UNIVERSIDAD NACIONAL DE QUILMES

Departamento de Ciencia y Tecnología

Ingeniería en Automatización y Control Industrial

Diseño de software y hardware de un controlador lógico programable (PLC) y su entorno de programación

Alumno: ERIC NICOLÁS PERNIA. **Legajo**: 15925.

Director: CARLOS LOMBARDI.

Quilmes, Buenos Aires, Argentina.

Presentación:
Octubre de 2013

RESUMEN

En este trabajo final se presenta el diseño de un PLC y su entorno de programación. Los principios generales de este diseño son: el estricto seguimiento de la Norma IEC 61131, la posibilidad de construir un PLC de bajo costo de Hardware, el uso de herramientas que puedan correr en múltiples sistemas operativos, la adaptabilidad del software generado desde el entorno de programación a cualquier arquitectura de controlador, y la inclusión de facilidades de edición de programas (en particular en lenguajes gráficos) que se correspondan con el estado del arte reflejado en las herramientas comerciales de uso más extendido.

De esta forma, se se obtiene un diseño de PLC y de su entorno de programación los cuales se independizan del Hardware logrando una gran compatibilidad, permitiendo al mismo tiempo la edición ágil de los programas a ser ejecutados en el controlador.

Se incluyen en esta propuesta:

- Un diseño general del hardware para un micro PLC que favorece implementaciones de bajo costo y adecuadas prestaciones.
- Un diseño de interfaz de usuario del entorno de programación, con formatos de pantalla, interacciones entre los distintas pantallas, e indicaciones generales de usabilidad.
- Una especificación del modelo computacional de los conceptos de programación PLC incluidos en la norma IEC 61131, concebida a partir de los principios fundamentales de la programación con objetos.
- La definición del entorno de software de ejecución a montar sobre el hardware, de forma tal que los programas generados desde el entorno de programación se ejecuten sobre el entorno de ejecución definido.

Para demostrar la validez del diseño se desarrollaron: un prototipo electrónico de PLC (Hardware) en el cual sólo se consideran entradas y salidas booleanas y sin funciones de comunicación, un entorno de programación desarrolado de acuerdo al diseño propuesto, y una implementación del entorno de ejecución de software en el PLC sobre el que se montan los programas desarrollados desde el entorno de programación. Esta implementación cumple con las pautas incluidas en el diseño, respetando los principios generales anteriormente descriptos.

En particular, para garantizar independencia de Hardware se eligió freeRTOS como sistema operativo del PLC que posee "ports"[1] para la mayoría de los microcontroladores del mercado. El entorno de programación se implementó sobre Pharo-Smalltalk, que es un ambiente de desarrollo que permite que el software construido funcione en Windows, Linux y MAC OS X; de esta forma se logra también la independencia del sistema operativo del equipo en el que se ejecuta dicho entorno. Ambos software (Pharo y freeRTOS) son herramientas libres minimizando los costos de desarrollo.

[1] En freeRTOS se refiere como "ports.ª la capa del software dependiente del Hardware, es decir, del microcontroaldor.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

INTRODUCCIÓN E IMPORTANCIA DEL TEMA

1.1. Marco temático Controladores Lógicos Programables (PLC)

1.1.1. Definición y características principales

Un Controlador Lógico Programable (en adelante PLC, llamado así por sus siglas en inglés), o Autómata Programable, es un sistema electrónico programable diseñado para controlar diversos tipos de máquinas o procesos en tiempo real en un entorno industrial. Está compuesto por un microprocesador o microcontrolador, memorias, interfaces de entradas/salidas y circuitos adicionales. Puede programarse utilizando distintos lenguajes, tanto textuales como gráficos.

1.1.2. Programación de un PLC

Para programar un PLC, se debe comunicar al usuario con el Hardware PLC, mediante un dispositivo que permita escribir y poner a punto el programa que se ejecutará. Éste dispositivo se conoce como "consola de programación", que puede ser un dispositivo con pantalla diseñado específicamente para tal fin, o bien, una computadora de propósito general, corriendo un Software con las mismas habilidades (más común en la actualidad).

1.1.3. Funciones de un PLC

Un PLC es capaz de:

1. Adquirir datos del proceso por medio de sensores conectados eléctricamente a sus entradas digitales y analógicas.

2. Almacenar datos en memoria.

3. Tomar decisiones mediante un programa en memoria ingresado por el usuario que consiste de instrucciones las cuales implementan funciones especificas tales como lógicas, secuenciales, temporización, contadores y matemáticas.

4. Actuar sobre el proceso mediante sus salidas digitales y analógicas conectadas eléctricamente a actuadores.

5. Comunicarse con otros sistemas externos.

1.1.4. Aplicaciones de los PLC

El PLC es el equipo más utilizado en la automatización industrial y domótica. Es por esto, que en el desarrollo de la práctica de su profesión un Ingeniero en Automatización y Control Industrial, utiliza estos equipos con mucha frecuencia. Existen seis áreas generales de aplicación de PLC:

1. Control secuencial.
2. Control de movimiento.
3. Control de procesos.
4. Monitoreo y supervisión de procesos.
5. Administración de datos.
6. Comunicaciones.

1.1.5. Contexto histórico

En 1968 GM Hydramatic (la división de transmisiones automáticas de General Motors) ofertó un concurso para una propuesta del reemplazo electrónico de los sistemas cableados. En respuesta a estas necesidades el ingeniero Estadounidense Dick Morley invento el PLC.

Un PLC permite realizar, mediante un cambio en el programa que ejecuta, modificaciones que antes implicaban un cambio físico de dispositivos en los sistemas cableados. Esto permite ahorrar tiempo y costos de mano de obra. El PLC es de tamaño reducido, con costos inferiores de mantenimiento y existe la posibilidad de controlar más de una máquina con el mismo equipo.

Por otro lado, el PLC requiere de conocimientos tanto de electricidad, como de lógica y programación para el desarrollo de automatismos. En consecuencia, es necesario contar con técnicos calificados para ocuparse de su operación y mantenimiento.

1.1.6. Estado actual de los PLC

Existen muchos fabircantes de PLC en el mercado, desde grandes empresas multinacionales, hasta pequeños desarrollos.

En Argentina, el PLC a utilizar se elige según el tipo de industria a la cual se aplica, sus costos y especificaciones de seguridad entre otros factores. Una encuesta realizada a profesionales dedicados a la programación de éstos equipos, sobre qué marcas son las más utilizadas en el país, se resumen en la tabla [1.1].

Los entornos de programación de PLC han evolucionado ampliamente desde sus comienzos hasta hoy en día. En el sitio web [22], pueden observarse diferentes entornos de programación de PLC.

En la actualidad, las empresas líderes en el mercado tienden a intentar resolver el problema completo de automatización; esto es, en un mismo entorno proveen programación y simulación tanto del software que contiene el PLC, como de sus módulos de entradas, salidas, de comunicación, conexiónes de red entre los dispositivos y Paneles para HMI/SCADA. Así, se pueden resolver muchos problemas de automatización utilizando una única herramienta de software, reduciendo los tiempos de desarrollo y entrenamiento del personal encargado de utilizarlos.

Como desventaja, cada software es distinto, creando una dependencia de una cierta marca a las empresas que brindan servicios de automatización.

	Fabricantes		
	Rockwell	Siemens	Schneider Electric - Telemecanique
Rangos de precios	Desde U$S 500 hasta los U$S 20.000	Desde U$S300 hasta los U$S 20.000	Desde U$S 300 hasta los U$S 15.000
Ejemplo de modelos de PLC de bajas prestaciones	RS1500	S7 200 / S7 1200	Twido
Ejemplo de modelos de PLC de altas prestaciones	Contrologic	S7 400 / S7 1500	CPU Premiun / M340
Software de programación	RS 5000	TIA V12	Unity 7.0
Empresas que los utilizan	Petroleras, Mobile (ex ESSO), Clorox (fabrica lavandina Ayudín entre otros).	Envasadoras, por ejejmplo, Tetrapack, La salteña, Aluar, Cervecería Quilmes, Volkswagen, Panamerican Energy (extracción de petroleo), CEPAS (fabrica la bebida Gancia entre otros).	Automotrices, Aysa, Sadesa Curtiembre.

Tabla 1.1: Marcas de PLC más utilizadas en el país.

1.2. Norma IEC 61131 - Controladores Programables

Como fue anticipado en la sección 1.4, existen diversos fabricantes de PLC, tipos y modelos adecuados a las distintas necesidades. Esto implica muchas incompatibilidades entre sistemas. Para solventarlo, se creó la norma IEC 61131, que es el resultado del gran esfuerzo realizado por siete multinacionales, con muchos años de experiencia en el campo de la automatización industrial; logrando el primer paso en la estandarización de los controladores programables y sus periféricos, incluyendo los lenguajes de programación que se deben utilizar.

La finalidad de esta norma es:

1. Definir e identificar las características principales que se refieren a la selección y aplicación de los PLC y sus periféricos.
2. Especificar los requisitos mínimos para las características funcionales, las condiciones de servicio, los aspectos constructivos, la seguridad general y los ensayos aplicables a los PLC y sus periféricos.
3. Definir los lenguajes de programación de uso más corriente, las reglas sintácticas y semánticas, el juego de instrucciones fundamentales, los ensayos y los medios de ampliación y adaptación de los equipos.
4. Dar a los usuarios una información de carácter general y unas directrices de aplicación.
5. Definir las comunicaciones entre los PLC y otros sistemas.

Esta norma, está formada por las siguientes partes:

1. IEC 61131-1. Información general.
2. IEC 61131-2. Especificaciones y ensayos de los equipos.
3. IEC 61131-3. Lenguajes de programación.

4. IEC 61131-4. Guías de usuario.
5. IEC 61131-5. Comunicaciones.
6. IEC 61131-6. Reservada.
7. IEC 61131-7. Fuzzy Control.
8. IEC 61131-8 Guías de programación.

1.3. IEC 61131-3 Lenguajes de programación.

En el presente Trabajo Final se utiliza la segunda edición de esta parte de la norma, publicada en el año 2003 (IEC 61131-3:2003). La misma especifica los modelos de software, comunicación, programación; cuatro lenguajes de programación y elementos SFC utilizados para la organización de programas confeccionados mediante dichos lenguajes. En las siguientes secciones se realiza un resumen de estas características.

1.3.1. Modelo de Software

El modelo de Software definido en la norma IEC 61131-3 se expone en la figura 1.1.

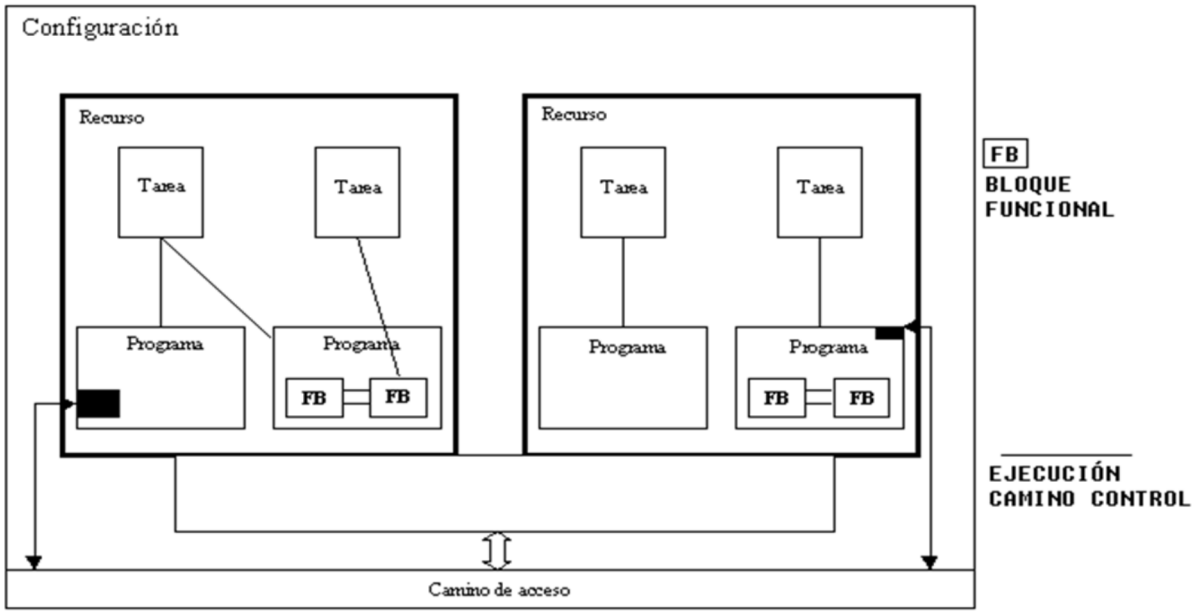


Figura 1.1: Modelo de Software de PLC según norma IEC 61131-3.

El elemento de Software de mayor jerarquía, requerido para solucionar un problema de control particular, se conoce como **Configuración**. Una configuración, es específica para un tipo de sistema de control, incluyendo las características del hardware: procesadores, direccionamiento de la memoria para los canales de E/S y otras capacidades del sistema.

Dentro de una configuración, se pueden definir uno o más **Recursos**. Éste se puede entender como un procesador con su respectivo Sistema Operativo capaz de ejecutar programas IEC.

Cada Recurso puede contener una o más **Definiciones de Tareas**, que controlan la ejecución de un conjunto de **Programas** y/o **Bloques de Función**.

1.3.2. Modelo de comunicación

Define brevemente cómo se comunican dos configuraciones, mediante **Caminos de Acceso**, o bien, entre Programas o Bloques de Función. La descripción detallada del modelo de comunicación se encuentra en la parte 5 de la norma.

1.3.3. Modelo de programación

El modelo de programación define:

- Tipos de datos
- Literales
- Variables
- Categorías de variables
- Unidades de Organización de Programa

Las Unidades de Organización de Programa, (POU), están formadas por distintas categorías de Declaraciones de Variables, y un cuerpo de programa confeccionado en alguno de los cuatro lenguajes propuestos por la norma. Existen 3 tipos de POU:

- **Funciones:** IEC 61131-3 especifica funciones estándar y funciones definidas por el usuario (o funciones derivadas); las primeras son, por ejemplo, ADD (suma), ABS (valor absoluto), SQRT (raíz cuadrada), SIN (seno), y COS (coseno), y las restantes, una vez implementadas por el usuario, pueden ser utilizadas dentro de cualquier POU. Las funciones no pueden contener ninguna información de estado interno, es decir, que la invocación de una función con los mismos argumentos (parámetros de entrada) debe suministrar siempre el mismo valor (salida).
- **Bloques de Función:** Representan funciones de control especializadas. Contienen, datos, instrucciones, y además, pueden guardar los valores de las variables. Debido a esto, es un bloque altamente reutilizable, ya que permite definir instancias del mismo independientes entre sí, las cuales contienen la información de su estado interno. De la misma forma que en el caso de las Funciones, existen Bloques de Función estándar (por ejemplo, biestables, detección de flancos, contadores, temporizadores, etc.) y el usuario puede definir sus propios. En su interior puede contener llamados a Funciones.
- **Programas:** Se define como un conjunto lógico de todos los elementos y construcciones del lenguaje de programación, que se requiere para el control de una máquina o proceso mediante el sistema PLC. Un programa puede contener en su interior llamados a Funciones e instancias de Bloques de Función.

1.3.4. Elementos de SFC

SFC son las siglas en inglés de Secuencial Function Charts, que significa Gráfico Funcional Secuencial. SFC deriva de las Redes de Petri y Grafcet (IEC 848). Los elementos del SFC proporcionan un medio para subdividir una POU de un autómata programable, en un conjunto de etapas y transiciones interconectadas por medio de enlaces directos. Las etapas llevan asociadas un conjunto de bloques de acción y transiciones entre las mismas, las cuales contienen una condición para permitir efectuar dicha transición. Cuando esta condición se cumple, causa la desactivación de la etapa anterior a la transición y la activación de la siguiente. Los bloques de acción permiten realizar el control del proceso. Cada elemento puede ser programado en alguno de los lenguajes IEC,

incluyéndose el propio SFC. Dado que los elementos del SFC requieren almacenar información, las únicas POU que se pueden estructurar utilizando estos elementos son los Bloques de Función y Programas.

Se pueden usar secuencias alternativas y paralelas. Debido a su estructura general, de sencilla comprensión, SFC permite la transmisión de información entre distintas personas, con distintos niveles de preparación y responsabilidad dentro de una empresa. Un ejemplo gráfico de SFC se ofrece en la figura 1.2.

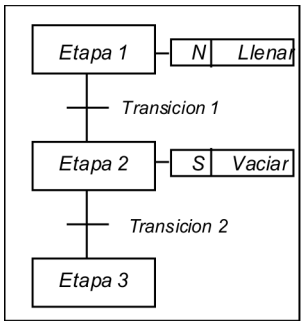


Figura 1.2: Ejemplo de SFC, definido en la norma IEC 61131-3.

1.3.5. Lenguajes de programación

Se define la sintaxis y semántica de cuatro lenguajes de programación. Comprendidos éstos, en dos de tipo textual y dos de tipo gráfico. En el primer grupo se encuentran:

- **Instruction List (IL)**: Lista de instrucciones, lenguaje similar al lenguaje Ensamblador de un CPU con un único acumulador.
- **Structured Text (ST)**: Texto estructurado, es un lenguaje de medio nivel de características similares al lenguaje Pascal.

Mientras que en el segundo:

- **Ladder Diagram (LD)**: Diagrama de escalera o contactos, representa gráficamente un circuito eléctrico de contactores y relés.
- **Function Block Diagram (FBD):** Diagrama de bloques funcionales, forma gráficamente circuitos eléctricos del tipo lógicos.

En la figura 1.3 se ejemplifica el mismo programa creado en los cuatro lenguajes.

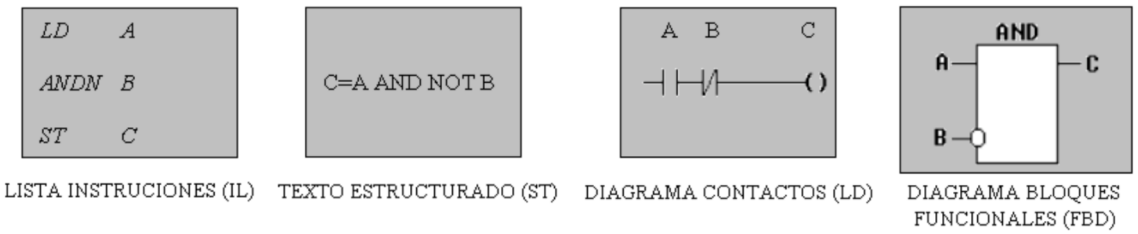


Figura 1.3: Un mismo programa creado en los cuatro lenguajes IEC 61131-3.

1.4. Justificación del tema seleccionado

Mediante el análisis, práctica de uso, e instalación de PLC a nivel industrial, se revelan problemas, cuya determinación propician el siguiente diagnóstico:

- En la actualidad existen diferencias entre los lenguajes de los software de programación de los distintos fabricantes de PLC del mercado, en contradicción con norma IEC 61131-3 (ver sección 1.3), la cual pretende estandarizar los lenguajes de programación en la automatización industrial. Estas diferencias, se deben en su mayor parte a implementaciones parciales de la norma. En consecuencia un usuario debe aprender a programar cada uno de los PLC según el fabricante, retrasando tiempos de desarrollo, e impidiendo la portabilidad de los programas creados por el usuario entre las diferentes marcas.
- Otro inconveniente de muchos PLC radica en la nula portabilidad de su software de programación. En su mayoría éstos funcionan sólo en PC, con microprocesador de 32 bits, bajo el sistema operativo Microsoft Windows; debiéndose así utilizar una máquina virtual en computadoras con otros sistemas operativos o con procesadores de 64 bits para poder programar estos PLC.
- El costo actual resulta elevado, en particular para instituciones educativas en cuyas currículas se incluya la capacitación en el uso de los mismos.

Por lo tanto, se establece que la alternativa óptima, es el diseño de un PLC, prototipo y su software de programación que resulten independientes, tanto de los componentes utilizados en su construcción, como del Hardware o Sistema Operativo del equipo en el que se ejecuta el entorno de programación; permitiendo así la concepción de soluciones de bajo costo.

Este Trabajo Final ha sido encarado desde la convicción de que un diseño con las características planteadas puede impulsar una mayor utilización de PLC dentro de currículas educativas, con las consiguientes ventajas para la formación de recursos humanos capacitados. Además se cree que puede facilitar el desarrollo, dentro del ámbito nacional, de la actividad ligada a la construcción y utilización de estos equipos.

Finalmente, la realización de un proyecto de la envergadura propuesta implica enfrentar el desafío profesional y personal (del autor), de crear una herramienta superadora a las dificultades y limitaciones encontradas.

Como estrategia para la acción, se recurre a los aprendizajes adquiridos en el trayecto universitario y la obtención de nuevos conocimientos surgidos por la demanda de una solución. Es decir, que a partir de la realización de este Trabajo Final se obtendrá una mayor capacidad material, técnica y humana para contribuir a mejorar la situación actual referente a estos equipos.

Capítulo 2

OBJETIVO

2.1. Objetivo principal

El objetivo del Trabajo Final es la obtención de un diseño de PLC, la implementación de un prototipo y la construcción del software de computadora necesario para la programación de distintos PLC que respondan al diseño mencionado. En las siguientes secciones se describen las características principales en los productos a desarrollar.

2.2. Diseño del PLC

Realizar un diseño del sistema operativo y funciones de PLC, que permita la portabilidad a distintos controladores disponibles en el mercado creando una herramienta flexible independientemente de su hardware.

2.3. Prototipo de PLC

Construir un prototipo de bajo costo de fabricación, que permita a los estudiantes de Ingeniería en Automatización y Control Industrial, poder utilizarlo en las prácticas de sus materias.

2.4. Diseño del Software de computadora para la programación del PLC

Crear un software portable entre distintos sistemas operativos y arquitecturas de computadoras, permitiendo de esta forma, su utilización tanto en PC como MAC, con distintos sistemas operativos.

Modelado e implementación de la mayoría de los conceptos y lenguajes de programación de PLC, especificados en la norma IEC 61131-3, permitiendo al usuario decidir que lenguaje se adecua más a sus conocimientos y requerimientos.

Obtención de un archivo de programa textual en lenguaje IL que cumpla con con la norma IEC 61131-3 para lograr compatibilidad con otros PLC que adhieran a la misma.

Capítulo 3

DISEÑO

3.1. Descripción general

En el presente Trabajo Final se presenta el diseño del software y hardware de un Controlador Lógico Programable (PLC) y un IDE[1] destinado a su programación. El sistema se compone de una computadora ejecutando el Entorno de Programación de PLC, y un Hardware PLC conectado a dicha computadora, por medio de una interfaz de comunicación, que será donde el IDE descargue el programa creado por el usuario (figura [3.1]).

Figura 3.1: Modelo conceptual.

3.1.1. Diseño de PLC

Se elije realizar un diseño de Micro PLC, es decir, con menos de 64 E/S, de construcción compacta o integral, esto es, que posee en el mismo gabinete CPU y módulos de entradas y salidas[2]. De esta forma, puede realizarse un prototipo de bajo costo, con un diseño de Hardware más simple en comparación con un PLC de mayor cantidad de entradas y salidas del tipo modular. Su diseño se expone en la sección 3.6.

El modelo de Software de PLC se expone en la figura [3.2]. En esta primer aproximación puede observarse que se compone de dos capas:

[1] Siglas en inglés de Entorno de Desarrollo Integrado. También llamado Entorno de Programación.
[2] En la implementación no se incluye la fuente de alimentación dentro del gabinete del equipo.

- **Programa de usuario.** El Programa de usuario corresponde a la capa de software realizada por el usuario, en consecuencia, es inherentemente cambiante.
- **Entorno de Ejecución.** Se llamará Entorno de Ejecución al conjunto de componentes de software que permite al Programa de usuario funcionar sobre el Hardware PLC. Esta capa de software es fija, su diseño se discute en la sección 3.5.

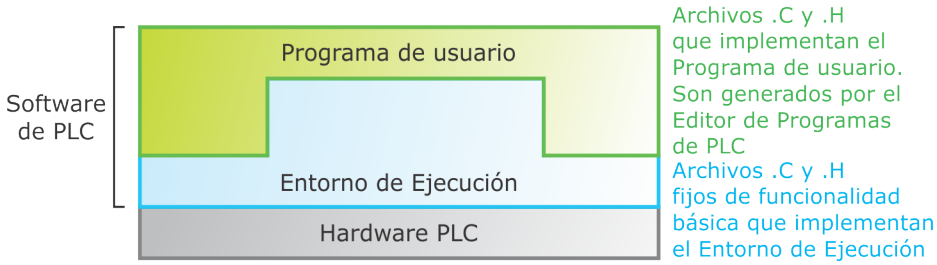


Figura 3.2: Modelo de Software de PLC.

Este Software está diseñado íntegramente en lenguaje de programación C. Se modulariza en varios archivos .C y .H, parte de ellos son generados desde el Editor de Programas de PLC (los cuales implementan el Programa de usuario, que se definirán en la sección 3.4.11), y el resto, son archivos fijos que implementan el Entorno de Ejecución (sección 3.5).

3.1.2. Entorno de Programación de PLC

El Entorno de Programación debe ser capaz de permitir realizar al usuario las siguientes tareas:

- Crear y editar un programa utilizando un lenguaje de programación de PLC.
- Compilar el programa.
- Descargar el programa a la memoria del Hardware PLC.

Para cumplir con estos requerimientos, se plantea el esquema general de la solución en la figura [3.3]. Aquí se exponen las partes que constituyen el sistema y sus relaciones.

El usuario desarrolla su programa, en alguno de los lenguajes definidos en la norma IEC 61131-3, utilizando el *Editor de programas de PLC*. Este editor es responsable de la traducción del programa de usuario al lenguaje de programación C. Luego, combinando estos archivos generados, con archivos fijos preexistentes, completa el Software de PLC en lenguaje C. Posteriormente, invoca al compilador de C enviándole la dirección de dicho Software. El compilador genera el código ejecutable, compilando el programa completo. Este código ejecutable se incorpora al Hardware del PLC mediante un componente al que se llamará *Programador de PLC*.

De los componentes propuestos en la figura [3.3] se desarrollarán:

- Editor de programas de PLC.
- Entorno de Ejecución.
- Hardware Controlador Programable.

En cuanto al Compilador de C y el Programador se utilizarán alternativas existentes en el mercado.

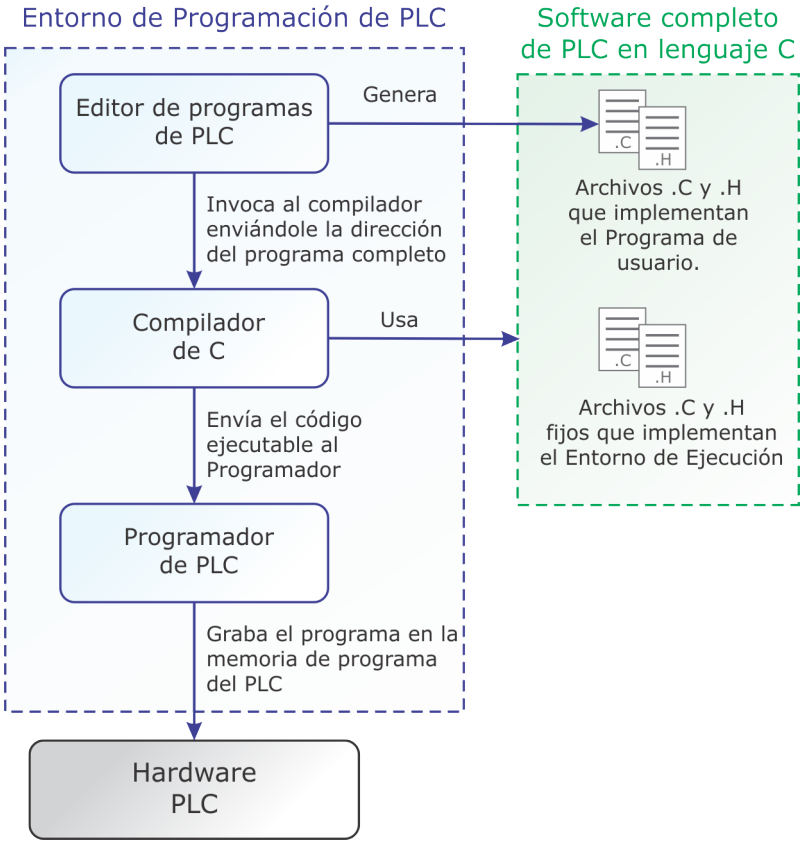


Figura 3.3: Modelo de Sistema.

3.2. Editor de programas de PLC

El diseño del editor de programas se realizó utilizando el patrón de arquitectura de software MVC (Modelo/Vista/Controlador). Este patrón indica una separación lógica en:

- Modelo: es el conjunto de objetos que modela el dominio de aplicación, en este caso se trata de un editor de programas de Controladores Programables.
- Vista: es el conjunto de objetos que modela la presentación de la información en pantalla.
- Controlador: es el conjunto de objetos que procesa las acciones del usuario, propagándolas adecuadamente al modelo y a la vista.

En el caso particular de este diseño se utilizan múltiples vistas y controladores los cuales se conectan donde les compete con el modelo. El diseño de la Interfaz Gráfica de Usuario del editor de programas se plantea en la sección 3.3 y su Modelo Computacional se expone en la sección 3.4.

Una característica general de esta propuesta que tiene consecuencias en el diseño del editor de programas es la decisión de respetar las definiciones de los modelos de software y programación incluidos en la norma IEC 61131-3 tanto como sea posible, evitando la introducción de características provenientes de un diseño o implementación particular. Se considera que esta determinación marca una diferencia entre el diseño propuesto y la mayor parte de las alternativas disponibles hoy en el área.

3.3. Interfaz Gráfica de Usuario (GUI)

El diseño elegido para la interfaz gráfica presenta elementos similares a los que se encuentran en herramientas ampliamente utilizadas en la industria, como las mencionadas en la Sección 1.1.6.

De esta forma se presenta a los profesionales del área un entorno que les resulta familiar, facilitando la aceptación de los controladores que se desarrollen de acuerdo con el diseño propuesto.

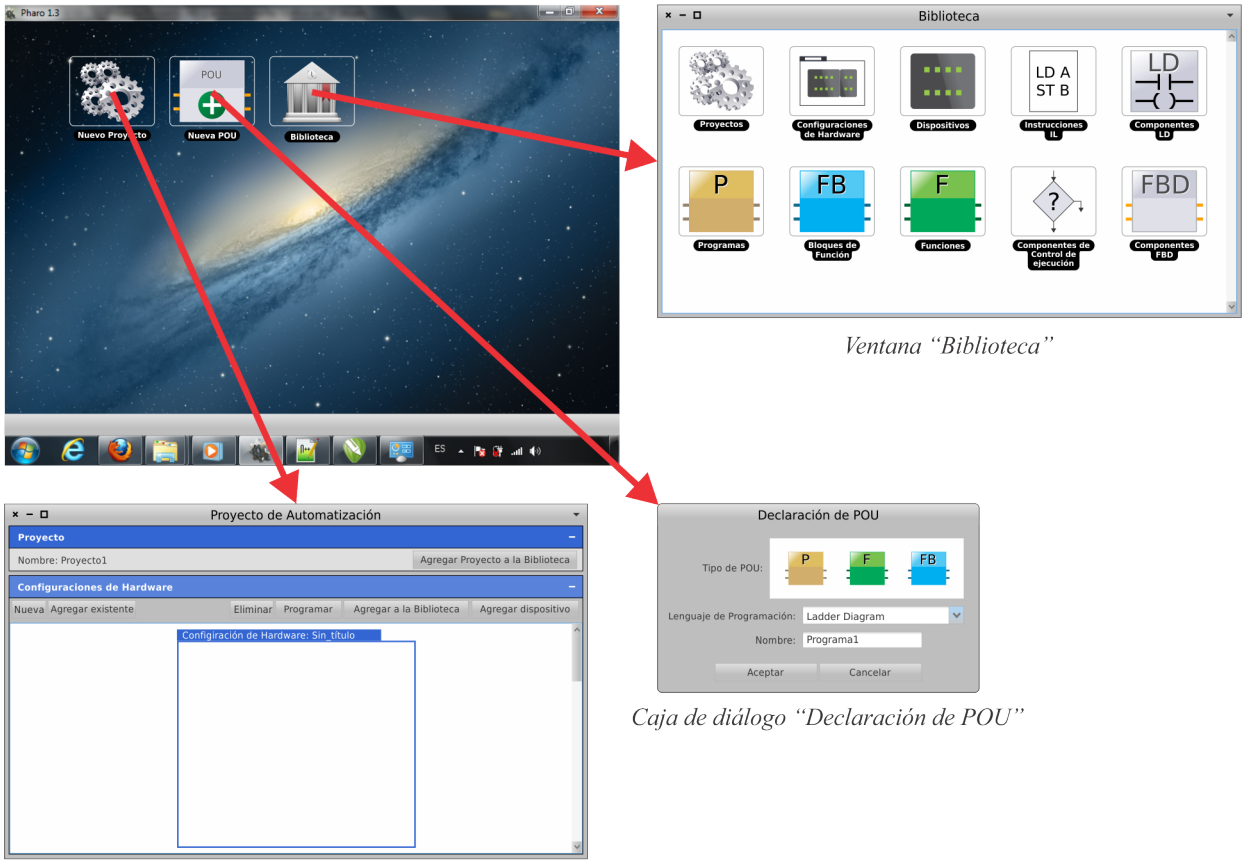
Por otra parte, recordando que uno de los objetivos de este trabajo final es proporcionar a los estudiantes de carreras de automatización y electrónica un entorno para el aprendizaje de la programación de controladores; la decisión de adoptar un tipo de interfaz gráfica que rescata elementos de uso común a las utilizadas en la industria apunta a brindar a los estudiantes una herramienta que, siendo de bajo costo, les permita al mismo tiempo experimentar con interfaces de usuario similares a las que probablemente deberán utilizar en su carrera profesional.

Al iniciar la aplicación el usuario se encuentra con una pantalla similar al "escritorio" de cualquier sistema operativo de Computadora actual que posee íconos y una barra inferior como se muestra en la figura [3.4].

Figura 3.4: Inicio de la aplicación.

Dichos íconos corresponden a "Nuevo Proyecto", "Nueva POU" y "Biblioteca" respectivamente. El primero, abre la ventana de edición de "Proyecto de Automatización" la cual se utiliza para la configuración de la estructura general de un proyecto de automatización (Sección 3.3.1). El segundo, abre la caja de diálogo de "Declaración de POU" para configurar los parámetros de una nueva Unidad de Organización de Programa (POU). Una vez configurados los parámetros, la edición de dicha POU se realiza mediante la ventana "Unidad de Organización de Programa (POU)" focalizada

en la especificación de la lógica de una POU utilizando un lenguaje de programación según define la norma IEC 61131-3 (Sección 3.3.2). Finalmente, el tercero, abre la ventana "Biblioteca" donde se muestran tanto los elementos estándar como los elementos definidos por el usuario (Sección 3.3.3). Las ventanas pueden minimizarse a la barra inferior o solaparse unas con otras. En la figura [3.5] se muestra un esquema de las ventanas que abre cada ícono.



Ventana "Biblioteca"

Caja de diálogo "Declaración de POU"

Ventana "Proyecto de Automatización"

Figura 3.5: Ventanas abiertas por cada ícono del escritorio.

De esta manera se modulariza la aplicación en ventanas más simples en comparación con la mayoría de las herramientas actuales basadas en una única ventana dividida en una barra superior con menú, una zona central de edición y dos columnas laterales de árbol de proyecto (a izquierda) y área de biblioteca (a derecha). En la figura [3.6] se muestra una comparativa entre el entorno de programación TIA[3] V10.5 de Siemens y el editor propuesto a baja resolución (800px x 600px) pudiéndose observar el mayor espacio disponible para la edición de una POU en el caso propuesto.

Otro efecto deseado, es focalizar la atención del usuario en la acción que realiza en particular, mostrando solamente información pertinente a dicho módulo. Ésto logra una mayor abstracción y simplificación del problema, facilitando al usuario el desarrollo de su programa.

Además se simplificó cada ventana teniendo en cuenta la futura utilización del editor de programas en dispositivos móviles tales como Tablets y Smathphones los cuales poseen pequeñas pantallas.

[3]Totally Integrated Automation.

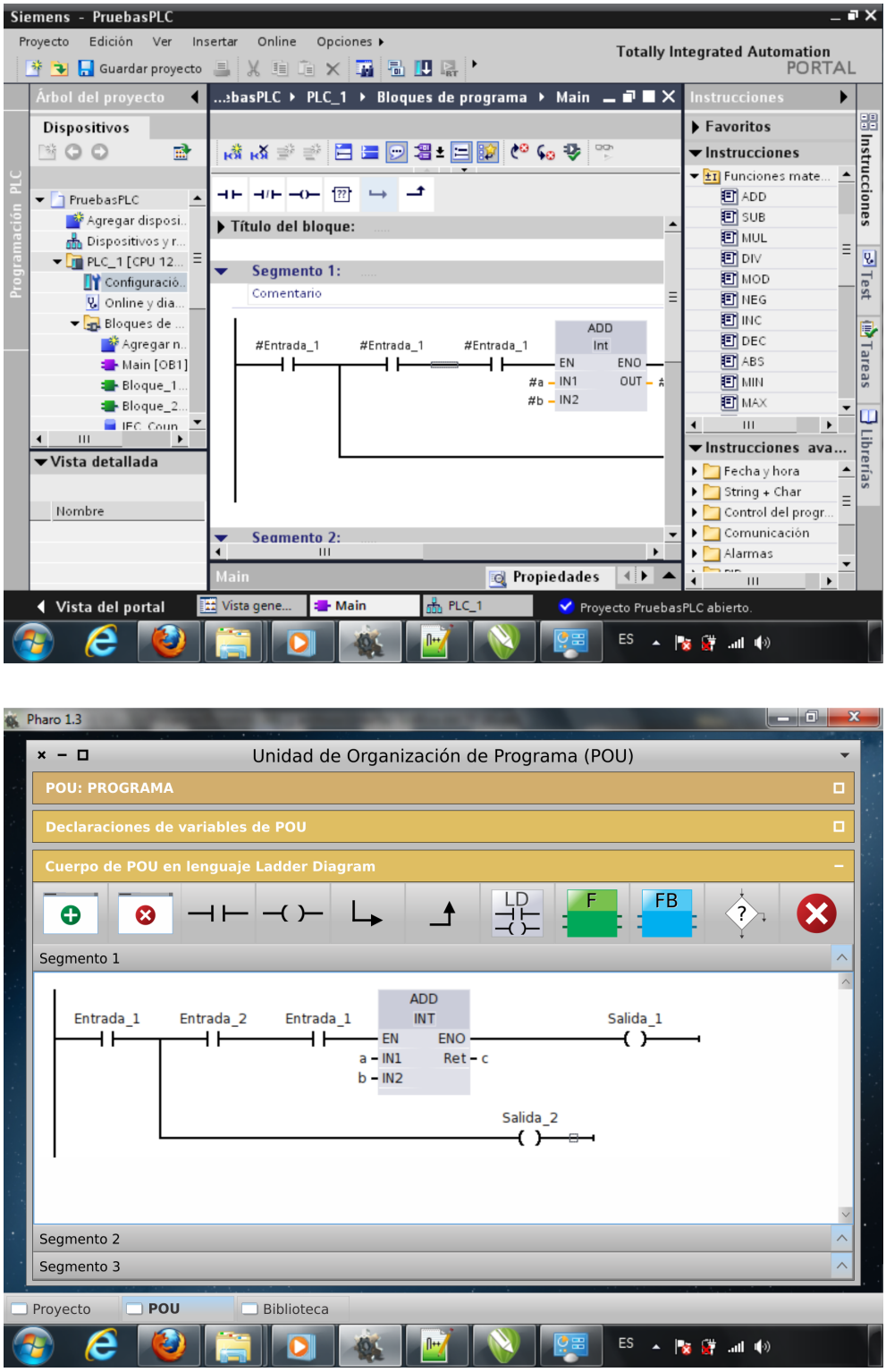


Figura 3.6: Programación de POU a resolución 800px x 600px. Imagen superior programa Siemens TIA. Imagen inferior diseño propuesto.

Cada vez que se agrega, edita o borra un componente en alguna de las ventanas, el editor de programas debe reconfigurar correctamente el modelo computacional y redibujar la visualización en pantalla. Uno de los aspectos más complejos respecto de la implementación del diseño planteado es lograr un comportamiento correcto del editor de programas, para la edición de las configuraciones de hardware y los lenguajes gráficos de programación de Unidades de Organización de Programa, ante las distintas acciones que puede llevar a cabo un usuario. Los desafíos presentados, y las características generales de la solución encontrada, se describen en la Sección 4.5. En el diseño propuesto, se ofrece al usuario una interfaz ágil para la programación de PLC.

A continuación se describe cada sección de la interfaz de usuario del editor de programas.

3.3.1. Edición de un Proyecto

Para crear un "Nuevo Proyecto" se utiliza la ventana de edición de "Proyecto de automatización". Esta ventana se divide en dos secciones, "Proyecto" y "Confguraciones de Hardwarwe". Una representación de la misma con un nuevo proyecto vacío se muestra en la figura [3.7].

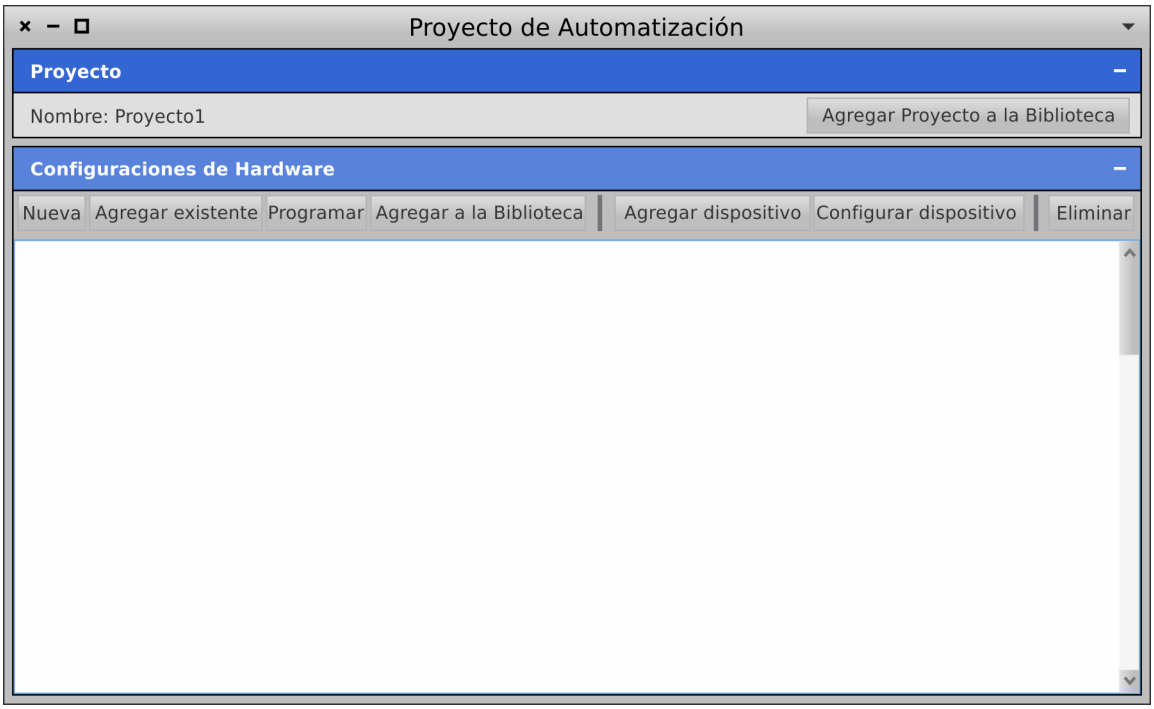


Figura 3.7: Ventana de edición de "Proyecto de Automatización".

La primera sección permite al usuario cambiar el nombre del proyecto y guardarlo en la biblioteca. La segunda sección es donde el usuario desarrolla el Proyecto. Éste se basa en la confección de Configuraciones de Hardware de Controladores Programables y su programación. Una configuración de hardware se compone de uno o más dispositivos interconectados permitiendo describir el hardware de un Controlador Programable. Por ejemplo, puede agregarse un Controlador del tipo compacto el cual está conformado por un único dispositivo (figura [3.8]), o bien, un Cotrolador del tipo modular con un módulo de comunicación, una CPU y un módulo de entradas y salidas (figura [3.9]). Cada dispositivo se coloca en una "Ranura" numerada. Estos números de ranura se utilizan para el direccionamiento de las entradas y salidas físicas del Controlador Programable en la Configuración de Software.

Una vez creada la configuración de hardware esta ventana permite su programación. También permite agregar una configuración de hardware existente y su edición.

