UNIVERSIDAD DE ORIENTE
NÚCLEO DE SUCRE
ESCUELA DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICAS
PROGRAMA DE LA LICENCIATURA EN INFORMÁTICA

SOFTWARE MULTIPLATAFORMA DE PROGRAMACIÓN VISUAL BASADO EN EL LENGUAJE ESCALERA (LADDER) CUMPLIENDO CON LA NORMA IEC 61131-3, QUE SERVIRÁ PARA PROGRAMAR AUTÓMATAS, EN EL PROYECTO PLCLAB DE SUCRE ELECTRÓNICA C.A. (Modalidad: Pasantía)

ANDY DAVID VÁSQUEZ MACHACÓN

TRABAJO DE GRADO PRESENTADO COMO REQUISITO PARCIAL PARA OPTAR AL TÍTULO DE LICENCIADO EN INFORMÁTICA

CUMANÁ, 2008

SOFTWARE MULTIPLATAFORMA DE PROGRAMACIÓN VISUAL BASADO EN EL LENGUAJE ESCALERA (LADDER) CUMPLIENDO CON LA NORMA IEC 61131-3, QUE SERVIRÁ PARA PROGRAMAR AUTÓMATAS, EN EL PROYECTO PLCLAB DE SUCRE ELECTRÓNICA C.A.

APROBADO POR:

_____

Prof. Julio Martínez

(Asesor)

_____

Ing. Hayah García

(Asesor Institucional)

_____

(Jurado)

_____

(Jurado)

INDICE GENERAL

DEDICATORIA i
AGRADECIMIENTO ii
LISTA DE TABLAS iii
LISTA DE FIGURAS iv
RESUMEN vi
INTRODUCCIÓN 1
CAPÍTULO I. 4
PRESENTACIÓN 4
1.1. Planteamiento Del Problema 4
1.2 Alcance Y Limitaciones 5
1.2.1. Alcance 5
1.2.2. Limitaciones 5
CAPÍTULO II. 7
MARCO DE REFERENCIA 7
2.1. Marco Teórico 7
2.1.1. Antecedentes de la investigación 7
2.1.2. Antecedentes de la organización 8
2.1.3. Área de estudio 8
2.1.4. Área de investigación 8
2.1.5. Tipo de Investigación 9
2.1.6. Técnicas de recolección de datos 9
2.1.7. Automatismo 9
2.1.8. Controladores lógicos programables (PLC) 9
2.1.8.1. IEC 12
2.1.8.2. IEC 61131-3: Un recurso de programación estándar 13
2.1.8.3. Elementos comunes 14

2.1.8.3.1. Tipos de datos 14
2.1.8.3.2. Variables 14
2.1.8.3.3. Configuración, recursos y tareas 15
2.1.8.3.4. Unidades de organización de programa 16
2.1.9. Funciones 16
2.1.9.1. Bloques funcionales (BF) 17
2.1.9.2. Programas 18
2.1.9.3. Gráfico funcional secuencial (GFS) 18
2.1.9.4. Lenguajes de Programación del IEC61131-3 19
2.1.9.5. Lista de instrucciones (LI) 21
2.1.9.6. El diagramas de bloques funcionales (DBF) 21
2.1.9.7. Texto estructurado (TE) 21
2.1.9.8. Elementos de programación del lenguaje escalera 22
2.1.9.9. Elementos básicos 22
2.1.10. Variables internas y *bits* de sistema 24
2.1.10.1. Software libre 24
2.1.10.2. Decreto 3390 25
2.1.10.3. Multiplataforma 25
2.1.10.4. Perl 26
2.1.10.5. Perl/Tk 28
2.1.10.6. Pilas 28
2.1.11. Programación orientada a objeto 28
2.1.11.1 Lenguaje de programación orientado a objetos 29
2.1.11.2. Análisis orientado a objetos 29
2.1.11.3. Diseño orientado a objetos 29
2.1.11.4. Objeto 30
2.1.11.5. Herencia 30
2.1.11.6. Abstracción 31
2.1.11.7. Abstracción de datos 31

