceNameSetDirectly'
attribute
                        {
                            if (attinfo.Name == "PnDeviceNameSetDirectly")
                            {
                                node.SetAttribute("PnDeviceNameSetDirectly",
false);
                                break;
                            }
                        }
                        node.SetAttribute("PnDeviceNameAutoGeneration", false);
                        node.SetAttribute("PnDeviceName", nodeName);
                    }
                }
            }
            else Console.WriteLine("Subnet not found. Profinet Name has not been
changed (" + nodeIP + ", " + nodeName + ").");
        }

        static public void ActiveLanguages(Project project, List<string>
languages)
        {
            LanguageSettings languageSettings = project.LanguageSettings;
            LanguageComposition supportedLanguages = languageSettings.Languages;
            LanguageAssociation activeLanguages =
languageSettings.ActiveLanguages;
            foreach (string language in languages)
            {
```

```
                if (supportedLanguages.Find(CultureInfo.GetCultureInfo(language))
!= null &&
                    activeLanguages.Find(CultureInfo.GetCultureInfo(language)) ==
null)
                {

activeLanguages.Add(supportedLanguages.Find(CultureInfo.GetCultureInfo(language))
);
                }
            }
        }

        static public int SoftwareCompilationErrors(Project project)
        {
            int errors = 0;
            foreach (Device device in project.Devices)
            {
                Software softwareBase;
                PlcSoftware plcSoftware = null;

                // Returns PlcSoftware
                DeviceItemComposition deviceItemComposition = device.DeviceItems;
                foreach (DeviceItem deviceItem in deviceItemComposition)
                {
                    SoftwareContainer softwareContainer =
deviceItem.GetService<SoftwareContainer>();
                    if (softwareContainer != null)
                    {
                        softwareBase = softwareContainer.Software;
                        plcSoftware = softwareBase as PlcSoftware;
                    }
                }

                ICompilable compileService =
plcSoftware.GetService<ICompilable>();
                CompilerResult result = compileService.Compile();
                errors += result.ErrorCount;
            }
            return errors;
         }


        static public void WriteCompilerResults(CompilerResult result)
        {
            Console.WriteLine("State:" + result.State);
            Console.WriteLine("Warning Count:" + result.WarningCount);
            Console.WriteLine("Error Count:" + result.ErrorCount);
            RecursivelyWriteMessages(result.Messages);
        }
        static public void
RecursivelyWriteMessages(CompilerResultMessageComposition messages, string indent
= "")
        {
            indent += "\t";
            foreach (CompilerResultMessage message in messages)
            {
                Console.WriteLine(indent + "Path: " + message.Path);
                Console.WriteLine(indent + "DateTime: " + message.DateTime);
                Console.WriteLine(indent + "State: " + message.State);
                Console.WriteLine(indent + "Description: " +
message.Description);
```

```
                Console.WriteLine(indent + "Warning Count: " +
message.WarningCount);
                Console.WriteLine(indent + "Error Count: " + message.ErrorCount);
                RecursivelyWriteMessages(message.Messages, indent);
            }
        }

        static public void ImportHardware(Project project, string projectpath,
string logpath)
        {
            FileInfo amlPath = GetAMLPath(new DirectoryInfo(projectpath));
            string hardwarepath = amlPath.FullName;
            string logpathcomplete = logpath + @"\import.log";
            CaxProvider caxProvider = project.GetService<CaxProvider>();
            if (caxProvider != null)
            {
                caxProvider.Import(new FileInfo(hardwarepath), new
FileInfo(logpathcomplete), CaxImportOptions.MoveToParkingLot);
            }
        }

        static public void ImportSoftwareBlock(Project project, string blockPath,
ImportOptions importOptions)
        {
            string groupPath = GetSoftwareGroupPath(new FileInfo(blockPath));
            foreach (Device device in project.Devices)
            {
                Software softwareBase;
                PlcSoftware plcSoftware = null;

                // Returns PlcSoftware
                DeviceItemComposition deviceItemComposition = device.DeviceItems;
                foreach (DeviceItem deviceItem in deviceItemComposition)
                {
                    SoftwareContainer softwareContainer =
deviceItem.GetService<SoftwareContainer>();
                    if (softwareContainer != null)
                    {
                        softwareBase = softwareContainer.Software;
                        plcSoftware = softwareBase as PlcSoftware;
                    }
                }
                PlcBlockGroup BlockGroup = CreateBlockGroups(plcSoftware,
groupPath);
                BlockGroup.Blocks.Import(new FileInfo(blockPath), importOptions);
            }
        }

        static public void ImportTagTable(Project project, string tagTablePath,
ImportOptions importOptions)
        {
            string groupPath = GetSoftwareGroupPath(new FileInfo(tagTablePath));
            foreach (Device device in project.Devices)
            {
                Software softwareBase;
                PlcSoftware plcSoftware = null;

                // Returns PlcSoftware
                DeviceItemComposition deviceItemComposition = device.DeviceItems;
                foreach (DeviceItem deviceItem in deviceItemComposition)
                {
```