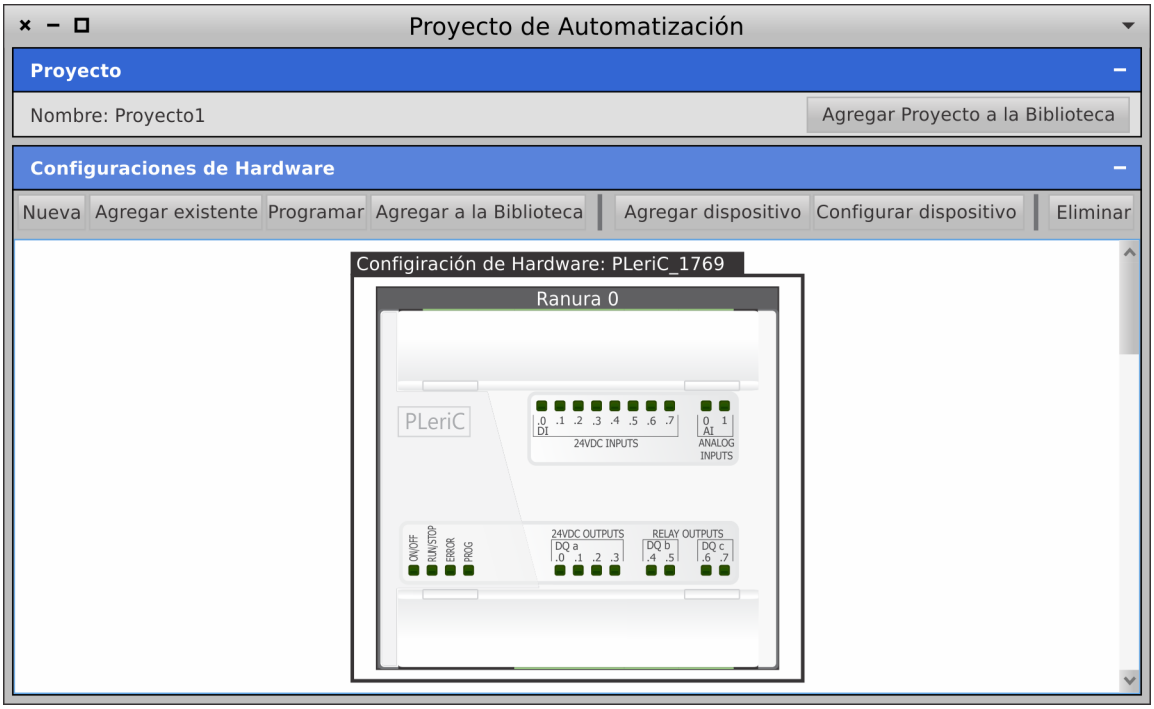


Figura 3.8: Ventana de edición de "Proyecto de Automatización". En la misma se muestra la edición de una configuración de hardware que posee un único dispositivo Controlador Programable del tipo compacto.

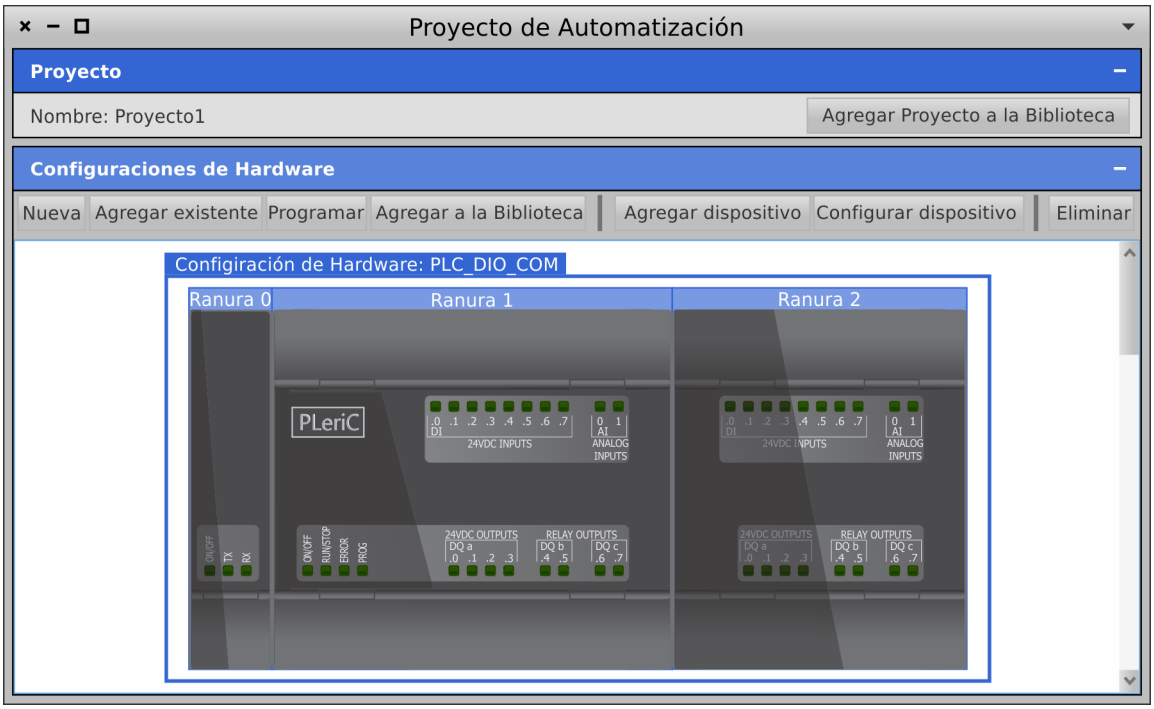


Figura 3.9: Ventana de edición de "Proyecto de Automatización". En la misma se muestra la edición de una configuración de hardware que posee tres dispositivos conectados entre si conformando un Controlador Programable del tipo modular.

Se puede acceder a programar una configuración de hardware mediante el botón “Programar”. Este botón abre la ventana de edición de “Configuración de software de Controlador Programable”. En la figura [3.10] se muestra la ventana de configuración de software correspondiente a la configuración de hardware de la figura [3.9].

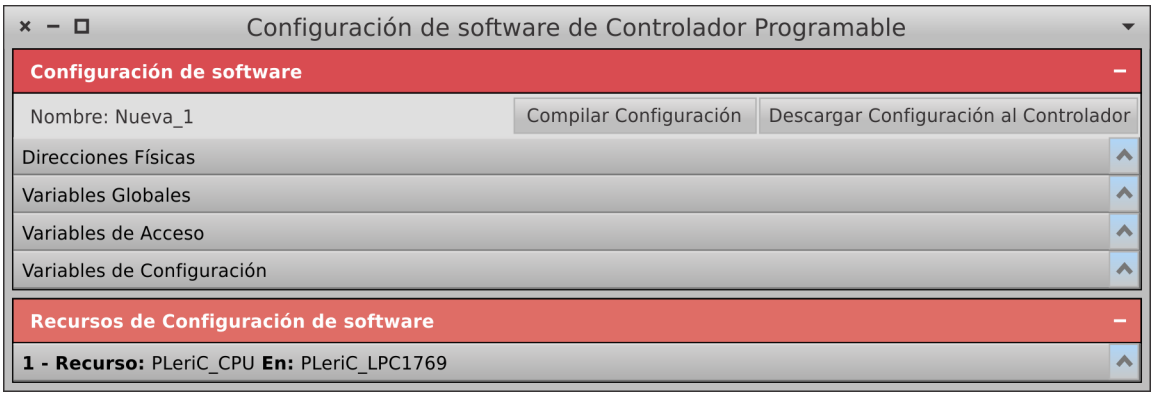


Figura 3.10: Ventana de edición de “Configuración de software de Controlador Programable”.

Aquí el usuario edita la configuración de software del Controlador Programable definida en la norma IEC 61131-3. Esta configuración contiene las direcciones físicas de los dispositivos, Recursos y las declaraciones de variables: globales, de acceso y de configuración. Cada Recurso incluye variables globales de recurso, configuraciones de Tareas y de Programas. Una configuración de programa puede estar asociada a una configuración de tarea.

Se ofrece un ejemplo de Recurso en la figura [3.11].

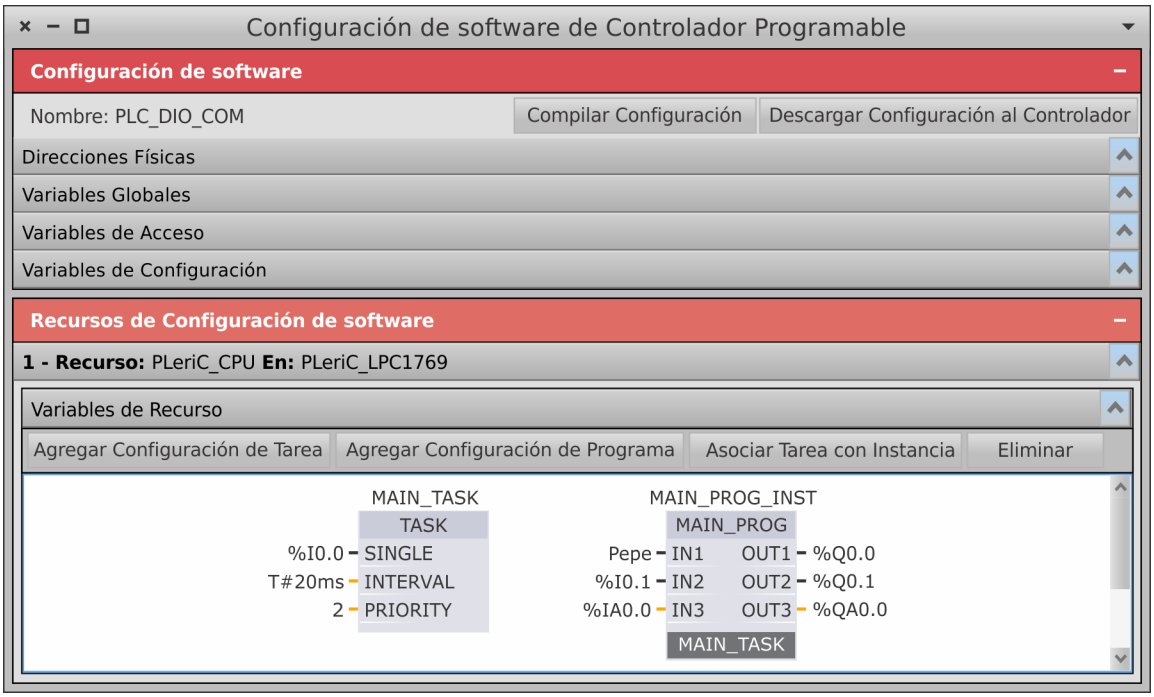


Figura 3.11: Ventana de edición de “Configuración de software de Controlador Programable” con una configuración de Tarea y una configuración de Programa asociada a la Tarea.

En este ejemplo existe una configuración de tarea llamada “MAIN_TASK”, una configura-

ción de programa utilizando la instancia “MAIN_PROG_INST” de la POU de tipo Programa “MAIN_PROG”, y la asociación entre estas dos configuraciones en la parte inferior de la configuración de programa. Una configuración de tarea como “MAIN_TASK” se agrega presionando sobre el botón “Agregar Configuración de Tarea” que abre la caja de diálogo “Configuración de Tarea” (figura [3.12]).

Figura 3.12: Caja de diálogo “Configuración de Tarea”.

La configuración de programa utilizando una instancia de una POU de tipo “Programa”, que ha sido definida por el usuario previamente utilizando la ventana de edición de “Unidad de Organización de Programa (POU)” (detallada en la Sección 3.3.2), se agrega mediante el botón “Agregar Configuración de Programa”. Este botón abre la biblioteca de POU del tipo Programa (ver Sección 3.3.3) que permite elegir alguna de estas POU para agregarla al Recurso. Cuando se elige una, se abre automáticamente la caja de diálogo ‘Declaración de Instancia de Programa o FB” (figura [3.13]) permitiendo declarar una variable de instancia de esa POU donde se almacenan todas sus variables internas persistentes.

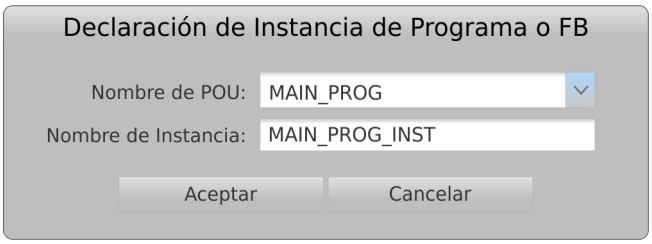


Figura 3.13: Caja de diálogo “Declaración de Instancia de Programa o Bloque de Función”.

Luego, mediante el botón “Asociar Tarea con Instancia” se especifica cuál es la tarea que invoca la instancia de Programa cuando dicha tarea es planificada para su ejecución por el Sistema Operativo del Controlador Programable. Si no se especifica ninguna asociación (el parámetro inferior de la instancia “MAIN_PROG_INST” de la figura [3.11] se encuentra vacío), la instancia de programa se ejecuta cíclicamente con la menor prioridad del sistema.

Una vez definida la configuración de software, la ventana permite su compilación y su posterior descarga a la memoria del Hardware del Controlador Programable.

3.3.2. Creación y edición de una “Unidad de Organización de Programa (POU)”

Cuando se accede al ícono “Nueva POU” del escritorio se abre la caja de diálogo “Declaración de POU” (ilustrada en la figura [3.14]) la cual permite elegir el tipo de POU a crear (Programa, Bloque de Función o Función), su lenguaje de programación y su nombre. Una vez escogidos estos parámetros, se abre la ventana de edición de “Unidad de Organización de Programa (POU)” con una nueva POU vacía del tipo elegida.

Mediante su utilización el usuario puede editar las POU definidas en la norma IEC 61131-3. Esta ventana se divide en tres secciones como se observa en la figura [3.15].

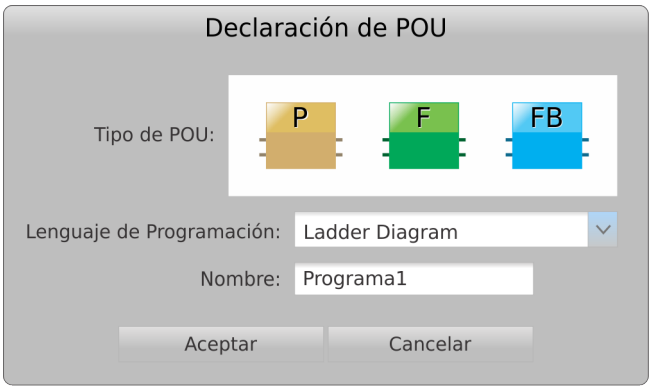


Figura 3.14: Caja de diálogo de "Declaración de POU".

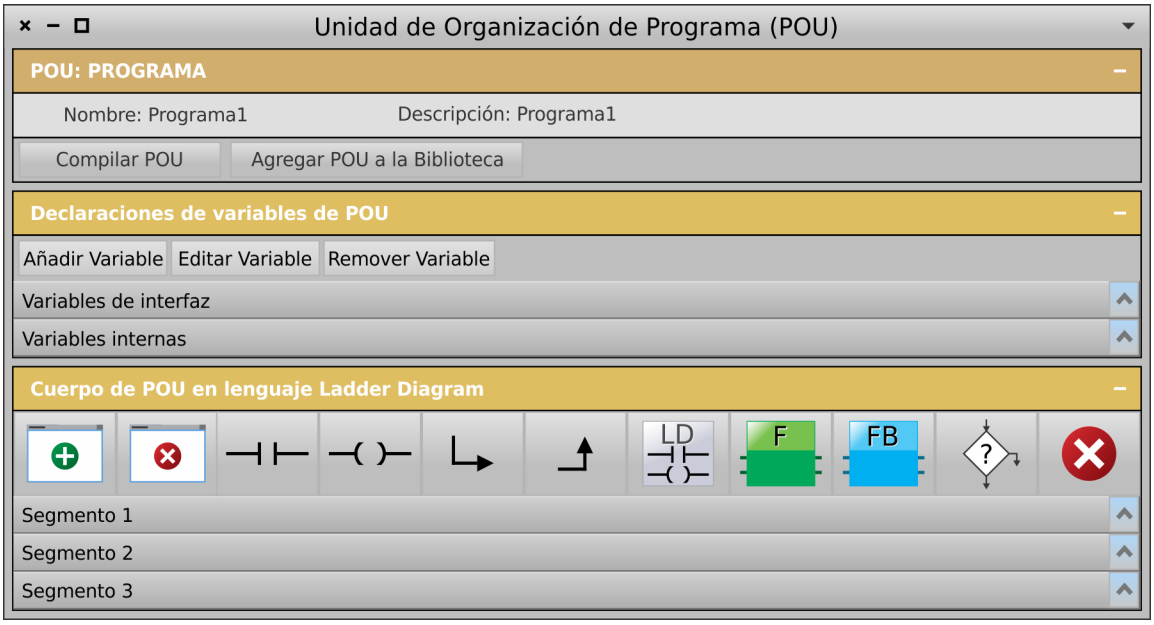
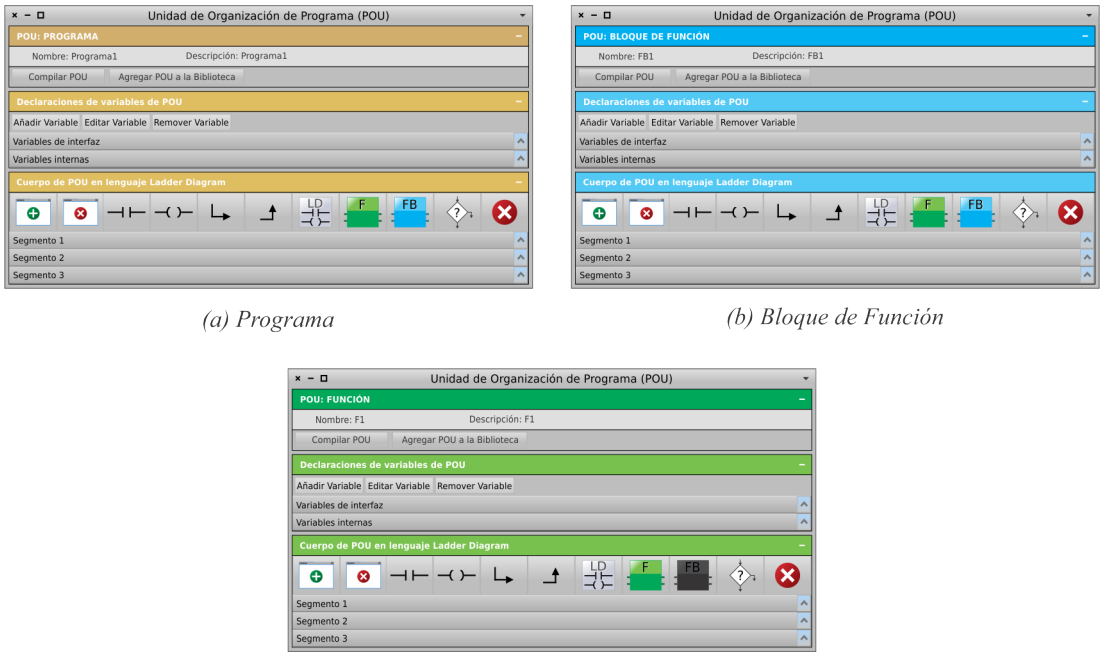


Figura 3.15: Ventana de edición de "Unidad de Organización de Programa (POU)".

La barra de título de cada sección cambia de color según el tipo de POU que se esté editando. Si se edita un Programa, éstas se muestran en tonos de dorado, en el caso de editar un Bloque de Función se muestran en tonos de cian y para una Función en tonos de verde. De esta forma se identifica fácilmente cuál es el tipo de POU que se está editando, como se ilustra en la figura [3.16].



(a) Programa *(b) Bloque de Función*

(c) Función

Figura 3.16: Ventanas de edición de "Unidad de Organización de Programa (POU)" para los tipos de POU: (a) Programa, (b) Bloque de Función y (c) Función.

En la primer sección el usuario puede establecer el nombre de la POU, una descripción de la misma, compilarla y guardarla en la Biblioteca. La segunda sección está destinada a la edición de las variables de interfaz e internas de la POU. Finalmente, en la tercer sección, se edita el cuerpo del programa en alguno de los lenguajes de programación definidos por la norma IEC 61131-3.

A continuación se describe en detalle la sección "Declaraciones de variables de POU" y la sección de "Cuerpo de POU".

3.3.2.1. "Declaraciones de variables de POU"

Esta sección de la ventana (ilustrada en la figura [3.17]) contiene una barra de herramientas y dos secciones expansibles las cuales corresponden a las grillas de "Variables de interfaz" y "Variables internas" respectivamente.

Las variables de interfaz corresponden a las categorías de variables de "Entrada", "Salida" y "Entrada-Salida", que son accesibles fuera de la POU (parámetros formales). En los lenguajes gráficos, estas variables se muestran como pines de conexión.

Las variables internas corresponden a las categorías "Temporal", "Interna" y "Externa" que se utilizan únicamente en el interior de la POU. Una declaración de variable "Externa" hace referencia a una declaración de variable "Global" dentro de un Recurso o Configuración de Software. Las POU del tipo Función sólo incluyen la categoría "Interna". Las del tipo Bloque de Función incluyen "Temporal" e "Interna". Las del tipo Programa incluyen todas las categorías de variables descriptas. Para las POU de los tipos Programa y Bloque de Función el valor de las variables de categoría "Interna" persiste entre ejecuciones de estas POU.

La barra de herramientas permite añadir una variable (ver figura [3.18]) o eliminar una variable seleccionada en la grilla utilizando los botones destinados a tal fin.

Figura 3.17: Sección "Declaraciones de variables de POU" de la ventana de edición de Unidad de Organización de Programa.

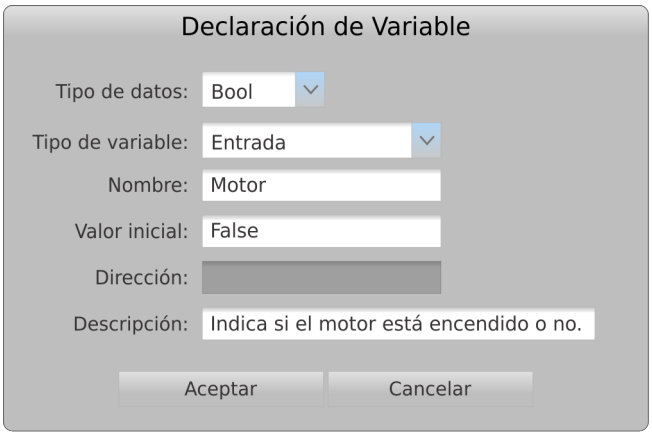


Figura 3.18: Diálogo "Declaración de variable". Este permite crear una nueva declaración de variable.

La grilla muestra una declaración de variable por fila. Cada columna de la misma representa una característica de la variable declarada como se observa en la cabecera de la grilla (figura [3.17]).

Las declaraciones de variables se encuentran ordenadas por categoría. Además, permite la edición directa de cualquier celda de declaración mediante la acción de doble clic sobre la misma. De esta forma, puede cambiarse fácilmente alguna característica de la declaración de variable, como por ejemplo, el nombre de una variable mostrado en la figura [3.19].

Figura 3.19: Edición del nombre de una variable.

3.3.2.2. "Cuerpo de POU"

La sección "Cuerpo de POU" cuenta con una barra de herramientas en la parte superior seguida de secciones expansibles correspondientes a los segmentos de programa. Estos segmentos se utilizan para modularizar el programa de usuario y cada uno puede utilizarse como destino de salto de ejecución en las instrucciones de control de ejecución de programa. Estas instrucciones se utilizan agregando una "etiqueta de segmento" única por cada segmento la cual lo identifica.

Se plantean cuatro versiones de esta sección, una por cada lenguaje de programación definidos en la norma IEC 61131-3. Dos de estas versiones corresponden a lenguajes de programación gráficos, mientras que las restantes, a lenguajes de programación del tipo textual. En los siguientes apartados se exponen las características de la interfaz de usuario para para cada uno de ellos.

Programación con lenguajes gráficos

Los lenguajes gráficos definidos en la norma IEC 61131-3 son Ladder Diagram (LD) y Function Block Diagram (FBD). La interfaz definida para programación con los mismos se basa en la manipulación directa de los elementos de programa, tales como contactos, bobinas y llamados a funciones, entre otros. Cada uno de estos, posee un dibujo en pantalla que lo representa.

El usuario construye cada segmento que conforma una POU ubicando sucesivamente cada componente (contacto, llamado a función, bobina, etc.) en el lugar que le corresponde según el flujo deseado creando un esquema de circuito (electromecánico en LD, o eléctrico-lógico en FBD) en el segmento.

Debe destacarse que en la norma se define el término "Networks" que corresponde al "máximo conjunto de elementos gráficos interconectados, excluyendo las barras de alimentación en el caso del lenguaje Ladder Diagram". En consecuencia el término no siempre es aplicable a los segmentos definidos en este trabajo final. Es decir una "Network" siempre puede ser un segmento, pero el recíproco no siempre lo es. Es responsabilidad del usuario el separar los circuitos en dichos segmentos. Esta aclaración se expone debido a que en la norma cada "Network" puede etiquetarse y ser destino de salto de ejecución de programa en los lenguajes gráficos.

Ladder Diagram (LD)

Una visión general de la sección de programación en lenguaje Ladder Diagram (LD) se ofrece en la figura [3.20].

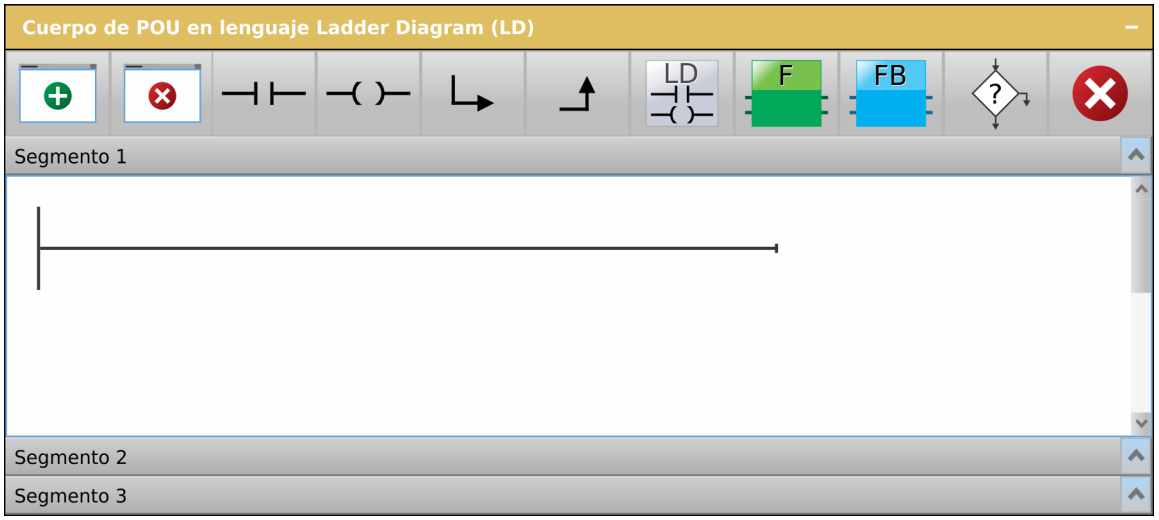


Figura 3.20: Sección "Cuerpo de POU en lenguaje Ladder Diagram (LD)".

En la figura [3.20] se puede apreciar cómo se muestra un segmento LD al que aún no se le ha agregado ningún elemento. Sin embargo, posee por defecto tres elementos gráficos de programa los cuales son "Barra de alimentación derecha" y "Barra de alimentación izquierda", que se encuentran unidos por una "Conexión horizontal". También se puede apreciar en esta figura la barra de herramientas superior. Esta contiene los siguientes botones de izquierda a derecha:

- Agregar Segmento.
- Eliminar Segmento.
- Agregar Contacto normal abierto.
- Agregar Bobina.
- Abrir rama paralela.
- Cerrar rama paralela.
- Agregar elemento Ladder.
- Agregar llamado a Función.
- Agregar llamado a instancia de Bloque de Función.
- Agregar elemento de Control de ejecución de programa.
- Eliminar elemento.

El usuario incorpora un elemento a un segmento seleccionando el punto dentro del circuito definido para el segmento en el que debe insertarse el nuevo elemento, y luego eligiendo el ícono correspondiente al tipo de elemento a agregar (contacto, bobina, llamada a función, etc.) en la barra de herramientas.

En este momento, el editor de programas inserta el elemento seleccionado, haciendo las reconfiguraciones necesarias en las conexiones entre los elementos que forman parte del segmento, y reacomodando la ubicación de cada elemento en pantalla de acuerdo al nuevo grafo de conectividad. Esto significa que en un segmento en lenguaje LD es el entorno quien sen encarga de distribuir en pantalla los elementos que componen el circuito.

En la figura [3.21] se muestran las distintas etapas en el agregado de un Contacto dentro de un segmento.

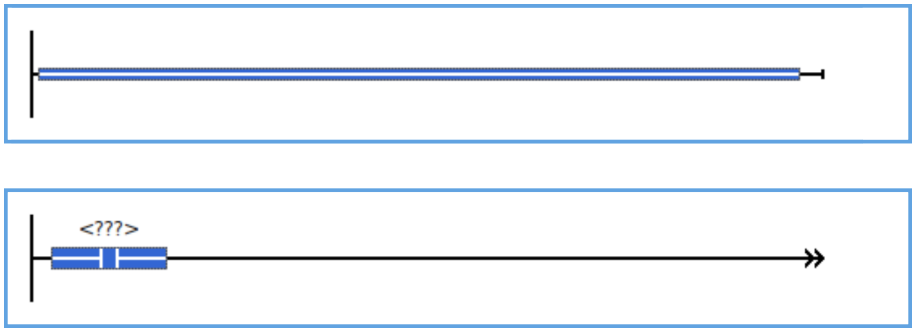


Figura 3.21: Ladder Diagram. Agregado de un contacto sobre una conexión.

Estas etapas son:

1. Seleccionar el punto de inserción. Este debe ser una conexión entre dos elementos o un pin (figura [3.21] parte superior).

2. Elegir elemento a insertar en la barra de herramientas (en este caso un Contacto normal abierto).

3. Se redibuja automáticamente el segmento mostrando el nuevo elemento insertado en el circuito (figura [3.21] parte inferior).

Un efecto a destacar es el cambio del dibujo del elemento "Barra de alimentación derecha" indicando que el circuito no puede terminar en un Contacto y, en cambio, debe terminar en llamado a Función, llamado a Bloque de Función, Bobina o elemento de Bifuración de programa. Es por ello que en la figura [3.21] se muestra el elemento "Barra de alimentación derecha" con una doble punta de flecha. Esto indica que se producirá un error al compilar ésta POU. De modo semejante, la figura [3.22] expone las distintas etapas en el agregado de un llamado a Función suma ("ADD") dentro de un segmento.

Los argumentos de los elementos pueden configurarse mediante la acción de "doble clic" sobre el mismo. Cada argumento admite como valor un literal o variable de cierto tipo. Las variables además, deben encontrarse en el "alcance" de la POU, esto es, declaradas en la sección de "Declaraciones de variables de POU" descripta anteriormente. Estos chequeos deben ser realizados por este Editor. En caso que el argumento ingresado sea inválido, se cambia el color del valor ingresado para destacar el error y se produce un error al compilar la POU. En la figura [3.23] se expone la secuencia de edición del único argumento que posee un Contacto normal abierto.

El editor de programas también debe soportar la *eliminación* de un elemento en un segmento. Para realizarlo debe seleccionarse el elemento a elminar, y luego, presionar el botón "Eliminar elemento". Esta secuencia se muestra en la figura [3.24]. En la parte superior de la misma se muestra seleccionado el componente Contacto y en la parte inferior el circuito resultante luego de su eliminación. Nótese cómo debe reacomodar el Contacto que se encontraba conectado en paralelo con el que fue eliminado.

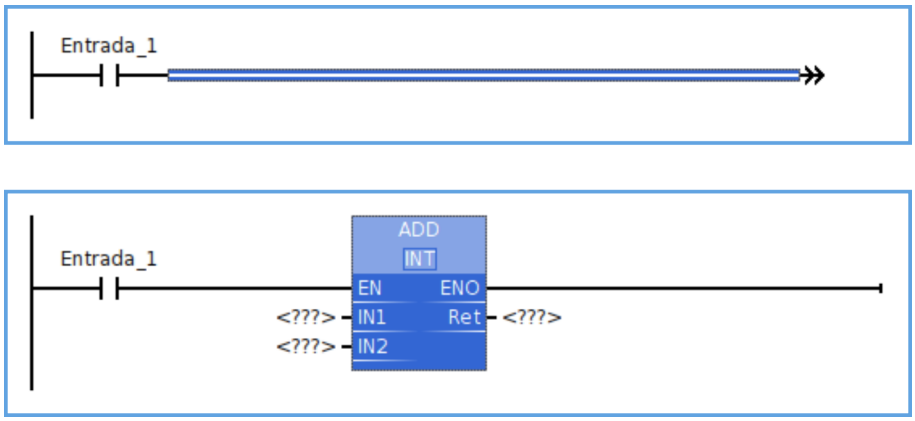


Figura 3.22: Ladder Diagram. Agregado de un llamado a Función Suma ("ADD") sobre una conexión.

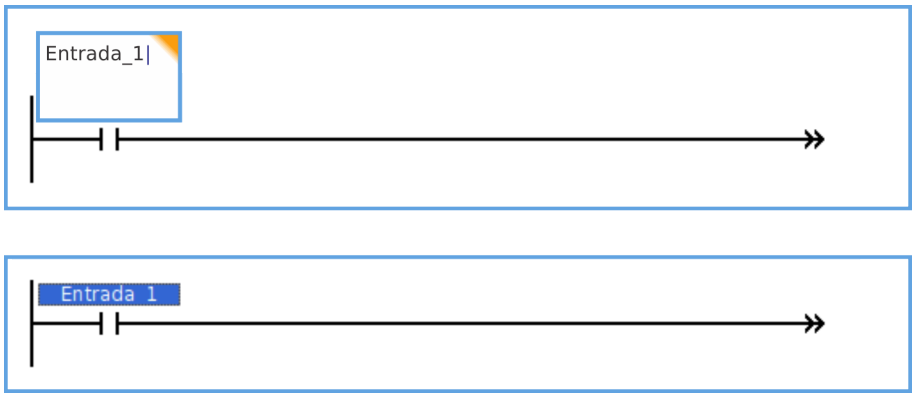


Figura 3.23: Ladder Diagram. Edición del argumento de un contacto.

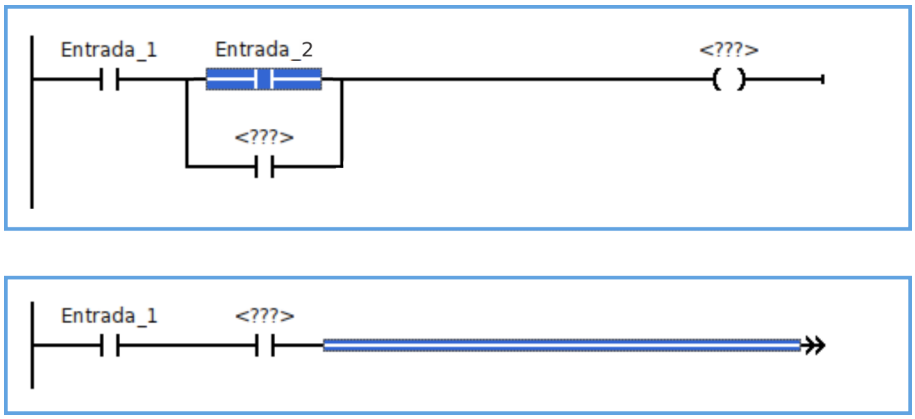


Figura 3.24: Ladder Diagram. Borrado de Contacto.

En un circuito Ladder se permiten conexiones en serie y paralelo de los elementos. Estas conexiones se realizan mediante los botones de "Apertura de rama paralela" y "Cierre de rama paralela" contenidos en la barra de herramientas. En ambos casos se inserta un componente "Enlace Vertical". La figura [3.25] muestra las dos acciones que producen como resultado la apertura de una rama paralela. En la parte superior de esta figura puede observarse la conexión seleccionada; mientras que en la parte inferior de la misma se da el resultado, luego de presionar el botón "Apertura de rama paralela".

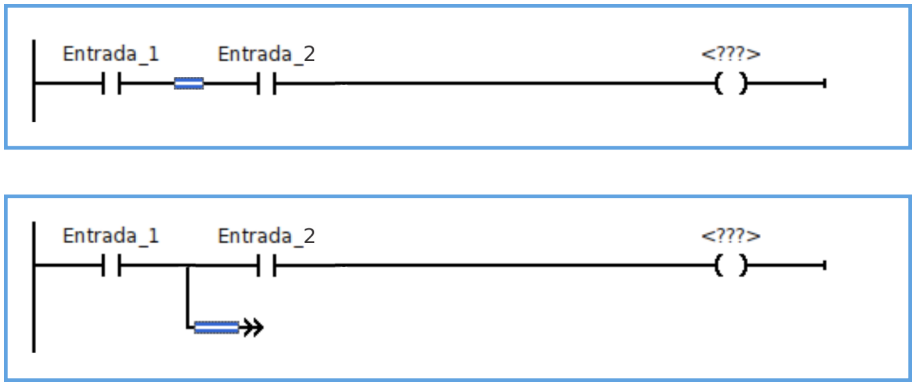


Figura 3.25: Ladder Diagram. Abrir rama paralela sobre una conexión.

Una rama paralela al igual que la principal debe terminar en un llamado a Función, llamado a Bloque de Función, Bobina o elemento de Control de ejecución de programa. Para añadir un elemento en la misma, se procede de igual forma que en el caso de la rama principal como se expone en la figura [3.26].

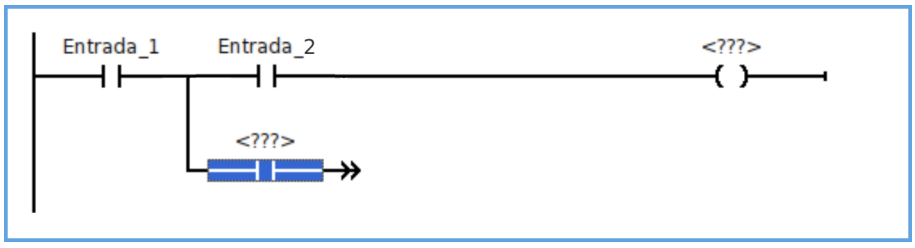


Figura 3.26: Ladder Diagram. Agregado de un contacto en una rama paralela.

Para poder cerrar una rama paralela, creando en consecuencia una conexión en paralelo, es

requisito que exista un elemento en la misma para no producir un “cortocircuito” como el de la figura [3.27].

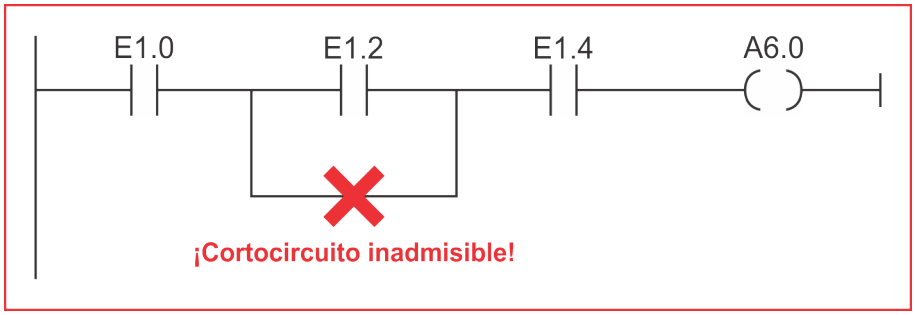


Figura 3.27: Ladder Diagram. Circuito inválido por cortocircuito.

En consecuencia, el entorno debe chequear que exista al menos un elemento en la rama paralela cuando el usuario intenta realizar esta acción. Los pasos para cerrar una rama paralela se ofrecen en la figura [3.28], estos consisten en:

1. Seleccionar el componente “Barra de alimentación derecha” que se quiere utilizar como origen para cerrar la rama(figura [3.28] parte superior).

2. Presionar la tecla shift, que permite selección múltiple de elementos (otro factor que debe ser soportado por el entorno), y mientras se encuentra esta tecla presionada seleccionar una conexión destino hacia dónde se cerrará la rama (figura [3.28] parte central).

3. Teniendo estos dos elementos seleccionados, presionar el boton “Cierre de rama paralela” de la barra de herramientas.

4. En caso de permtir el cerrado de la rama, el entorno redibuja automáticamente el segmento, mostrando el resultado de la conexión en paralelo entre ambos Contactos (figura [3.28] parte inferior).

Una variante deseable, consiste en cerrar parcialmente dos ramas seleccionando las conexiones que deben unirse mediante un “Enlace Vertical”, creando un cierre de rama paralela utilizando el mismo botón. Se muestra un ejemplo en la figura [3.29].

Para la eliminación de ramas paralelas se decidió que el entorno debe borrarlas automáticamente, al suprimir todos los elementos situados en las mismas.

Un factor de diseño de la interfaz de Ladder deviene de la especificación de la norma IEC 61131-3, la cual indica que “en un circuito Ladder la corriente debe circular de izquierda a derecha” tomando como polos positivo y negativo a los componentes “Barra de alimentación izquierda” y “Barra de alimentación derecha” respectivamente. Un ejemplo de circuito inválido se da en la figura [3.30]. Aquí un estado de señal “0” en E 1.4 causaría un flujo de corriente de derecha a izquierda en E 6.8, lo cual no es admisible.

Para asegurar este comportamiento, se tomó como decisión de diseño que el entorno sólo debe permitir conexiones en serie o paralelo de elementos. Este chequeo debe ser realizado junto con los anteriores al presionar el botón “Cierre de rama paralela”.

Otro efecto gráfico deseable, es que el entorno no cree en pantalla elementos redundantes; por ejemplo, en la figura [3.31] se indica el comportamiento esperado en el caso de que el usuario requiera cerrar una rama paralela antes de una apertura de otra rama creada previamente. Aquí en vez de crear otro elemento “Enlace Vertical” solamente debe conectarse al mismo. Esto puede considerarse como una absorción de un “Enlace Vertical” con respecto al otro formando un solo “Enlace Vertical”.

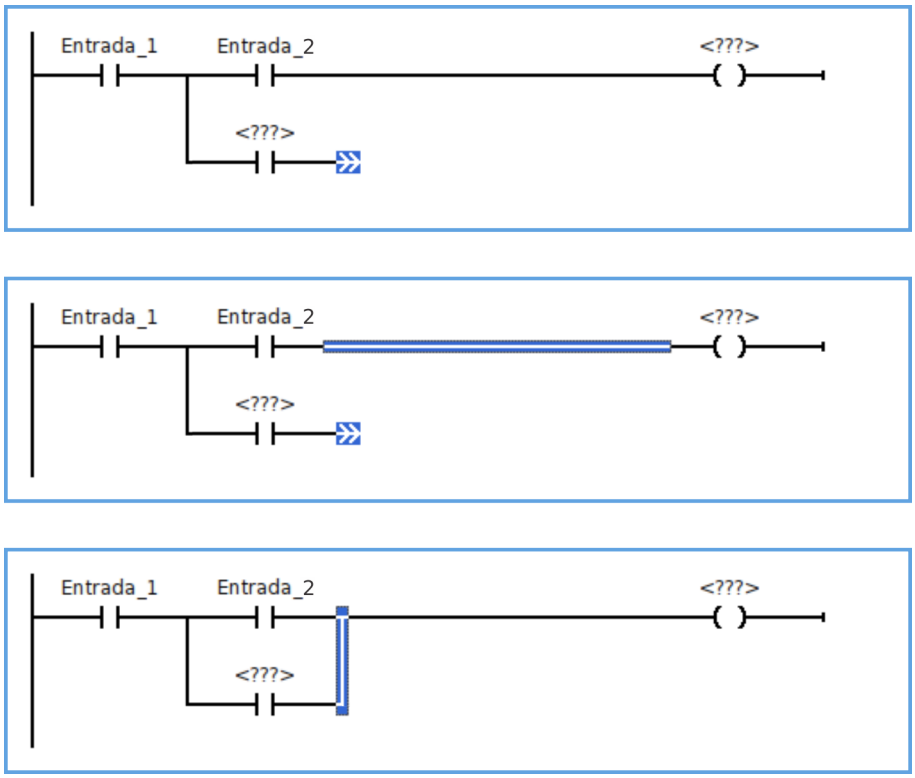


Figura 3.28: Ladder Diagram. Cerrar rama paralela para formar una conexión en paralelo.

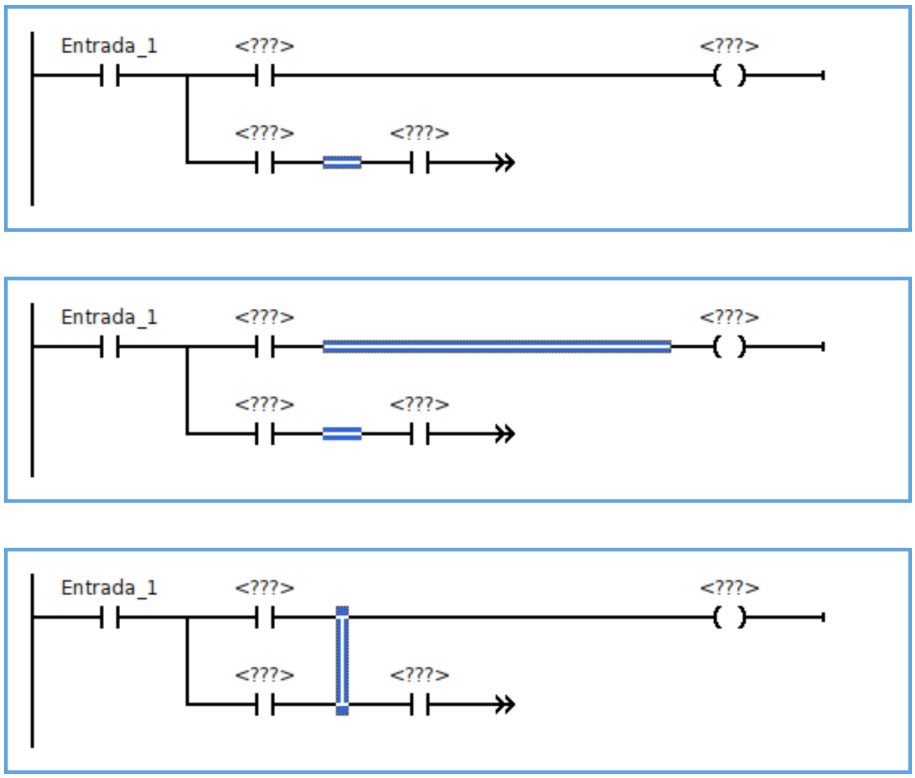


Figura 3.29: Ladder Diagram. Cerrar rama paralela entre dos conexiones.