2.1.11.8. Polimorfismo 31
2.1.11.9. Encapsulamiento 31
2.1.11.10. Método 32
2.1.11.11. Eventos 32
2.1.11.12. Mensaje 32
2.1.11.13. Modelo 32
2.1.11.14. UML 33
2.1.11.15. Estereotipo 33
2.1.11.16. Casos de uso 34
2.1.11.17. Actores 35
2.1.11.18. Diagramas de casos de uso 35
2.1.11.19. Clase 38
2.1.11.19.1. Diagramas de clase 39
2.1.11.19.2. Asociación (*Association*) 40
2.1.11.19.3. Composición (*composition*) 41
2.1.11.19.4. Generalización (Generalitation) 41
2.1.11.19.5. Agregación (Agregation) 42
2.1.11.19.6. Dependencia (Dependency) 43
2.1.11.19.7. Diagramas de secuencia 43
2.2. Marco Metodológico 46
2.2.1. Metodología de la investigación 46
2.2.1.2. Metodología del área de aplicación 46
2.2.2. Modelado del negocio 47
2.2.3. Requerimientos 47
2.2.4. Análisis 48
2.2.5. Diseño 48
2.2.6. Implementación 48
2.2.7. Prueba 49
2.2.8. Administración y configuración de cambios 49

2.2.9. Ambiente 49
2.2.10. Evaluación 49
2.2.11. Fase de Inicio 51
2.2.12. Fase de elaboración 51
2.2.13. Fase de construcción 52
CAPÍTULO III 53
DESARROLLO 53
3.1. Fase de inicio 54
3.1.1. Modelado del negocio 55
3.1.2. Requerimientos 56
3.1.3. Análisis y diseño 57
3.1.4. Implementación 57
3.1.5. Pruebas 58
3.1.6. Evaluación 60
3.2. Fase de elaboración 61
3.3. Fase de construcción 62
CONCLUSIONES 65
RECOMENDACIONES 66
BIBLIOGRAFÍA 67
APÉNDICES 69
APÉNDICE A: Diagramas y documentación de los casos de uso del sistema 70
APÉNDICE B:Diagrama de clase de dominio del sistema 95
APÉNDICE C: Diagrama de clase de diseño del sistema 97
APÉNDICE D: Diagrama de clase de implementación del sistema 99
APÉNDICE E: Diagramas de Secuencia del sistema 101
APÉNDICE F: Interfaces del sistema 109
APÉNDICE G: Interfaz del primer beta del Visual Ladder 111
APÉNDICE H: Cuestionario de las entrevistas no estructuradas 113
APÉNDICE I: Diseño de los prototipos de iconos finales 114

APÉNDICE J: Cronograma Interno dentro PUDS 116
APÉNDICE K: Mapeo de clases al lenguaje Perl..... 118
APÉNDICE L: Pruebas y ajustes multiplataforma 120
APÉNDICE M: Manual de usuarios del sistema 121

DEDICATORIA

A

Dios, por guiarme y darme las fuerzas para mantenerme en el camino del bien a pesar de las tentaciones.

Mi madre Olga, por sus consejos y sacrificios, sin duda mi mayor estímulo en la vida, por enseñarme valores como la honradez el trabajo la tenacidad y fe de un futuro mejor.

Mis hermanos por todo su apoyo y unión, a pesar de todos los obstáculos que tuvimos en la vida. Gracias por ser mis hermanos y saben que siempre los tengo presentes y que los quiero mucho.

Mis sobrinos Jeimar, Jeffrey, Jeremy y Steven que han alegrado mi vida.

Mi esposa Mariangeles, que con su amor y apoyo me impulsa a ser mejor persona cada día.

Mi padre Silfredo, por inculcarme una aptitud y convicción inexorable ante cualquier reto.

La Coordinación del Programa de la Licenciatura en Informática de la UDO-SUCRE por estimularme a mejorar para ser un profesional digno.

La Universidad de Oriente, por darme la preparación para representarla en cualquier escenario.

Cualquiera que sienta a pesar de la pobreza y los obstáculos que puede hacer su vida diferente con mucha voluntad y disciplina.

AGRADECIMIENTO

A

Dios por darme la fuerza y guía en esta etapa de formación profesional, señor todos mis logros son tuyos.