```
                    SoftwareContainer softwareContainer =
deviceItem.GetService<SoftwareContainer>();
                    if (softwareContainer != null)
                    {
                        softwareBase = softwareContainer.Software;
                        plcSoftware = softwareBase as PlcSoftware;
                    }
                }
                PlcTagTableGroup tagTableGroup =
CreateTagTableGroups(plcSoftware, groupPath);
                tagTableGroup.TagTables.Import(new FileInfo(tagTablePath),
importOptions);
            }
        }

        static public void ImportDataType(Project project, string dataTypePath,
ImportOptions importOptions)
        {
            string groupPath = GetSoftwareGroupPath(new FileInfo(dataTypePath));
            foreach (Device device in project.Devices)
            {
                Software softwareBase;
                PlcSoftware plcSoftware = null;

                // Returns PlcSoftware
                DeviceItemComposition deviceItemComposition = device.DeviceItems;
                foreach (DeviceItem deviceItem in deviceItemComposition)
                {

                    SoftwareContainer softwareContainer =
deviceItem.GetService<SoftwareContainer>();
                    if (softwareContainer != null)
                    {
                        softwareBase = softwareContainer.Software;
                        plcSoftware = softwareBase as PlcSoftware;
                    }
                }

                string dataTypeName =
dataTypePath.Substring(dataTypePath.LastIndexOf('\\') + 1); //
thenameofthetype.xml
                dataTypeName =
dataTypeName.Remove(dataTypeName.LastIndexOf('.')); // thenameofthetype
                if (plcSoftware.TypeGroup.Types.Find(dataTypeName) == null) // If
a type is created automatically with the CPU it will be already in the project in
the main group, so only import it when its not there yet.
                {
                    PlcTypeGroup dataTypeGroup =
CreateDataTypeGroups(plcSoftware, groupPath);
                    dataTypeGroup.Types.Import(new FileInfo(dataTypePath),
importOptions);
                }
                else Console.WriteLine(dataTypeName + " not imported. It is
already in the project.");
            }
        }

        public struct LogStruct
        {
            public List<string> errors;
            public List<string> warnings;
            public string logtxt;
```

```
            public LogStruct(string logpath)
            {
                logtxt = File.ReadAllText(logpath);
                errors = new List<string>();
                warnings = new List<string>();
                IEnumerable<string> Lines = File.ReadLines(logpath);
                foreach (string line in Lines)
                {
                    if (line.Length > 29 && line.Substring(24, 5) == "ERROR")
errors.Add(line); // Example: "08.02.2018 10:37:02 AM: ERROR: device item 'DP
interface_1' could not be processed."
                    else if (line.Length > 28 && line.Substring(24, 4) == "WARN")
warnings.Add(line); // Example: "08.02.2018 10:37:02 AM: WARN : Address Attribute
value '2.4.6.91' could not be set for node 'X3'."
                }
            }
        }

        static public PlcTypeGroup CreateDataTypeGroups(PlcSoftware plcSoftware,
string softwareGroupPath)
        {
            string groupPath = softwareGroupPath;
            PlcTypeGroup dataTypeGroup = plcSoftware.TypeGroup;
            int n = softwareGroupPath.Split('\\').Length - 1; // Number of
subgroups
            if (n != 0)
            {
                for (int i = 0; i < n; i++)
                {
                    string name = groupPath.Substring(0,
groupPath.IndexOf(@"\"));
                    if (dataTypeGroup.Groups.Find(name) == null)
dataTypeGroup.Groups.Create(name);
                    groupPath = groupPath.Remove(0, name.Length + 1);
                    dataTypeGroup = dataTypeGroup.Groups.Find(name);
                }
            }
            return dataTypeGroup;
        }

        static public PlcTagTableGroup CreateTagTableGroups(PlcSoftware
plcSoftware, string softwareGroupPath)
        {
            string groupPath = softwareGroupPath;
            PlcTagTableGroup tagtablegroup = plcSoftware.TagTableGroup;
            int n = softwareGroupPath.Split('\\').Length - 1; // Number of
subgroups
            if (n != 0)
            {
                for (int i = 0; i < n; i++)
                {
                    string name = groupPath.Substring(0,
groupPath.IndexOf(@"\"));
                    if (tagtablegroup.Groups.Find(name) == null)
tagtablegroup.Groups.Create(name);
                    groupPath = groupPath.Remove(0, name.Length + 1);
                    tagtablegroup = tagtablegroup.Groups.Find(name);
                }
            }
            return tagtablegroup;
        }
```