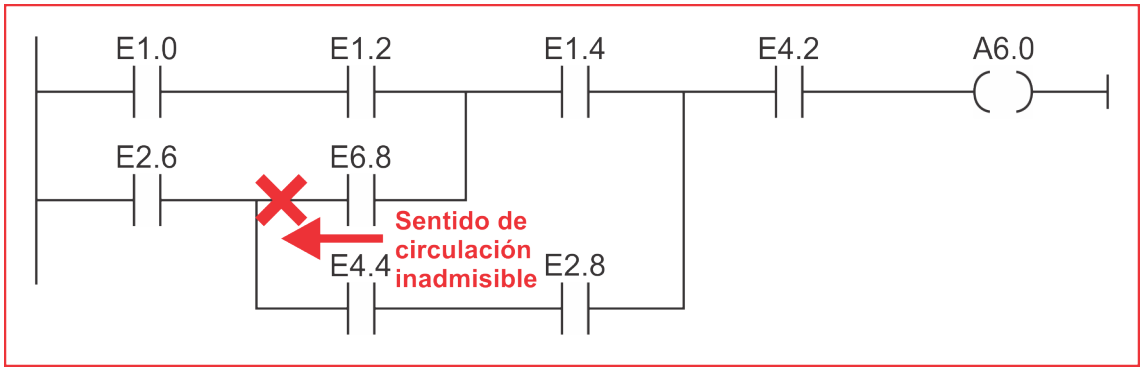


Figura 3.30: Ladder Diagram. Circuito inválido por flujo de corriente de derecha a izquierda.

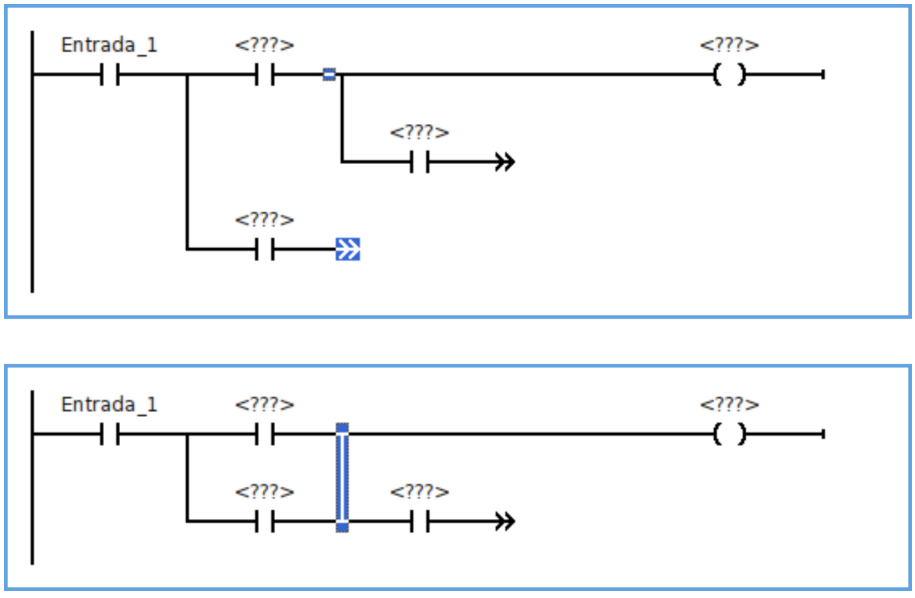


Figura 3.31: Ladder Diagram. Cerrar rama antes de una rama abierta.

Finalmente, en la figura [3.32] se ilustra un ejemplo más complejo de circuito Ladder para indicar el comportamiento esperado de la interfaz. En el circuito se muestran varias aperturas de rama en cascada y además, cómo deben "empujar" los Contactos al Enlace Vertical para no pisarse en pantalla.

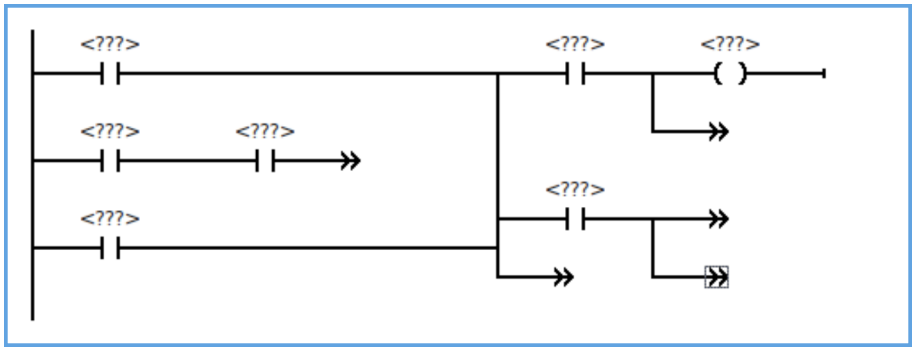


Figura 3.32: Ladder Diagram. Circuito complejo.

Function Block Diagram (FBD)

Mientras que en un segmento Ladder es el entorno quien se encarga de distribuir los componentes en pantalla calculando sus posiciones, en un segmento FBD es el usuario quien debe posicionarlos en pantalla insertando cada elemento en la posición deseada. El entorno solamente se encarga de posicionar las conexiones entre los mismos. Debido a la similitud con Ladder, en este apartado sólo se destacan las particularidades de este lenguaje.

En la figura [3.33] se ofrece la visión completa de la sección de programación en lenguaje Function Block Diagram (FBD).

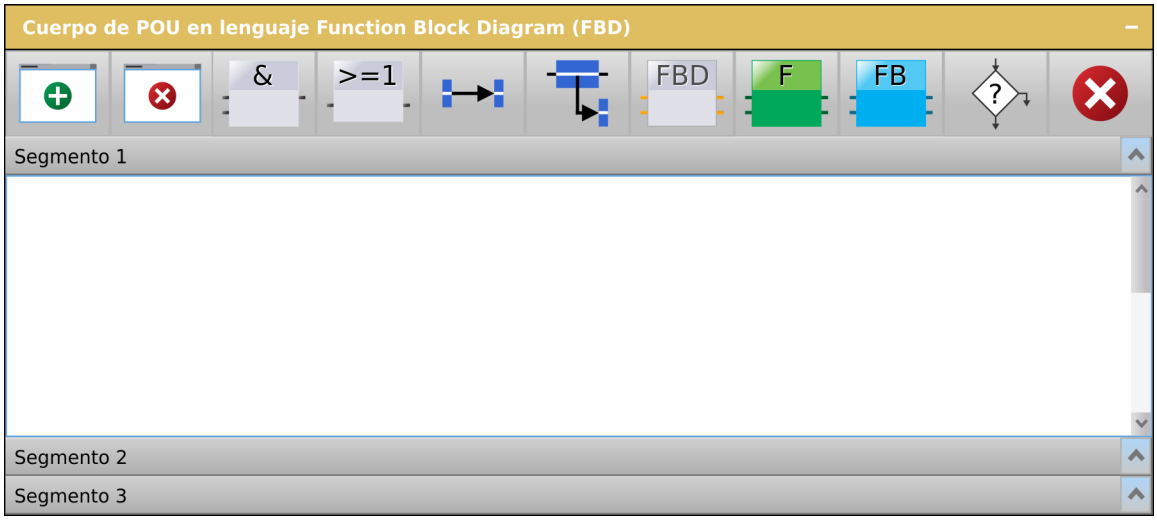


Figura 3.33: Sección "Cuerpo de POU en lenguaje Function Block Diagram (FBD)".

En este caso, los botones provistos por la barra de herramientas de izquierda a derecha son:

- Agregar Segmento.
- Eliminar Segmento.
- Agregar Bloque lógico "AND".

- Agregar Bloque lógico “OR”.
- Conectar pines.
- Conectar Pin de Entrada con conexión previa.
- Agregar elemento FBD.
- Agregar llamado a Función.
- Agregar llamado a instancia de Bloque de Función.
- Agregar elemento de Control de ejecución de programa.
- Eliminar componente.

Para la inserción de componentes en el segmento se distinguen los siguientes pasos:

1. Elegir el componente a insertar presionando el botón adecuado en la barra de herramientas.
2. Una vez presionado el botón, el componente aparece “pegado” al cursor.
3. Situar el cursor sobre la posición del segmento dónde se quiere depositar el componente y se hacer clic.
4. Al hacer clic, el componente se “despega” del cursor y queda fijado en esa posición del segmento.

Estos pasos se ilustran en la figura [3.34].

Si el usuario desea cambiar la posición de un componente previamente posicionado, el mismo debe ser seleccionado haciendo clic con el mouse, arrastrarlo a la nueva posición deseada y soltar el botón del mouse. Esta acción se conoce comúnmente por su nombre en inglés “drag & drop”.

La conexión entre dos elementos se realiza mediante el botón “Conectar pines” de la barra de herramientas. Los pasos para realizarlo son:

1. Selecciónar un pin de salida de un componente.
2. Mediante la utilización de la tecla shift seleccionar el segundo elemento creando una selección múltiple.
3. Finalmente, presionar el botón “Conectar pines” de la barra de herramientas.

Estos pasos se ilustran en la figura [3.35].

En este lenguaje, la norma no permite crear conexiones cableadas en paralelo[4] , en reemplazo existe el Bloque lógico “OR”. También se ofrece un bloque “XOR”.

Una decisión de diseño para este lenguaje, es que no se permitan los lazos de realimentación cableados. Esta elección, facilita el desarrollo de la interfaz gráfica y no agrega limitaciones ya que siempre se puede realimentar el circuito FBD mediante la utilización de variables.

Programación con lenguajes textuales

En la interfaz planteada para programación en lenguajes textuales como IL o ST el usuario simplemente ingresa el texto correspondiente al programa. Todos los chequeos de errores se realizan durante la compilación de la POU.

[4]Nombrada como OR cableada en la norma IEC 61131-3

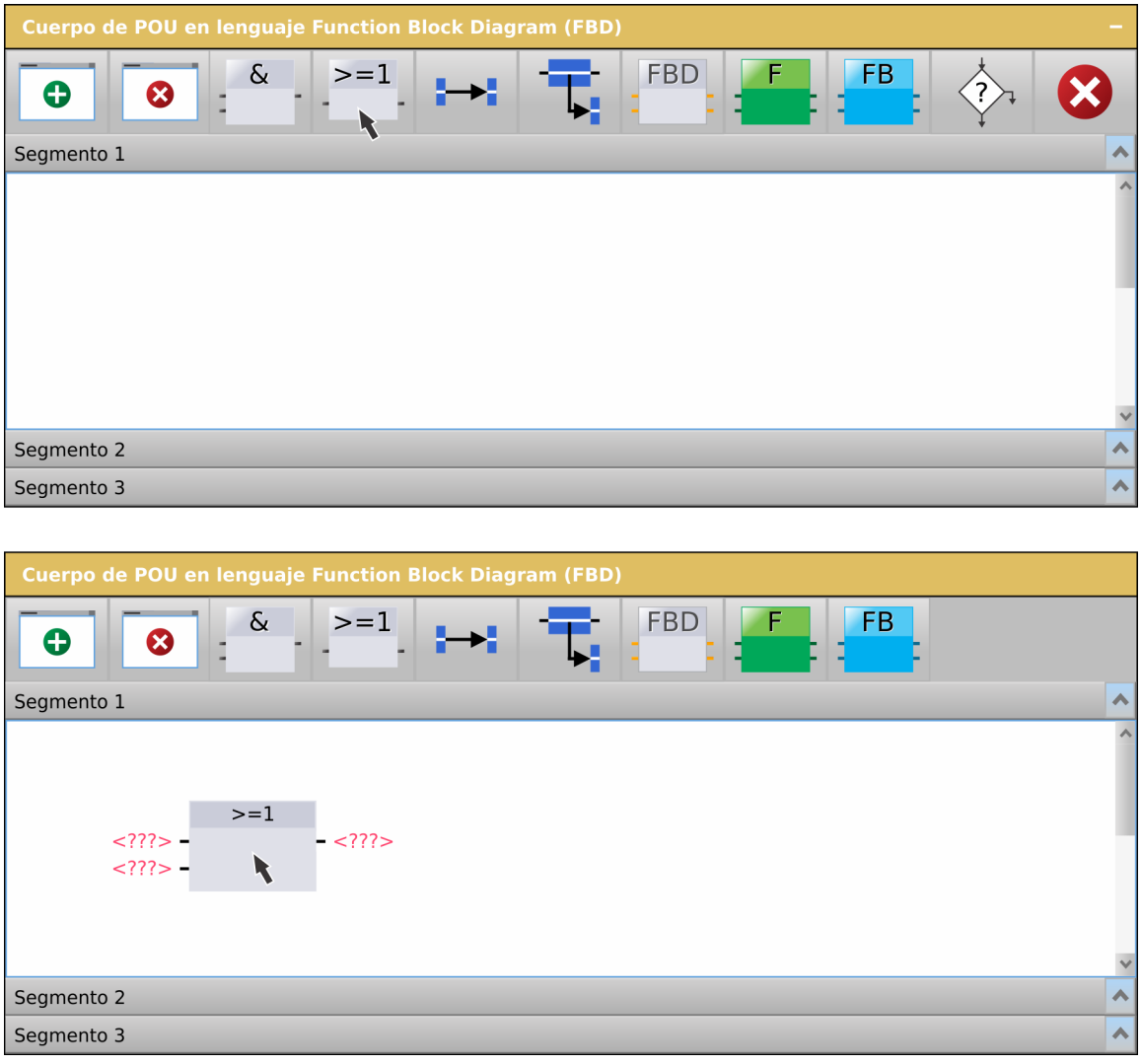


Figura 3.34: Ventana de edición de "Unidad de Organización de Programa (POU)" en lenguaje Function Block Diagram (FBD).

Si bien en la norma IEC 61131-3 no existe una definición de “Network” para lenguajes textuales, se tomó la decisión de permitir la división del programa textual en segmentos. Como restricción no presente en la norma, solo se permite agregar etiquetas de destino de salto a los mismos como en los lenguajes gráficos. De esta manera se simplifica la compilación de estos lenguajes y se homogeneíza la interfaz de usuario. Como desventaja, se restringe al usuario a dividir el programa en segmentos que en algunos casos podrían ser de una o muy pocas líneas.

La barra de herramientas es común para ambos lenguajes, con el agregado de un único botón para el lenguaje Instruction List. Esta barra de herramientas presenta características habituales halladas en una barra de herramientas de un editor de texto y además características específicas de estos lenguajes idénticas a las de los lenguajes gráficos. De esta manera, se expone al usuario una interfaz gráfica similar para los cuatro lenguajes de programación facilitando la comprensión de los mismos. Las funcionalidades de los botones contenidos en la barra de herramientas son, de izquierda a derecha:

- Agregar Segmento.
- Eliminar Segmento.
- Agregar Instrucción IL (no presente en ST).
- Copiar texto seleccionado.
- Cortar texto seleccionado.
- Pegar texto seleccionado.
- Comentar/descomentar texto seleccionado.
- Agregar llamado a Función.
- Agregar llamado a instancia de Bloque de Función.
- Agregar elemento de Control de ejecución de programa.

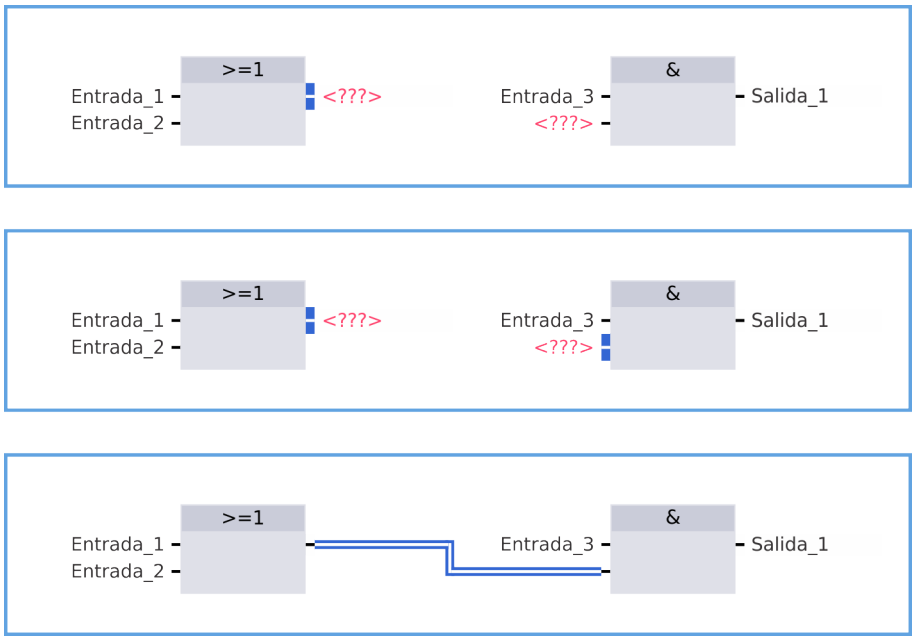


Figura 3.35: Ventana de edición de “Unidad de Organización de Programa (POU)” en lenguaje Function Block Diagram (FBD).

- Eliminar texto seleccionado.

Instruction List (IL)

En adhesión a la norma IEC 61131-3, el usuario debe ingresar una única instrucción por línea de texto. Con excepción de un llamado a Función, o instrucción de llamado a Bloque de función, que permiten separar los argumentos en varias líneas. En la figura [3.36] se expone un ejemplo de programa en lenguaje IL.

Como se ha anticipado, se ofrece en la barra de herramientas el botón "Agregar Instrucción IL" que permite al usuario elegir alguna de las instrucciones IL para incorporarla al programa.

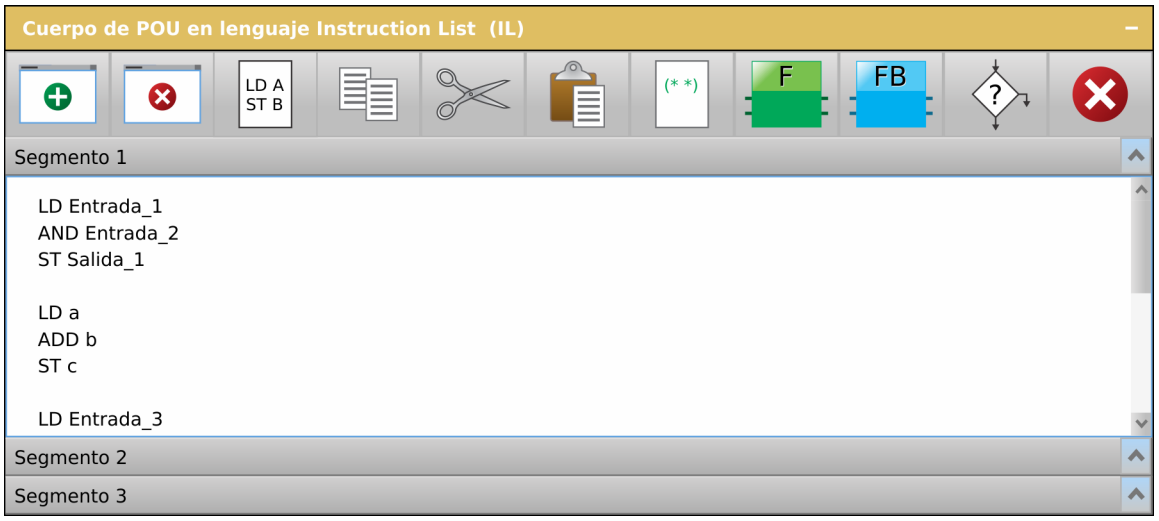


Figura 3.36: Sección "Cuerpo de POU en lenguaje Instruction List (IL)".

Structured Text (ST)

En este caso, al tratarse de un lenguaje similar a C o Pascal, no existen restricciones de líneas de texto y espaciados. Un ejemplo de programa en lenguaje ST se muestra en la figura [3.37].

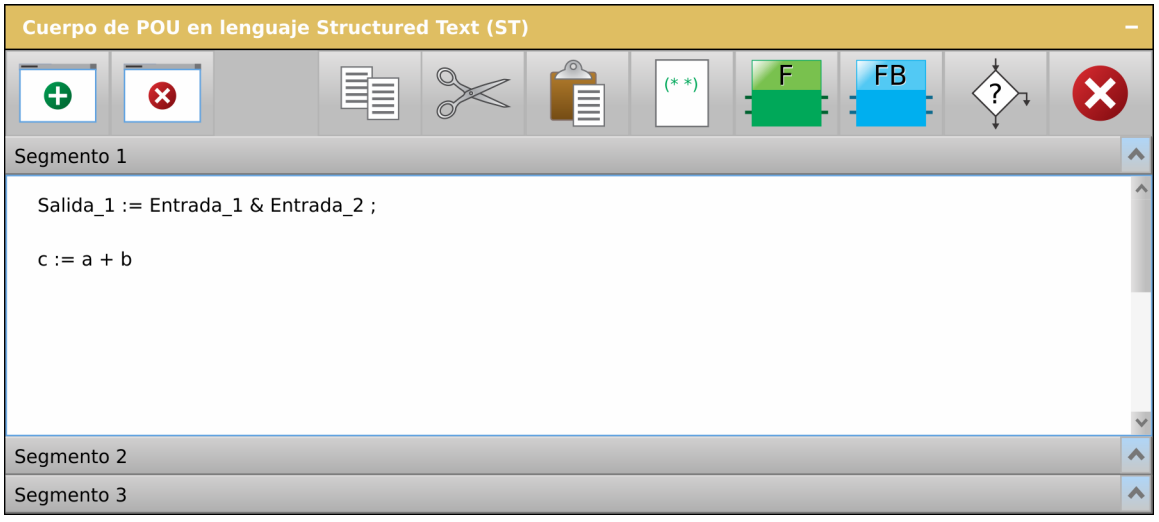


Figura 3.37: Sección "Cuerpo de POU en lenguaje Structured Text (ST)".

3.3.3. Ventana de Biblioteca

Permite visualizar al usuario todos los elementos utilizados para crear el programa del Controlador Programable. Estos elementos se encuentran agrupados en las categorías:

- Proyectos.
- Dispositivos.
- Configuraciones de Hardware.
- Programas.
- Funciones.
- Bloques de Función.
- Instrucciones IL.
- Componentes LD.
- Componentes FBD.
- Componentes de control de ejecución.

Como se muestra en la figura [3.38], cada categoría es representada por un ícono que permite acceder a la ventana en particular que muestra todos los elementos contenidos.

Cada ventana de categoría posee una lista de elementos y un área de "vista previa" del elemento seleccionado. Algunas de éstas poseen sólo elementos estándar de la norma IEC 61131-3 (como por ejemplo la categoría "Componentes LD"); otras, sólo elementos definidos por el usuario (ejemplo "Configuraciones de Hardware") y otras poseen ambos separados en dos secciones expansibles. Los elementos definidos por el usuario pueden editarse o borrarse. Estas ventanas de categoría son las mismas que se acceden al agregar cualquier componente en las barras de herramientas de las demás ventanas de la interfaz de usuario. Cuando esto sucede, se agrega un botón inferior que permite añadir al elemento seleccionado a la ventana invocante.

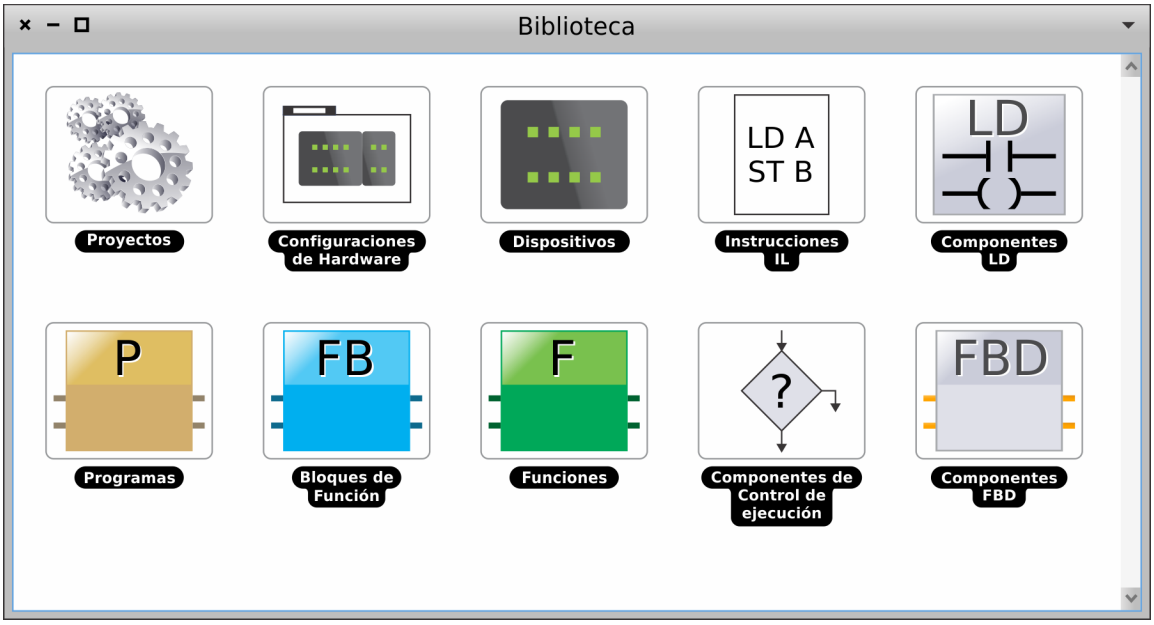


Figura 3.38: Ventana "Biblioteca".

En la figura [3.39] se muestra como ejemplo la ventana de Biblioteca de la categoría “Funciones” que muestra tanto Funciones estándar como Funciones definidas por el usuario.

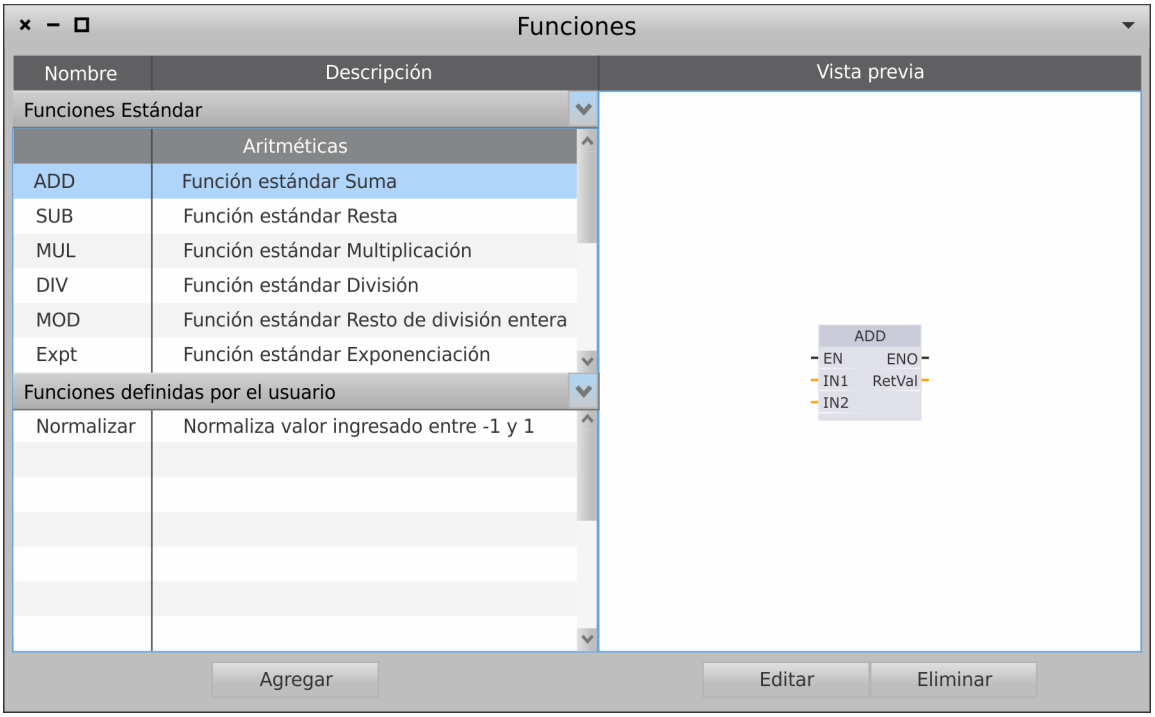


Figura 3.39: Ventana de Biblioteca de la categoría “Funciones”.

3.4. Modelo computacional

Con este modelo se plantea un marco donde el estándar encaja de manera natural, es decir, los mismos conceptos especificados en la norma IEC-61131-3 son los utilizados en el modelo computacional. Para describirlo, se elije el paradigma de objetos.

Se intenta que el modelo propuesto respete las siguientes características:

1. *Facilitar el desarrollo de su implementación*, dado que un buen modelo computacional, puede ayudar a que la implementación del editor de programas esté bien organizada.

2. *Reflejar los conceptos que estructuran un programa* en la definición de los distintos lenguajes dentro de la norma.

3. *Contribuir a una buena integración con la interfaz de usuario*, en particular para la edición de programas en lenguajes gráficos. Se destacan los siguientes aspectos: claridad en cómo debe modificarse un modelo de programa a partir de las distintas acciones que puede realizar un usuario y; posibilidad de definir, a partir del modelo de un programa gráfico, cómo deben organizarse sus elementos y conexiones al mostrarlo en pantalla.

Cabe destacar que para realizar el diseño de este modelo se aislaron las diferencias entre los cuatro lenguajes, logrando un modelo que es igual en muchos aspectos para programas en cualquiera de los lenguajes definidos en la norma y difiere solamente en el modelo del cuerpo de POU, sus segmentos y elementos específicos de cada lenguaje.

En las siguientes secciones se describen las partes fundamentales que componen el modelo de editor de programas de Controladores Programables.

3.4.1. Modelo de Tipos de Datos

En la norma IEC 61131-3 se definen tres categorías de tipos de datos: “Elementales” (3.4.1.1), “Derivados” (3.4.1.2) y “Genéricos” (3.4.1.3). El tipado en la norma es muy estricto y sólo se permite la conversión explícita entre tipos de datos mediante el uso de funciones de conversión de tipos.

Para modelar estos tipos de datos se utiliza el esquema de la figura [3.40].

El modelo de “Tipo de Datos” se encarga de realizar el chequeo de tipos de datos; es decir, si acepta a un determinado tipo de datos el cual recibe como parámetro. También puede chequear si un determinado valor es aceptado por el tipo de datos.

Cada tipo de datos es capaz de generar código en lenguaje textual de PLC (IL o ST) o C.

Para los lenguajes gráficos Ladder y FBD, los tipos de datos son capaces de indicar si permiten crear una conexión a un cierto pin de un componente o únicamente pueden asignar un argumento.

Los tipos de datos se agrupan en “Tipos de Datos Genéricos” y “Tipos de Datos No Genéricos”. Los genéricos contienen uno o más tipos no genéricos. Los no genéricos son capaces de indicar un valor inicial de tipo de datos. Este valor se utiliza como valor inicial de una variable cuando su valor inicial no es establecido por el usuario.

Existe un tipo de datos especial que modela la ausencia de tipo de datos, llamado “Tipo de Datos Vacío”. Si bien este tipo no se encuentra definido en la noma IEC 61131-3, se necesita para establecer un tipo de datos al “Operando No definido” que se define en la sección 3.4.2.

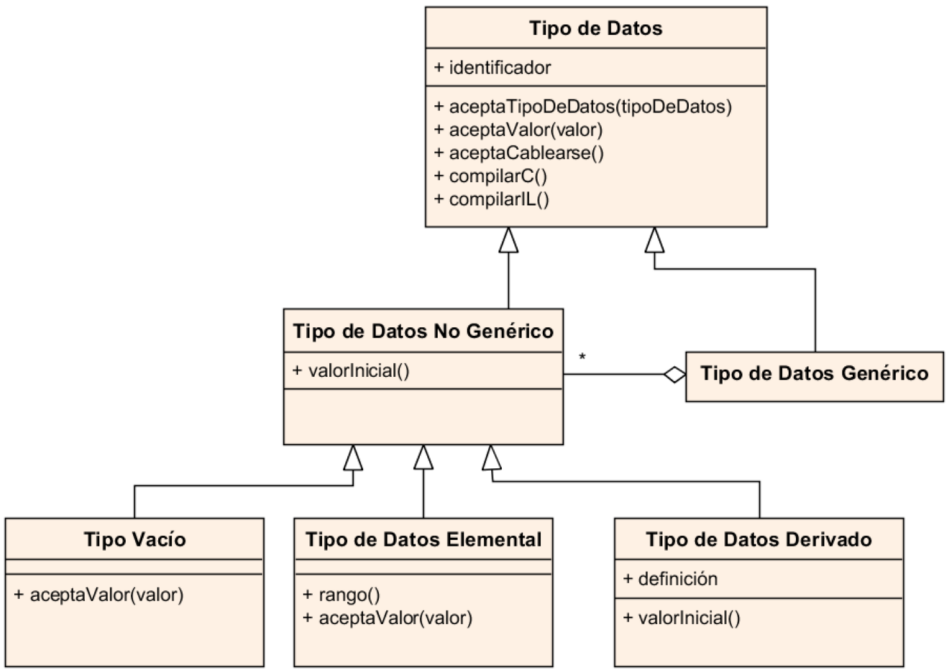


Figura 3.40: Modelo de Tipos de Datos.

3.4.1.1. Tipos de Datos Elementales

Los tipos de datos elementales que se encuentran en la norma corresponden a los tipos de datos base que puede utilizar el usuario.

El modelo "Tipo de Datos Elemental" provee servicios para discernir si acepta un determinado valor recibido como parámetro, es decir si éste se encuentra en su rango de valores; dar sus valores mínimo y máximo, y su valor incicial.

Existen varias categorías de tipos de datos elementales y sus tipos correspondientes:

- Cadena de bits: BOOL, BYTE, WORD, LWORD y DWORD.
- Duración: TIME.
- Fecha y Hora: DATE, TIME_OF_DAY y DATE_AND_TIME.
- Cadena de caracteres: STRING y WSTRING.
- Enteros con signo: SINT, INT, LINT y DINT.
- Enteros sin signo: USINT, UINT, ULINT y UDINT.
- Reales: REAL y LREAL.

Cada especificación (tipo de dato en particular) implementa los servicios anteriormente descriptos junto a las sutilezas de cada tipo de dato. Sólo se necesita un único objeto por cada tipo elemental, todas las referencias a cada tipo de datos corresponden al mismo objeto.

En este diseño se decidió que únicamente el tipo de datos elemental Booleano (BOOL) permita crear una conexión en un pin. Debido a esto, en los lenguajes gráficos sólo existirán "cables" entre pines de componentes del tipo booleano.

3.4.1.2. Tipos de Datos Derivados

Los tipos de datos derivados contemplan todos los tipos de datos creados por el usuario a partir de los tipos de datos elementales. En la norma IEC 61131-3 se definen dos tipos de datos derivados que contienen un único elemento de datos, siendo "Derivado directamente" y "Sub-rango"; y tres tipos de datos derivados con múltiples elementos de datos que corresponden a "Enumeración", "Arreglo" y "Estructura".

El modelo de tipos de datos derivados se expone en la figura [3.41].

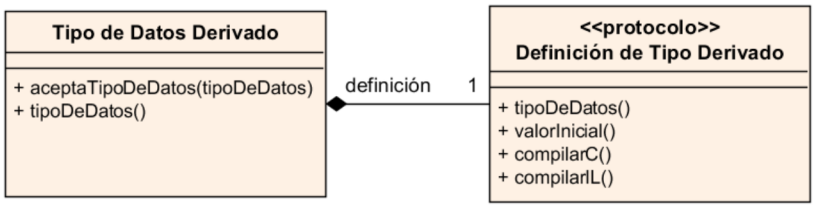


Figura 3.41: Modelo de tipos de datos Derivados.

Un "Tipo de Datos Derivado", se encarga de modelar una definición de tipo de datos y es capaz de generar código en lenguaje textual de PLC, o C. Posee un atributo identificador del tipo y una referencia a una definición de tipo de datos. Esta definición puede ser cualquier objeto que implemente el protocolo "Definición de Tipo Derivado".

A modo de ejemplo se describe cómo utilizar el diseño general recién descripto para modelar los tipos "Derivado directamente" y "Estructura".

Los tipos elementales implementan el protocolo "Definición de Tipo Derivado" modelando el tipo "Derivado directamente". De esta forma, una definición de tipo de dato con identificador 'ESTADO_MOTOR' y tipo de dato booleano se representa en el modelo de objetos de la figura [3.42].

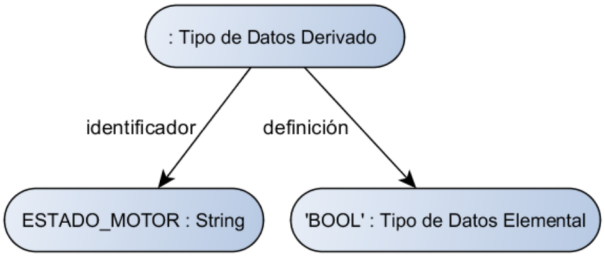


Figura 3.42: Ejemplo de tipo "Derivado directamente". Modelo de objetos.

Este modelo genera el código:

- En lenguaje C:

```
typedef PLC_BOOL ESTADO_MOTOR;
```

- En lenguaje IL o ST:

```
TYPE
  ESTADO_MOTOR : BOOL;
END_TYPE
```

Para modelar la “Estructura” se define una clase “Definición de Estructura” que implementa el protocolo “Definición de Tipo Derivado”, como se muestra en la figura [3.43].

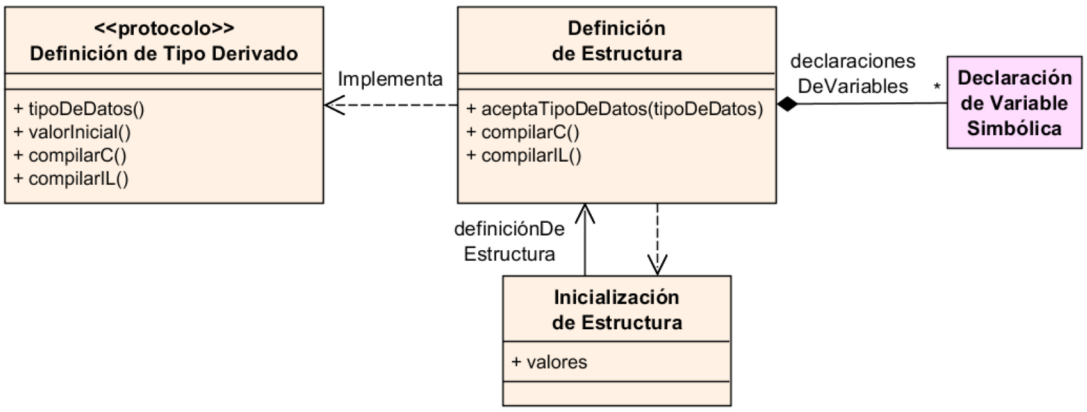


Figura 3.43: Modelo de “Definición de Estructura”.

La “Definición de Estructura” genera código en lenguajes textuales de PLC, o C. Contiene una o más “Declaraciones de variables Simbólicas” que se definen en la sección 3.4.4. Puede generar una “Inicialización de Estructura” como valor inicial.

En la figura [3.44] se muestra un ejemplo de modelo de objetos para una definición de estructura con identificador ‘Motor’, que contiene las declaraciones de variables simbólicas ‘RPM’ y ‘EstadoMotor’ de tipos de datos entero sin signo y booleano respectivamente.

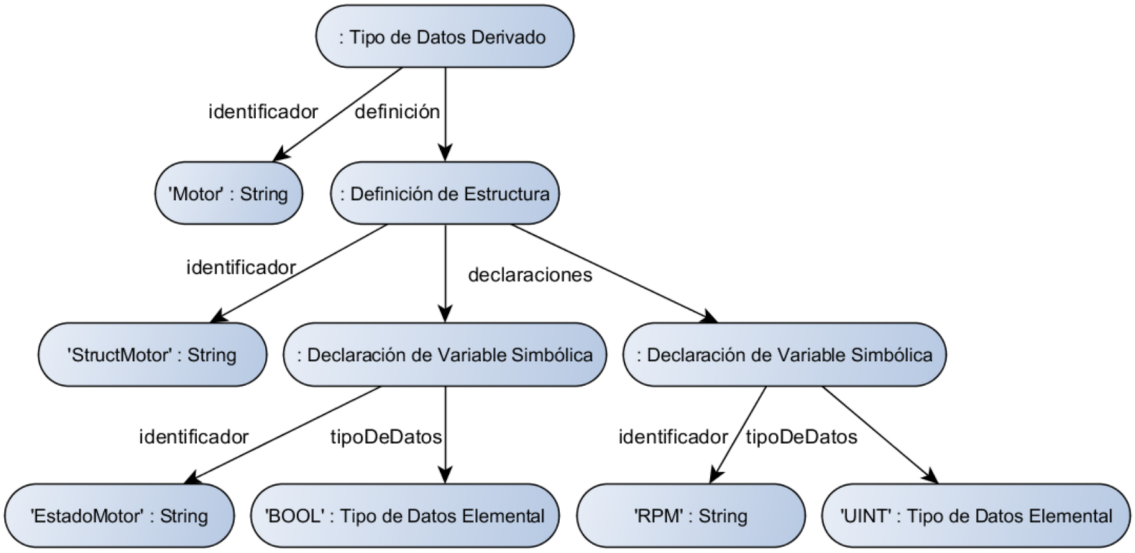


Figura 3.44: Modelo de objetos para una“Definición de Estructura”.

Este modelo genera el código:

- En lenguaje C:

```
typedef
  struct StructMotor{
    PLC_UINT RPM;
    PLC_BOOL EstadoMotor;
    } Motor;
```

- En lenguaje IL o ST:

```
TYPE
  Motor :
    STRUCT
      RPM : UINT;
      EstadoMotor : BOOL;
    END_STRUCT
END_TYPE
```

Una inicialización de estructura, modela la asignación de valores iniciales de las variables contenidas en la definición de estructura cuando se declara una variable de tipo estructurado. Esta inicialización de estructura conoce a una "Definición de estructura" y su conjunto de valores. Como ejemplo, el código que genera la inicialización de la estructura generada por 'StructMotor' es:

- En lenguaje C:

```
{
  0,
  TRUE
}
```

- En lenguaje IL o ST:

```
(
  0,
  TRUE
)
```

3.4.1.3. Tipos de Datos Genéricos

Los tipos de datos genéricos son tipos de datos que agrupan tipos Elementales y Derivados. Se definen en la norma para permitir la sobrecarga de tipos de datos en Funciones o Bloques de función estándar, operadores o instrucciones. Esto significa que puedan operar con diferentes tipos de datos utilizando un tipo de datos genérico. Por ejemplo, en la figura [3.45] se ilustra la función estándar ADD (suma), sobrecargada para cualquier tipo de datos numérico (ANY_NUM).

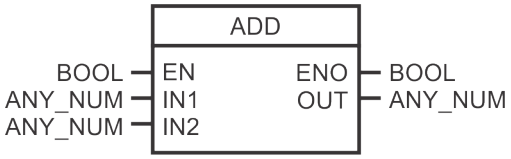


Figura 3.45: Ejemplo de POU Función ADD sobrecargada para cuelquier numérico.

La existencia de los tipos de datos genéricos no rompe con la especificación de tipado estricto, ya que sólo se permite que la POU estándar pueda tener asignadas variables de un solo tipo en cada uso de la misma. En el ejemplo anterior, esto significa que en un llamado a dicha función todas las variables asignadas deben ser del mismo tipo de datos. No se permite utilizar los tipos de datos genéricos en ninguna POU definida por el usuario.

En la figura [3.46] se muestra el modelo de tipos de datos genéricos. Si bien en este modelo todos han sido diseñados como especificaciones de "Tipo de Datos Genérico" se ordenan los mismos

genéricos de manera jerárquica como especifica la norma IEC 61131-3. Estos tipos utilizan a los tipos de datos no genéricos, para determinar si un tipo genérico dado, acepta a un cierto tipo no genérico.