Mi madre y toda mi familia, por haber sido y ser mi impulso para nunca claudicar en mi búsqueda del progreso y de un mejor horizonte para todos.

Mi Universidad, por darme la oportunidad de demostrarme que podía superarme y hacer realidad su lema; pues del pueblo vengo y al pueblo iré con los conocimientos que ella me ha brindado; mil gracias UDO.

La Coordinación del Programa de la Licenciatura en Informática de la UDO-SUCRE y a su cuerpo docente, por brindar su mejor esfuerzo en pro de la formación de un recurso humano cada vez mejor.

Todo el personal de SUTRONIX, en especial al profesor, Rodolfo Acosta, por darme la oportunidad de descubrir nuevos horizontes en el campo industrial al ver la magia que ocurre cuando se unen la informática con la electrónica.

Todos los oficiales y tropa del cuartel “Gran Mariscal Antonio José de Sucre” por su hospitalidad y ayuda durante todo el año 2000 después de llegar al estado Sucre procedente de la tragedia de Vargas.

Preparadores y compañeros de la Universidad por su ayuda y enseñanza durante toda la carrera, muchas gracias a todos.

LISTA DE TABLAS

Tabla 1. Símbolos de los elementos básicos del lenguaje escalera. 22
Tabla 2. Relación de hitos por fases 50

LISTA DE FIGURAS

Figura 1.División del estándar IEC61131-3.....14
Figura 2. Configuración, recursos y tareas. 15
Figura 3. Etapas de gráfico funcional secuencial.....18
Figura 4. Lenguajes de programación gráficos y literales. 21
Figura 5. Icono para el modelaje de un caso de uso en UML.....35
Figura 6. Icono para el modelaje de un actor en UML. 35
Figura 7. Diagrama de casos de uso.....36
Figura 8. Asociación entre el actor y el caso de uso. 37
Figura 9. Relación de inclusión entre casos de uso.....37
Figura 10. Relación de extensión entre casos de uso. 37
Figura 11. Relación de generalización entre casos de uso. 38
Figura 12. Icono que representa una clase en UML.....39
Figura 13. Diagrama de clases. 40
Figura 14. Relación de asociación entre clases. 41
Figura 15. Relación de composición entre clases. 41
Figura 16. Relación de generalización entre clases. 42
Figura 18. Relación de dependencia entre clases. 43
Figura 19. Diagrama de secuencia. 44
Figura 20. Ciclo Iterativo de PUDS. 47
Figura 21. Representación gráfica del PUDS. 50
Figura 22. Cronograma para el desarrollo del sistema. 54
Figura 23. Diagrama de casos de uso preliminar. 55
Figura 24. Explosión del caso de uso del sistema. 57
Figura 25. Primer prototipo de iconos del sistema. 58
Figura 26. Primeras pruebas para el ajuste multiplataforma. 59

Figura 27. Cronograma interno para las iteraciones en PUDS. 61
Figura 28. Interfaz final del primer beta del sistema. 64

RESUMEN

Se desarrolló, un software multiplataforma de programación visual basado en el lenguaje escalera (*ladder*) cumpliendo con la norma IEC 61131-3, el mismo, se realizó siguiendo las fases del Proceso Unificado de Desarrollo de *Software* (PUDS). Durante la aplicación metodológica se llevó a cabo todo el levantamiento de información necesario para recolectar, los requisitos del sistema. El PUDS, fue la base metodológica de ingeniería que permitió definir el esquema disciplinado para asignar las tareas y responsabilidades en el proceso de desarrollo. Por medio de la metodología, se trabajó con una serie de ciclos ejecutados repetidamente; cada ciclo o iteración estuvo divido en fases. El contenido de las iteraciones cambiaba para acomodarse a los objetivos de cada fase, adicionalmente se utilizó el lenguaje unificado de modelado (UML), para generar todos los planos arquitectónicos del sistema. En la etapa final, el volumen de la programación fue bajando a medida que se alcanzaban los objetivos y simplemente, se aumentaron las pruebas y la interacción con los técnicos de Sucre Electrónica C.A de forma mucho más permanente y así se cumplieron todas las fases de la metodología. El *software,* fue desarrollado con el lenguaje *Perl* en su versión 5.7.8, probado y ajustado todo su código fuente en plataformas *Windows* y *Linux* de forma estable y en *Solaris* y *Mac* de forma muy esporádica por no ser el objeto a corto plazo del PLCLAB, pero donde igual puede operar. El sistema se ajusta a lo solicitado por la empresa y permite crear la base del proyecto PLCLAB en su primera etapa, donde los analista podrán generar sus programas y proyectos en lenguaje ladder y en un futuro cuando se construya el PLC se podrá integrar el *hardware* con el *software.*