```
        static public PlcBlockGroup CreateBlockGroups(PlcSoftware plcSoftware,
string softwareGroupPath)
        {
            string groupPath = softwareGroupPath;
            PlcBlockGroup blockGroup = plcSoftware.BlockGroup;
            int n = softwareGroupPath.Split('\\').Length - 1; // Number of
subgroups
            if (n != 0)
            {
                for (int i = 0; i < n; i++)
                {
                    string name = groupPath.Substring(0,
groupPath.IndexOf(@"\"));
                    if (blockGroup.Groups.Find(name) == null)
blockGroup.Groups.Create(name);
                    groupPath = groupPath.Remove(0, name.Length + 1);
                    blockGroup = blockGroup.Groups.Find(name);
                }
            }
            return blockGroup;
        }
    }
}
```

15.8 **Others.cs**

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Http;
using System.Web;
using System.Text;

namespace TIAPortalApplication.Utils.Generate
{
    class Others
    {
        #region FileInfo[] of a directory and its subdirectories

        /* Se utiliza una lista para poder añadir elementos dinámicamente sin
especificar el tamaño total
         * Se utiliza la lista como variable global para que en el método
recursivo se mantenga de una iteración a otra
         * WalkDirectoryTreeList(...) - Rellena esa lista (private)
         * WalkDirectoryTree(...) - Método con el que interactúa el usuario
         *                          · Resetea la lista;
         *                          · Rellena la lista;
         *                         · La devuelve en forma de array
         */

        private static List<FileInfo> fileslist;

        private static void WalkDirectoryTreeList(DirectoryInfo root, string
searchPattern = "*.*")
        {
            DirectoryInfo[] subDirs = null;
            FileInfo[] files = null;

            // First, process all the files directly under this folder
            files = root.GetFiles(searchPattern);

            if (files != null)
            {
                foreach (FileInfo fi in files)
                {
                    fileslist.Add(fi);
                }

                // Now find all the subdirectories under this directory.
                subDirs = root.GetDirectories();

                foreach (DirectoryInfo dirInfo in subDirs)
                {
                    // Recursive call for each subdirectory.
                    WalkDirectoryTreeList(dirInfo);
                }
            }

        }

        public static FileInfo[] WalkDirectoryTree(DirectoryInfo root, string
searchPattern = "*.*")
```

```
        {
            FileInfo[] filesarray;
            fileslist = new List<FileInfo>();
            WalkDirectoryTreeList(root, searchPattern);
            filesarray = fileslist.ToArray();
            return filesarray;
        }

        #endregion

        static public string GetSoftwareGroupPath(FileInfo file, bool
withFileName = true)
        {
            string directory = file.FullName;
            string deviceSoftwareRoot =
directory.Substring(file.DirectoryName.IndexOf(@"\Software\") +
@"\Software\".Length);// PLC_1\PLC Tags\...\...xml
            string plcTagsRoot =
deviceSoftwareRoot.Substring(deviceSoftwareRoot.IndexOf(@"\") + 1); // PLC
Tags\...\...xml
            string groupRoot = plcTagsRoot.Substring(plcTagsRoot.IndexOf(@"\") +
1); // ...\....xml
            if (groupRoot.IndexOf(@"\") == -1) return ""; // retrun "" if
filename.xml is in the main Group.
            else if (withFileName) return groupRoot; // return
"Group1\Group2\...\GroupN\filename.xml" if withFileName is true
            else
            {
                groupRoot = groupRoot.Remove(groupRoot.LastIndexOf(@"\") + 1);
                return groupRoot; // rerutn "Group1\Group2\...\GroupN\" if
withFileName is false
            }
        }
    }
}

        private static string user = "admin";
        private static string pass = "1234";
        private static string UrlExistDB =
"http://localhost:8080/exist/rest/db/";

        public async static Task<string> ExistGet(string extraUrl) //
"http://localhost:8080/exist/rest/db/ + extraUrl"
        {
            var credenciales = new HttpClientHandler { Credentials = new
NetworkCredential(user, pass) };
            using (var httpClient = new HttpClient(credenciales) { Timeout =
TimeSpan.FromMilliseconds(10000) })
            {
                var GetResponse = await
httpClient.GetStringAsync(string.Concat(UrlExistDB, extraUrl));
                return GetResponse;
            }
        }

        public async static Task<string> ExistGetCollection(string
rootCollection, string Destination) //Download everything inside a collection and
save it in the Destination directory.
        {
            string actual = rootCollection;
            string response = (await ExistGet(actual));
            XmlDocument xmlResult = new XmlDocument();
```

```
            try { xmlResult.LoadXml(response); }
            catch { return "Invalid XML response"; }

            if (xmlResult.DocumentElement.Name == "exist:result") // Have more
collections or resources inside
            {
                XmlNodeList nodes =
xmlResult.SelectNodes("child::node()/child::node()/*");
                foreach (XmlNode node in nodes)
                {
                    actual = actual + @"\" + node.Attributes["name"].InnerText;
                    string task = await ExistGetCollection(actual, Destination);
                    actual = actual.Remove(actual.LastIndexOf('\\'));
                }
            }
            else // XML or AML file
            {
                string saveDir = Uri.UnescapeDataString((Destination +
actual).Remove((Destination + actual).LastIndexOf('\\')));
                Directory.CreateDirectory(saveDir);
                xmlResult.Save(Uri.UnescapeDataString(Destination + actual));
            }
            return "";
        }
```