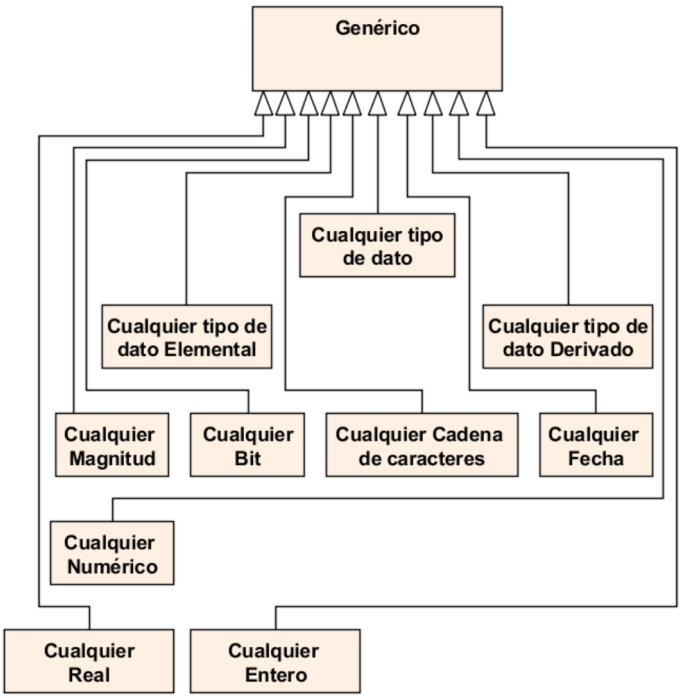


Figura 3.46: Modelo de tipos de datos Genéricos.

3.4.2. Modelo de Operandos

Un Operando representa el valor que va a ser ingresado como argumento en llamados a POU (los cuales se definen en la sección 3.4.7.2), instrucciones IL (sección 3.4.7.1) y elementos gráficos Argumentos (sección 3.4.8.1). Los operandos pueden clasificarse como:

- Literal.
- Operando No Definido.
- Operando Dirección.
- Operando Variable.

El Modelo de Operandos se ilustra en la figura [3.47]. Todos los operandos pueden retornar un tipo de datos y un valor. Permiten generar código tanto en lenguaje IL como en lenguaje C.

Un operando **Literal** modela un valor constante de cualquier tipo de datos, por ejemplo, 'BOOL#TRUE', 'INT#10', 'REAL#58.5e-3', etc. Contiene un valor y un Tipo de Datos (definido en la sección 3.4.1).

El **Operando No Definido** modela el caso específico de un operando que aún no ha sido ingresado por el usuario. Es ampliamente utilizado en los lenguajes gráficos. Contiene un tipo de datos vacío (sección 3.4.1). Su valor es fijo, la cadena '<???>'.

Un **Operando Dirección** representa el valor de una dirección física de un Controlador Programable, por ejemplo, '%I0.0', '%Q1.3', '%M5.0', etc. Contiene una "Dirección Física" como se define en la sección 3.4.10.

El **Operando Variable** contiene un valor y una "Declaración de Variable Simbólica" la cual se define en la sección 3.4.4. Se subcategoriza en:

- **Operando Variable Simple**. Ejemplo 'Var'.
- **Operando Item de Enumeración**. Ejemplo 'segundo'.
- **Operando Acceso a Variable Interna de Estructura**. Ejemplo 'STR.VAR1'.
- **Operando Acceso a Indice de Arreglo**. Ejemplo 'ARR[5]'.

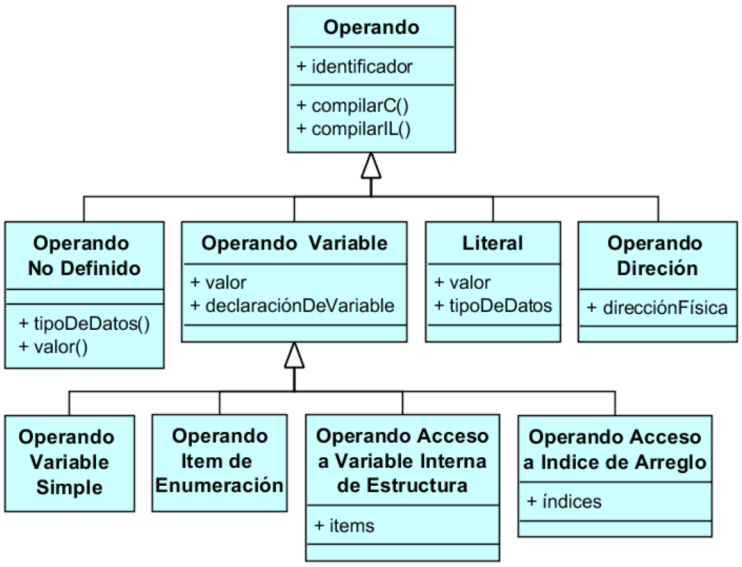


Figura 3.47: Modelo de Operandos.

3.4.3. Modelo de Asignaciones de POU

Las asignaciones de POU se utilizan en los llamados a POU cuyo modelo se define en la sección 3.4.7.2. Existen tres tipos:

- Asignación de entrada
- Asignación de salida
- Asignación de entrada-salida

Este modelo se muestra en la figura [3.48].

Todas las asignaciones incluyen un parámetro formal y un argumento. El parámetro formal es una "Declaración de Variable" (ver sección 3.4.4), mientras que el argumento es un operando (definido en la sección 3.4.2).

Además, tiene la capacidad de indicar si se asigna un "Argumento Válido" a la declaración de variable. Este chequeo es delegado a la declaración de variable, que utiliza su categoría de declaraciones de variables y su tipo de datos para resolverlo como se detalla en la sección 3.4.5.

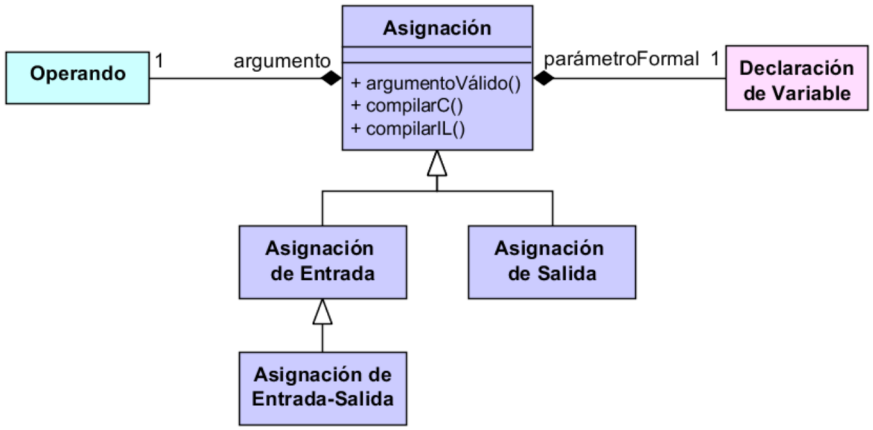


Figura 3.48: Modelo de Asignaciones de POU.

Como ejemplos de asignaciones de Entrada, teniendo un parámetro formal cuyo identificador es 'IN1' y un operando literal '10' genera:

- En lenguaje C

```
IN1 = 10
```

- En lenguaje IL o ST

```
IN1 := 10
```

Si el operando es un ítem de estructura 'Motor.RPM' genera:

- En lenguaje C:

```
IN1 = Motor.RPM
```

- En lenguaje IL o ST:

```
IN1 := Motor.RPM
```

En estos ejemplos si el tipo de datos de la declaración de variable de IN1 es entero (al igual que el tipo del ítem RPM), ambos argumentos serían válidos.

Como ejemplo de asignación de Salida cuyo parámetro formal sea 'Salida7' y utilizando el literal del ejemplo anterior generaría:

- En lenguaje C:

```
10 = Salida7
```

- En lenguaje IL o ST:

```
Salida7 => 10
```

En este caso el argumento es inválido ya que la categoría de salidas no admite literales ya que no tendría sentido como puede observarse en el código C.

3.4.4. Modelo de Declaraciones de Variables

En la norma IEC 61131-3 se definen tres formas de declarar variables: “Representación directa”, “Simbólica” y “Simbólica Localizada”.

La primera corresponde a una declaración de variable que corresponde a una dirección lógica en memoria especificada por el usuario, por ejemplo, ‘AT: %I0.1 : BOOL’ representa una entrada de un bit de tamaño en la dirección 1 del byte 0. Existen tres áreas lógicas de memoria, donde pueden declararse variables de representación directa en un Controlador Programable y, por lo tanto, Entradas (se identificada con la letra ‘I’), Salidas (la letra ‘Q’) y Memoria Interna (letra ‘M’).

La segunda, Declaración de Variable Simbólica, es una declaración de variable con un nombre simbólico ubicada automáticamente en el área de Memoria Interna por el Controlador Programable.

Finalmente, una declaración Simbólica Localizada corresponde a una declaración de variable con un nombre simbólico pero cuya dirección lógica es especificada por el usuario; por ejemplo, ‘Salida0 AT: %Q0.0 : BOOL’ representa una salida de un bit de tamaño en la dirección 0 del byte 0 cuyo nombre simbólico es ‘Salida0’.

El modelo de Declaraciones de Variables se muestra en la figura [3.49].

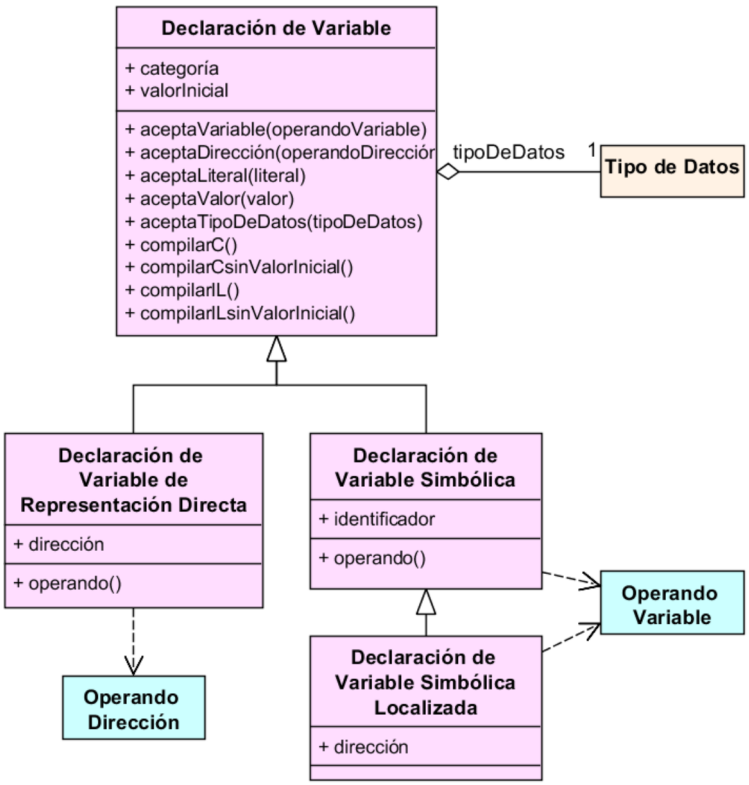


Figura 3.49: Modelo Declaraciones de Variables.

Como puede observarse, el comportamiento en común de las declaraciones de variables se agrupa en “Declaración de Variable”. Contiene un tipo de datos (definidos en la sección 3.4.1), y un valor inicial. Es capaz de realizar chequeos acerca de si acepta un determinado operando, valor o tipo de datos recibido como parámetro. Además, permite su compilación a los lenguajes textuales de PLC (IL o ST), y al lenguaje C incluyendo, o no, el valor inicial. En esta sección y en la sección 3.4.1 se han expuesto varios ejemplos de su compilación.

Para especificar el comportamiento en particular de las declaraciones de variables se utilizaron los mismos nombres especificados al comienzo de esta sección.

Una “Declaración de Variable de Representación directa” contiene un atributo dirección que hace referencia a una dirección lógica en memoria. Es capaz de generar un Operando Dirección (definido en la sección 3.4.2).

La “Declaración de Variable Simbólica” posee un atributo identificador, que se utiliza para dar un nombre a la misma. También es capaz de generar un Operando, en este caso, un Operando Variable (ver sección 3.4.2).

En el caso de una “Declaración de Variable Simbólica Localizada” se diseñó como una especificación de la Declaración de Variable Simbólica ya que posee el mismo comportamiento con el agregado de un atributo dirección.

3.4.5. Modelo de Categoría de Declaraciónes de Variables

La norma IEC 61131-3 define muchas formas de agrupar declaraciones de variables y sendos significados. Con el modelo propuesto en la figura [3.50] se intenta abarcar todas estas formas.

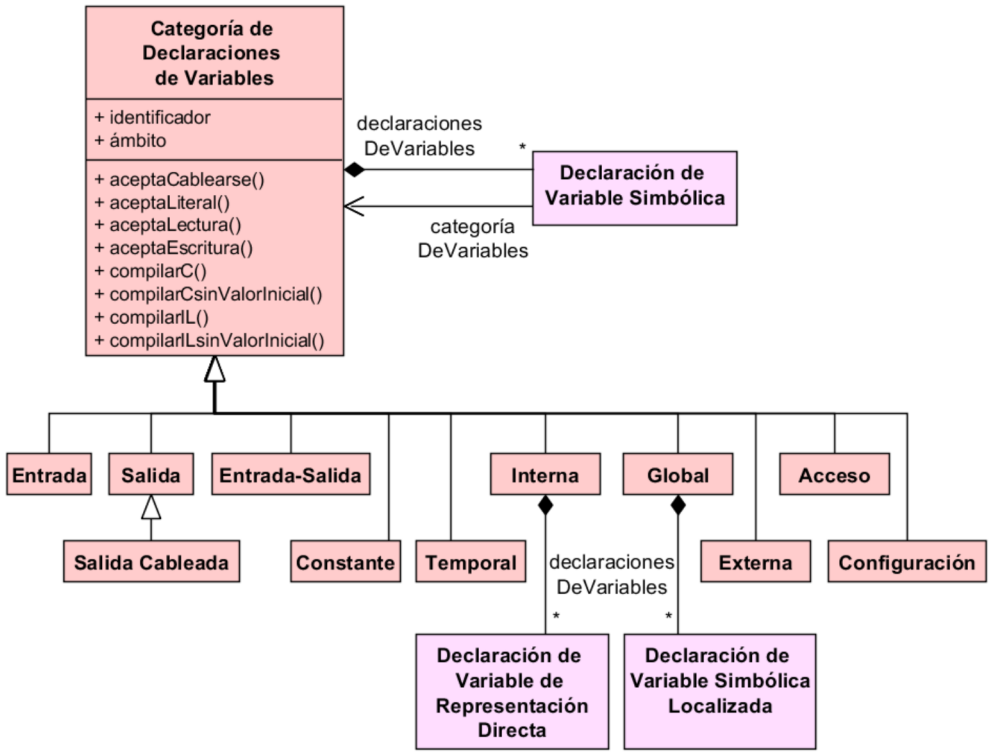


Figura 3.50: Modelo de Categorías de Declaraciones de Variables.

Una categoría de declaraciones de variables modela un grupo de declaraciones de variables que comparten características en común y tienen el mismo alcance (o ámbito). Contiene un atributo identificador, que utiliza para generar su código en lenguajes textuales de PLC (IL o ST), y una colección de una o más “declaraciones de variables simbólicas”. Genera código en lenguajes textuales de PLC o C, incluyendo en el código otorgado por sus declaraciones de variables con, o sin, valores iniciales. Como se observa en la figura [3.50] toda la funcionalidad en común de las categorías de variables se agrupan en “Categoría de Declaraciones de Variables” mientras que el comportamiento específico es contenido por cada especificación. En particular la especificación “Interna”, además de declaraciones de variables simbólicas, puede contener “Declaraciones de Variables de Representación Directa”. La especificación Global puede contener “Declaraciones de Variables Simbólicas Localizadas”. Por ejemplo, una categoría de variable con identificador ‘INPUT’ y dos declaraciones de variables genera:

- En lenguaje C:

```
PLC_UINT RPM ;
PLC_BOOL EstadoMotor ;
```

- En lenguaje IL o ST:

```
VAR_INPUT
  RPM : UINT;
  EstadoMotor : BOOL;
END_VAR
```

Este ejemplo corresponde a la categoría de variables 'entradas' de una POU. En lenguaje C no existen bloques del tipo VAR_INPUT ... END_VAR definidos en la norma IEC 61131-3 para indicar el ámbito de una variable y , por lo tanto, otros objetos del modelo deberán encargarse de ubicar el código C donde sea necesario para establecer su ámbito.

En conjunto, las categorías de variables, declaraciones de variables y tipos de datos son responsables de los chequeos que se exponen en las siguientes secciones.

3.4.5.1. Operando utilizado como Argumento

Cuando se utiliza un Operando Variable como argumento, una categoría de variables puede indicar si el mismo acepta lectura o escritura para indicar si es un argumento válido. Por ejemplo, esto sirve para realizar el chequeo si puede utilizarse, o no, como argumento en una Asignación de Salida un Operando Variable cuya Declaración de Variable Simbólica se encuentre contenida en una Categoría de variables 'VAR_INPUT'. En este ejemplo no se puede; pues las variables de entradas, no pueden escribirse en el interior del cuerpo de donde se encuentran declaradas, como se especifica en la norma IEC 64431-3.

3.4.5.2. Declaración de variable utilizada como Parámetro Formal

Las "Declaraciones de variables" pueden ser utilizadas como Parámetros Formales dentro de declaraciones de POU, Recurso o Configuraciones de Software.

Junto con los tipos de datos, las categorías de variables establecen qué pines pueden ser conectados mediante cables en las POU (tipos Función o Bloque de Función) en los lenguajes gráficos. Mientras que la norma IEC 61131-3 establece que a excepción de las variables declaradas como 'VAR_IN_OUT' todas pueden ser cableadas, en este Trabajo Final se tomo como decisión que las categorías de variables que permiten ser cableadas son todas las de entrada (VAR_INPUT) y una única declaración de variable de salida (VAR_OUTPUT).

Las categorías de variables también establecen si una cierta categoría permite asignar, o no, literales como valores de argumentos de sus pines. Aquí la norma establece que sólo la categoría 'VAR_INPUT' permite asignar literales a sus argumentos.

Ejemplificando lo expuesto, se ofrece en la figura [3.51] una declaración gráfica de una POU de tipo Función. En la misma se indican las categorías de declaraciones de variables, que definen las declaraciones de variables de interfaz (pines de conexión externos en los lenguajes gráficos) de esta función. También se indican qué pines permiten ser cableados de acuerdo a las restricciones tanto de categorías de variables, como de tipo de datos de las declaraciones de los pines.

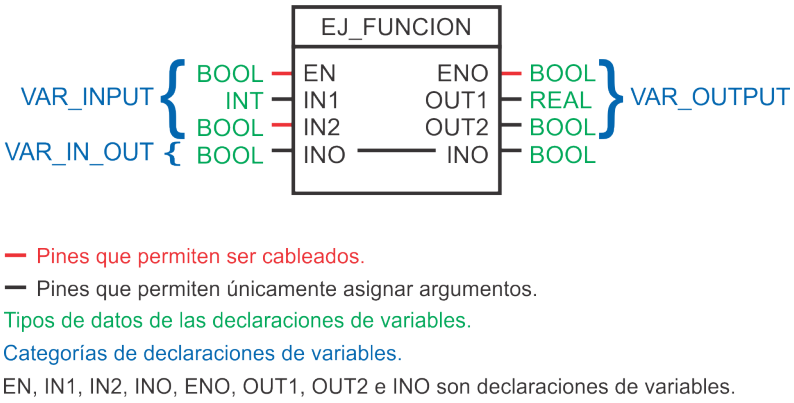


Figura 3.51: Ejemplo de categorías de variables de una Función.

3.4.6. Modelo de POU

Una POU modela tanto una definición, como una de declaración de una Unidad de Organización de Programa (POU), las cuales se definen en la norma IEC 61131-3. El modelo general de POU se ilustra en la figura [3.52].

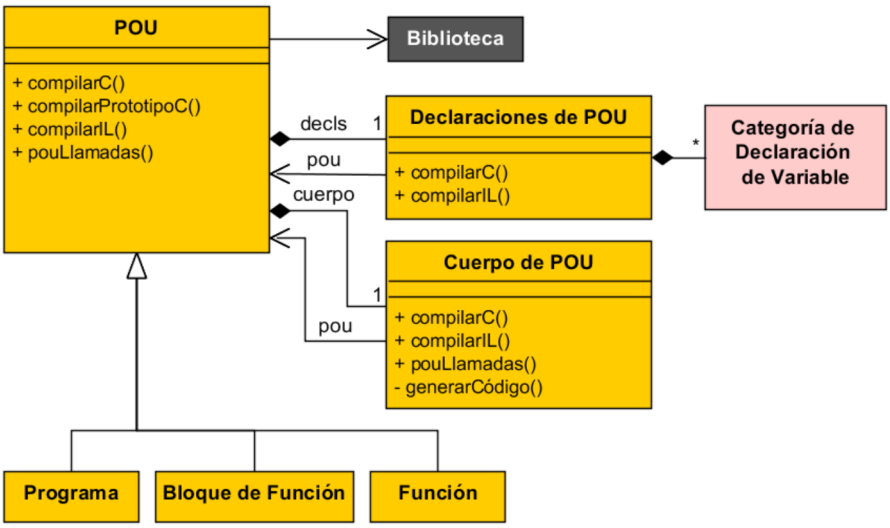


Figura 3.52: Modelo de POU.

Cada POU contiene un "Cuerpo de POU" y "Declaraciones de POU". También conoce a la **Biblioteca** donde se almacenan las mismas.

El **Cuerpo de POU** es el modelo de programa que crea el usuario utilizando uno de los lenguajes definidos en la norma IEC 61131-3. Se especifica en la sección 3.4.6.1.

Declaraciones de POU modela el ámbito de variables asociado a la POU. En este ámbito las variables se encuentran agrupadas en categorías cuyo modelo fue descripto en la sección 3.4.4. Las categorías soportadas por cada tipo de POU se describieron en la sección 3.3.2.1.

Como se ha anticipado, existen tres tipos de POU: "Programas", "Bloques de Función" y "Función". Cada uno controla que categorías de declaraciones de variables puede contener y cuáles tipos de POU pueden ser invocadas en su cuerpo de programa.

La generación de código de una POU se realiza utilizando sus "Declaraciones de POU", como fue expuesto en las secciones 3.4.4 y 3.4.5, y su "Cuerpo de POU" (ver sección 3.4.6.1). Es capaz

de retornar una definición de POU en lenguaje IL, una definición de función en C que implementa la POU en este lenguaje y su prototipo. Además, puede solicitarse que retorne una colección de POU invocadas dentro de su cuerpo. A continuación se ofrece un ejemplo del código generado por una POU del tipo Programa.

Ejemplo de generación de código en una POU del tipo Programa

- Definición de POU en lenguaje IL:

```
PROGRAM ProgEj

  (* Declaraciones de POU *)

  VAR_INPUT
    Ent1 : BOOL;
    Ent2 : BOOL;
  END_VAR

  VAR_OUTPUT
    Sal : BOOL;
  END_VAR

  (* Cuerpo de POU *)

  LD  Ent1
  AND Ent2
  ST  Sal

END_PROGRAM
```

- Definición de Función en C que implementa la POU:

```
void ProgEj(PLC_P_Struct_ProgEj *pxPOU)
{
  // Cuerpo de POU

  // LD  Ent1
  PLC_IL_LD( &(pxPOU->Ent1), sizeof(pxPOU->Ent1), BOOL, NullModifier);
  // AND Ent2
  PLC_IL_AND( &(pxPOU->Ent2), BOOL, NullModifier);
  // ST Sal
  PLC_IL_ST( &(pxPOU->Sal), sizeof(pxPOU->Sal), NullModifier);

}
```

- Prototipo de Función en C que implementa la POU:

```
void ProgEj(PLC_P_Struct_ProgEj *);
```

3.4.6.1. Modelo de Cuerpo de POU

El "Cuerpo de POU" contiene el funcionamiento común para todos los lenguajes. Existen cuatro especificaciones de cuerpo de POU, uno para el lenguaje textual IL, dos para los lenguajes gráficos (Ladder y FBD) y el último para modelar las POU estándar definidas en la norma IEC 61131-3. El modelo de cuerpo de POU se ilustra en la figura [3.53].

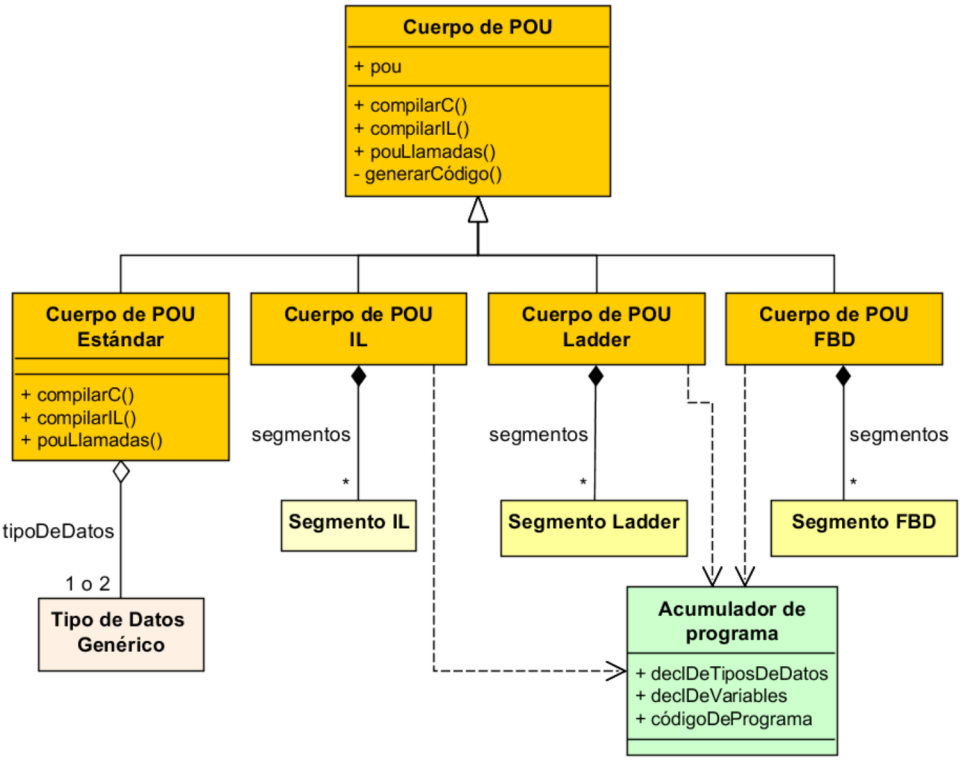


Figura 3.53: Modelo de Cuerpo de POU.

El contenido del cuerpo de POU Estándar no es accesible por el usuario debido a que se trata de un cuerpo de POU escrito directamente en lenguaje C. Debido a esto sólo permite retornar código en lenguaje C. Contiene un Tipo de Datos Genérico (definidos en la sección 3.4.1) que se utiliza para indicar los tipos de datos soportados (sobrecarga) por una POU estándar.

Tanto el cuerpo de POU para el lenguaje IL, como el cuerpo de POU para los lenguajes gráficos, se compone de uno o más "Segmentos". Cada uno de estos representa un segmento de programa hecho en el lenguaje correspondiente a su cuerpo y puede contener una "Etiqueta de segmento" cuyo identificador debe ser único en toda a POU. Todos los segmentos de un Cuerpo de POU deben ser Segmentos del mismo lenguaje de programación, esto es controlado por el cuerpo de POU, en consecuencia, se requiere que exista un cuerpo de POU para cada lenguaje de programación.

Para generar código, los cuerpos de programa IL, Ladder y FBD crean un objeto "Acumulador de programa" y se lo envían a cada uno de sus segmentos los cuales se encargan de rellenar con objetos que permiten compilarse tanto a IL como a C. Como se verá en las siguientes secciones, la estrategia de generación de código es distinta para un Segmento de POU en lenguajes textuales y gráficos.

NOTA: El lenguaje ST no ha sido modelado debido a la decisión de no implementarlo en el presente Trabajo Final. La Razón de esta decisión radica en que el lenguaje ST es un lenguaje muy similar a C y su implementación carece de aportes significativos. En caso de requerir su implementación, simplemente debería realizarse un traductor de expresiones y declaraciones de ST a lenguaje C, cuya estructura semántica es similar pero varía su sintaxis.

En las siguientes secciones se describen los segmentos de POU para el lenguaje IL y los lenguajes gráficos.

3.4.7. Modelo de Segmento de POU en lenguaje IL

El modelo de segmento en lenguaje IL se expone en la figura [3.54].

Figura 3.54: Modelo de Segmento IL.

Un "Segmento IL" conoce al cuerpo de POU que lo contiene y posee un atributo 'programa IL', que es una referencia una colección de "Instrucciones IL" y "Llamados a POU", los cuales generan código en lenguaje IL o C como se expondrá en las próximas secciones.

Cuando un Cuerpo de POU le pide que genere código a un Segmento IL, éste simplemente guarda sus instrucciones y llamados en el Acumulador de Programa que recibe como parámetro desde el cuerpo de POU.

3.4.7.1. Modelo de Elementos de lenguaje IL

Los Elementos de lenguaje IL modelan las instrucciones y sus modificadores en este lenguaje de programación. El esquema del modelo se muestra en la figura [3.55].

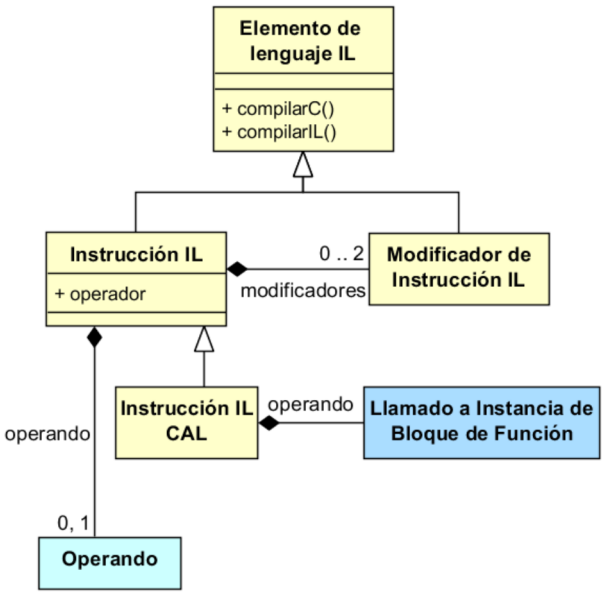


Figura 3.55: Modelo de Elementos de lenguaje IL.

Como se ha anticipado, cada elemento de lenguaje IL es capaz de generar su código en lenguajes IL y C.

Una "Instrucción IL" contiene un atributo operador, que indica el código de operación de la instrucción (por ejemplo, 'LD', 'S', 'OR', etc.), puede contener un operando (definido en la sección 3.4.2) y hasta 2 "Modificadores de Instrucción IL".

Las instrucciones se pueden clasificar a partir de sus operadores de la siguiente manera:

- Carga y almacenamiento - LD y ST.
- Set y Reset - S y R.
- Lógicas - AND, OR, XOR y NOT.
- Aritméticas - ADD, SUB, MUL, DIV y MOD.
- Comparación - GT, GE, EQ, NE, LE y LT.
- Control de ejecución de programa - JMP y RET.
- Cerrar paréntesis -).

Entre los modificadores de instrucción se encuentran:

- Modificador No asignado: Modela el modificador no asignado.
- Modificador "N": Modela el modificador que aplica una negación lógica bit a bit al operando antes de aplicar la operación. Se aplica a las instrucciones lógicas, de carga y almacenamiento, control de ejecución de programa y a la "Instrucción IL CAL".
- Modificador "C": Modela el modificador que aplica una condición a las instrucciones de control de ejecución de programa. Esta condición chequea el acumulador "Resultado Actual" y en base al resultado booleano obtenido ejecuta, o no, la instrucción a la cual se le aplica. Este modificador puede ser utilizado en las instrucciones de control de ejecución de programa y a la "Instrucción IL CAL" y combinarse con el modificador N.
- Modificador Abrir Paréntesis "(": Este modificador indica que la evaluación de un operador será retrasada hasta que se encuentre el operador cerrar paréntesis ")". De esta manera se resuelve primero la operación entre paréntesis y luego la operación externa. Puede combinarse con el modificador N.

Ejemplo de instrucción de carga - LD

El modelo de objetos para una instrucción 'LD' con operando 'var1' cuya declaración de variable simbólica contiene un tipo entero se muestra en la figura [3.56].
Este modelo genera:

- En lenguaje C:

```
1 PLC_IL_LD( &(pxPOU->var1), sizeof(pxPOU->var1), INT, NullModifier );
```

- En lenguaje IL:

```
1 LD var1
```

Como puede inferirse mediante este ejemplo, las instrucciones en IL se implementan como funciones en C en este diseño.

Existe una "Instrucción IL CAL" que modela un llamado a Instancia de Bloque de Función en este lenguaje. A diferencia de las demás instrucciones, ésta contiene como operando un "Llamado a Instancia de Bloque de Función" el cual se explica en la siguiente sección.

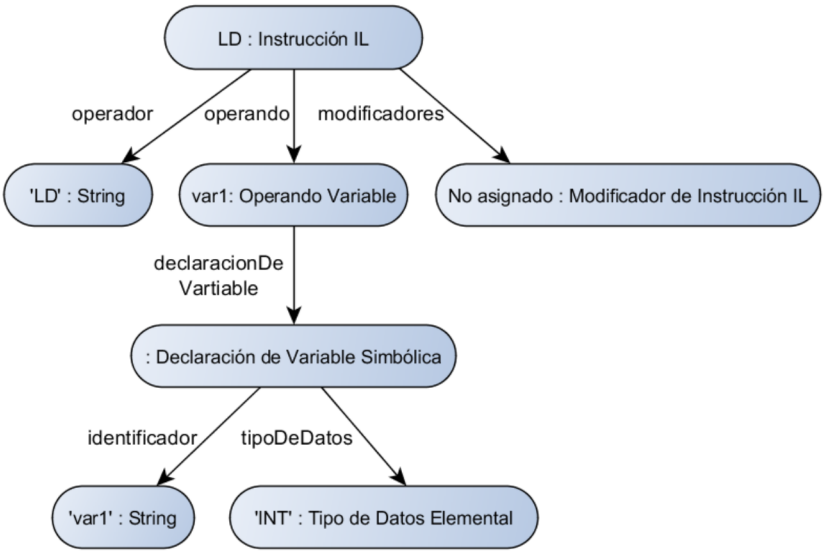


Figura 3.56: Ejemplo de Instrucción IL de carga 'LD'.

3.4.7.2. Modelo de Llamado a POU

El "Llamado a POU" modela un llamado o invocación a POU. Este modelo se muestra en la figura [3.57].

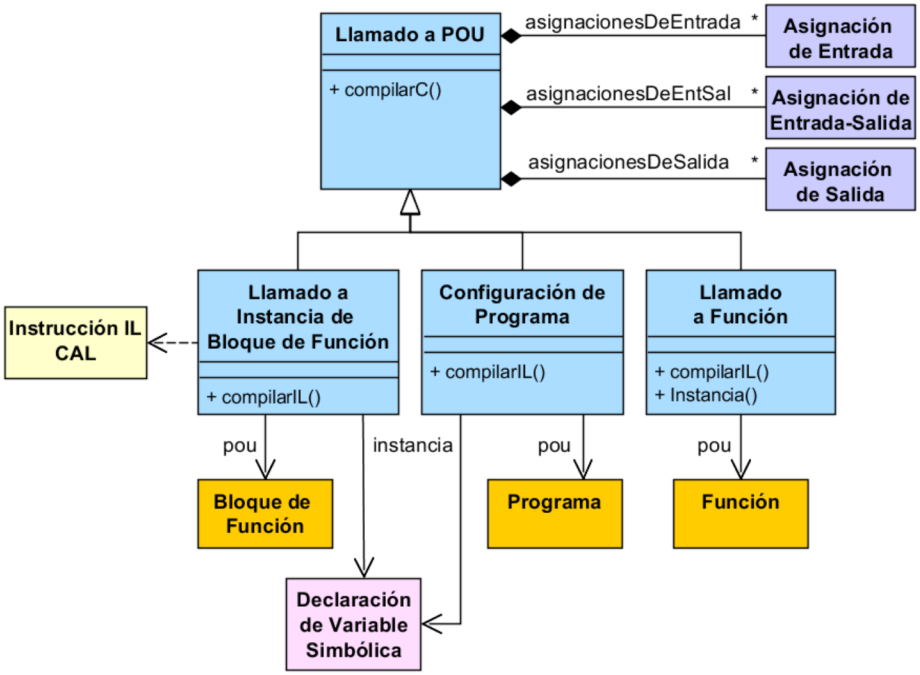


Figura 3.57: Modelo de Llamado a POU.

Un llamado a POU contiene asignaciones (definidas en 3.4.3).

Existen tres especificaciones, una por cada tipo de POU, estas corresponden a "Llamado a Función", "Llamado a Instancia de Bloque de Función" y "Configuración de Programa[5]". Cada

[5]En particular el nombre puede resultar un poco confuso, ya que se trata de un llamado a instancia de programa, pero se eligió así en concordancia con el nombre otorgado por la norma IEC 61131-3.

una de ellas contiene una referencia a la POU que llama. Las últimas dos contienen además, una referencia a una declaración de variable simbólica (definida en 3.4.4), que corresponde a una instancia de la POU que llama.

El llamado a instancia de Bloque de Función es capaz de generar una instrucción IL CAL y asignarse a ésta como operando.

Un ejemplo de llamado se ofrece a continuación.

Ejemplo de llamado a Instancia 'TON_1' del Bloque de Función Temporizador 'TON'

Un llamado a instancia 'TON_1' con POU del tipo Bloque de función 'TON'. Con dos asignaciones de entrada ('EN := VAR1' y 'PT := TIME#10s') y una asignación de salida ('Q => VAR2') genera:

- En lenguaje C:

```
TON_1.IN = VAR1 ;       // Asignación de entrada
TON_1.PT = 10000 ;      // Asignación de entrada (tiempo en ms)
PLC_TON( &(TON_1) );    // Llamado a función en C de TON
VAR2 = TON_1.Q ;        // Asignación de salida
```

- En lenguaje IL:

```
CAL TON_1(
  EN := VAR1 ,
  PT := TIME#10s ,
  Q => VAR2
  )
```

Su modelo de objetos se muestra en la figura [3.58].

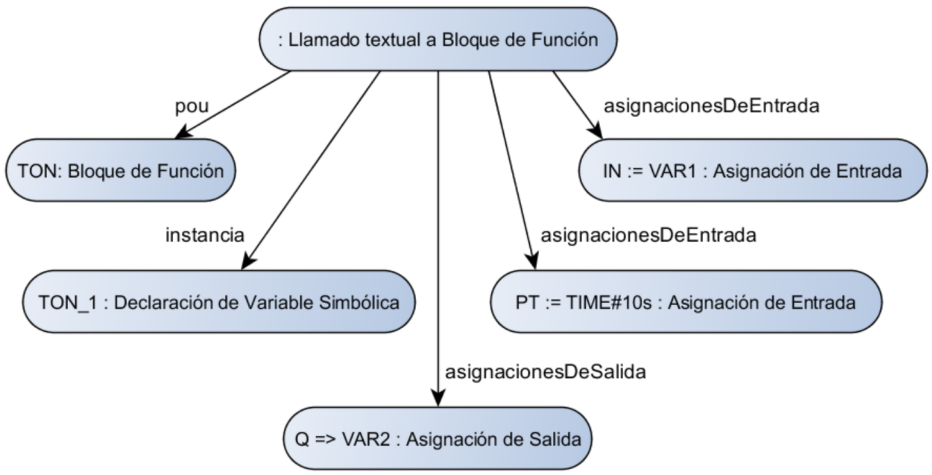


Figura 3.58: Ejemplo de llamado a Instancia 'TON_1' del Bloque de Función Temporizador 'TON'.

Para generar el código en lenguaje IL internamente este llamado genera una Instrucción IL CAL a la que se agrega como operando y le solicita que entregue su código generado. El modelo de objetos de esta instrucción se ilustra en la figura [3.59].

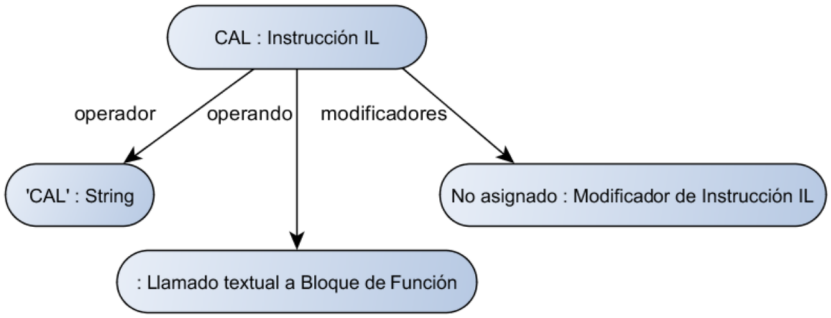


Figura 3.59: Ejemplo de Instrucción IL CAL.

3.4.8. Modelo de Segmentos de POU en lenguajes gráficos

Los lenguajes gráficos Ladder y FBD se basan en la construcción de un diagrama de componentes interconectados, que contienen muchas partes en común. De esta forma, se define un modelo general para un segmento gráfico y dos especificaciones con las particularidades de cada uno. El modelo general de Segmento en lenguajes gráficos se ilustra en la figura [3.60].

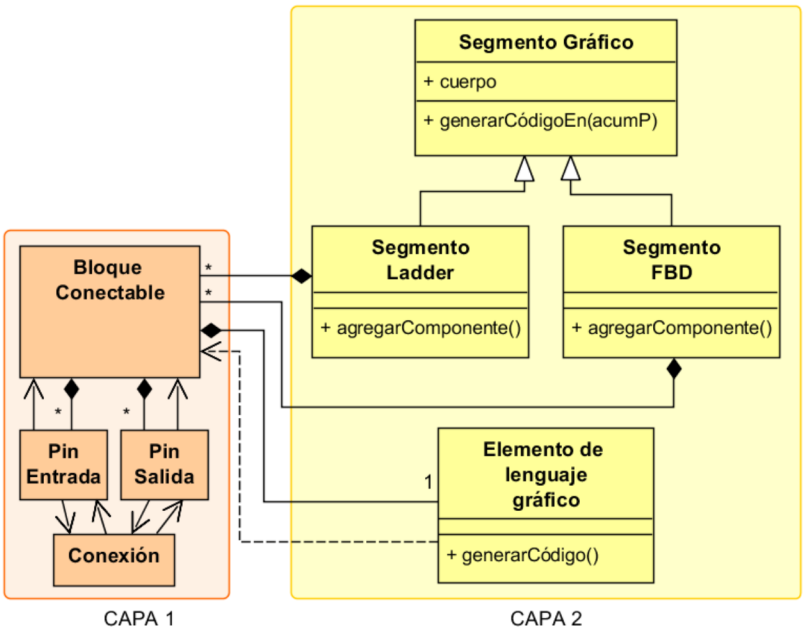


Figura 3.60: Modelo de Segmento en lenguejes gráficos

Este modelo se compone de los siguientes elementos:

1. **Bloque conectable:** Modela cada parte intercambiable de un componente. Cada uno posee un Elemento de lenguaje gráfico y Pines de conexión.

2. **Elemento de lenguaje gráfico:** Representa la función de cada bloque conectable. Por ejemplo, en lenguaje Ladder: contactos, bobinas, llamadas a funciones, argumentos, etc. Cada elemento incluye la información de cómo debe ser recorrida la red de bloques conectables, para convertir el modelo de segmento gráfico a uno textual.

3. **Pin de conexión:** Describe las entradas y salidas de los Bloques conectables. Existen "Pines de Entrada" y "Pines de Salida".

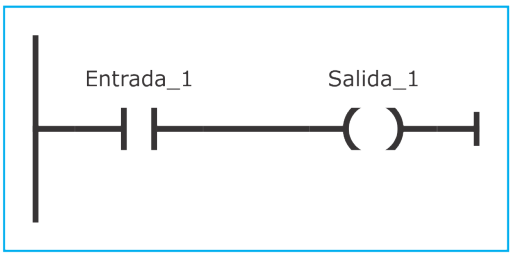
4. **Conexión:** Modela la conexión entre un Pin de salida de un bloque conectable y un pin de entrada de otro.

Mediante este modelo, un **Componente de lenguaje gráfico** se modela como un bloque conectable, o varios bloques conectables interconectados.

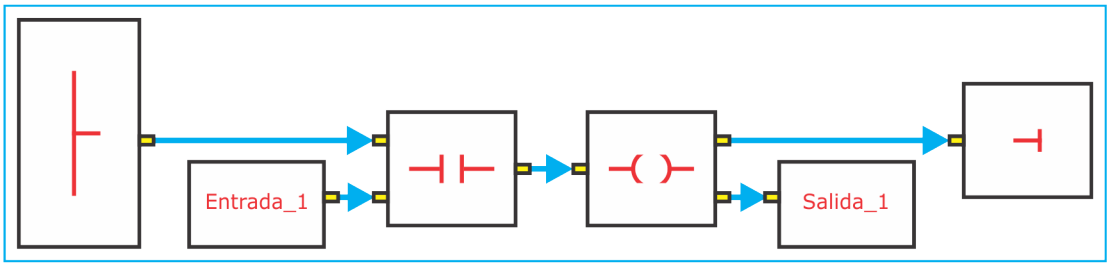
Finalmente, los componentes se combinan formando una de **Red de Bloques conectables** que modela un segmento de programa en los lenguajes gráficos.