INTRODUCCIÓN

Desde la revolución computacional en la década de los 50, las organizaciones han tratado de utilizar las tecnologías informáticas con el fin de automatizar sus actividades, para así obtener fiabilidad en las operaciones, rapidez, reducción en los costos de producción y seguridad. Debido a estas demandas corporativas, comienzan a instituirse grupos de programadores a nivel mundial, que con el pasar de los años se les definió como la industria del *software* (Gates, 2000). Con el tiempo la industria del *software* se ha encargado de resolver problemas específicos en determinadas organizaciones, utilizando la ingeniería de desarrollo de *software* (Pressman, 2002).

La ingeniería de desarrollo de *software*, es un enfoque sistemático de producción, operación, mantenimiento y retiro del *software*, regido por un conjunto de etapas parcialmente ordenadas con intención de lograr un objetivo, en este caso la obtención de un producto de *software* de calidad (Pressman, 2002).

Con el avance tecnológico, la ingeniería de *software* ha tenido algunos cambios significativos en la concepción de los paradigmas, propósitos comerciales y formas de programación. En la actualidad las organizaciones requieren aplicaciones fáciles de usar y que sean compatibles con múltiples plataformas, tomando en cuenta que el término es usado para referirse a los programas, que puedan funcionar en diversos entornos. Por ejemplo, una aplicación multiplataforma podría ejecutarse en *Windows* en un procesador Intel x86, en *GNU/Linux* en un procesador Intel x86, en *Mac OS X* en uno Intel x86, en un *Sun SPARC* con Solaris o en un *PowerPC*, esa característica de portabilidad es muy solicitada sobre todo en las industrias automatizadas que manejan una gran cantidad de equipos interconectados que en su mayoría son manejados por autómatas programables. A los autómatas también se les conoce como

Controladores Lógicos Programables (PLC), Los PLC son un *hardware* similares al de un computador, que ejecuta las instrucciones que se graban en él. Son muy utilizados en la automatización industrial para abrir puertas, encender bombas, activar pozos petroleros, sincronizar motores de producción entre otras. A lo largo de la historia, muchos fabricantes de PLC comercializaban sus productos con un lenguaje de programación propio, situación que dificultaba el trabajo con distintas marcas de autómatas en las empresas. Es por eso que la Comisión Electrotécnica Internacional (IEC), realizó un esfuerzo por estandarizar estos lenguajes, creando un conjunto de normas denominadas como IEC 61131-3, con la cual se pueden expresar aquellos elementos esenciales para manipulación de autómatas a través de cuatro lenguajes.

Entre los lenguajes regidos por la norma 61131 del IEC en su apartado 3, está el diagrama en escalera (*ladder diagram*) conocido también como diagrama de contactos. El diagrama en escalera, es uno de los lenguajes preferidos por los operadores para la programación de autómatas, gracias a la facilidad y a la sencillez que brinda (Montejo, 2005). Es por ello que dentro de todas las especificaciones del proyecto PLCLAB, SUTRONIX seleccionó a el lenguaje escalera para desarrollar el *software* multiplataforma de programación visual, que permite crear proyectos y programas a través con una impecable asociación gráfica de los elementos escalera, manejar los tipos de datos y variables de la norma, construir un archivo de conexión con el PLC que actualmente está en construcción, y generar un código intermedio en lenguajes de alto nivel.

Todo esto, le permitió a la empresa completar la primera fase del proyecto PLCLAB que consistía en desarrollar un *software* propio de manipulación para sus autómatas con tecnología nacional, apegado al decreto 3.390 y que permita una fácil portabilidad a cualquier plataforma; para así colaborar en el desarrollo tecnológico del estado Sucre y de Venezuela.