Para comprender como colaboran entre sí las diferentes partes que constituyen la red de bloques conectables, y como se aplica este modelo para ambos lenguajes; se ofrecen los siguientes ejemplos:

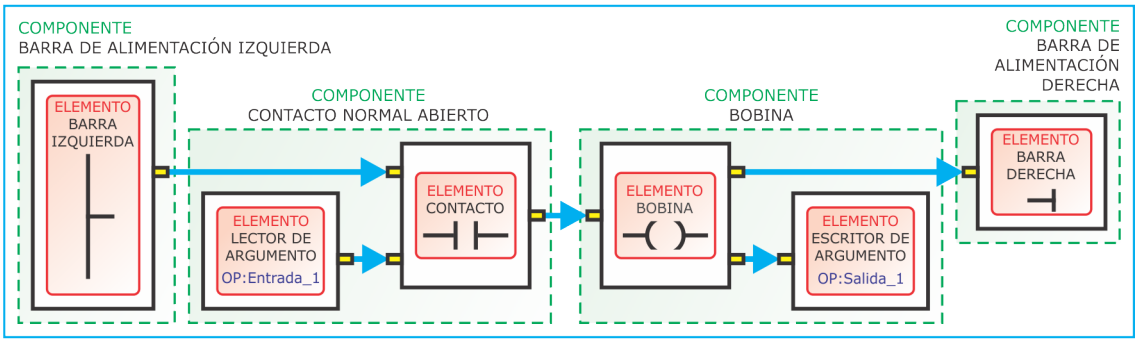
En la figura [3.61], se ofrece un gráfico conceptual esta red para un Segmento sencillo en lenguaje Ladder.



(a) Segmento Ladder.



(b) Red de bloques conectables equivalente al Segmento Ladder.



(c) Red de bloques conectables con nombres de Componentes y Elementos de programa gráfico destacados.

Figura 3.61: Red de bloques conectables de un Segmento Ladder.

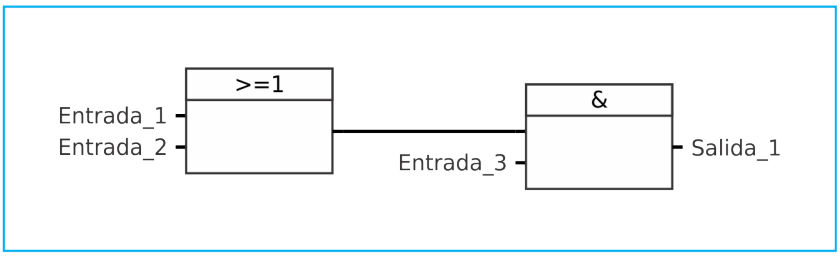
Esta figura consta de tres partes:

(a) El Segmento Ladder a modelar formado por cuatro componentes: "Barra de alimentación izquierda", "Contacto", "Bobina" y "Barra de alimentación derecha".

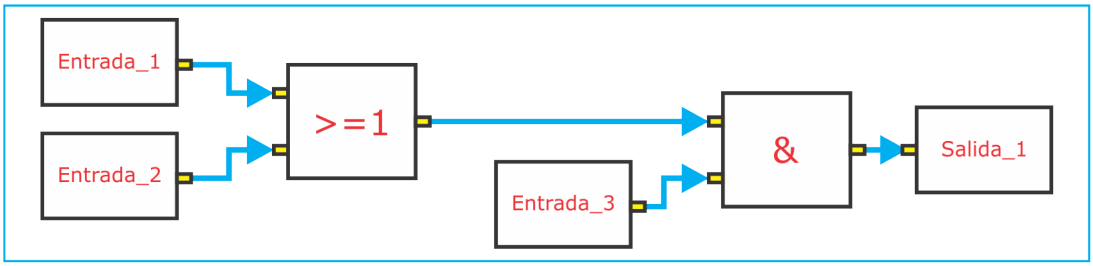
(b) Red de bloques conectables equivalente. Las flechas de color cian representan *conexiones*. Cada rectángulo blanco simboliza un *bloque conectable*, que contiene un *elemento de lenguaje gráfico* (dibujado en color rojo) y rectángulos amarillos a izquierda y derecha representando sus *pines de conexión* de entrada y salida respectivamente.

(c) Se destacan los nombres de cada *elemento* y los *componentes*. Como puede observarse, para modelar un componente contacto se utilizan dos bloques conectables, uno con elemento "Lector de Argumento" y el otro con elemento "Contacto".

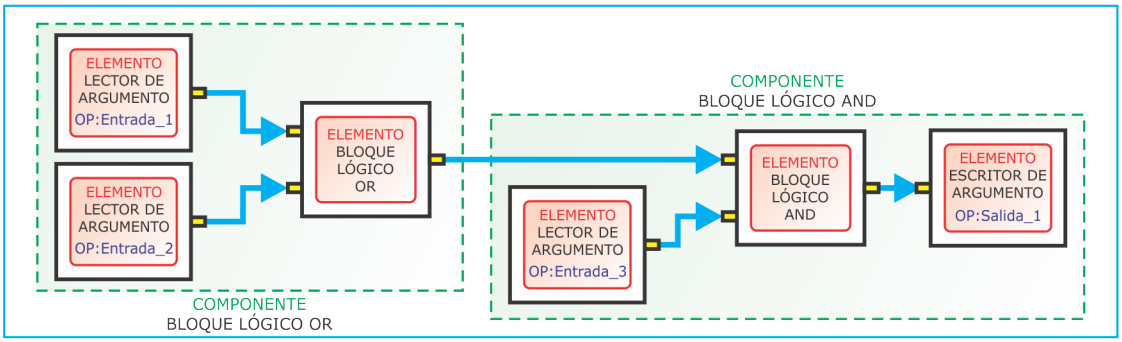
Un ejemplo de "red de bloques conectables" para un Segmento FBD se muestra en la figura [3.62]. Esta figura posee la misma estructura que la figura [3.61]. En este caso, el Segmento FBD a modelar consta de dos componentes, un "Bloque lógico OR" y un "Bloque lógico AND".



(a) Segmento FBD.



(b) Red de bloques conectables equivalente al Segmento FBD.



(c) Red de bloques conectables con nombres de Componentes y Elementos de programa gráfico destacados.

Figura 3.62: Red de bloques conectables de un Segmento FBD.

De los ejemplos anteriores se destaca que existen elementos de programa gráfico comunes a ambos lenguajes. En los mismos pueden observarse que los elementos de programa gráfico "Lector de Argumento" y "Escritor de Argumento" se repiten en ambos lenguajes. Otros elementos comunes son "Llamado a Función" y "Llamado a Instancia de Bloque de Función".

3.4.8.1. Capas del modelo de Segmento gráfico y responsabilidades

La Red de Bloques conectables que modela un segmento gráfico se diseñó con la intención de separar el modelo de Segmento en dos capas diferenciables:

- *Capa 1*: Mantiene la información de los componentes y sus conexiones (en los *bloques conectables*, *pines* y *conexiones*), y brinda servicios para recorrer la red a otros objetos (en el *segmento*).
- *Capa 2*: Corresponde al *segmento Ladder o FBD*, que es responsable del agregado o borrado de componentes completos de lenguajes gráficos manteniendo la lógica del circuito resultante, y a los *elementos de lenguajes gráficos*, responsables de la generación de código de un segmento.

Esta separación facilita definir nuevos componentes para los lenguajes Ladder y FBD, o bien, definir nuevos lenguajes gráficos que se basen en componentes interconectados. Para realizarlo, sólo se debería modificarse la capa 2, dejando intacta la capa 1.

Bloques conectables, pines y conexiones

Cada bloque conectable se construye solicitando a su elemento de programa gráfico el número de entradas y salidas. Tiene la capacidad de agregar y eliminar pines. Además controlan la conexión y desconexión con otro bloque.

En este diseño sólo se permite como conexión, un único pin de salida de un bloque conectable con un único pin de entrada de otro bloque conectable diferente. Esto es controlado por los pines y conexiones colaborando mutuamente. De esta manera se evitan muchos casos de error.

Elementos de lenguaje gráfico

Se diseñaron todos los elementos que forman parte de los lenguajes de programación Ladder y FBD. Pueden clasificarse de la siguiente forma:

Elementos comunes a ambos lenguajes gráficos

Estos elementos son elementos que se utilizan en ambos lenguajes. Los mismos corresponden a:

- Lector de Argumento
- Escritor de Argumento
- Llamado gráfico a Función
- Llamado gráfico a Instancia de Bloque de Función
- Retorno de función
- Salto a segmento etiquetado

Elementos de lenguaje Ladder

Los elementos específicos de lenguaje Ladder son:

- Barra de alimentación derecha
- Barra de alimentación izquierda

- Contacto
- Bobina
- Enlace vertical

Elementos de lenguaje FBD

En este caso, los elementos específicos de lenguaje FBD son:

- Bloque lógico
- Nodo de conexión

Características de los Elementos de lenguajes gráficos

Cada elemento gráfico (exceptuando los Lectores y Escritores de argumento, y los Llamados gráficos) contiene Categorías de Variables y Declaraciones de Variables. Se utilizan para indicar a un bloque conectable cuántos pines de entradas y salidas debe tener cuando se construye. El elemento, contiene la información del mapeo entre número de Pin y declaración de Variable correspondiente. Esta información la utiliza durante el proceso de generación de código.

Los elementos "Lector de Argumento" y "Escritor de Argumento", aunque no contienen Categorías de Variables, indican al bloque conectable que debe construirse con un Pin de Salida y un Pin de entrada respectivamente. Contienen "Operandos" definidos en 3.4.2. Son capaces de chequear si un argumento es válido. Para realizar el chequeo, crean una Asignación (definida en 3.4.3) y delegan en la misma el chequeo como se informó en la sección 3.4.5.

Existen varios tipos de elementos gráficos contactos y bobinas en el lenguaje Ladder. Los contactos incluyen "Contacto normal abierto", "Contacto normal cerrado", "Contacto sensor de transición positiva" y "Contacto sensor de transición negativa"; mientras que las bobinas son "Bobina", "Bobina negada", "Bobina Set", "Bobina Reset", "Bobina sensor de transición positiva" y "Bobina sensor de transición negativa". En cuanto a los elementos gráficos de FBD, existen tres tipos de bloques lógicos, "AND", "OR" y "XOR".

Los elementos "Llamado gráfico a Función" y "Llamado gráfico a Instancia de Bloque de Función" mantienen las conexiónes en los lenguajes gráficos. Son capaces de generar un "Llamado a Función" y "Llamados a Instancia de Bloque de Función" respectivamente, definidos en la sección 3.4.7.2.

Para completar la explicación de cómo colaboran conjuntamente los Elementos Gráficos, se toma como ejemplo el componente "Llamado a Función ADD" como se muestra en la figura [3.63].

Este componente está formado por un Elemento gráfico "Llamado gráfico a Función" y Elementos gráficos "Lectores y Escritores de Argumento" conectados al primero. El Llamado gráfico a Función conoce a la POU que llama del tipo Función, en este caso la Función "ADD", obteniendo de la misma sus declaraciones de variables de "Entrada" y "Salida". Además, contiene la información de como se mapean los pines de su bloque conectable contenedor, con las declaraciones de variables obtenidas de la Función.

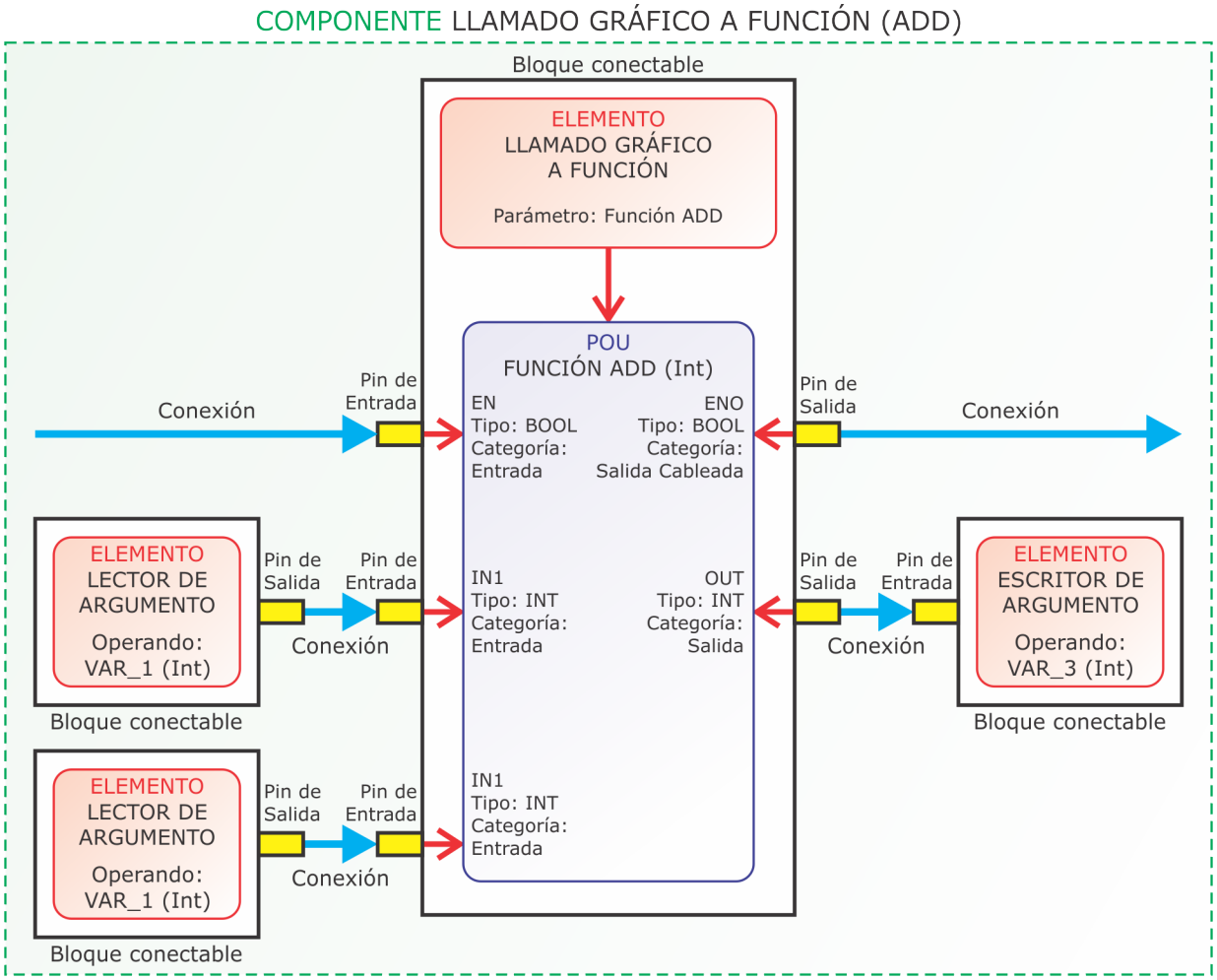


Figura 3.63: Ejemplo de Componente "Llamado gráfico a Función ADD".

3.4.8.2. Dinámica de un Segmento Ladder

Como se ha mencionado, al implementar el modelo se debe reconfigurar al mismo correctamente ante modificaciones del usuario. Esto es responsabilidad del Segmento Ladder. Sus tareas son:

- Agregar un Componente.
- Eliminar un Componente.
- Abrir una Rama.
- Cerrar una Rama.

Estas tareas implican el agregado o eliminación de varios bloques conectables y sus conexión/-desconexión de manera específica. Para realizar cada una de ellas, existen varios chequeos complejos como consecuencia de las especificaciones de la interfaz gráfica.

En esta sección se detalla como cambia el modelo ante acciones del usuario y qué chequeos debe realizar un Segmento Ladder para mantener una red de bloques conectables válida.

Un Segmento Ladder inicia con un componente "Barra de Alimentación Izquierda" conectada a una "Barra de Alimentación Derecha" como se muestra en la figura [3.64].

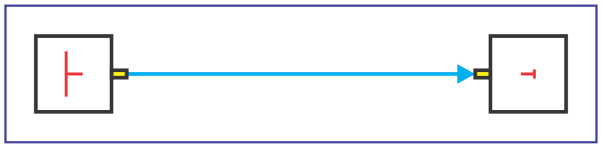


Figura 3.64: Segmento inicial en lenguaje Ladder.

Agregado de componentes

En un Segmento Ladder solamente pueden agregarse componentes sobre una conexión gráfica, conectando su primer entrada y salida a los componentes de los extremos de dicha conexión. Los componentes que pueden agregarse de esta forma son: Contactos, Bobinas, Llamados a Funciones o Bloques de función, Retorno de función y Salto a segmento etiquetado.

Cuando el usuario indica al Segmento Ladder que debe agregar un componente específico en cierta conexión, busca a través de los bloques conectables si se trata de una rama en paralelo, o no. Si es una rama conectada en paralelo, chequea si el elemento principal del componente a agregar permite, o no, conectarse en paralelo. Los componentes que permiten conexión en paralelo son los Contactos y Llamados a Bloques de Función de Comparación (por ejemplo, mayor, menor, igual). Luego de validar todos estos chequeos, crea los bloques conectables pertenecientes al componente, los agrega al Segmento Ladder, conecta internamente los bloques conectables que lo conforman y finalmente abre la conexión donde se debía agregar el componente y conecta el mismo integrándolo a la red de bloques conectables.

Ejemplo de agregado de un Contacto Normal Abierto

Se solicita a la única conexión disponible en el Segmento de la figura [3.64], que se agregue un "Contacto Normal Abierto"; se realizan los chequeos descriptos, se crean y conecta los dos Elementos que forman el Contacto Normal Abierto, como se muestra en la figura [3.65].

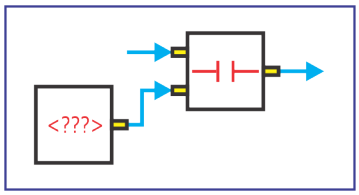


Figura 3.65: Componente Contacto Normal Abierto.

Finalmente se conecta donde fue previsto, resultando en el modelo de la figura [3.66].

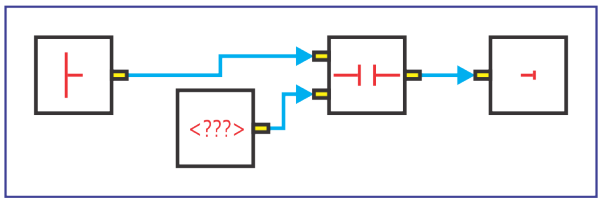


Figura 3.66: Segmento Ladder con Componente Contacto Normal Abierto.

Abrir una Rama

Esta operación se utiliza para abrir una rama paralela, sobre una conexión. En la figura [3.67] se expone un ejemplo de bloques conectables de un Segmento Ladder donde se desea Abrir una Rama sobre la conexión indicada.

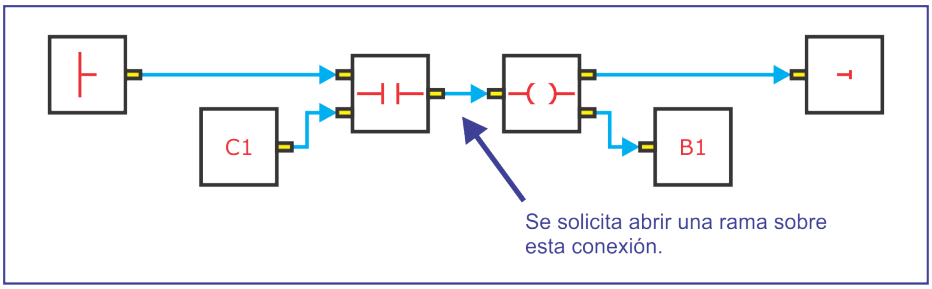


Figura 3.67: Ejemplo de Segmento Ladder donde se desea Abrir una Rama.

Para abrir una rama, crea un Componente “Enlace Vertical” con una entrada y dos salidas. Crea un Componente Barra de Alimentación Derecha y se la conecta a la segunda salida del Enlace Vertical, luego donde se encontraba la conexión, conecta la primer entrada y salida del Enlace Vertical, resultando en la red de bloques conectables de la figura [3.68].

En esta figura también puede observarse que existen algunas conexiones numeradas.

En caso de requerir abrir una rama en la conexión ‘1’, en lugar de crear un nuevo Enlace Vertical como se realizo previamente, se agrega un Pin de Salida al Componente Barra de Alimentación Izquierda, se crea una nueva Barra de Alimentación Derecha, se conecta al recientemente creado Pin de Salida de la Barra de Alimentación Izquierda.

Si en cambio se solicita abrir rama sobre las conexiones ‘2’, ‘3’ o ‘4’, se agrega un nuevo Pin de Salida al Enlace Vertical y conecta al mismo una nueva Barra de Alimentación Derecha.

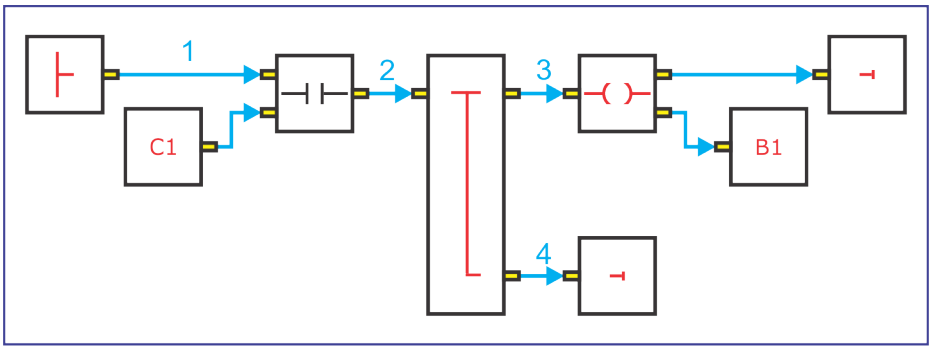


Figura 3.68: Ejemplo de Segmento Ladder con una Apertura de Rama.

Cerrar una Rama

Existen dos opciones de cerrado de rama para crear una conexión en paralelo, la primera, es cerrar rama desde una Barra de Alimentación Derecha hacia una Conexión, la segunda, cerrar una rama entre dos Conexiones.

Cuando se requiere realizar una de estas dos acciones, se chequea que todos los elementos contenidos en las ramas, que formarán una conexión en paralelo, permitan estar conectados en paralelo. Además, chequea que no se forme un cortocircuito al cerrar la rama paralela. En caso afirmativo, realiza la conexión en paralelo modificando adecuadamente el modelo.

Ejemplo de Cerrar Rama Paralela entre una Barra de Alimentación Derecha y una Conexión

En la figura [3.69] se expone la red de bloques conectables, indicando entre cuáles elementos se desea cerrar una rama paralela.

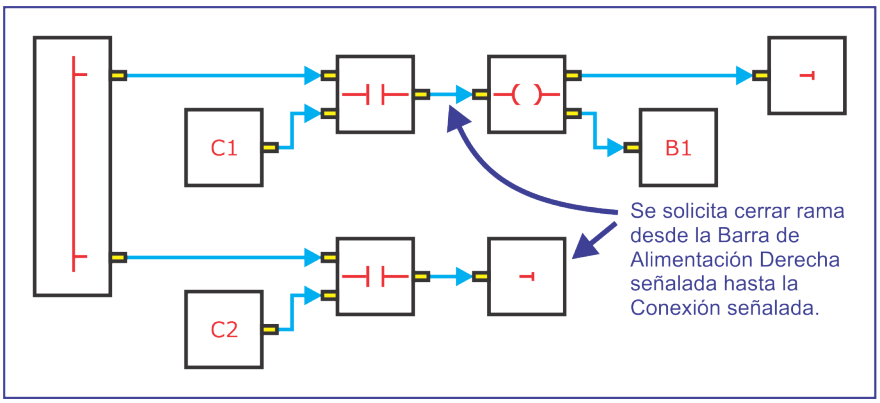


Figura 3.69: Ejemplo de Segmento Ladder donde se desea Cerrar una Rama.

Ambos contactos permiten estar conectados en paralelo y no se forman cortocircuitos. En consecuencia, puede cerrar la rama paralela resultando en la red de bloques conectables que se muestra en la figura [3.70].

Para facilitar los chequeos, se establecieron casos en los cuales se determina los lugares del circuito donde es posible cerrar una rama paralela.

Otro chequeo que realiza se da cuando por ejemplo, se desea cerrar una rama paralela en una conexión antes de una apertura de rama. En este caso, debe agregar un Pin de Entrada al enlace Vertical de la apertura de rama y conectarse en el mismo.

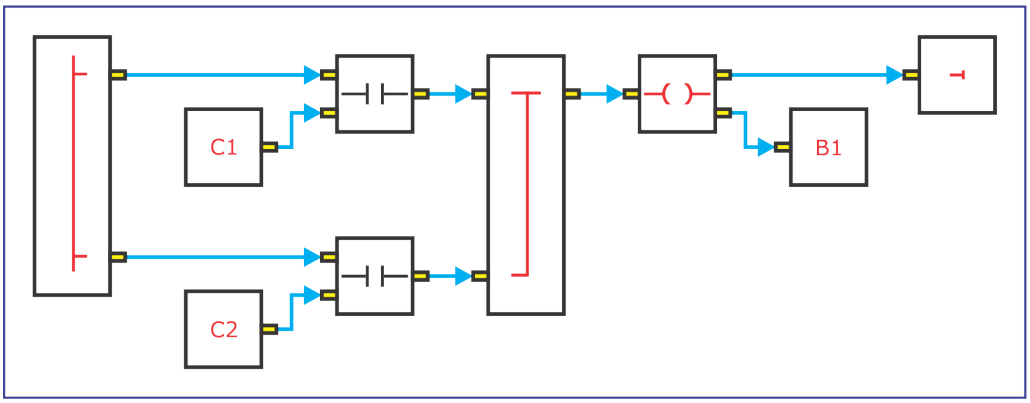


Figura 3.70: Ejemplo de Segmento Ladder con conexión en paralelo.

Eliminación de componentes

Los Componentes que permiten ser borrados son:

- Contactos
- Bobinas
- Llamados
- Barra de Alimentación Derecha cuando se encuentra conectada a un Enlace Vertical o Barra de Alimentación Izquierda.
- Salto a Segmento.
- Retorno.

Para eliminar elementos deben realizarse muchos de los chequeos expuestos previamente y conectar la salida y entrada que quedan sin conexión donde se borra el Componente para mantener un circuito lógico.

Los chequeos que realiza son:

- Chequear que no se produzca cortocircuito al eliminar el Componente.
- Al eliminar una Barra de Alimentación Derecha, cuando se encuentra conectada a un Enlace Vertical, o Barra de Alimentación Izquierda, se elimina además el Pin de Salida del componente al cual estaba conectada la Barra de Alimentación Derecha. En el caso de que este componente sea la Barra de Alimentación Izquierda debe chequear que siempre quede al menos un Pin de salida, si es el último no lo borra pues se perdería el segmento. Si en cambio el componente es un Enlace Vertical, debe chequear que quede un Enlace Vertical coherente luego de eliminar el correspondiente Pin, esto es; que quede al menos, con un Pin de Entrada y Dos Pines de Salida, o bien dos Pines de Entrada y uno de Salida. En caso contrario, elimina también el Enlace Vertical y conecta los extremos donde se encontraban conectados su primer Pin de Entrada y primer Pin de Salida.
- El chequeo de Enlace Vertical coherente lo realiza cada vez que borra cualquier Componente. De esta forma, si por ejemplo se borra un Contacto el cual era el único componente en paralelo, se borran consecuentemente ambos Enlaces Verticales conectados a sus extremos como puede apreciarse en el ejemplo siguiente.

Ejemplo de Borrado de Componente Contacto en paralelo

Partiendo de la red de bloques conectables de la figura [3.71], se desea eliminar el contacto con la cruz.

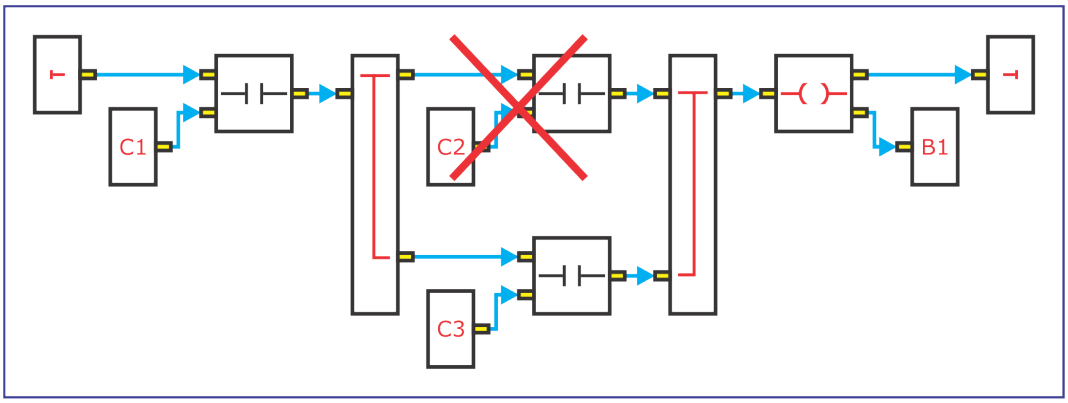


Figura 3.71: Ejemplo de Borrado de Componente Contacto en paralelo.

Al eliminarlo, elimina consecuentemente el primer Pin de Salida del Enlace Vertical a su izquierda y el primer Pin de Entrada del Enlace Vertical a su derecha. Cuando chequea si ambos enlaces son coherentes, descubre que no y en consecuencia los elimina. Luego, reconecta todo los bloques conectables, obteniendo finalmente, la red de bloques conectables que se muestra en la figura [3.72].

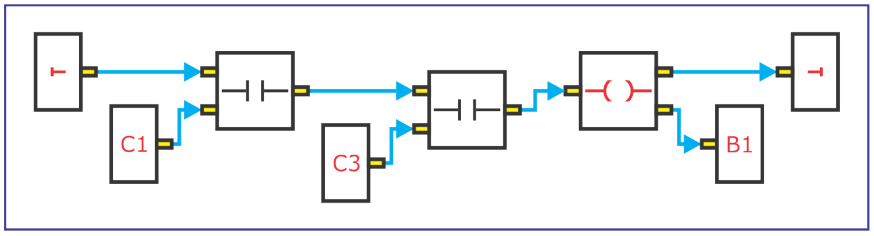


Figura 3.72: Ejemplo de Borrado de Componente Contacto en paralelo luego del borrado.

3.4.8.3. Dinámica de un Segmento FBD

De la misma forma que para lenguaje Ladder, el segmento FBD controla las operaciones referentes a sus respectivos Componentes. Sus tareas son:

- Agregar un Componente.
- Eliminar un Componente.
- Conectar Pin de Salida con Pin de Entrada de dos Componentes distintos.
- Conectar pin de Entrada a una Conexión entre dos pines.

Se describe a continuación cómo cambia el modelo ante acciones del usuario y qué chequeos debe realizar un Segmento FBD.

Un Segmento FBD inicia con la pantalla totalmente vacía.

Agregado de componentes

A diferencia de Ladder, en FBD no debe realizar ningún chequeo para agregar un nuevo Componente ya que los mismos se crean y agregan con elementos Lectores y Escritores de Argumento conectados a los pines del elemento principal[6] del Componente.

Conexión entre un Pin de Salida de un Componente y un Pin de Entrada de otro

Para poder realizar esta conexión, interroga a ambos componentes si las declaraciones correspondientes a los pines que desean conectarse, aceptan cablearse como se describió en la sección 3.4.5. Además, chequea que no se forme un lazo de realimentación cableado.

Ejemplo de Conexión entre un Pin de Salida de un Componente "Bloque Lógico OR" y un Pin de Entrada de un Componente "Bloque Lógico AND"

En la red de bloques conectables de la figura [3.73] se desea conectar los pines indicados.

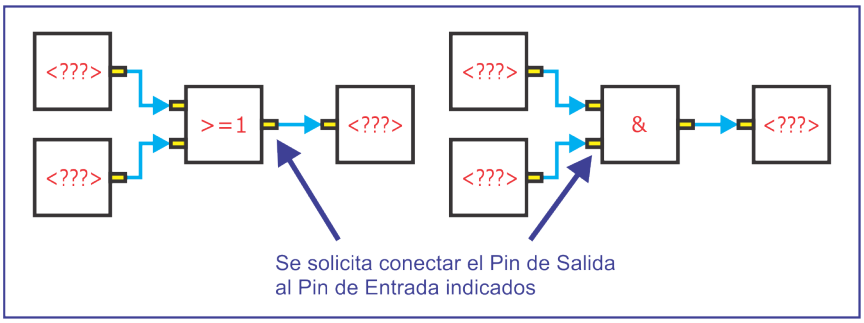


Figura 3.73: Ejemplo de Conexión entre un Pin de Salida de un Componente "Bloque Lógico OR" y un Pin de Entrada de un Componente "Bloque Lógico AND" indicando qué se debe conectar.

Antes de conectarlos, verifica si estos pines admiten ser cableados. En este caso ambos son pines booleanos que admiten conexión. Luego, chequea que al realizar la conexión no se forme un lazo de realimentación. En este caso también se cumple. Como superó ambos chequeos, ahora puede conectarlos. Para Realizar la conexión, elimina el Escritor de Argumento del Bloque Lógico OR y el segundo Lector de Argumento del Bloque Lógico AND. Luego conecta los pines resultando en la red de bloques conectables de la figura [3.74].

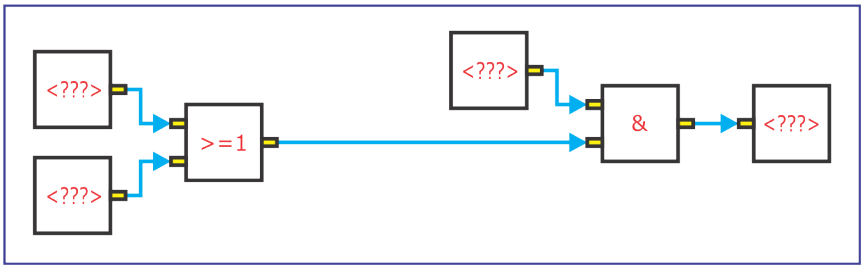


Figura 3.74: Ejemplo de Conexión entre un Pin de Salida de un Componente "Bloque Lógico OR" y un Pin de Entrada de un Componente "Bloque Lógico AND".

[6]Se llama elemento principal del Componente al elemento que contiene su funcionalidad, por ejemplo, de un Componente Llamado a Función el elemento Llamado a Función, y no sus Lectores o Escritores de Argumento.

Conexión de un Pin de Entrada hacia una Conexión previa entre dos pines

En la figura [3.75] se muestra, por ejemplo, una red de bloques conectables indicando cuál es la conexión y el Pin de Entrada que desean conectarse.

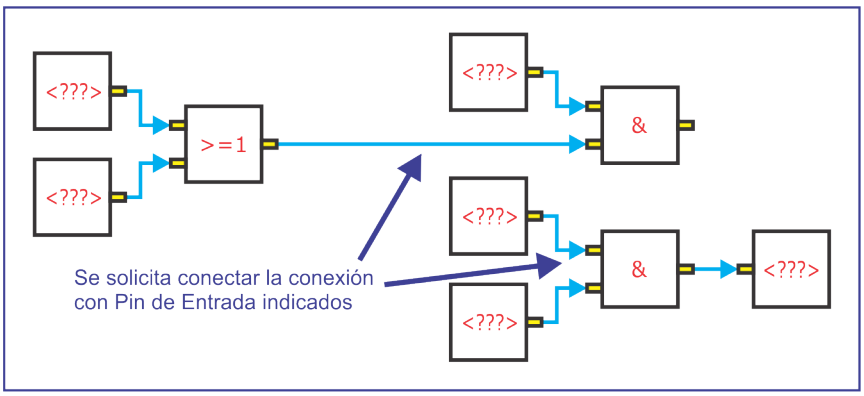


Figura 3.75: Red de bloques conectables FBD donde desea conectarse un Pin de Entrada con una Conexión previa entre dos pines.

Para realizarlo, examina si el Pin de Entrada acepta cablearse, y que no se forme un lazo de realimentación al igual que el caso anterior. Si esto se cumple, debe crear y agregar un Componente "Nodo[7]", con una entrada, y dos salidas. Dicho Nodo, conecta su primer Pin de Entrada y su primer Pin de Salida, a los pines donde se encontraba la conexión, e su segundo Pin de Salida al pin de Entrada seleccionado. El resultado se expone en la figura [3.76].

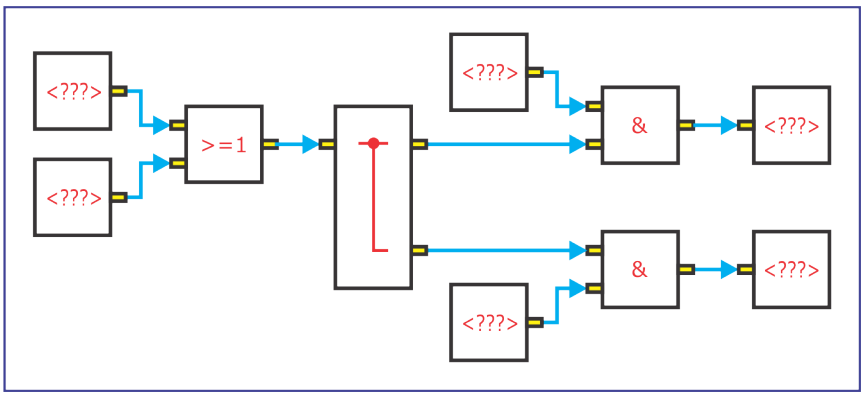


Figura 3.76: Red de bloques conectables FBD con un Nodo.

Eliminación de componentes

En FBD puede eliminarse cualquier Componente, incluso Nodos.

Cuando se elimina un Componente distinto de Nodo, debe chequearse si se encontraba conectado a un Nodo; si es así, borra el Pin de Salida o Entrada del Nodo según corresponda y realiza verificación que quede un Nodo coherente de la misma forma que lo realizaba en Ladder con el Enlace Vertical. Además, agrega Lectores y Escritores de Argumento a otros componentes diferentes de Nodos donde se ha borrardo el componente.

Si el componente a borrar se trata de un Nodo, borra el Componente y agrega a cada conexión entrante un Escritor de Argumento y Cada Salida un Lector de Argumento.

[7]Este Nodo tiene características similares al Enlace Vertical de Ladder, pero no puede tener más de una Entrada.

3.4.8.4. Generación de código de un Segmento gráfico

Cuando un segmento gráfico debe generar código, se utiliza como paso intermedio construir: Instrucciones IL, Declaraciones de Variables, Llamados a POU, etc.; aprovechando así, que los mismos saben generar código tanto en lenguaje IL como C. De esta forma, no sólo es posible crear código C a partir de un segmento gráfico, sino que también puede obtenerse el modelo en lenguaje IL equivalente, permitiendo la conversión de un segmento en lenguaje gráfico Ladder o FBD a un segmento IL.

Para poder generar el modelo IL a partir del modelo gráfico, se debe recorrer la *red de bloques conectables* que conforma el segmento gráfico inteligentemente, para lograr una lista secuencial de Instrucciones IL, Llamados a Funciones y Bloques de Función que mantenga la semántica del modelo gráfico. La información de cómo, recorrer la red está contenida en los *elementos de programa gráfico*.

Generación de código de un Segmento Ladder

En este lenguaje, cuando el segmento recibe el pedido de generar código por parte del Cuerpo de POU que lo contiene, enviándole como parámetro un "Acumulador de Programa" como ha sido anticipado; el segmento mismo busca su Elemento Gráfico "Barra de alimentación izquierda" dentro de su red de bloques conectables. Una vez hallado le envía la misma petición de generar código en su acumulador a dicho elemento. Este elemento envía la petición a cada elemento conectado a cada una de sus salidas, a través de las conexiones de los bloques conectables, y así sucesivamente hasta cubrir toda la red. Cada elemento es responsable de rellenar este acumulador de programa, decidir cuándo terminó de generar su parte y a qué salida rutear la petición de generación de código en un acumulador. Una regla general para todos los elementos, es que debe estar generado previamente el código de todos los elementos cableados a sus entradas, esto es, todos los elementos a excepción de "Lectores de Argumento".

Ejemplo de Generación de código de un Segmento Ladder

Utilizando la red de bloques conectables del ejemplo de Segmento Ladder de la figura [3.61] que se muestra en la figura [3.77] se expone cómo genera código este segmento.

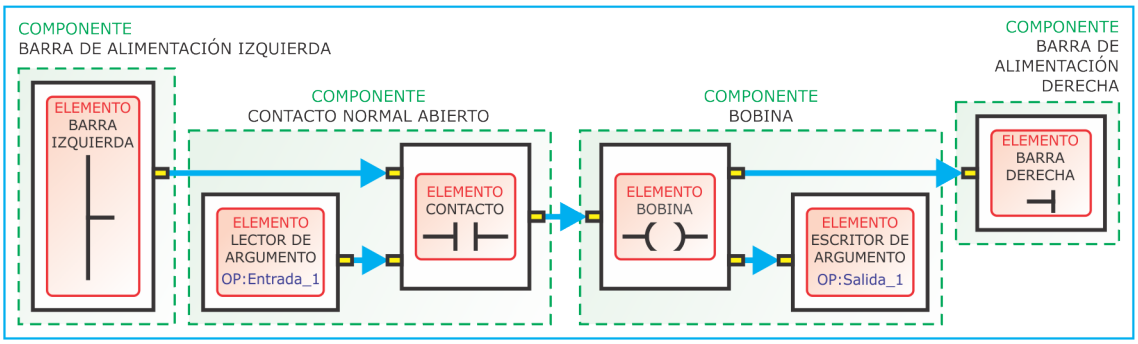


Figura 3.77: Red de bloques conectables de un Segmento Ladder.

1. Como se ha indicado, en todo Segmento Ladder la generación de código comienza con el Cuerpo de POU enviándole el mensaje 'generáTuCódigoEn(AcumuladorDePrograma)' al Segmento Ladder.

2. El segmento busca su elemento gráfico "Barra de alimentación izquierda" (el cual es único en cada segmento Ladder) y su bloque conectable. Le envía el mensaje 'generáTuCódigoEn (

AcumuladorDePrograma, BC_BAI)' al elemento encontrado, donde el parámetro 'BC_BAI' es el bloque conectable de la Barra de alimentación izquierda[8].

3. Al recibir el mensaje, la Barra de alimentación izquierda lo reenvía al elemento de cada bloque conectable, conectado a sus salidas buscando cada bloque conectable y elemento conectado a las mismas. En este ejemplo posee sólo una así que simplemente le envía el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_CONTACTO)' al "Contacto Normal Abierto" conectado a su salida.

4. El contacto normal abierto agrega una 'Instrucción de carga IL' al acumulador de programa, obteniendo de su segunda entrada el Operando desde el "Lector de Argumento", que tiene el Operando Variable simbólica 'Entrada_1'. Como resultado genera la instrucción 'LD Entrada_1'. Luego, busca el bloque conectable y elemento conectado a su única salida y le envía el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_BOBINA)' a la "Bobina" conectada a su salida.

5. La bobina agrega una 'Instrucción de almacenamiento IL' al acumulador de programa, obteniendo de su segunda salida el Operando desde el "Escritor de Argumento", que tiene el Operando Variable simbólica 'Salida_1'. Como resultado genera la instrucción 'ST Salida_1'. Luego, busca el bloque conectable y elemento conectado a su única salida y envía el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_BAD)' a la "Barra de Alimentación Derecha" conectada a su salida.

6. La barra de alimentación derecha, es un elemento que simplemente termina la generación de código, dándole la oportunidad a otro elemento que tenga más de una salida seguir generando código.

7. Finalmente, dentro del acumulador de programa quedan las instrucciones:

```
1   LD Entrada_1
2   ST Salida_1
```

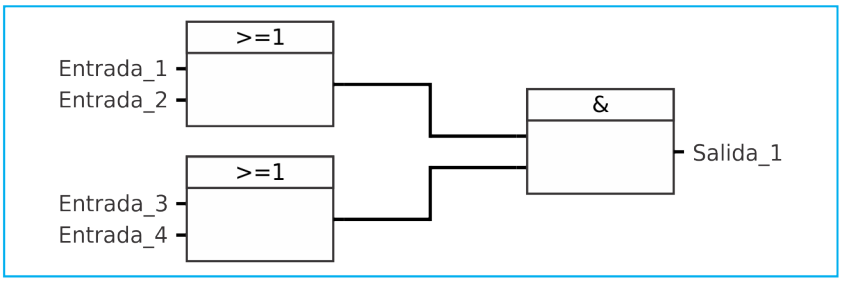
Generación de código de un Segmento FBD

La estrategia de generación de código es muy parecida al caso de Ladder. La única diferencia radica en los elementos buscados para iniciar la generación de código. En este caso se buscan todos los elementos "Lectores de Argumentos" y les envía a cada uno la petición de generación de código para comenzar con la misma. Los elementos terminadores son los "Escritores de Argumentos"

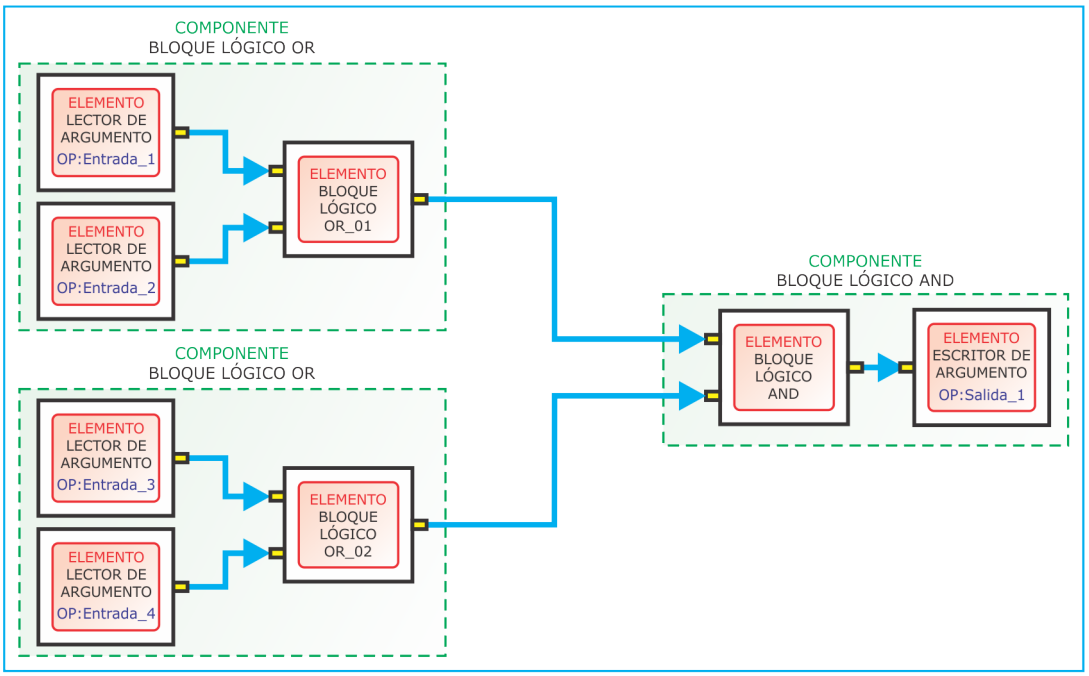
Ejemplo de Generación de código de un Segmento FBD

Como ejemplo se utiliza la red de bloques conectables de la figura [3.78] se expone cómo genera código este segmento.

[8] Esto es necesario pues los elementos gráficos no tienen un atributo de su respectivo bloque conectable, sino que lo reciben como parámetro. De esta manera, pueden existir elementos únicos en el sistema reduciendo la utilización de recursos los cuales se encuentran referenciados por uno o más bloques conectables.



(a) Segmento FBD.



(b) Red de bloques conectables con nombres de Componentes y Elementos de programa gráfico destacados.

Figura 3.78: Ejemplo de Red de bloques conectables de un Segmento FBD.

1. Al igual que en Ladder, en todo Segmento FBD la generación de código comienza con el Cuerpo de POU enviándole el mensaje 'generáTuCódigoEn(AcumuladorDePrograma)' al Segmento FBD.

2. El segmento busca todos los elementos "Lectores de Argumentos", en este ejemplo, encuentra cuatro elementos y sus respectivos bloques conectables. Comienza por el primero enviándole el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_Entrada_1)' al lector de argumento con Operando Variable Simbólica 'Entrada_1'.

3. Al recibir este mensaje un lector de argumento, establece a verdadero el atributo 'generado', que indica que ya generó sus instrucciones y, simplemente reenvía el mensaje al elemento conectado a su única salida cambiando el parámetro por el bloque conectable de dicho elemento conectado. De esta manera, envía el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_OR_01)' al "Bloque Lógico OR" número 1.

4. Este bloque chequea cuántas Entradas 'cableadas'[9] tiene. En este caso como no tiene ninguna (son dos lectores de argumento), les pide a cada entrada su operando y genera las instrucciones:

[9] Se explicó previamente que significaba una conexión con un bloque conectable cuyo elemento no sea "Lector de Argumento"

```
LD Entrada_1
OR Entrada_2
```

Las guarda en el acumulador de programa. Como ya generó sus instrucciones, establece a verdadero el atributo 'generado' en si mismo y en el lector de argumento con Operando 'Entrada_2'. Luego, busca el elemento y bloque conectable a su salida y le envía el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_AND)' al "Bloque Lógico AND".

5. Al chequear la cantidad de entradas cableadas, el Bloque Lógico AND encuentra dos entradas. Necesita al menos almacenar los resultados de N - 1 entradas cableadas. En este ejemplo, al ser N - 1 = 1, genera una[10] Declaración de Variable Simbólica Simple 'AND_00' y la agrega al Acumulador de Programa. Genera la instrucción:

```
ST AND_00
```

Posteriormente, la guarda en el Acumulador de Programa. Como aún no tiene todos los resultados de sus entradas cableadas, detiene la generación de código.

6. El segmento, envía el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_Entrada_3)' al tercer lector de argumento encontrado, con Operando Variable Simbólica 'Entrada_3', pues el lector con Operando 'Entrada_2' ya generó su código.

7. El lector de argumento, establece a verdadero el atributo 'generado' en si mismo y envía el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_OR_02)' al Bloque Lógico OR número 2.

8. Este bloque genera las instrucciones:

```
LD Entrada_3
OR Entrada_4
```

Luego, las guarda en el Acumulador de Programa y establece a verdadero el atributo que indica que ya generó sus instrucciones en si mismo y el lector de argumento 'Entrada_4'. Busca el elemento y bloque conectable a su salida y le envía el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_AND)' al Bloque Lógico AND.

9. En este momento, el elemento Bloque Lógico AND ya contiene la información de todas sus entradas cableadas y genera la instrucción:

```
AND AND_00
```

Después, establece a verdadero el atributo que indica que ya generó sus instrucciones, busca el elemento y bloque conectable a su salida y le envía el mensaje 'generáTuCódigoEn(AcumuladorDePrograma, BC_Salida_1)' al "Escritor de Argumento" con Operando 'Salida_1'.

10. Este escritor genera:

```
ST Salida_1
```

Las guarda en el Acumulador de Programa, concluyendo la generación de código.

[10] Si el resultado fuese mayor a 1, generaría una declaración de Tipo de Datos Estructurado con N - 1 miembros y una Declaración de Variables de este tipo y guardaría ambos en el Acumulador de Programa.

11. Finalmente, en el Acumulador de Programa quedan las instrucciones:

```
1    LD Entrada_1
2    OR Entrada_2
3    ST AND_00
4    LD Entrada_3
5    OR Entrada_4
6    AND AND_00
7    ST Salida_1
```

y la Declaración de Variable:

```
1    AND_00 : BOOL ;
```

3.4.9. Modelo de Recurso

Un "Recurso", según la norma IEC 61131-3, modela una Unidad Central de Procesamiento (CPU) con un Sistema Operativo listo para ejecutar tareas y programas de un Controlador Programable. Por este motivo, existe un recurso por cada dispositivo CPU de un PLC.

El sistema operativo controla la ejecución de programas creados por el usuario mediante la planificación de tareas, las cuales se encuentran asociadas con dichos programas. La norma IEC 61131-3 además, propone como esquemas de planificación de tareas, un modo cooperativo y uno apropiativo.

El modelo de Recurso se muestra en la figura [3.79].

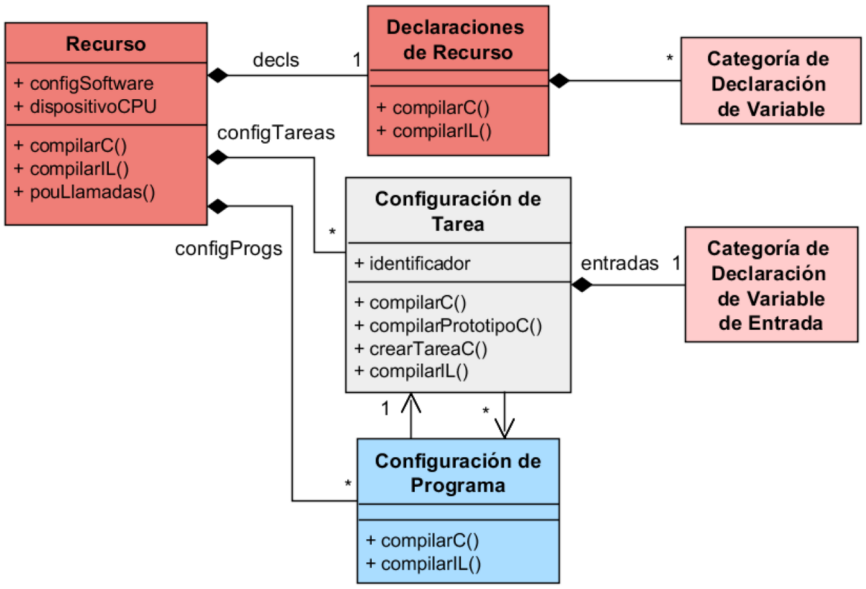


Figura 3.79: Modelo de Recurso.

El Recurso está formado por "Declaraciones de Recurso" y uno o varias "Configuraciones de Tarea". Además contiene "Configuraciones de Programa" las cuales fueron definidas en la sección 3.4.7.2.

Las declaraciones de recurso están formadas por las categorías de declaraciones de variables "Globales", "Externas" y "Acceso".

Una configuración de tarea, contiene un identificador único y una categoría de variable de "Entrada"(definida en la sección 3.4.5), donde se incluyen tres declaraciones de variables correspondientes a los parámetros de control de ejecución de tarea (SINGLE, INTERVAL y PRIORITY). Posee asociaciones a una o más POU del tipo Programa, señalando de esta manera, los Programas que le corresponde ejecutar a la tarea.

Cada configuración de tarea es capaz de generar el código de configuración de tarea en lenguaje IL, la definición de Función de tarea en C que implementa la Configuración de Tarea, su prototipo de función, y la llamada a la API de creación de la tarea en el sistema operativo.

Ejemplo de generación de código en una Configuración de Tarea

En este ejemplo, la Configuración de Tarea tiene asociada a la POU del tipo Programa 'ProgEj' utilizada en el ejemplo de la sección 3.4.6. El código C generado podría variar ligeramente según la implementación. En particular, este ejemplo utiliza tareas y funciones de la API del Sistema Operativo freeRTOS, el cual se ha elegido para la implementación como se expone en el capítulo 4.

- Configuración de Tarea en lenguaje IL:

```
TASK TareaEj(INTERVAL := T#50ms, PRIORITY := 1) ;
```

- Definición de Función de tarea en C que implementa la Configuración de Tarea:

```
void vPeriodicTask_TareaEj(void *pvParameters)
{
  portTickType xLastWakeTime;
  xLastWakeTime = xTaskGetTickCount();
  // xTaskGetTickCount() es la función API del S.O.
  // freeRTOS para obtener el conteo de Ticks del Sistema.

  /* DECLARACIÓN DE INSTANCIAS DE LOS PROGRAMAS ASOCIADOS
     PLC_P_Struct_ProgEj es un tipo de datos estructurado
     que contiene todas las declaraciones de variables de
     ProgEj como miembros */
  PLC_P_Struct_ProgEj ProgEj_Instance;

  /* Cada tarea del S.O. contiene un bucle infinito */
  for( ;; )
  {
    /* Lectura de las entradas físicas */
    entradas = Read_IN();
    I0 = Convert_Uint8_t_2_PLC_Digital_PORT_8(entradas);

    /* LLAMADOS A PROGRAMAS ASOCIADOS A LA TAREA PLC */
    // Lectura de parámetros de entrada
    ProgEj_Instance.Ent1 = I0.b0;
    ProgEj_Instance.Ent2 = I0.b1;
    // Llamo al Programa
    ProgEj(&ProgEj_Instance);
    // Escritura de parámetros de salida
    Q0.b0 = ProgEj_Instance.Sal;

    /* Escritura de salidas físicas */
    salidas = Convert_PLC_Digital_PORT_8_2_Uint8_t(Q0);
    Write_OUT(salidas);

```

```
34      /* Trarea periódica cada 20ms utilizando la API
35         del S.O. freeRTOS de Dealay Periódico. */
36      vTaskDelayUntil(&xLastWakeTime, (20/portTICK_RATE_MS));
37    }
38  }
```

- Prototipo de Función de tarea en C que implementa la Configuración de Tarea:

```
1  void vPeriodicTask_TareaEj(void *)
```

- Llamada a la función API de creación de la tarea en el sistema operativo:

```
1  /* Creación una instancia de la tarea periódica con prioridad 2 en el
       Sistema Operativo de Tiempo Real freeRTOS. */
2
3  xTaskCreate( vPeriodicTask_TareaEj, "Tarea periódica TareaEj", 512,
       NULL, 1, NULL );
```

Un Recurso es capaz de generar código en lenguaje IL agregando el código generado por sus Declaraciones de Recurso, Configuraciones de Tareas y Configuraciones de Programas. Además puede retornar el código correspondiente a sus Declaraciones de Recurso en lenguaje C. A continuación se expone un ejemplo de generación de código IL en un Recurso.

Ejemplo de generación de código IL en un Recurso

Para ejemplificar esta generación de código IL, se utiliza un Recurso que contiene la Configuración de Tarea del ejemplo anterior, y la POU del tipo Programa 'ProgEj' (sección 3.4.6).

- Recurso en lenguaje IL:

```
1  RESOURCE RecursoEj ON ProcesadorEj
2
3    (* Declaraciones de Recurso *)
4    VAR_GLOBAL
5      AT %I0.0 : BOOL ;
6      AT %I0.1 : BOOL ;
7      AT %Q0.0 : BOOL ;
8    END_VAR
9
10   (* Configuraciónes de Tareas *)
11   TASK TareaEj(INTERVAL := T#50ms, PRIORITY := 1) ;
12
13   (* Configuraciónes de Programas *)
14   PROGRAM ProgEj WITH TareaEj :
15         ProgEj_Instance
16         (
17           Ent1 :=  %I0.0;
18           Ent2 :=  %I0.1;
19           Sal  =>  %Q0.0;
20         ) ;
21
22 END_RESOURCE
```

3.4.10. Modelo de Proyecto y Configuraciones

Esta parte del modelo corresponde a la parte externa del Editor de Programas. Los componentes del modelo y sus relaciones pueden apreciarse en la figura [3.80].

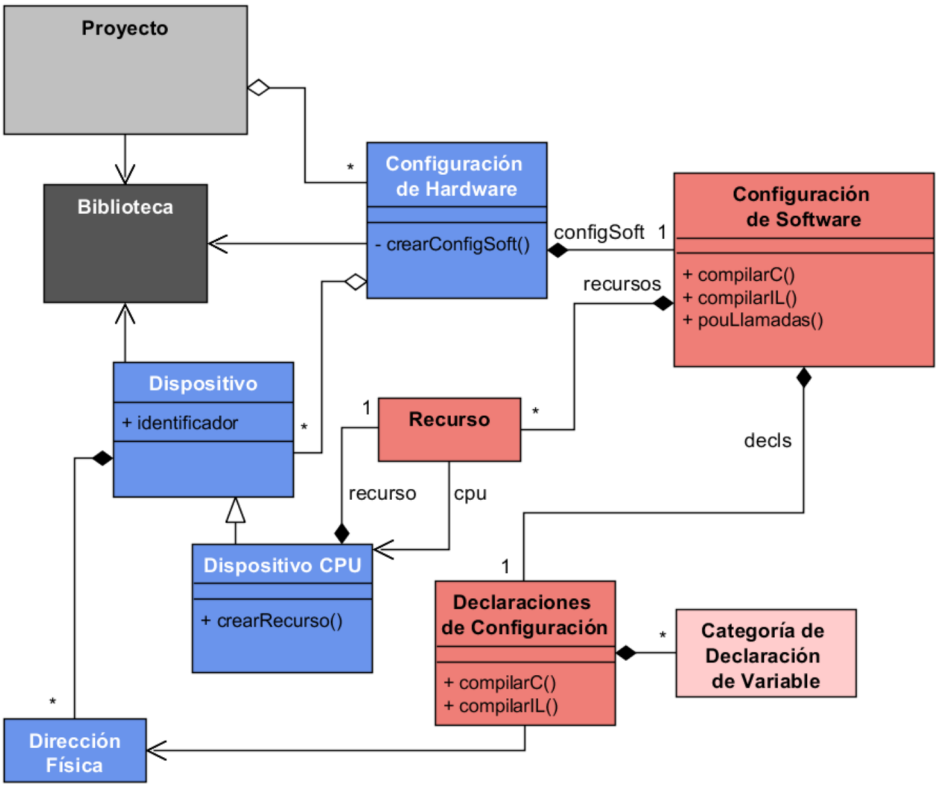


Figura 3.80: Modelo de Proyecto.

Un "Proyecto" modela la descripción completa de software y hardware que conforman un Controlador Programable. Contiene varias "Configuraciones de Hardware" y conoce a la "Biblioteca" en la que se agrupan los distintos elementos que componen el Editor.

La configuración de Hardware modela un equipo físico Controlador Programable completo. La misma contiene uno o más "Dispositivos", que se corresponden con un equipo físico el cual forma parte del Controlador Programable (por ejemplo, un módulo de entradas y salidas , una CPU o una fuente de alimentación de PLC); y una "Configuración de Software" que modela el concepto de "Configuración" definido en la norma IEC 61131-3. Esta configuración incluye todo el software que se ejecutará en los dispositivos. Una configuración de hardware crea su configuración de software utilizando información contenida en sus dispositivos. Si se agregan o eliminan dispositivos en la configuración de hardware, entonces la misma modificará su configuración de software para mantener la coherencia del modelo.

La configuración de software está compuesta principalmente por varios "Recursos", que tienen correspondencia de uno a uno con los dispositivos del tipo CPU, y "Declaraciones de Configuración". Las declaraciones de configuración se forman a partir de "Categorías de Declaraciones de Variables" las cuales agrupan "Declaraciones de Variables". Las categorías admitidas por esta configuración son: "Global", "Acceso" y "Configuración" (detalladas en la sección 3.4.5).

Una Configuración de Software, al igual que el Recurso, puede generar el código correspondiente a sus Declaraciones de Configuración en lenguaje C, y su código completo en lenguaje IL, agregando el código IL generado por cada uno de sus recursos, al código generado por sus Declaraciones de Configuración.

Ejemplo de generación de código IL en una Configuración

Esta configuración utiliza al Recurso del ejemplo anterior.

- Recurso en lenguaje IL:

```
CONFIGURATION ConfiguracionEj

  (* Declaraciones de Configuración *)

    VAR_GLOBAL
      AT %M0.0 : BOOL ;
    END_VAR


  (* Recursos *)

  RESOURCE RecursoEj ON ProcesadorEj

    (* Declaraciones de Recurso *)
    VAR_GLOBAL
      AT %I0.0 : BOOL ;
      AT %I0.1 : BOOL ;
      AT %Q0.0 : BOOL ;
    END_VAR

    (* Configuraciónes de Tareas *)
    TASK TareaEj(INTERVAL := T#50ms, PRIORITY := 1) ;

    (* Configuraciónes de Programas *)
    PROGRAM ProgEj WITH TareaEj :
        ProgEj_Instance
        (
          Ent1 :=  %I0.0;
          Ent2 :=  %I0.1;
          Sal  =>  %Q0.0;
        ) ;

  END_RESOURCE

 END_CONFIGURATION
```

3.4.11. Modelo de Generación de archivos, compilación y grabación del programa

En esta sección se describe la parte del modelo computacional responsable de la generación de los archivos en lenguaje C que implementan la Configuración de Software, su compilación a código ejecutable y su grabación en la memoria del Hardware PLC. El mismo se presenta en la figura [3.81]

En este modelo, existe un "Compilador y Grabador" el cual conoce al "Sistema de Archivos", que encapsula la funcionalidad tanto de direccionamiento como lectura y escritura de archivos, y al "Proceso del Sistema Operativo", que es capaz de ejecutar comandos en el Sistema Operativo donde corre el Editor de Programas.

Este Compilador y Grabador es capaz de generar código en lenguajes IL y C de una Configuración de Software recibida como parámetro.

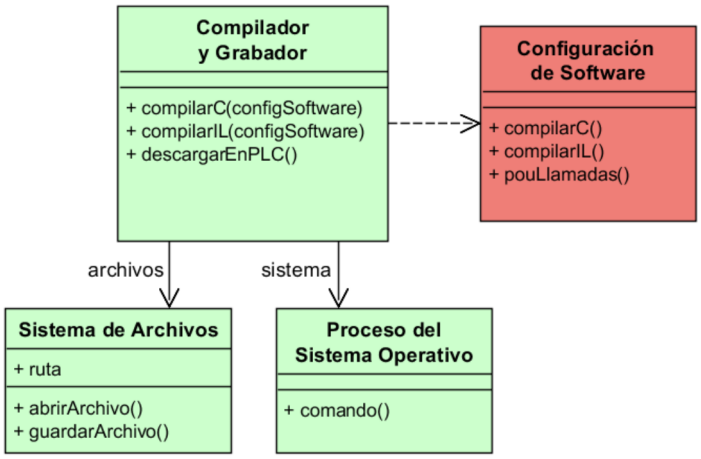


Figura 3.81: Modelo de Generación de archivos, compilación y grabación del programa.

Para generar código en lenguaje IL, simplemente solicita a dicha configuración que compile a IL, generando su código como se expuso en la sección 3.4.10.

En lenguaje C, en cambio, el Compilador y Grabador debe explorar la Configuración de Software, solicitando el código C de cada parte contenida en la misma, con el fin de ir recopilando y ubicando el código en el archivo .C o .H correspondiente, utilizando el Sistema de Archivos. Recorre la Configuración llegando hasta el nivel de las POU. La responsabilidad de generar el código C de las POU recae en éstas, como ha sido descripto en la sección 3.4.6.

Los archivos que utiliza el Compilador y Grabador para volcar el Programa de usuario traducido a lenguaje C son:

- main.c
- tareasDeUsuario.c
- tareasDeUsuario.h
- programasDeUsuario.c
- programasDeUsuario.h
- bloquesDeFuncionDeUsuario.c
- bloquesDeFuncionDeUsuario.h
- funcionesDeUsuario.c
- funcionesDeUsuario.h
- tiposDeDatosDefinidosPorElUsuario.h

En la figura [3.82] se muestra como encajan los mismos en el modelo de Software de PLC.

El Programa de usuario, definido en la sección 3.1.1, corresponde entonces a una determinada Configuración de Software implementada en lenguaje C.

Además, el Compilador y Grabador es capaz de invocar la compilación del Programa completo en lenguaje C, que incluye los archivos generados por el editor y archivos fijos de funcionalidad básica, como se describe en la sección 3.1. Para realizarlo, utiliza un programa compilador de C

desarrollado por terceros, a través de comandos enviados al Sistema Operativo con su Proceso del Sistema Operativo. De esta manera genera el código ejecutable del Programa completo del PLC.

Finalmente, puede invocar también mediante comandos, a otro programa externo cuyo fin es la comunicación con el Hardware PLC y descarga del código ejecutable a la memoria de del dispositivo.

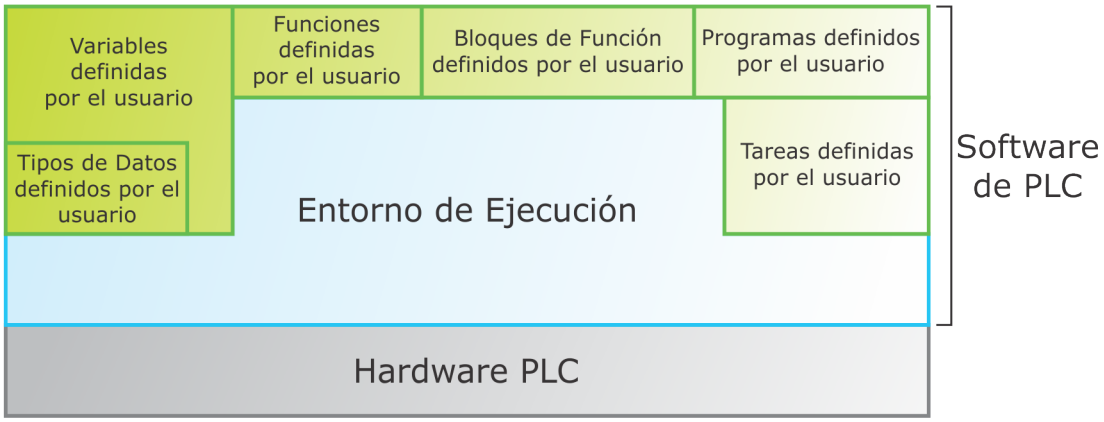


Figura 3.82: Modelo de Software de PLC con detalle de componentes pertenecientes al Programa de usuario.

3.5. Entorno de Software de Ejecución

En esta sección se expone Entorno de Software de Ejecución completando el modelo de Software de PLC. El entorno está formado por los siguientes componentes de Software:

- **Tipos de Datos estándar.** Contiene las definiciones de todos los tipos de datos Elementales establecidos por la norma IEC 61131-3.
- **Instrucciones IL.** Está formado por la implementación de todas las instrucciones del lenguaje IL, implementadas como funciones en lenguaje C.
- **Registros.** Contiene definiciones de los registros de Entradas (I) y salidas (Q) del PLC y el registro acumulador "Resultado Actual[11]" del lenguaje IL .
- **Funciones estándar.** Contiene las definiciones de funciones en C que implementan las Funciones estándar y las definiciones de Estructuras de datos de las mismas.
- **Bloques de Función estándar.** Contiene las definiciones de funciones en C que implementan los Bloques de Función estándar y las definiciones de Estructuras de datos de las mismas.
- **Sistema Operativo y Drivers.** Aquí se agrupa un sistema Operativo de Tiempo Real (RTOS) y los Drivers encargados de controlar a bajo nivel los distintos periféricos de Hardware PLC. Ambos deben estar diseñados de forma tal que se cree una capa de abstracción entre el resto de los componentes de Software y el Hardware subyacente, permitiendo aislar el Software dependiente del Hardware.

En consecuencia, el modelo completo de Software de PLC resulta en esquema de la figura [3.83].

Cada uno de estos componentes está implementado en uno o más archivos .C y .H. De estos componentes de software se han diseñado todos excepto el Sistema Operativo y los Drivers.

El sistema Sistema Operativo requerido debe cumplir los siguientes requisitos:

[11] Llamado CR por sus siglas en inglés.

- Debe ser un Sistema Operativo de Tiempo Real (RTOS). Esto es condición necesaria para asegurar que se cumplan las tareas de usuario en tiempos establecidos.
- Ofrezca herramientas de temporización, comunicación entre tareas, sincronización entre tareas e interrupciones, definición de secciones críticas.
- Permita planificar tareas de forma cooperativa o apropiativa conforme a la norma IEC 61131-3.
- Sea fácilmente portable a distintas arquitecturas de Hardware PLC.
- Debe estar Escrito en lenguaje C, salvo las partes específicas de cada máquina, para poder lograr que brinde una interfaz estándar con las funciones específicas del PLC adaptándose a sus necesidades.

Las Tareas definidas por el usuario son Tareas del Sistema Operativo, las cuales gestiona el planificador de este sistema, y por lo tanto, tienen comunicación directa con el Sistema Operativo, mediante llamadas a funciones de la API de este sistema.

Entre los Registros a diseñar se encuentra el acumulador "Resultado Actual" de lenguaje IL que es utilizado por todas las Instrucciones IL, Funciones y Bloques de Función. Este acumulador, es llamado "acumulador universal" pues puede contener un valor de cualquier tipo de datos. Debe ser fuertemente tipificado para lograr detección de errores, tanto en tiempo de compilación como de ejecución. En consecuencia su implementación propone un desafío interesante.

Otro desafío, es dar soporte a la sobrecarga de tipos de datos tanto en las instrucciones IL como en las Funciones y Bloques de Función estándar.

Finalmente, cabe destacar que los bloques de Función estándar Temporizadores (TP, TON y TOF), requieren obtener el tiempo absoluto mediante una llamada a la API del Sistema Operativo.

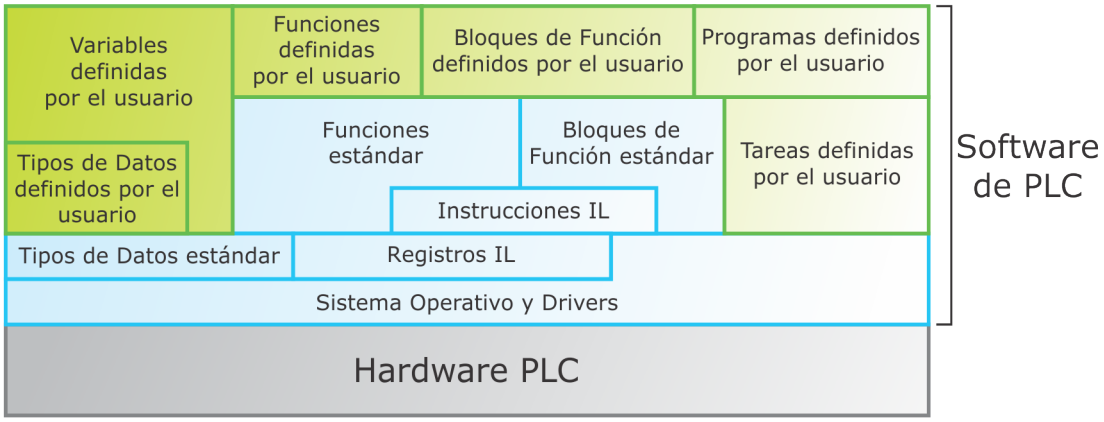


Figura 3.83: Modelo completo de Software de PLC.

3.6. Hardware PLC

Como se expuso en la sección 3.1.1 se diseña un micro PLC compacto. El modelo de Hardware propuesto se expone en la figura [3.84].

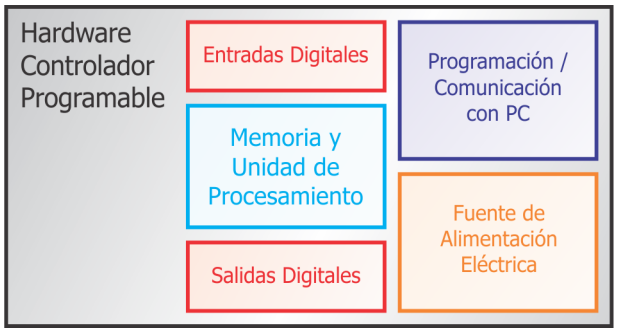


Figura 3.84: Modelo de Hardware Propuesto.

Este modelo respeta el modelo de Hardware contenido en la norma IEC61131-1. En la figura [3.84] se observa que el modelo está compuesto por:

- Memoria y Unidad de Procesamiento
- Entradas Digitales
- Salidas Digitales
- Programación / Comunicación con PC
- Fuente de Alimentación Eléctrica

En las siguientes secciones se describe cada una de estas partes.

NOTA: No se han diseñado Entradas o Salidas analógicas o comunicaciones industriales, como por ejemplo RS 485, dejando estos para futuros desarrollos.

3.6.1. Memoria y Unidad de Procesamiento

Como bloque de Unidad de Procesamiento y Memoria se propone la utilización de un Microcontrolador. Otras alternativas podrían ser un DSP, un Microprocesador y Memorias o una FPGA. La selección de un Microcontrolador se debe a las siguientes características:

- **CPU:** En la actualidad las características de una CPU de los Microcontroladores han dado un importante salto utilizando arquitecturas de 32 bits y frecuencias de trabajo entre 100 MHz y 1 GHz, permitiendo su utilización en aplicaciones más complejas.
- **Memorias:** Incluye memorias RAM, FLASH y EEPROM en el mismo circuito integrado.
- **Periféricos:** Incluye varios periféricos integrados, por ejemplo, Conversor Analógico Digital, Temporizador, Ethernet, SPI, UART, CAN, USB, etc.
- **Velocidad de Operación:** La velocidad de operación de los Microcontroladores es más que suficiente para las aplicaciones de control industrial y en particular para el PLC diseñado. Cabe aclarar, que si se compara con un Mircoporcesador, su velocidad es mucho menor. Además, los Microcontroladores carecen, en general, de Unidad de Punto Flotante para realizar cálculos, efectuándolos mediante la utilización de bibliotecas de software.

- **Tamaño:** Como se ha anticipado, el Microcontrolador incluye memorias y periféricos en un único Circuito Integrado. Esto implica una gran ventaja en varios factores como por ejemplo, la disminución en el tamaño del circuito impreso por la reducción de los circuitos externos.
- **Costos:** El costo para un sistema basado en Microcontrolador es mucho menor en comparación a un Microprocesador, y es menor a los de dispositivos DSP o FPGA.
- **Interferencias:** El alto nivel de integración reduce los niveles de interferencia electromagnética.
- **Tiempo de desarrollo:** El tiempo de desarrollo de un sistema basado en Microcontrolador es veloz, debido a la amplia documentación disponible por parte de los fabricantes y grupos de desarrollo en Internet. Además, los mismos fabricantes ofrecen kits de desarrollo de hardware y su correspondiente IDE de Programación permitiendo la creación de prototipos rápidos mediante la programación de los mismos.

3.6.2. Programación / Comunicación con PC

Se determina la utilización de un puerto de comunicación USB, como interfaz de comunicación con la PC, para la programación del PLC. Esta decisión se basa en la gran popularidad que los puertos USB gozan en la actualidad. Este puerto de comunicación se encuentra hoy en día en cualquier computadora, smartphone o tablet. Si bien, el diseño se centra en la utilización con una computadora para la programación del PLC, se prevee en el diseño su futura implementación en otros dispositivos.

3.6.3. Entradas Digitales

El modelo general de una Entrada Digital (o booleana) se expone en la figura [3.85].

Figura 3.85: Modelo de Entradas Digitales.

La señal proveniente del exterior es rectificada en una primer etapa. En segundo lugar se acondiciona la señal eléctricamente. En tercer lugar, se muestra el estado lógico de la señal al usuario (normalmente a través de un diodo LED). En cuarto lugar, se aísla galvánicamente[12] el circuito de potencia externo del circuito lógico interno. Finalmente se introduce la señal en el circuito lógico de Entrada.

En este Trabajo final se elige que el PLC cuente con ocho Entradas Digitales de 24 VDC[13], tipo 1 (IEC61131-2) sumidero de corriente (sinking). Este tipo de entradas son muy utilizadas en la automatización industrial existiendo muchos tipos de sensores con salidas compatibles a las mismas. Un sensor compatible, se debe conectar el terminal positivo de una fuente de 24 VDC y el terminal de entrada del PLC. En el capítulo 4 se expone un ejemplo de circuito eléctrico que responde a este modelo de Hardware.

[12] El aislamiento galvánico consiste en la separación de partes funcionales de un circuito eléctrico para prevenir el traspaso de portadores de carga. Este tipo de aislamiento se utiliza cuando se desea que se transmitan señales entre las distintas partes funcionales, pero las masas tienen que mantenerse separadas. Este aislamiento entre las masas o tierra se hace por motivos de seguridad.

[13] VDC siglas en inglés de Volts de Corriente Continua.

3.6.4. Salidas Digitales

El modelo general de una Salida Digital se muestra en la figura [3.86].

Figura 3.86: Modelo de Salidas Digitales.

En primer lugar, existe un circuito lógico de salida que proporciona un nivel de tensión correspondiente al nivel lógico de la misma. Seguidamente la señal se aísla galvánicamente de la etapa de potencia de salida. Luego se indica al usuario el estado de la misma. Finalmente, se inyecta en un circuito de conexión, que posee a su salida algún tipo de protección eléctrica.

Se elige que el PLC posea dos grupos de Salidas Digitales, el primer grupo corresponde a cuatro Salidas Digitales de 24 VDC sumidero (sinking) a transistor NPN, la carga se conecta entre el terminal positivo de la fuente de 24 VDC y el terminal de salida del PLC; mientras que el segundo grupo corresponde a cuatro Salidas a Relé de contacto seco, la carga se conecta entre la salida del PLC, y el neutro de una fuente de alimentación de tensión alterna; formando un total de ocho salidas. De esta manera, se soporta un amplio rango de tipos de cargas de actuadores conectados a las salidas del PLC. Al igual que para las entradas, en el capítulo 4 se muestran ejemplos de circuitos eléctricos que responden al modelo de Salidas Digitales para ambos tipos concretos de salidas.

Características de las salidas de corriente continua comparadas con Salidas a Relé

- Mayor velocidad de respuesta.
- Ausencia de desgaste mecánico.
- Más compactas.
- Permiten protección contra cortocircuitos.
- Sin protección contra cortocircuitos son más sensibles a picos de corriente de carga.
- Menor capacidad de carga a la salida.
- Caída de tensión en el transistor de salida, es mayor a la que produce el contacto de un relé.
- Corriente de fuga pequeña en estado “0”.

3.6.5. Fuente de Alimentación Eléctrica

Como tensión de alimentación del PLC se elige una alimentación de tensión de 24 VDC. De esta manera, se simplifica la alimentación del circuito de Salidas Digitales debido a que utiliza directamente el nivel de tensión de su entrada. Como desventaja, se requiere una Fuente de Alimentación externa que provea al PLC de 24 VDC.

El circuito de alimentación debe suministrar además, los niveles de tensión requeridos por el Microcontroaldor. Estos niveles son generalmente inferiores, situándose en el rango de 1,2 VDC a 5 VDC según el Microcontrolador. En consecuencia, se debe diseñar un circuito que realiza la transformación entre estos niveles acorde al Microcontrolador seleccionado. Un ejemplo de circuito se ofrece en el capítulo 4.

3.6.6. Capacidad de ampliación

A este diseño pueden agregarse módulos de Comunicación o Entradas Analógicas. Los requisitos solicitados para los mismos son:

- Respetar los niveles de tensión de las entradas y salidas del Microcontrolador (requerimiento de compatibilidad interna).
- Respetar niveles estándar de tensión en aplicaciones industriales (requerimiento de compatibilidad externa).
- Contener protecciones eléctricas para evitar daños al PLC.

Capítulo 4

IMPLEMENTACIÓN

Para comprobar la factibilidad del diseño que conforma este Trabajo Final, se desarrolla una implementación de referencia de los distintos aspectos, descriptos en el Capítulo 3, referidos al diseño de un PLC y de su entorno de programación.

Se implementan todas las partes que permiten una prueba completa del diseño; desde la edición de un programa en un lenguaje gráfico o textual, hasta la ejecución del programa editado en un micro PLC. En particular:

- Un entorno de programación que soporta la edición de segmentos tanto para el lenguaje de programación Ladder como para el lenguaje textual IL.

  El desarrollo de este entorno de programación está organizado de acuerdo al patrón MVC tal cual se describe en la sección 3.2, implementa el diseño de modelo computacional de un programa PLC descripto en la sección 3.4, e incluye una interfaz de usuario que respeta las indicaciones de la sección 3.3.

  Particularmente, esta implementación permite realizar todas las acciones correspondientes a la dinámica de un segmento Ladder descriptas en las secciones 3.3.2.2 y 3.4.8.2

- Se construye el Hardware completo de un micro PLC, respetando los lineamientos descriptos en la sección 3.6, y además, un sencillo entorno de prueba para ensayar programas en dicho PLC.

- Una implementación completa del entorno de Software de ejecución de un programa Ladder o IL que corre sobre el micro PLC recién descripto. Esta implementación incluye todos los componentes descriptos en la sección 3.5.

- La generación del código C correspondiente a una configuración de software creada por el usuario dentro del entorno de programación. Este código se integra con el entorno de Software de ejecución tal como se describe en la sección 3.5. También implementa la funcionalidad de compilación del Software completo y de descarga del programa generado en el micro PLC construido.

Se destacan, a continuación, los criterios que guiaron la selección de las herramientas para la implementación:

- Obtener un micro PLC a partir de piezas de hardware de bajo costo.

- Que los distintos componentes de software puedan utilizarse desde distintos tipos de PC y de sistemas operativos.

- Respetar las definiciones de la norma IEC 61131 en el entorno de programación.

En este capítulo se describe sucintamente algunos puntos relevantes de esta implementación de referencia. En las secciones 4.1 a 4.3 se detallan las herramientas de distinto tipo utilizadas en la implementación: paradigmas y lenguajes de programación, entornos de desarrollo de software, hardware del micro PLC y software de base (sistema operativo, drivers) a ejecutarse sobre el mismo. Luego se dan algunos detalles de la implementación del modelo computacional, de la interfaz de usuario del entorno de programación PLC y el hardware del mismo.

4.1. Paradigmas y lenguajes de programación utilizados

4.1.1. Programación estructurada en lenguaje C

Como ha sido expuesto en el capítulo 3, para la programación del código generado desde el Editor de Programas de PLC, Entorno de Ejecución y Sistema Operativo se ha escogido la utilización del lenguaje C. La motivación para el uso de este lenguaje frente al Ensamblador se basa fundamentalmente en la **independización del Hardware**, pues, hoy en día existen compiladores de C para la mayoría de los Microcontroladores del mercado. Otras ventajas son:

- Facilidad de Entendimiento, depuración y mantenimiento.
- Tiempo de desarrollo.
- Soporte.
- Reutilización de código (bibliotecas).
- Aprovechamiento de bibliotecas provistas por terceros.

En la actualidad C es el lenguaje común para programar sistemas embebidos (como este PLC). El código ligero que un compilador C genera, combinado con la capacidad de acceso a capas del software cercanas al hardware, son la causa de su popularidad en estas aplicaciones.

Además del lenguaje C, se elije seguir el paradigma de programación estructurada. Este paradigma da lugar a programas fiables y eficientes, escritos de manera que facilitaba su mejor comprensión. Provee ventajas durante la fase de desarrollo y también posibilita una más sencilla modificación posterior.

En particular se aprovecha la Modularización de código que impone este paradigma para organizar el programa mediante funciones y el uso exclusivo de estructuras de código secuencia, instrucción condicional e iteración con condición al principio.

4.1.2. Programación Orientada a Objetos en lenguaje Smaltalk

La POO difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, debido a que el proósito buscado es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en las estructuras de datos que esos procedimientos manejan. En la programación estructurada sólo se escriben funciones que procesan datos. Los programadores que emplean POO, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

Para el Editor de Programas de PLC, la utilización de la Programación Orientada a Objetos (POO) facilita la concepción y luego la implementación de un modelo en el cual existen muchos elementos distintos con combinaciones no triviales.

Entre los conceptos que ofrece este paradigma existen dos que son de grán importancia:

- **Polimorfismo:** Se utiliza ampliamente debido a que los lenguajes gráficos están compuestos por muchos elementos con una misma interfaz pero con distinta implementación.
- **Herencia:** Este concepto también se ha utilizado ampliamente, ya que existen muchos elementos con pequeños cambios en su funcionamiento o que agregan algún tipo de funcionalidad.

Como lenguaje POO se selecciona el lenguaje Smalltalk. Sin embargo, Smalltalk es mucho más que un lenguaje de programación. Es un sistema que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales.

Un sistema Smalltalk está compuesto por:

- Máquina virtual.
- Un archivo llamado "Imagen", que contiene a todos los objetos del sistema.
- Lenguaje de programación (también conocido como Smalltalk).
- Una enorme biblioteca de "objetos reusables".
- Entorno de desarrollo que además funciona como un sistema en tiempo de ejecución.

4.2. Entorno Pharo 2.0

Pharo es una implementación completa, moderna y de código abierto, del lenguaje y ambiente de programación Smalltalk. Se esfuerza en ofrecer una plataforma limpia y de código abierto para el desarrollo profesional de Software, y una plataforma robusta y estable para investigación y desarrollo en lenguajes y ambientes dinámicos. En la figura [4.1] se muestra una captura de pantalla del Entorno Pharo 2.0.

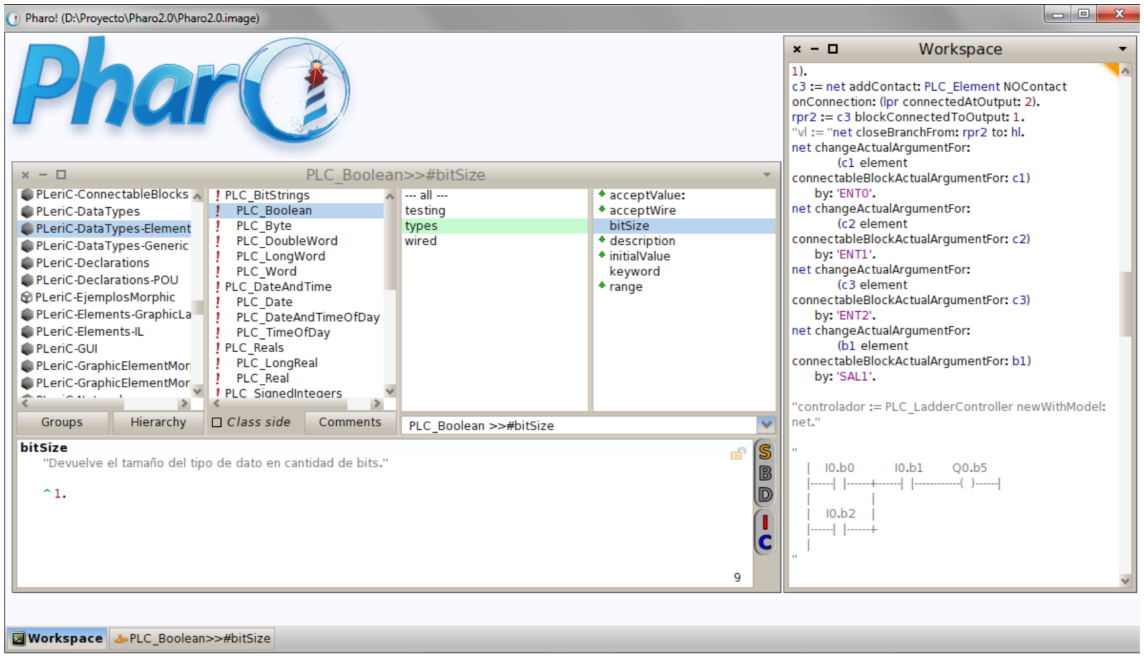


Figura 4.1: Entorno Pharo 2.0.