El presente trabajo está estructurado en tres capítulos, los cuales se describen a continuación:

Capítulo I: Presentación. Este capítulo describe los requerimientos de la empresa o el problema que fue abordado durante la investigación, el alcance y las limitaciones presentadas durante el desarrollo de la misma.

Capítulo II: Marco de referencia. Presenta las bases teóricas para el soporte de la investigación y la metodología utilizada para lograr el objetivo propuesto.

Capítulo III: Desarrollo. Se desarrolla, de manera detallada, cada una de las fases de la metodología aplicada, así como la codificación y documentación del *software*.

Finalmente, se presentan las conclusiones y las recomendaciones del trabajo realizado.

CAPÍTULO I.
PRESENTACIÓN

1.1. Planteamiento Del Problema

Desde hace muchos años, las empresas que prestan servicio en el área de automatismo industrial han tenido que trabajar con controladores lógicos programables de distintas marcas, lo que las obliga a manejar cada programa de manipulación de los fabricantes más populares, situación que requiere una gran inversión tanto en personal, como en dinero para la adquisición de estos PLC, que en su mayoría no cumplen con todos los requisitos de las empresas del área, además el *software* de los fabricantes es de código cerrado lo que hace prácticamente imposible modificarlos para ajustarlos a las nuevas exigencias de cualquier prestador de servicio, esta situación motivó a SUTRONIX a partir del año 2005 a arrancar con el proyecto PLCLAB.

A simple vista se, puede confundir el proyecto, con la fabricación de un controlador lógico programable más, pero la situación real y lo que persigue SUTRONIX, es un producto con tecnología nacional a un costo accesible y que el programa del PLC sea de código abierto, para que así los propietarios del autómata puedan modificarlo y ajustarlo cuando ellos lo deseen a sus necesidades, requerimiento que SUTRONIX siempre anheló cuando estrictamente se dedicaba a comprar equipos y que ahora como futuro fabricante se propone cumplir.

Además, de que la mayoría de los fabricantes producen el *software* de manipulación con código cerrado, únicamente desarrollan para plataformas *Windows*, lo que presenta un inconveniente para todas aquellas empresas que deseen migrar

apegadas al decreto 3390, firmado por el presidente Hugo Chávez el 23 de diciembre de 2004 y publicado en Gaceta Oficial de la República Bolivariana de Venezuela N° 38.095 de fecha 28 de diciembre de 2004 (Montejo, 2005), Por tal motivo, SUTRONIX necesitaba en su primera etapa que el *software* de sus PLC pudiese ser ejecutado en múltiples plataformas sin amarrarse a ningún sistema operativo y así facilitar, la migración de toda la parte automatizada de cualquier empresa a entornos *Linux*, *Solaris* o *Mac*. También que el *software se* diseñara en las herramientas existentes de forma que los usuarios no sintieran un cambio muy marcado a lo que están acostumbrados a usar y que el lienzo para los programas fuese de (15) filas por (16) columnas. Es por ello que se desarrolló una aplicación que cumpla todos esos requerimientos y así completar la primera etapa del proyecto con éxito.

1.2 Alcance Y Limitaciones

1.2.1. Alcance

El *software* fue desarrollado para cumplir con la primera etapa del PLCLAB, que persigue sembrar las bases a nivel de aplicación, que soporten la visión multiplataforma del proyecto, está orientado a todos aquellos técnicos, usuarios, universidades, liceos y empresas, que necesiten enseñar y construir lógica de programas con el lenguaje gráfico escalera para la manipulación de autómatas desarrollados con tecnología venezolana por SUTRONIX, o por cualquier fabricante que tome el código y lo ajuste a su propio PLC.

1.2.2. Limitaciones

Al momento de la investigación no se contaba con el *hardware* del PLC venezolano, lo que obligó a redefinir el alcance de la investigación.

La falta de conocimiento de muchos términos del área electrónica influyó al inicio de la investigación en la compresión del verdadero alcance del *software*.