El núcleo de Pharo contiene sólo código que ha sido contribuido bajo licencia MIT. Tanto el software como la documentación pueden descargarse y utilizarse libremente.

Pharo es altamente portable, funciona sobre los sistemas operativos de computadoras Windows, Linux y OS X. Actualemente existen desarrollos para los sistemas operativos de Smartphones Android e iOS.

En Pharo, un programa se diseña extendiendo (mediante herencia) las clases preexistentes y creando nuevas. Todas ellas Heredan de la clase Object[1], la cual se encuentra en la raiz de la jerarquía de clases.

Para la implementación del Entorno de Programación de PLC en este Trabajo Final se utilizó en su comienzo la versión 1.4 de Pharo. Luego, se migró a la versión 2.0, que fue lanzada durante la realización del mismo.

En las siguientes secciones se detallan las características y Frameworks[2] de Pharo 2.0 utilizados.

4.2.1. Morphic

Morphic es el nombre de la interfaz gráfica de Pharo. Morphic está escrito en Smalltalk, por lo que es complementamente portable entre distintos sistemas operativos; como consecuencia de ello, Pharo se ve exactamente igual en Unix, MacOS y Windows. Lo que distingue a Morphic de la mayoría de los otros frameworks de interfaz de usuario, es que no tiene modos separados para "composición" y "ejecución" de la interfaz: todos los elementos gráficos pueden ser ensamblados o desensamblados por el usuario en cualquier momento. Los Morph son objetos capaces de componerse, formando Morph compuestos, y manejar eventos de teclado y mouse sobre ellos. La clase base de Morphic es "Morph", todos los Morph heredan de esta clase.

Morphic es la base de la implementación de la GUI del Editor de Programas de PLC, en donde cada componente gráfico es un Morph.

4.2.2. PetitParser

Un parser (o analizador sintáctico) es una de las partes constituyentes de un compilador. El análisis sintáctico convierte el texto de entrada en otras estructuras, que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada.

En el presente Trabajo Final se construye un parser para analizar tanto los Argumentos en los lenguajes gráficos como el texto en lenguaje IL ingresados por el usuario en la GUI. Como resultado del análisis se obtienen los objetos que forman parte del modelo computacional.

Para la implementación del parser se utiliza Framework PetitParser. Este Framework combina muchas ideas de diferentes tecnologías de análisis para modelar gramáticas y analizarlas como objetos que pueden ser reconfigurados dinámicamente. PetitParser hace que sea fácil de definir parsers mediante código Smalltalk y reutilizar de forma dinámica, componer, transformar y ampliar gramáticas. Las gramáticas resultantes pueden ser modificadas sobre la marcha, por lo tanto, PetitParser se ajusta a la naturaleza dinámica de Smalltalk.

4.2.3. FileStream

En Pharo, los Streams son objetos que soportan el acceso secuencial a las colecciones y los archivos de datos secuenciales.

Cuando se crea un Stream a partir de un archivo se realiza mediante el uso de la clase FileStream, que opera sobre el sistema de archivos de la plataforma. Los nombres de archivo y directorios están restringidos por el sistema operativo de la plataforma.

[1] Pharo soporta únicamente Herencia simple.

[2] Un framework o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software.

Existen tres tipos de File Stream que permiten, respectivamente:

- **ReadFileStream:**Leer archivos únicamente.
- **WriteFileStream:** Escribir archivos solamente.
- **ReadWriteFileStream:** Leer y escribir archivos.

En esta implementación se utiliza FileStream para crear y escribir los archivos .C y .H generados por el Compilador y Grabador (descripto en la sección 3.4.11).

4.2.4. OS Proces

OS Process proporciona acceso a funciones del sistema operativo, incluyendo tuberías[3] y la creación de procesos hijos. Se implementa utilizando primitivas conectables en una biblioteca compartida para Unix o Linux, y una DLL para Windows. El código Smalltalk, incluyendo las clases que conforman las primitivas conectables para funciones del sistema operativo Unix o Win32, puede ser cargado en cualquier imagen de Pharo, pero las primitivas sólo son útiles en sistemas Unix y Windows. Se imparten clases contenedoras para MacOS, OS/2 y RiscOS, pero todavía no están implementadas.

Mediante la utilización de OSProcess, en esta implementación se ejecuta un programa externo a Pharo que se encarga de la compilación de código C y grabación del código ejecutable resultante en la memoria de programa (Flash) del Microcontrolador. De esta manera se implementa el proceso descripto en la sección 3.4.11. Este programa externo es el LPCXpresso IDE & "development tools", que se detalla en la sección 4.3.1.

4.3. LPCXpresso

LPCXpresso es una cadena de herramientas completa para evaluación y desarrollo, de bajo costo, para los Microcontroladores ARM[4] Cortex-M0 y M3 de NXP.

Esta cadena de herramientas es un desarrollo conjunto de NXP (empresa fabricante de Microcontroladores), Embedded Artists (provee el Hardware completo del LPCXpresso incluyendo al Microcontrolador) y Code Red Technologies (suministra el Software, un IDE para su programación y depuración).

Esta compuesto por:

- Software LPCXpresso IDE & "development tools".
- Hardware LPCXpresso target board.

Estas herramientas son las utilizadas tanto para desarrollar el software en C del PLC, como para la implementación del Hardware. Además se utiliza el IDE como Compilador de C y Programador de PLC (definidos en la sección 3.1.2). El costo del target board en el año 2011 (cuando fue adquirido), era de u$s 25, el software correspondiente se puede descargar gratuitamente permitiendo realizar programas con un límite de 128 Kb. En consecuencia, permite utilizar solamente un cuarto de la capacidad de memoria Flash que dispone el Microcontrolador sin pagar una licencia del IDE. Sin embargo, los 128 Kb disponibles son más que suficientes para los programas testeados en esta implementación.

[3] En este contexto, una tubería (en inglés pipe) consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos.

[4] ARM es una empresa que se dedica al diseño de procesadores y la venta de licencias de fabricación de los mismos.

4.3.1. LPCXpresso IDE & "development tools"

El LPCXpresso IDE fue desarrollado por CodeRed junto a NXP. El mismo incluye un entorno de Eclipse específicamente adaptado para interactuar con el target board, un Compilador y Linkeador GNU (GCC[5]) y un depurador GDB[6]. Una captura de pantalla de la versión 5 del entorno puede apreciarse en la figura [4.2].

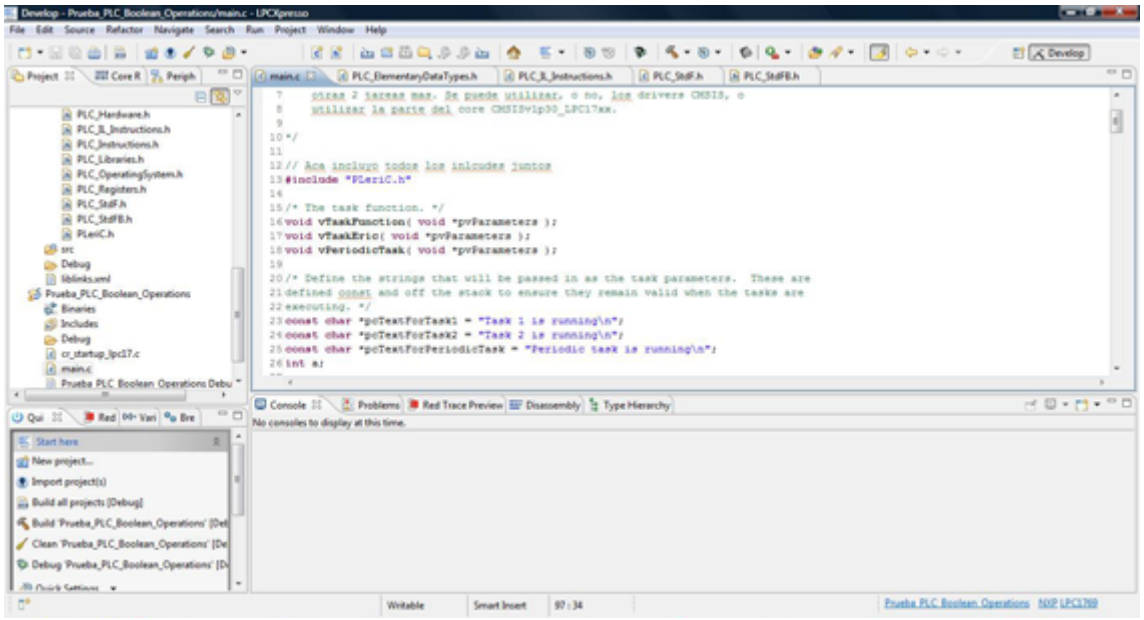


Figura 4.2: IDE CodeRed LPCxpresso v5.

Este entorno permite crear aplicaciones en lenguaje C, compilarlas, descargarlas al Microcontrolador y realizar depuración en caliente sobre la placa LPCXpresso target board.

Debido a que esta basado en Eclipse, utiliza algunos conceptos que no siempre son comunes a otros entornos de desarrollo. Se definen a continuación algunos de ellos:

- **Workspace:** Es el contenedor de los proyectos. Estos proyectos pueden ser aplicaciones y/o bibliotecas. También almacena todas las configuraciones del entorno por lo que se puede mover muy fácilmente de computadora en computadora.
- **Proyecto:** Este puede ser de dos tipos, biblioteca estática o una aplicación ejecutable. Contiene archivos de código fuente (.c), encabezados (.h) y cualquier otro archivo que se desee.

4.3.2. LPCXpresso target board

El target board es un una placa de desarrollo que incluye un Micrcocontrolador de las familias LPC1100, LPC1300 o LPC1700 junto con electrónica mínima necesaria para su funcionamiento y un programador/debugger JTAG, llamado LPC-Link. En la figura [4.3] se puede apreciar una fotografía de la placa con sus partes más destacables.

Del lado del target incluye algunos periféricos básicos. Estos corresponden a un circuito integrado que implementa la capa física de Ethernet, una memoria EEPROM de 256 KB y circuito

[5]Sigras en inglés de GNU Compiler Collection (colección de compiladores GNU). Ess un conjunto de compiladores creados por el proyecto GNU (software libre).

[6]GDB (GNU Debugger) es el depurador estándar para el compilador GNU. Es un depurador portable y funciona para varios lenguajes de programación como C, C++ y Fortran. Ofrece la posibilidad de trazar y modificar la ejecución de un programa.

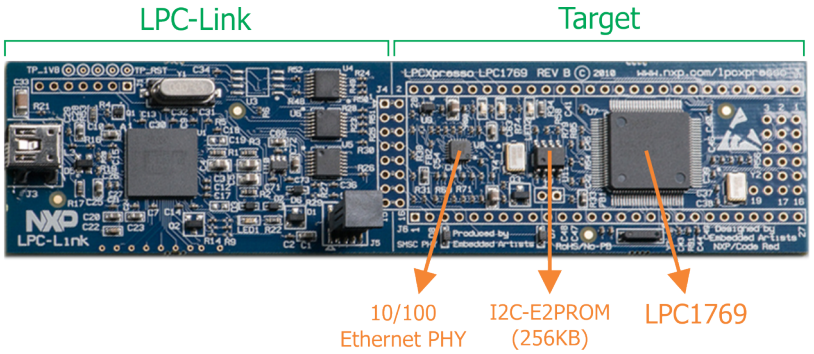


Figura 4.3: LPCXpresso target board.

oscilador. En este Trabajo Final se utiliza específicamente el target con Microcontrolador NXP LPC1769. Este Microcontrolador incluye: Núcleo ARM Cortex-M3 (hasta 120 Mhz), 512 KB Flash, 64 KB SRAM, Ethernet, USB On the go y más periféricos.

En las siguientes secciones se exponen el Sistema Operativo y Drivers usados en este Trabajo Final. Ambos están realizados por terceros e implementados como dos proyectos del tipo Biblioteca estática de LPCXpresso IDE.

4.3.3. Sistema Operativo freeRTOS

FreeRTOS es un Sistema Operativo de Tiempo Real (RTOS) desarrollado por Real Time Engineers LTD. Está diseñado para su utilización en Microcontroladores con soporte a más de 34 arquitecturas. Es un desarrollo profesional con estrictos controles de calidad, robustez, soporte y además es de licencia comercial libre (GPL) por lo que no impone ningún tipo de restricción legal para comercialización.

Características de freeRTOS:

- De código abierto
  - No hay costo de implementación
  - El código está ampliamente comentado, es muy sencillo verificar cómo hace su trabajo.
  - Es relativemente sencillo de portar a plataformas nuevas.
- Contiene una gran cantidad de demos, comunidades y foros de discusión.
- Existen dos versiones con soporte comercial, OpenRTOS® y SafeRTOS® (versión certificada SIL3, para sistemas de misión critica).
- Está pensado para Microcontroladores
  - Está escrito mayormente en C, salvo las partes específicas de cada plataforma.
  - Es liviano en tamaño de código y en uso de memoria RAM.
- Es un mini kernel[7] de tiempo real que puede trabajar en modos cooperativo, apropiativo o mixto.
- Permite compilar solo las funciones que se vayan a usar, acotando así su impacto en la memoria de código.

[7] Mini kernel significa que provee los servicios mínimos e indispensables.

- Se puede definir y acotar en tiempo de compilación el uso de memoria de datos por parte del sistema.
- Ofrece funciones de temporización, de comunicación entre tareas, de sincronización entre tareas e interrupciones, y de definición de secciones críticas.

Este RTOS cumple con todas las características requeridas para implementar el diseño expuesto en la sección 3.5.

Existe además un libro escrito por Richard Barry [1], creador de freeRTOS, dedicado exclusivamente a su utilización con Microcontroladores NXP 17xx, facilitando su comprensión e implementación.

4.3.4. CMSIS

CMSIS son las siglas de "Cortex Microcontroller Software Interface Standard". Es una capa de software para la abstracción de Hardware (HAL) para los procesadores Cortex-M definida por ARM en cooperación con vendedores de Silicio y de Software. Esta capa facilita la interfaz entre periféricos embebidos, sistemas operativos del real-tiempo y componentes externos. En la figura [4.4] se muestra como encaja el CMSIS en el modelo de software.

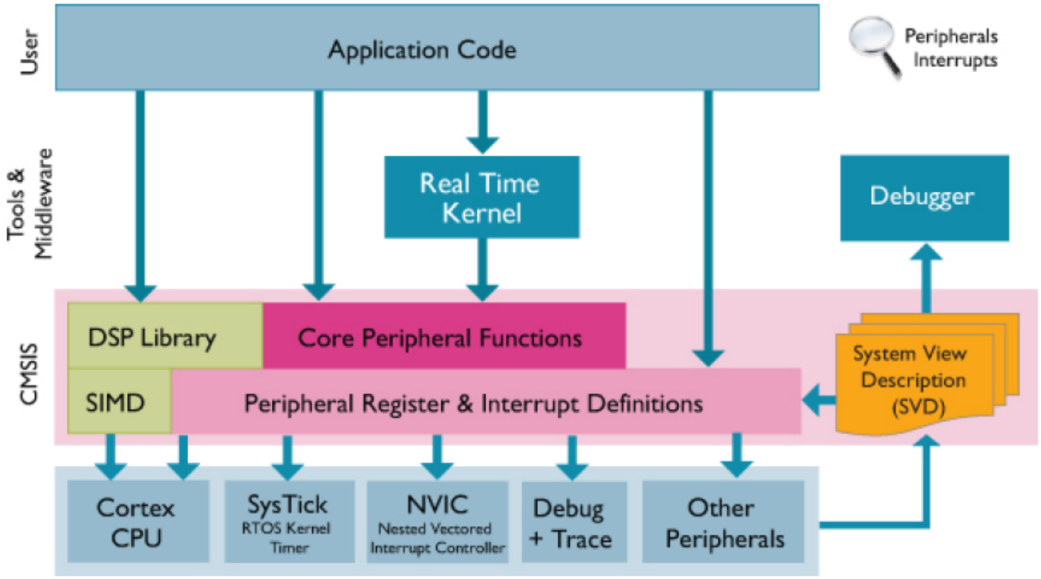


Figura 4.4: Ubicación de CMSIS en un modelo de software.

ARM proporciona una parte del CMSIS (capas del software están disponible para varios compiladores) llamada **Core & Peripheral Access Layer**. Esta parte contiene definición de nombres, direcciones y funciones de ayuda para acceder a los registros y periféricos del CORE (o núcleo). Define además una interfaz independiente del dispositivo para RTOS Kernels incluyendo definiciones del canal de depuración. Estas capas de software son expandidas por varios vendedores de Silicio con:

- **Device Peripheral Access Layer:** provee definiciones para dispositivos periféricos.
- **Access Functions for Peripherals:** suministra funciones de ayuda adicionales para periféricos.

CMSIS define para un sistema con Microcontrolador Cortex-M:

- Una manera común de acceso a registros de periféricos del Core y de definir vectores de excepción.
- Los nombres de los registros de los periféricos del Core y de los vectores de excepción del mismo.
- Una interfaz independiente del dispositivo para Kernels RTOS incluyendo un canal de depuración.
- Mediante el uso de software CMSIS compatible, el usuario puede fácilmente reutilizar el código (plantilla). CMSIS tiene por objeto permitir la combinación de componentes de software de múltiples proveedores de componentes.

Esta capa de software junto al Sistema Operativo freeRTOS forman en la implementación, la capa de Sistema Operativo y Drivers expuesta en la sección 3.5.

4.4. Implementación del modelo computacional

La construcción del entorno de edición de programas PLC sirvió, entre otras cosas, para testear y poner a punto el diseño del modelo computacional de las Configuraciones de Software y los lenguajes de programación definidos en la norma IEC 61131-3. Este modelo computacional, basado en los conceptos de la programación con objetos, fue detallado en profundidad en la sección 3.4.

El entorno de edición de programas PLC incluye una implementación del modelo computacional, programada en lenguaje Smalltalk, utilizando el entorno Pharo 2.0, introducido en la sección 4.2. Esta implementación refleja fielmente el diseño propuesto, donde cada concepto incluido dentro del modelo se corresponde con una clase Smalltalk, o bien, con una pequeña jerarquía de clases. Estas clases incluyen la implementación de los mensajes definidos para el concepto correspondiente. Las relaciones entre las clases también respetan las pautas definidas en el diseño. Del mismo modo, pueden existir simplificaciones, o pequeñas variaciones respecto de las jerarquías de clases definidas en el diseño. Esta implementación de los conceptos incluidos en el diseño forman la base del código del modelo computacional dentro del entorno de edición, siendo acompañados por otras clases con funcionalidades accesorias.

En esta implementación se toma la convención de utilizar el prefijo "**PLC_**" para distinguir las clases constituyentes frente a las clases incluidas en el entorno Pharo. Se eligió el idioma inglés para los nombres de clases y métodos.

En las siguientes secciones se ejemplifica la implementación construida, describiendo brevemente las clases que se corresponden con el modelo de Tipos de Datos, comentando algunos aspectos de la implementación de un Segmento de programa en el lenguaje Ladder, y finalmente, mostrando el código utilizado para una modificación sencilla en un Segmento Ladder.

4.4.1. Tipos de datos

Dentro del diseño propuesto, el modelo de Tipos de Datos se definió en la sección 3.4.1. Una versión resumida se presenta en la figura [3.40]. El código del entorno de edición de programas PLC incluye las siguientes clases, indentadas según su jerarquía de Herencia.

```
PLC_ClassWithIdentifier
  PLC_DataType
    PLC_ElementaryDataType
    PLC_DerivedDataType
    PLC_GenericDataType
```

Todas estas clases son abstractas, implementando parte del comportamiento requerido para el modelo de los tipos de datos dentro del editor de programas PLC. Sus subclases concretas (directas o indirectas) modelan cada uno de los tipos de datos. A modo de ejemplo, se expone la cadena de jerarquía para las clases **PLC_StructureDecl** y **PLC_Boolean**, que se corresponden con los tipos de datos Estructura y Booleano respectivamente.

```
PLC_ClassWithIdentifier
  PLC_DataType
    PLC_DerivedDataType
      PLC_StructureDecl

PLC_ClassWithIdentifier
  PLC_DataType
    PLC_ElementaryDataType
      PLC_BitStrings
        PLC_Boolean
```

Todos los objetos que representan tipos de datos entienden los mensajes correspondientes a los indicados para la clase Tipo de Datos como puede observarse en la figura [3.40], de acuerdo a la tabla de equivalencia [4.1].

Nombre del método en el diseño	Nombre del método en la inmplementación
identificador	`identifier`
aceptaTipoDeDatos(tipoDeDatos)	`acceptType:`
aceptaValor(valor)	`acceptValue:`
aceptaCablearse()	`acceptWire`
compilarC()	`cCompile`
compilarIL()	`ilCompile`

Tabla 4.1: Equivalencias entre nombres de métodos de tipos de datos en diseño e implementación.

La clase **PLC_ClassWithIdentifier** brinda la funcionalidad de poseer un identificador. Por lo tanto, todas las clases del modelo computacional que requieran de un identificador, serán subclases de ésta. Además de las que representan tipos de datos, otras clases que utilizan identificador son las que representan variables y categorías de declaraciones de variables (cuyo diseño ha sido descripto en las secciones 3.4.4 y 3.4.5 respectivamente).

La clase **PLC_DataType** incluye implementaciones genéricas de los métodos **acceptType:**, **acceptWire**, **cCompile** e **ilCompile**. Estas implementaciones se refinan en las subclases de la jerarquía que requieren comportamiento más específico o distinto para estas funcionalidades.

Un ejemplo sencillo es **acceptWire**. En base a lo enunciado en la sección 3.4.2, referente a que sólo el tipo de datos Booleano permite crear una conexión en un pin, implicando así, que las dos únicas implementaciones del método **acceptWire** son las siguientes.

En la clase **PLC_DataType**:

```
1 acceptWire
2   ^false
```

En la clase **PLC_Boolean**:

```
1 acceptWire
```

```
    ^true
```

Análogamente, el comportamiento genérico de **ilCompile** para un tipo de datos es muy sencillo:

En la clase **PLC_DataType**:

```
ilCompile
  ^self keyword asString
```

El mensaje **keyword** está definido en cada tipo de dato con el nombre del tipo correspondiente, por ejemplo:

```
keyword
  ^#BOOL
```

En cambio, para un tipo de datos estructura, la compilación a IL es más compleja. En la clase **PLC_StructureDecl**:

```
ilCompile

  | strucTypeDec |

  strucTypeDec := '  STRUCT'.
  self declarations do: [ :each |
    strucTypeDec := strucTypeDec , '    ' , each ilCompile , ';'.
      ].
  strucTypeDec := strucTypeDec , '  END_STRUCT'.
  ^ strucTypeDec.
```

Se observa que la definición de un tipo estructura incluye la lista de declaraciones de sus componentes, a la que se accede mediante el mensaje **declarations**.

4.4.2. Segmento de programa en lenguaje Ladder

El diseño de un segmento de POU para lenguajes de programación gráficos se ha descripto en la sección 3.4.8, un resumen puede apreciarse en la figura [3.60].

La implementación incluye clases que se corresponden con varios de los conceptos comprendidos en este modelo, de acuerdo a la tabla [4.2].

Nombre en el Modelo	Nombre de su clase
Segmento Ladder	PLC_LadderSegment
Bloque Conectable	PLC_ConnectableBlock
Pin de Entrada	PLC_ConnectableBlockInputPin
Pin de Salida	PLC_ConnectableBlockOutputPin
Conexión	PLC_ConnectableBlockConnection

Tabla 4.2: Conceptos del modelo y sus nombres de clase.

Los elementos de lenguaje Ladder se implementan mediante una jerarquía de clases, cuya raíz es la clase **PLC_Element**. En la tabla [4.3] se expone cada elemento representado junto a su nombre de clase:

Nombre del Elemento	Nombre de su clase
Contacto	PLC_Contact
Bobina	PLC_Coil
Barra de Alimentación izquierda	PLC_LeftPowerRail
Barra de Alimentación derecha	PLC_RightPowerRail
Invocación a función	PLC_FunctionCall

Tabla 4.3: Elementos Ladder y sus nombres de clase.

Estas clases siguen una variante del patrón de diseño Singleton [6], existiendo solamente una cantidad acotada de instancias de cada una. Por ejemplo, para una Bobina existen dos instancias, correspondientes **PLC_Coil** (representa un Elemento Bobina) y **PLC_NCoil** (representa un Elemento Bobina Negada). La información particular de cada contacto, bobina, etc., se encuentra dentro de sus bloques conectables.

Para compilar un segmento Ladder, se generan los objetos de modelo de programa IL correspondientes. Estos objetos son los responsables de construir el código en lenguaje IL o C como fue definido en la sección 3.4.8.4.

El responsable de generar las instrucciones IL correspondientes a un componente es el elemento del bloque conectable principal dentro de dicho componente. Los elementos son los encargados de dirigir la generación de código. Luego de generar su código, delegan la generación en los bloques siguientes, los cuales deben continuar con la misma (sección 3.4.8.4). Como el elemento es un Singleton, el método que genera las instrucciones IL recibe como parámetro el bloque conectable del cual obtiene la información.

A modo de ejemplo, se expone el código correspondiente a la generación de código de una bobina. Ésta genera una instrucción de almacenamiento ST en lenguaje IL.

En la clase **PLC_Coil**:

```
1  objectCompileIn: anObjectProgram for: aPLC_ConnectableBlock
2    "Agrega sus Objetos Instrucciones IL al acumulador de programa
        anObjectProgram"
3
4    | instruction |
5
6    "Crea una instruccion de almacenamiento ST"
7
8    instruction := PLC_ST_Instruction new.
9
10   "Agrega el modificador que contiene la Bobina a dicha instrucción."
11
12   instruction modifiers add: self modifier.
13
14   "Agrega el operando que tiene el Arguento conectado a su segunda
        salida a la instrucción ST."
15
16   instruction operands add: (self actualArgumentFor:
        aPLC_ConnectableBlock).
17
18   "Agrega el la instrucción al acumulador de programa."
19
20   aObjectProgram add: instruction.
21
```

```
    "Finalmente indica al elemento conectado a su primer salida que
       genere sus instrcciones y las guarde en el acumulador de programa
       anObjectProgram para seguir generando el código del Segmento
       Ladder."
    (aPLC_ConnectableBlock blockConnectedToOutput: 1) objectCompileIn:
       anObjectProgram.
```

```
actualArgumentFor: aPLC_ConnectableBlock
  "Devuelve el actualArgument que en este caso es una variable
  simbolica simple"

  ^(aPLC_ConnectableBlock blockConnectedToOutput: 2) element
     actualArgument.
```

En este ejemplo se observa la creación y configuración de la instrucción ST correspondiente a una bobina. El operando de la instrucción corresponde a la variable asociada a la bobina, que está conectada a la segunda salida del bloque conectable cuyo elemento es la bobina (ver figura [3.61(c)]). Finalmente, la compilación debe continuar por el componente que está conectado a la primer salida de la bobina.

4.4.3. Ejemplo de modificación de un segmento Ladder

Tal como se indica en el diseño, la responsabilidad de agregar nuevos componentes dentro de un Segmento Ladder es del objeto que lo representa. A título de ejemplo, se expone el método que permite agregar un Componente Bobina sobre una cierta conexión, con comentarios que describen las distintas acciones que implica esta modificación en el Segmento.

En la clase **PLC_LadderSegment**:

```
addCoil: aPLC_ElementCoil onConnection: aPLC_Connection
  "Crea y añade un componente Bobina."

  | connectableBlockActualArgumentWriter connectableBlockCoil |


  "Crea los Bloques Conectables correspondientes a la bobina
  y al escritor de la variable asociada"

  connectableBlockCoil :=
    PLC_ConnectableBlock newWithElement: aPLC_ElementCoil.

  connectableBlockActualArgumentWriter :=
    PLC_ConnectableBlock newWithElement: PLC_Element
       ActualArgumentWriter.


  "Conecta los 2 bloques conectables que forman el Componente:  el
     output
  2 de la bobina se conecta con el input 1 del escritor de variable"

  connectableBlockCoil
    output: 2
    connectTo: connectableBlockActualArgumentWriter input: 1.


  "Agrega los Bloques Conectables al segmento Ladder.
```

```
    El segmento mantiene el conjunto que incluye todos sus bloques,
    adicionalmente a la conexión de los bloques entre sí"

    self addBlock: connectableBlockCoil.

    self addBlock: connectableBlockActualArgumentWriter.


    "Conecta el nuevo componente a los Bloques Conectables entre los
    cuales estaba la conexion donde agrego el nuevo componente."

    aPLC_Connection
      replaceConnectionByConnectingToInputPin:
        (connectableBlockCoil inputs first)
          andOutputPin:
            (connectableBlockCoil outputs first).
```

En la clase **PLC_ConnectableBlockConnection**:

```
replaceConnectionByConnectingToInputPin: anInputPin andOutputPin:
    anOutputPin
    "Elimina la conexión que recibe este mensaje. Conecta el pin de
    salida de esa conexión al pin anInputPin (este pin puede un pin de
    un bloque conectado a un circuito complejo), y conecta el pin
    anOutputPin (que también puede ser un pin de salida de un bloque
    de un circuito complejo), al pin de entrada de la conexión.



    self es la conexión:  out ---> in


    se reemplaza por:

                          -------------
    out ---> anInputPin|  circuito  |anOutputPin ---> in
                        -------------


     "

    | connectionOutputPin connectionInputPin |


    "Busca los pines de los Bloques Conectables a entrada y salida
    para esta conexion."

    connectionOutputPin := self outputPin.
    connectionInputPin := self inputPin.
    "Desconecta la conexion"
    self disconnect.

    "Realiza las conexiones pertinentes"
    connectionOutputPin connectTo: anInputPin.
    anOutputPin connectTo: connectionInputPin.
```

4.5. Implementación de la GUI

La implementación de la interfaz gráfica se realiza utilizando el Framework Morphic, como se anticipó en la sección 4.2.1. En la sección 4.5.1 se exponen los Morph utilizados para la confección de la GUI. Luego, en la sección 4.5.2 se dan algunos detalles de la realización de los Morph más complejos creados para este Trabajo Final. Finalmente, en la sección 4.5.3 se explica cómo todos estos Morph se conectan con el Modelo Computacional.

4.5.1. Morph utilizados

Se utilizan varios Morph preexistentes y también debieron desarrollarse nuevos. En la figura [4.5] se muestra una captura de pantalla del Editor de Programas de PLC con la ventana de Edición de POU abierta. En la misma se destacan los nombres de las clases de los Morph que la forman.

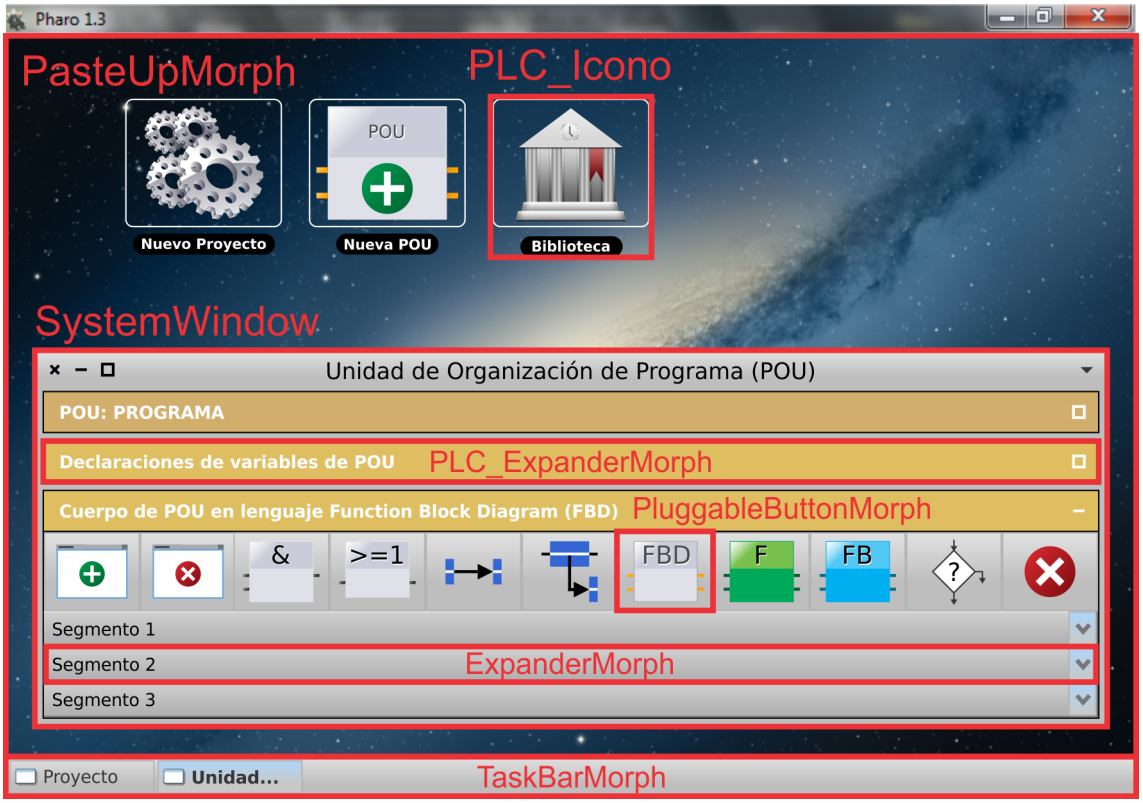


Figura 4.5: Captura de pantalla del Editor de Programas de PLC con la ventana de Edición de POU abierta. En la misma se destacan los Morphs que la forman.

Como se observa en la figura [4.5], toda la pantalla de Smalltalk es un Morph de la clase **PasteUpMorph** conocido como "World", que contiene en su interior todos los Morph que sean creados en pantalla. En la distribución de Pharo viene por defecto un Morph de la clase **TaskBarMorph** (figura [4.6]), que implementa la barra inferior donde se crea un rectángulo con el título de cada ventana abierta, y además, pueden abrirse las ventanas minimizadas.

Figura 4.6: Morph TaskBarMorph.

Los íconos son Morph cuya clase ha sido nombrada **PLC_Icono** y se desarrolla para este Trabajo Final como subclase de ImageMorph (un Morph capaz de mostrar imágenes en pantalla). Cada uno de ellos maneja eventos del mouse. En la figura [4.7] se muestra, como ejemplo, el ícono que abre el diálogo para crear una nueva POU. En la parte derecha de esta figura puede apreciarse como luce el ícono cuando se encuentra el puntero del mouse sobre el mismo; mientras que en la parte izquierda, se muestra como se ve cuando no tiene el puntero encima.

Figura 4.7: Morph PLC_Icono.

Cada ventana es un Morph de la clase **SystemWindow** (figura [4.8]), o bien, **StandardWindow**, que es subclase (hereda) de la primera.

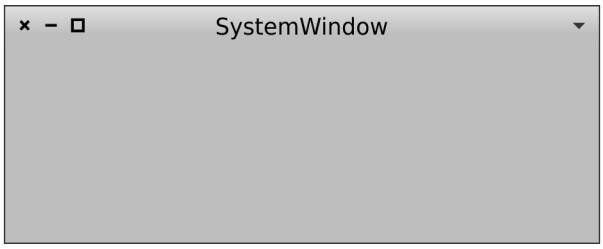


Figura 4.8: Morph SystemWindow.

En la ventana de la figura [4.5] se destacan, además, tres tipos de Morph que conforman el contenido de la misma:

- **PluggableButtonMorph:** Este Morph provee la funcionalidad de un botón. Puede tener una etiqueta de texto (figura [4.9] parte izquierda), o bien, una imagen (figura [4.9] parte derecha).

Figura 4.9: Morph PluggableButtonMorph.

- **ExpanderMorph:** El ExpanderMorph (figura [4.10]) es un Morph que se puede expandir, o contraer, para mostrar u ocultar su contenido, respectivamente. Es ampliamente utilizado en las diferentes ventanas para lograr una intefaz más limpia para el usuario.

Figura 4.10: Morph ExpanderMorph.

- **PLC_ExpanderMorph:** Este Morph (figura [4.11]) se realiza para el presente Trabajo, como subclase de ExpanderMorph, modificando únicamente su estilo visual.

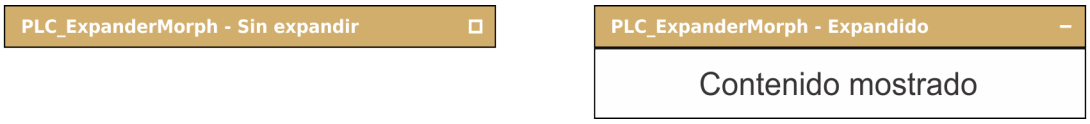


Figura 4.11: Morph PLC_ExpanderMorph.

Expadiendo la categoría de variables de interfaz, en la ventana de edicción de POU de la figura [4.5], y el Segmento 1, se presenta el contenido de estos ExpanderMorph en la figura [4.12].

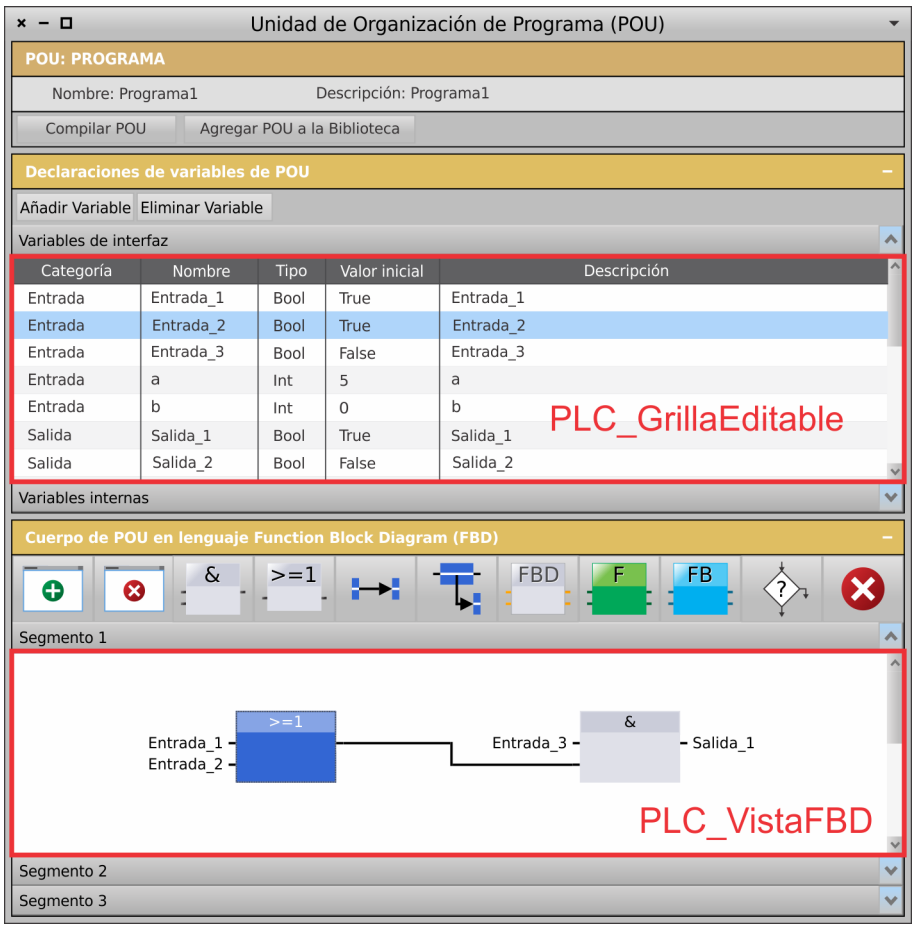


Figura 4.12: Ventana de edicción de POU con dos Mophs destacados.

Los Morph destacados que se crean para esta implementación son:

- **PLC_GrillaEditable:** Este Morph muestra las declaraciones de variables y permite la edición directa de las mismas.
- **PLC_VistaFBD:** El Morph PLC_VistaFBD brinda al usuario la vista de un segmento de programa en lenguaje gráfico FBD. Se compone de todos los Morph de los elementos de programa. Maneja eventos del mouse y teclado los cuales actúan sobre sus Morph contenidos.

En la sección 4.5.2 se ofrecen algunos detalles de la implementación de los mismos.

Otros Morph preexistentes utilizados se resumen en la figura [4.13].

Figura 4.13: Diálogo de Declaración de Variable con Morphs que lo forman destacados.

Estos son:

- **DialogWindow:** Este Morph representa un cuadro de diálogo[8].
- **LabelMorph:** Morph que muestra una etiqueta de texto.
- **DropListMorph:** El DropListMorph es un Morph que implementa una lista desplegable[9].
- **PluggableTextFieldMorph:** Es un Morph que realiza una caja de texto[10].

4.5.2. Morph PLC_Vista

La jerarquía de clases que implementa el Morph **PLC_Vista** y sus "descendientes" se muestra en la figura [4.14].

La clase PLC_Vista contiene varios atributos, entre ellos un atributo 'controlador', que es una referencia al controlador según el patrón MVC, el cual se detalla en la sección 4.5.3, otros atributos que hacen referencia a las listas de elementos contenidos seleccionados, no seleccionados y el elemento con el foco del teclado. Utilizando estas listas maneja los eventos de teclado y mouse sobre todos los Morph contenidos (submorph). Estos submorph son Morph que representan Elementos de programa gráfico del modelo computacional.

Eventos del teclado

Un Morph maneja la detección de teclas presionadas sobreescribiendo el método *handleKeystroke*. En esta implementación envía el mensaje correspondiente a la tecla, o combinación de teclas, al elemento que PLC_Vista contiene como elemento con el foco del teclado.

[8] Un cuadro de diálogo es un tipo de ventana. Suele ser pequeño, con dos o tres opciones en forma de botones, donde el usuario tiene que decidir cuál presionar y un breve texto explicativo. Generalmente no pueden minimizarse, ni maximizarse, su tamaño no varía, e incluso, en algunos casos, no pueden ser solapado por otras ventanas.

[9] Una lista desplegable es un elemento de control, similar a un cuadro de lista, que permite al usuario elegir un valor de dicha lista. Cuando una lista desplegable está inactiva, se muestra un solo valor. Cuando se activa, muestra (cae) una lista de los valores, de la cual el usuario puede seleccionar uno. Cuando el usuario selecciona un nuevo valor, el control vuelve a su estado inactivo, se muestra el valor seleccionado.

[10] Una caja de texto, o campo de texto, permite al usuario la entrada de información textual para ser utilizada por el programa.

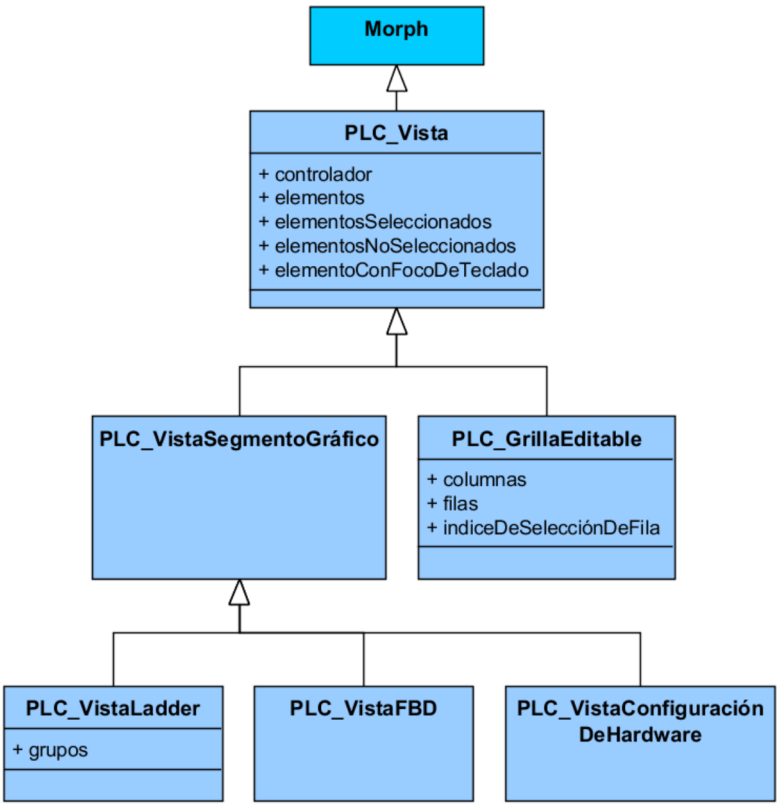


Figura 4.14: Modelo de vistas.

Eventos del mouse

- *mouseOver:* Evento que se activa (enviando el mensaje correspondiente) cuando el puntero del mouse se encuentra sobre el Morph.
- *mouseOut:* Evento que se activa cuando el puntero del mouse deja de estar sobre el Morph.
- *leftClick*: Evento que se activa cuando se presiona y suelta el botón izquierdo del mouse, una vez, con el puntero del mismo situado sobre el Morph.
- *leftDoubleClick:* Evento que se activa cuando se hacen dos leftClick seguidos en un corto lapso de tiempo.
- *rightClick:* Evento que se activa cuando se presiona y suelta el botón derecho del mouse, una vez, con el puntero del mismo situado sobre el Morph.
- *rightDoubleClick:* Evento que se activa cuando se hacen dos rightClick seguidos en un corto lapso de tiempo.
- *startDrag:* Evento que se activa cuando se presiona algún botón del mouse sobre el Morph y se arrastra

Para que un Morph maneje estos eventos, en primer lugar, deben sobrescribirse los siguientes métodos:

- *handlesMouseDown:* Permite que el Morph maneje los eventos correspondientes a presionar o soltar los botones del mouse.

- *handlesMouseOver:* Permite que el Morph maneje el evento MouseOver.

En segundo lugar, deben implementarse los métodos:

mouseUp, mouseLeave, mouseEnter, mouseDown.

Finalmente implementarse los métodos:

mouseOver, mouseOut, leftClick, leftDoubleClick, rightClick, rightDoubleClick y startDrag.

La clase **PLC_VistaSegmentoGráfico** contiene los métodos para obtener un Elemento de programa gráfico del modelo desde el Morph correspondiente que lo representa, y moverse a través de la red de bloques conectables.

Las clases **PLC_VistaLadder** (figura [4.15]), **PLC_VistaFBD** (figura [4.16]) y **PLC_VistaConfiguraciónDeHardware** (figura [4.17]), tienen los métodos para añadir o remover Morph que representan a los Elementos gráficos de lenguajes Ladder y FBD respectivamente.

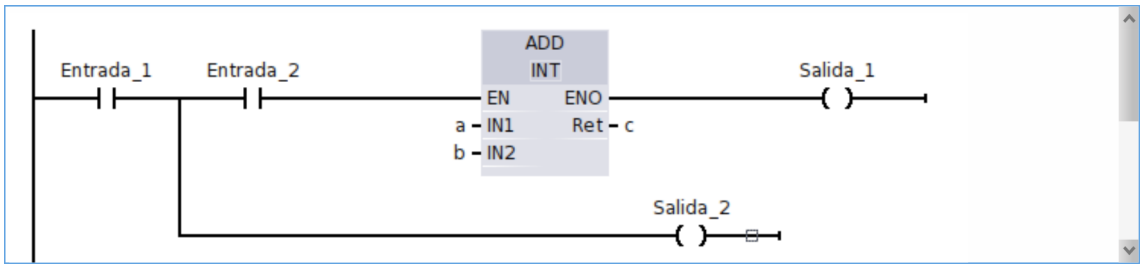


Figura 4.15: Morph PLC_VistaLadder.

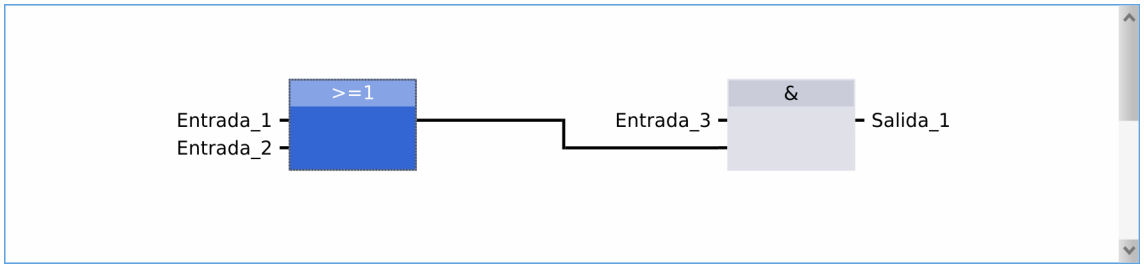


Figura 4.16: Morph PLC_VistaFBD.

En particular, la clase PLC_VistaLadder contiene un atributo 'grupos' que es una referencia a grupos de Morphs. Estos grupos los utiliza para resolver el layout en pantalla automático de Ladder junto con información contenida en sus submorphs.

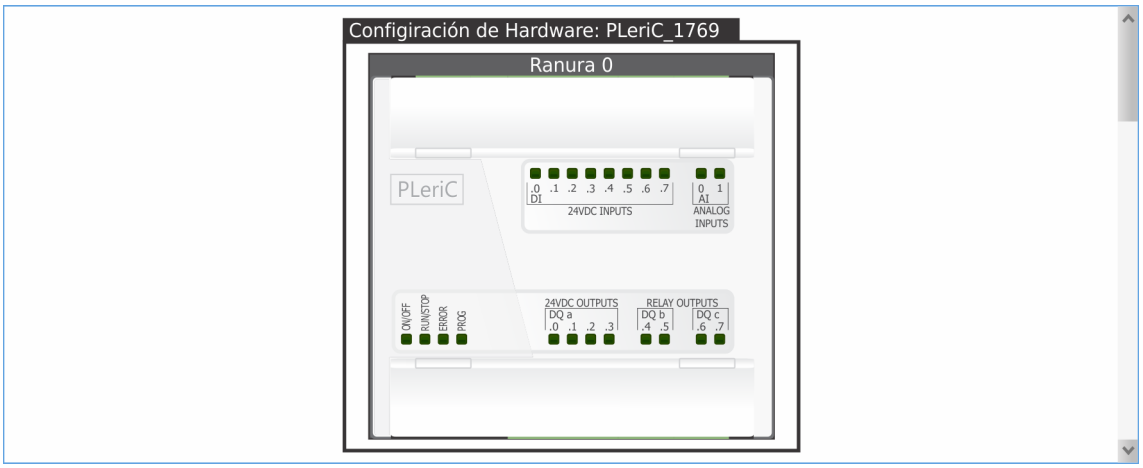


Figura 4.17: Morph PLC_Vista ConfiguraciónDeHardware.

La clase **PLC_GrillaEditable** (figura [4.18]) contiene los atributos 'columnas', 'filas' e 'índiceDeSelecciónDeFila'. Los primeros dos, son listas de Morph y el último es un número entero que indica el número de fila seleccionada.

Categoría	Nombre	Tipo	Valor inicial	Descripción
Entrada	Entrada_1	Bool	True	Entrada_1
Entrada	Entra	Bool	True	Entrada_2
Entrada	[illegible]	Bool	False	Entrada_3
Entrada	a	Int	5	a
Entrada	b	Int	0	b
Salida	Salida_1	Bool	True	Salida_1
Salida	Salida_2	Bool	False	Salida_2

Figura 4.18: Morph PLC_GrillaEditable.

Todos los Moph que representan Elementos de programa gráfico pueden cambiar de colores según su estado en la interfaz gráfica. Esto es, si están seleccionados u otros eventos similares.

4.5.3. MVC

En la sección 3.2 se anticipó la utilización del patrón MVC. Concretamente en esta implementación se utiliza para conectar los Morph de la GUI con el Modelo Computacional a través de "Controladores".

A distintos niveles de vista existen varios controladores conectándose con el modelo donde corresponde. Para ejemplificar esto se muestra un esquema simplificado en la figura [4.19].

Si bien en principio esta organización complica su implementación por el mayor número de controladores, en lugar de un único controlador para toda la ventana, tiene como ventaja que divide el problema en partes más pequeñas y desacopla los componentes de la vista facilitando futuras modificaciones.

Los eventos que reciben los controladores son de alto nivel, es decir, únicamente eventos que modifican el modelo, como por ejemplo, el agregado de algún componente o edición de alguna variable; la detección de eventos a bajo nivel (mouse y teclado) la realiza el Morph vista aprovechando las facilidades que otorga el Framework Morphic, como se mencionó en la sección 4.5.2.

Cada controlador, envía la petición de actualización a su respectiva vista cuando el modelo informa que ocurrió algún cambio.

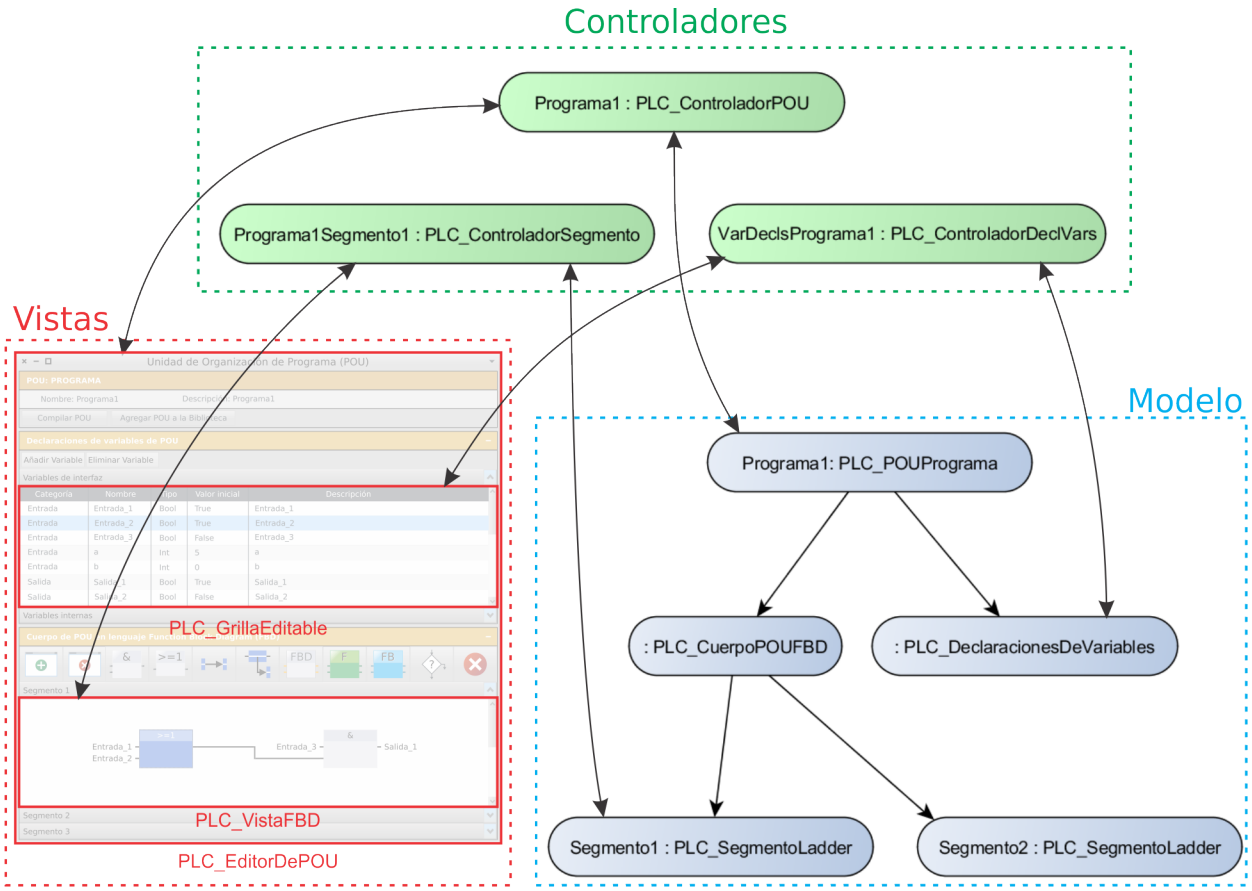


Figura 4.19: Ejmplo de MVC implementado.

4.6. Implementación del Entorno de Ejecución

Cada parte del Entorno de Ejecución se implementa como un proyecto del tipo Biblioteca estática de LPCXpresso IDE. Como se adelantó en la sección 4.3.1 se utilizaron FreeRTOS y CMSIS como Sistema Operativo y Drivers Respectivamente. Finalmente, el modelo de Software de PLC queda implementado como se expone en la figura [4.20].

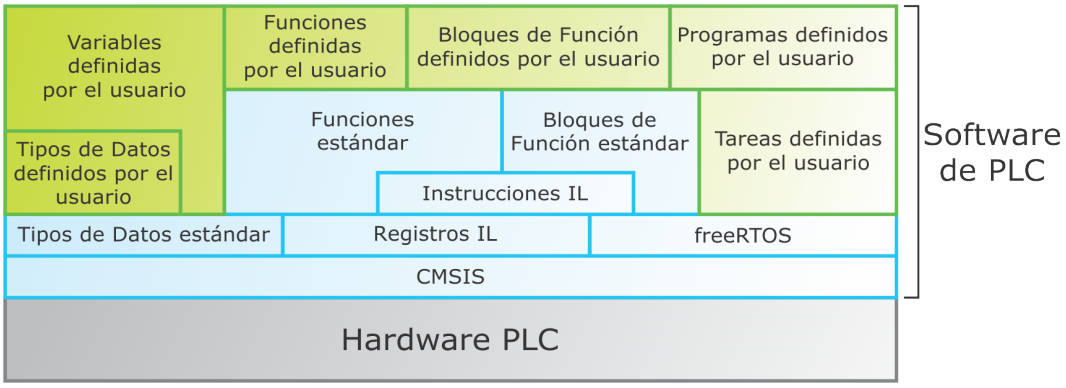


Figura 4.20: Modelo de Software de la implementación.

4.7. Implementación del Hardware

4.7.1. Microcontrolador NXP LPC1769

En la sección 4.3.2 se determinó utilizar el LPCXpresso Target Board con el Microcontrolador LPC1769. A continuación se describen las características técnicas de este Microcontrolador.

CPU:

- Núcleo ARM Cortex-M3 revisión 2.
- Velocidad de clock máxima: 120 MHz.

Integra controladores avanzados:

- MPU (Memory Protection Unit): Unidad de protección de memoria que permite dividirla hasta en ocho zonas con diferentes niveles de acceso.
- WIC (Wake-Up Interrupt Controller): Permite "despertar" al Microcontrolador casi instantáneamente aunque se encuentre en modo Deep Sleep.
- Flash Acelerator: Para "espera cero" a velocidad máxima de clock.
- DMA (Direct memory access): El acceso directo a memoria permite a algunos periféricos acceder a la memoria del sistema para leer o escribir independientemente de la unidad central de procesamiento (CPU) principal.

Modos de ahorro de energía:

- Sleep, Deep Sleep
- Power-Down, Deep Power-Down.

Otras características:

- Frente a competidores líderes posee en promedio un rendimiento 35 % mayor (según EEMBC[11]).
- Operación simultánea de Ethernet, USB y CAN sin cuellos de botella.
- Totalmente compatible con USB 2.0.
- ETM (Embedded Trace Macrocell).
- Real Time Clock (RTC) de 1uA con Función de Calendario.
- Full Debug/Trace.

Características del núcleo Cortex-M3

Cortex-M3 pone al alcance del programador de sistemas embebidos características que anteriormente se encontraban sólo en procesadores de alta gama. Posee un desempeño eficiente, mayor capacidad de trabajo frente a mismas condiciones de frecuencia o potencia y bajo consumo de

[11] EEMBC son las siglas en inglés del Embedded Microprocessor Benchmark Consortium. Este consorcio desarrolla software de benchmark (punto de referencia) para ayudar a diseñadores a elegir el procesador óptimo para una determinada aplicación.

energía. Es un procesador de 32-bits, con ALU, registros, buses, interfaces de memoria y data paths de 32-bits.

El modelo de memoria es del tipo Harvard, eso significa que admite accesos simultáneos a memoria de datos y programa. Su capacidad máxima de direccionamiento es de 4GB.

NVIC (Nested Vectored Interrupt Control): es el controlador de interrupciones anidadas. El vector de excepciones e interrupciones admite 256 entradas. Las primeras 16, son estándar de la arquitectura, mientras que las restantes 240 son específicas del Microcontrolador, definidas por el fabricante.

4.7.2. Circuito para la programación mediante USB

En base a la decisión de diseño la sección 3.6.2 se utiliza el puerto de comunicaciones USB para descargar el programa al PLC. Las características de este bus de comunicación son:

- USB significa "Universal Serial Bus", en español, bus serie universal..
- El bus se encuentra controlado por un único "Host".
- On-the-Go (Protocolo de negociación de host): permite a dos dispositivos negociar el rol de host.
- Topología estrella.
- Se pueden utilizar hubs para dividir alta y baja velocidad.
- Hasta 127 dispositivos pueden ser conectados a un bus USB en cualquier momento.
- Utiliza 4 líneas malladas: 2 son de alimentación (+5v y GND) y los otros 2, un par trenzado donde las señales se transmiten en modo diferencial.

En la figura [4.21] se ofrece el esquema de circuito eléctrico propuesto por el fabricante del Microcontrolador.

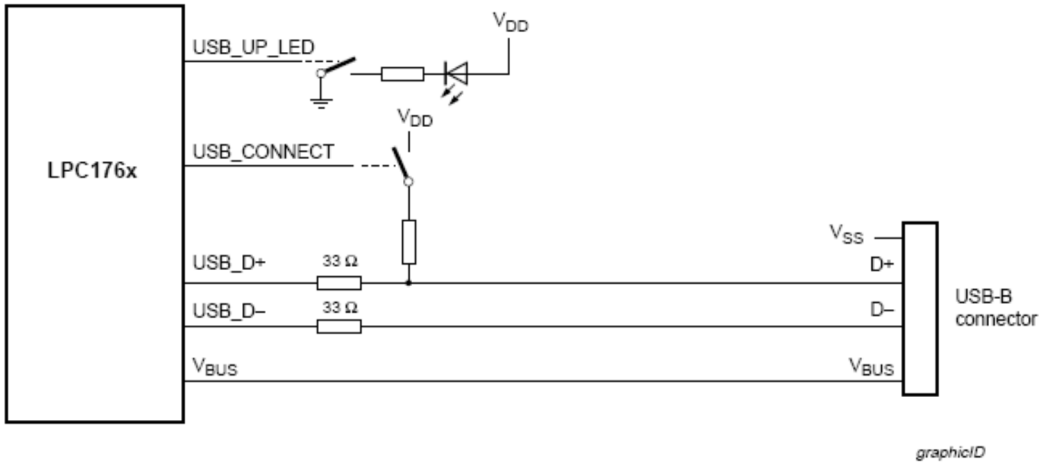


Figura 4.21: Esquemático de circuito de conexión USB.

4.7.3. Entradas Digitales

El circuito de Entradas Digitales (DI) del tipo sumidero, como se propuso en la sección 3.6.3, se muestra en la figura [4.22].

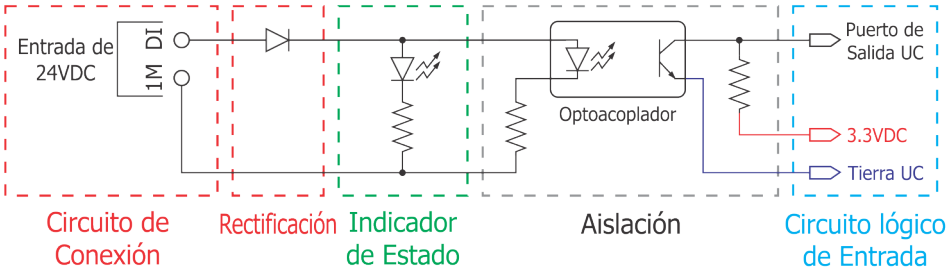


Figura 4.22: Esquemático de circuito de Entrada Digital De 24VDC.

En la misma, se han recuadrado las diferentes etapas correspondientes al diseño. Posee bornes de conexión, etapa de rectificación, indicación luminosa del estado de la entrada mediante un diodo LED, aislación galvánica por optoacoplador y conexión a un terminal de entrada del Microcontrolador y su respectiva fuente de alimentación de 3.3VDC.

4.7.4. Salidas Digitales

Para las Salidas Digitales (DQ) se han propuesto dos tipos (sección 3.6.4), Salidas Digitales de 24VDC y Salidas a relé.

El circuito propuesto para el primer tipo se expone en la figura [4.23].

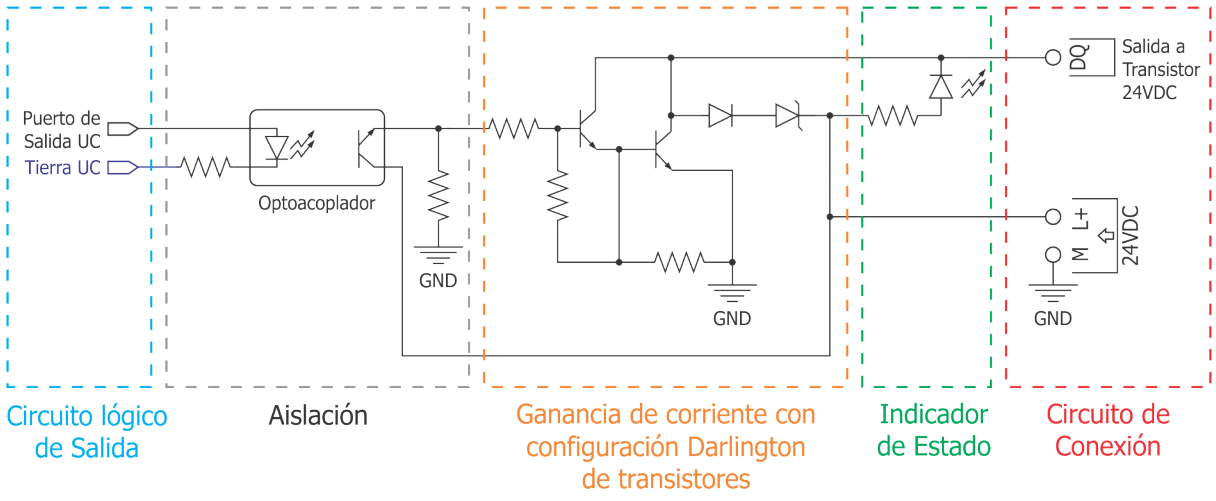


Figura 4.23: Esquemático de circuito de Salida Digital a transistor NPN de 24VDC.

Este tipo de salida (sumidero) utiliza un transistor NPN. En la figura [4.23] se recuadran las distintas etapas del circuito. Estas corrsponden a: terminal de salida del Microcontrolador y su terminal de tierra correspondiente, optoacoplado, una etapa de ganancia de corriente con Transistores en configuración Darlington[12], indicación del estado de la salida mediante Led y bornes de conexión.

[12]Permite que con la pequeña intensidad de corriente que puede manejar el transistor de salida del optoacoplador, se pueda manejar a la salida del circuito una grán intensidad de corriente.

Para el segundo tipo (salidas a relé) el esquema del circuito eléctrico (figura [4.24]) es igual, con el agregado del respectivo relé antes de los bornes.

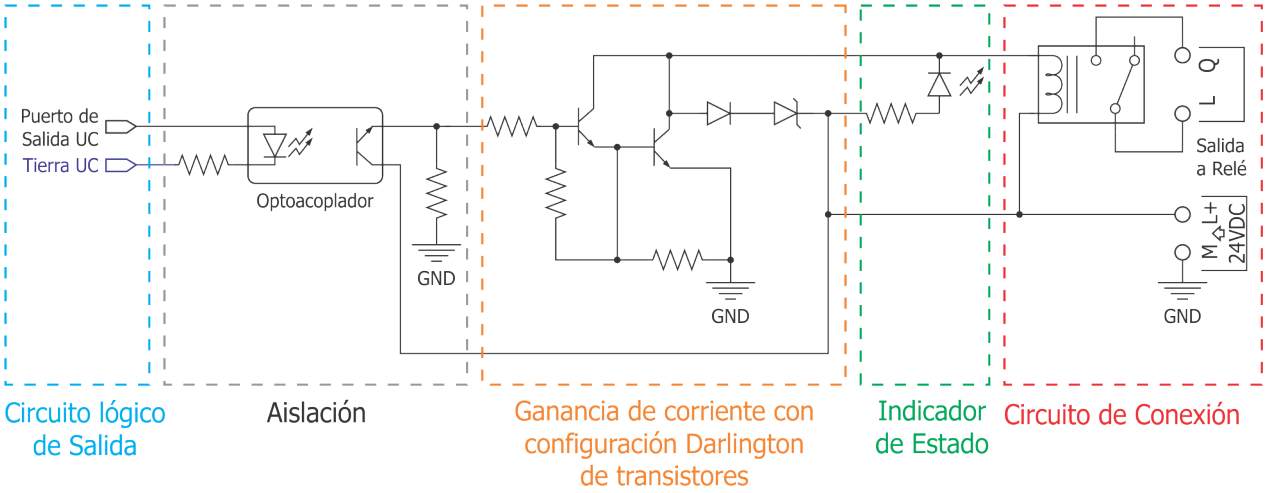


Figura 4.24: Esquemático de circuito de Salida Digital a Relé del PLC.

4.7.5. Fuente de Alimentación Eléctrica

En base a los requerimientos descriptos en la sección 3.6.5, dado el Microcontrolador elegido (que se alimenta con 3.3 VDC) y la utilización de USB (requiere una alimentación de 5 VDC); se diseña el circuito de fuente que se expone en la figura [4.25].

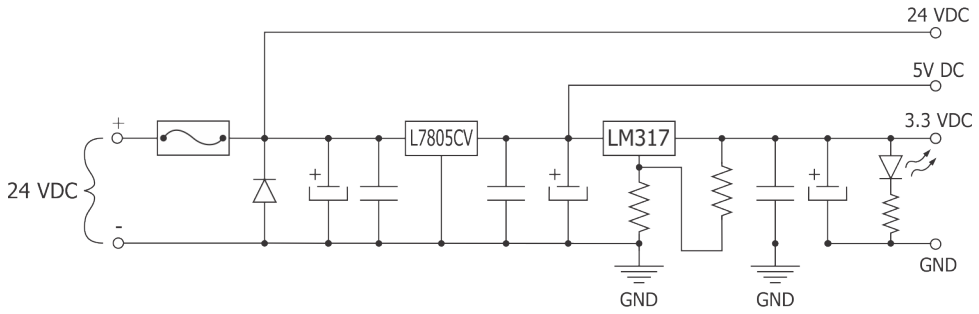


Figura 4.25: Esquemático de circuito de Fuente de alimentacion de 3.3 VDC, 5 VDC y 24 VDC.

En principio consta de una etapa de protección contra inversión de polaridad formada por un diodo en inversa y un fusible, a la salida de esta etapa, se obtienen 24 VDC. Luego, contiene una regulación de voltaje mediante el circuito integrado L7805CV, adquiriéndose 5 VDC a la salida para la alimentación del USB. Finalmente, existe una última etapa de regulación de voltaje mediante el circuito integrado LM317 logrando los 3.3 VDC que requiere el Microcontrolador para su alimentación. Además, posee un LED, que indica si la fuente se encuentra encendida.

4.7.6. Construcción del Equipo PLC

El frente del Equipo PLC se muestra en la figura [4.26]. En el mismo se observan un total de 19 LEDs, 8 corresponden a la señalización dele estado de las Entradas, 8 para el estadod e las Salidas y los últimos 3 para el estado general del PLC. Estos LEDs de estado son:

- **ON/OFF:** Indica que el PLC está alimentado.
- **RUN/STOP:** LED bicolor, en rojo el PLC está en modo STOP (sin ejecutar el programa de usuario), mientras que en verde se encuentra en modo RUN (ejecuta el programa de usuario).
- **ERROR:** Indica si ocurre un error de programa del usuario o de otra índole.

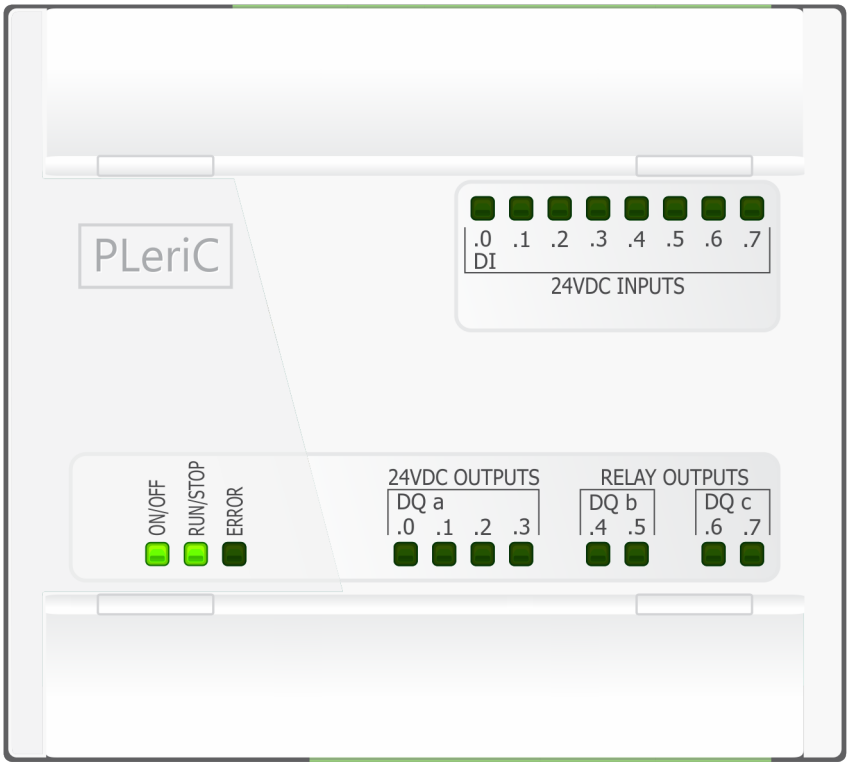


Figura 4.26: Frente del Equipo PLC.

Removiendo las tapas de borneras superior e inferior del equipo pueden observarse las borneras de conexión de Entradas, Salidas y Alimentación, así como el puerto de programación USB como se muestra en la figura [4.27]. Estas borneras puueden separarse fácilmente del equipo para su cableado.

De izquierda a derecha, las borneras superiores son para:

- Entrada de Fuente de alimentación de 24 VDC (dos bornes, el izquierdo es el positivo).
- Puesta a tierra del equipo (un borne).
- Salida de 24 VDC para alimentación de sensores (dos bornes, el izquierdo es el positivo).
- Tierra común a todas las Entradas Digitales (un borne).
- Entradas Digitales (ocho bornes).

Las borneras inferiores, de izquierda a derecha, son:

- Entrada de alimentación de 24 VDC para actuadores (dos bornes, el izquierdo es el positivo).
- Salidas Digitales de 24 VDC a transistor (cuatro bornes).
- Dos Salidas Digitales a relé (3 bornes, con entrada línea en el primer borne).
- Dos Salidas Digitales a relé, de la misma forma que las dos anteriores, con su línea independiente (3 bornes).

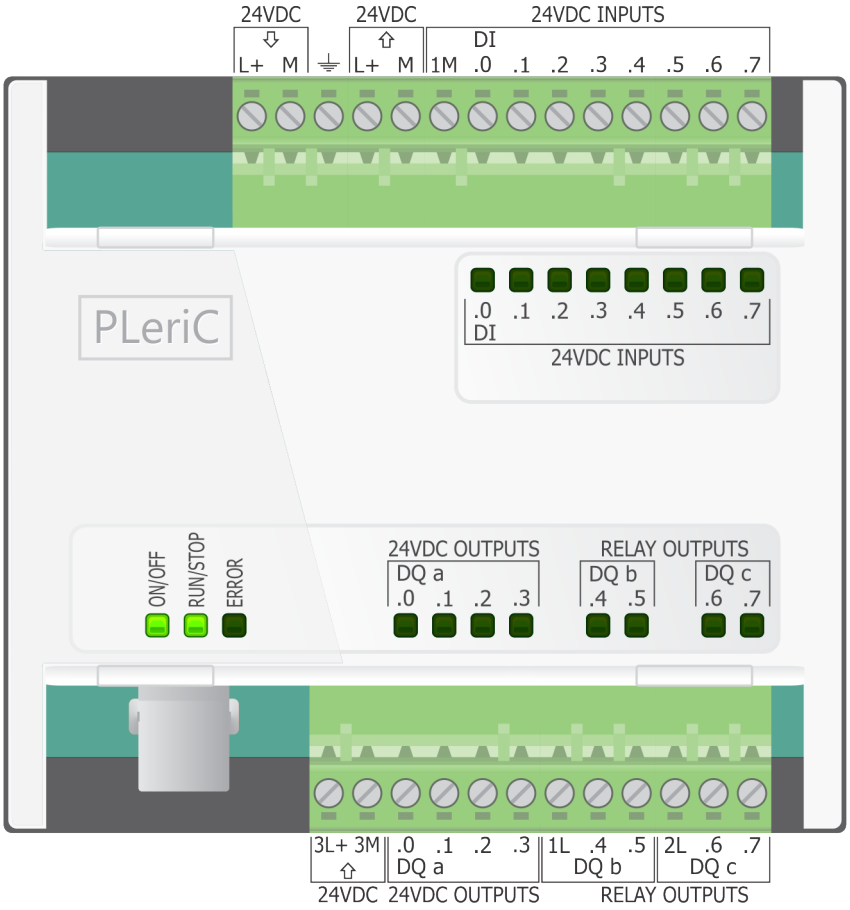


Figura 4.27: Frente del Equipo PLC sin tapas de borneras.

4.7.7. Cableado de Entradas del PLC

Los sensores se conectan a las Entradas Digitales del PLC como se expone en la figura [4.28]. La salida de 24 VDC puede ser utilizada para alimentar los sensores pasivos. Como fue descripto en la sección 3.6.3, deben conectarse entre el terminal positivo de una fuente de 24V DC y el terminal de entrada del PLC. En caso de sensores activos, los mismos poseen su propia fuente de alimentación, entregando un voltaje compatible con este tipo de entradas para indicar su estado.

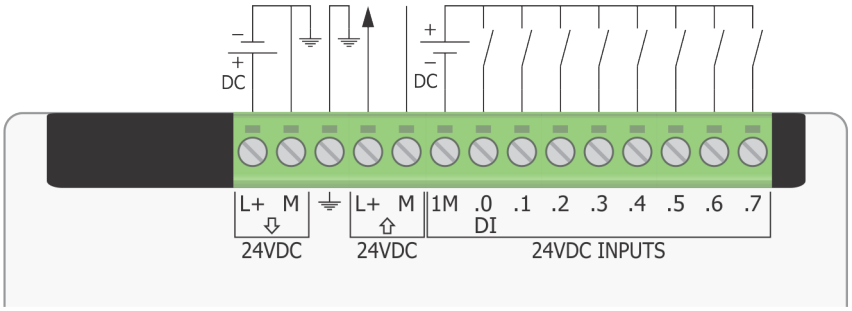


Figura 4.28: Cableado de Entradas del PLC.

4.7.8. Cableado de Salidas del PLC

En la figura [4.29] se ilustra como deben ser conectados las cargas o actuadores tanto a las Salidas Digitales de 24 VDC, como las Salidas Digiales a relé.

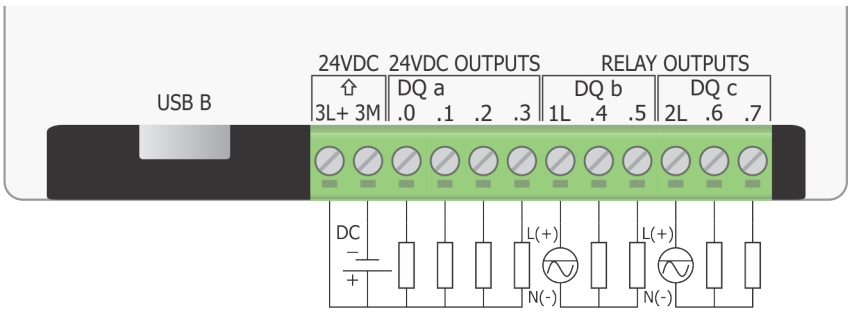


Figura 4.29: Cableado de Salidas del PLC.

En la sección 3.6.4, se adelantó que las cargas conectadas a las Salidas Digitales de 24VDC, se conectan entre el terminal positivo de la fuente de 24VDC y el terminal de salida del PLC. En el caso de las Salidas a Relé, las cargas se conectan entre la salida del PLC, y el neutro de una fuente de alimentación de tensión alterna cuyo terminal de línea, se conecta al terminal destinado a tal fin. Al tener dos grupos de dos relés con terminales en común pueden conectarse dos fuentes de tensión distintas para manejar cargas a diferentes tensiones.

Capítulo 5

CONCLUSIONES Y TRABAJO A FUTURO

5.1. Conclusión

En este trabajo se ha logrado llevar a cabo un diseño completo de un micro PLC, incluyendo especificaciones de Hardware, Software base (sistema operativo, drivers, implementaciones de funciones standard, etc.), y formato que debe respetar el resultado de la compilación de programas definidos en los lenguajes de la norma IEC61131-3, para poder ejecutarse en equipos que se correspondan con el modelo diseñado.

Por otra parte, se definen en detalle los siguientes aspectos salientes para la implementación de un entorno de edición de programas para PLC: comportamiento esperado de la interfaz de usuario, y requisitos que debe cumplir un modelo computacional de los conceptos y lenguajes de programación descriptos en la norma citada.

Se buscó generar especificaciones que no dependieran de determinados componentes o fabricantes para el Hardware, ni de un Sistema Operativo específico para el entorno de edición de programas. Permitiendo así, construir un PLC de bajo costo de Hardware, de forma tal que pueda ser utilizado en ámbitos académicos y que, al mismo tiempo, permitan la edición de programas en forma ágil y cómoda, en particular para los lenguajes gráficos (Ladder y FBD) incluidos en la norma.

Como parte del trabajo, se desarrolló una implementación que cumple con las pautas y definiciones incluidas en las distintas partes del diseño, utilizando componentes de Hardware económicos y fácilmente obtenibles; y un entorno de programación (para desarrollar el entorno de edición de programas) que puede obtenerse desde Internet en forma gratuita.

Por lo tanto, se llega a la conclusión que los objetivos planteados al comenzar el trabajo han sido alcanzados satisfactoriamente.

La definición del diseño fue llevada a cabo en conjunto, y en muchos casos influida por la construcción de implementaciones de referencia. Se culmina este Trabajo Final con la férrea convicción que esta manera de realizarlo, permitió llegar a diseños más acabados, brindando pautas concretas y realmente útiles para el desarrollo de futuras implementaciones.

Por otro lado, el trabajo resultó sumamente arduo en varios aspectos; entre los se destacan:

- La gran cantidad de iteraciones y versiones preliminares que culminaron en la especificación presentada del modelo computacional referente a los conceptos y lenguajes de programación incluidos en la norma IEC61131-3.
- La complejidad inherente a algunos detalles de la implementación de la interfaz de usuario para lenguajes gráficos, en particular el recálculo de las posiciones ante distintas acciones que puede llevar a cabo el usuario en un segmento de programa en lenguaje Ladder.

Considerando este último aspecto, si bien algunas características del Framework Morphic ayudaron en la implementación de referencia de la GUI, la poca documentación existente sobre esta herramienta, hizo que en muchas oportunidades fuera necesario revisar el código de sus clases constituyentes, recurriendo a las capacidades avanzadas de depuración de Pharo, que permiten examinar y modificar el código de su propia interfaz en tiempo de ejecución.

Cabe resaltar que la utilización de los conceptos principales del paradigma de programación orientada a objetos, facilitó el desarrollo del diseño y su implementación de referencia. En particular, contribuyó a manejar la gran complejidad del modelo computacional de una Configuración de Software y de los programas incluidos, permitiendo la obtención de componentes bien definidos e interfaces claras entre los mismos, logrando que componentes de distintas características (como por ejemplo, los segmentos que corresponden a distintos lenguajes de programación IEC61131-3) puedan ensamblarse en forma sencilla dentro de un mismo diseño general.

Además, el desarrollo de un modelo computacional adecuado referente a los conceptos de distintos lenguajes de programación, involucró investigar y adquirir conocimientos avanzados de programación, por ejemplo; tipos de datos, definiciones, declaraciones, parseo, compilación, etc.

Otra herramienta clave para llevar a cabo esta labor, fue la experiencia adquirida en programación de Microcontroladores, siendo, el autor de este trabajo, alumno y auxiliar académico en las materias correspondientes, hecho que contribuyó en la investigación de uno de los Microcontroladores (y su IDE de desarrollo) más novedosos disponible en el mercado, utilizado para la implementación de referencia del Hardware.

Concluyendo, la realización de este Trabajo Final demandó la articulación entre los conocimientos adquiridos a lo largo de la carrera y los aprendizajes incorporados, habilitando su puesta en práctica.

5.2. Trabajo a futuro

Como labor a futuro, pueden realizarse las siguientes tareas:

- Modelado del lenguaje ST (Structured Text) y de los elementos SFC (Secuencial Function Charts), definidos en la norma IEC 61131-3.
- Modelado de la opción Retenibilidad de variables como indica la norma IEC61131-3. Este tipo de variables se mantienen en memoria del PLC incluso luego de la pérdida de alimentación del equipo PLC.
- Diseño de Entradas y Salidas Analógicas.
- Diseño de módulos de comunicaciones industriales compatibles, siguiendo las pautas expuestas en la norma IEC 61131-5.
- Diseño de un simulador de PLC para ensayar el programa de usuario sin necesidad de poseer el Hardware específico.
- Diseño de un sistema de depuración en caliente y observación de variables internas del PLC desde la computadora, mientras el programa de usuario se ejecuta en el PLC.

REFERENCIA BIBLIOGRÁFICA

1. Barry, R. (2011).*Using the FreeRTOS TM Real Time Kernel NXP LPC17xx Edition ed. 3*: Real Time Engineers Ltd.

2. Bergel, A., Cassou, D., Ducasse, S., y Laval, J. (2013). *Deep into Pharo*. Switzerland: Square Bracket Associates. Consultado en 10-10-2013 en http://pharobooks.gforge.inria.fr/PharoByExampleTwo-Eng/latest/PBE2.pdf.

3. Black, A., Ducasse, S., Nierstrasz, O. y Pollet, D. (2009). *Pharo por Ejemplo*. Switzerland: Square Bracket Associates. Consultado en 10-10-2013 en http://pharobyexample.org/es/PBE1-sp.pdf.

4. Claus Gittinger Development & Consulting (1996). *Stream classes*. Consultado en 10-10-2013 en http://live.exept.de/doc/online/english/overview/basicClasses/streams.html.

5. Ducasse, S. (2008). *[Pharo-project] (Foro) - Collapsing panes*. Consultado en 10-10-2013 en http://lists.gforge.inria.fr/pipermail/pharo-project/2011-January/040520.html.

6. Gamma, E., Helm, R., Johnson, R. Vlissides, J. (1995). *Patrones de Diseño: Elementos de software orientado a objetos reutilizable*. Madrid: PEARSON - Addison Wesley.

7. Gemtalk Systems (2011). *Pharo the collaborActibe book*. Consultado en 10-10-2013 en http://pharo.gemtalksystems.com/book/.

8. Giner, G., Rafael, J. (2008). "*Programación estructurada en C ed. 1*". Pearson Prentice Hall.

9. Gough, B. (2005). An Introduction to GCC. Network Theory Ltd.

10. IEC (2003). *IEC 61131-3 Programmable controllers - Part 3: Programming languages ed2.0*. International Electrotechnical Commission.

11. Juarez, J. (2012). *UNQ - Apuntes de cátedra: Sistemas Digitales*. Consultado en 10-10-2013 en http://iaci.unq.edu.ar/materias/sistemas_digitales/index.htm.

12. Kosik, M. (2008). *Pluggable Morphs Demo*. Consultado en 10-10-2013 en http://wiki.squeak.org/squeak/2962.

13. Lewis, D. (2008). *OSProcess*. Consultado en 10-10-2013 en http://wiki.squeak.org/squeak/708.

14. LSE (2012). *UBA - Apuntes de cátedra: Sistemas embebidos*. Consultado en 10-10-2013 en http://laboratorios.fi.uba.ar/lse/.

15. Moreno Gomez, J. (2009). *What is the difference between Sinking and Sourcing Input Configuration - PLC?*. Consultado en 10-10-2013 en http://reliability-maintenance.blogspot.com.ar/2009/07/what-is-difference-between-sinking-and.html.

16. National Instruments (2011). *Digital I/O Sinking and Sourcing*. Consultado en 10-10-2013 en http://www.ni.com/white-paper/3291/en.

17. Radioaficionados (2010). *Protección contra inversiones de polaridad*. Consultado en 10-10-2013 en http://www.radioelectronica.es/radioaficionados/19-inversion-polaridad.

18. Ritchie, D. (1993). *The Development of the C Language*. Consultado en 10-10-2013 en http://cm.bell-labs.com/cm/cs/who/dmr/chist.html.

19. Siemens Industry Online Support (2013). *¿Qué significan los términos "sumidero"(alemán: "P-schaltend") y "fuente"(alemán: "M-schaltend") en los módulos digitales de SIMATIC?*. Consultado en 10-10-2013 en http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&objId=42616517&load=treecontent&lang=es&siteid=cseus&aktprim=0&objaction=csview&extranet=standard&viewreg=WW.

20. Stallman, R., Pesch, R., Shebs, S., et al. (2011). *Debugging with GDB*. Free Software Foundation.

21. Stallman, R. (2001). *Using and Porting the GNU Compiler Collection*. Free Software Foundation. Consultado en 10-10-2013 en http://gcc.gnu.org/onlinedocs/gcc-2.95.3/gcc.html.

22. Szirty (2013). *PLC programozás sokféleképpen (La programación de PLC de muchas maneras)*. Consultado en 10/10/2013 en http://szirty.taviroda.com/lang/index.html.

23. Wikipedia (2013). *Analizador sintáctico*. Consultado en 10-10-2013 en http://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico.

24. Wikipedia (2013). *Drop-down list*. Consultado en 10-10-2013 en http://en.wikipedia.org/wiki/Drop-down_list.

25. Wikipedia (2013). *Framework*. Consultado en 10-10-2013 en http://es.wikipedia.org/w/index.php?title=Framework&section=10.

26. Wikipedia (2013). *Programación orientada a objetos*. Consultado en 10-10-2013 en http://es.wikipedia.org/wiki/Programacion_orientada_a_objetos.

27. Wikipedia (2013). *Smalltalk*. Consultado en 10-10-2013 en http://es.wikipedia.org/wiki/Smalltalk.

28. Wikipedia (2013). *Tubería (informática)*. Consultado en 10-10-2013 en http://es.wikipedia.org/w/index.php?title=Tuber%C3%ADa_%28inform%C3%A1tica%29.