CAPÍTULO II.
MARCO DE REFERENCIA

2.1. Marco Teórico

2.1.1. Antecedentes de la investigación

En el área de automatismo industrial desde hace varios años se manejaban muchos lenguajes de programación, prácticamente uno por cada fabricante de PLC, lo que dificultaba trabajar con equipos de distintas marcas en una misma empresa automatizada, situación que implicaba un gasto en capacitación y la descoordinación de algunos procesos, es por ello que la Comisión Electrotécnica Internacional (IEC), realizó un esfuerzo por estandarizar estos lenguajes, creando un conjunto de normas denominadas IEC 1131-C, que luego de algunas revisiones, finalmente se trasforma en la norma IEC 61131-3, con la cual se pueden expresar aquellos elementos esenciales para manipulación de autómatas (Balcells y Romeral, 1998). Una vez definidas estas normas por la IEC, cualquier fabricante tiene los mecanismos para construir un PLC con su lenguaje de programación estandarizado y así mantener los lineamientos de automatización industrial moderna. Estas normas, son aplicadas y utilizadas por los mayores fabricantes a nivel mundial como por ejemplo *General Motors, Mitsubishi, General Electric, Stanley etc.*

En Venezuela han existido proyectos de fabricación de PLC, por ejemplo en el Instituto Universitario de Tecnología Industrial de Cumaná, el departamento de electrónica, presentó un trabajo de investigación para la fabricación de un PLC, pero, realmente nunca se ha hablado de la construcción de un software para manejar autómatas, este tipo de proyecto principalmente lo han desarrollado los grandes

fabricantes de autómatas.

2.1.2. Antecedentes de la organización

Sucre Electrónica Compañía Anónima, es una empresa fundada en el año 1987 por el profesor Rodolfo Acosta y que desde su creación ha estado en la búsqueda y utilización de modernas técnicas de desarrollo industrial para proveer servicios y generar tecnología propia en todo el oriente del país. Sucre electrónica C.A, es una empresa que se ha trazado como meta comandar el mercado de consumo en el área de automatismo industrial; para lograr esa meta, tiene por misión, hacer de su empresa una empresa más productiva y competitiva.

2.1.3. Área de estudio

El área de estudio está determinada, por los procesos y operaciones básicas que se realizan a nivel de sistematización con los autómatas programables, de acuerdo con las características que presenta la investigación este proyecto se ubica en el área de desarrollo de software.

2.1.4. Área de investigación

Debido a que en el proyecto PLCLAB, el *software* desarrollado será utilizado para controlar y programar de una forma visual y más sencilla los autómatas con las nociones gráficas de la actualidad, éste trabajo se encuentra en el área *software* multiplataforma y autómatas programables.

2.1.5. Tipo de Investigación

Este trabajo se ajusta a una investigación descriptiva, ya que se hará un análisis, interpretación y registro acerca de la naturaleza de los autómatas tomando en cuenta las realidades del contexto y para que usuario se destina, de igual forma, es una investigación de forma aplicada, debido a que confronta la teoría de desarrollo de sistema con la realidad de construir el *software* con las especificaciones de SUTRONIX.

2.1.6. Técnicas de recolección de datos

Las herramientas que se utilizaron para la recolección de datos durante el desarrollo de este trabajo fueron la observación directa y participativa y las entrevistas estructuradas (Apéndice H) dirigidas al personal de electrónica y automatismo de SUTRONIX así como la revisión bibliográfica.

2.1.7. Automatismo

Desarrollo de un proceso o funcionamiento de un mecanismo por sí solo, es la cualidad de lo que es automático.

2.1.8. Controladores lógicos programables (PLC)

Son un sistema, cuya conducta en un instante cualquiera queda definida por la totalidad de los valores que adoptan un conjunto de variables, algunas de las cuales están asociadas con el estado interno del autómata, mientras otras lo están con sus variables de entrada ó salida. Las entradas junto con el estado interno del autómata determinan una salida y un nuevo estado (Balcells y Romeral, 1998).

Un autómata programable, se puede considerar como un sistema basado en un microprocesador, siendo sus partes fundamentales la unidad central de proceso (CPU), la memoria y el sistema de entradas y salidas (E/S).

La CPU realiza el control interno y externo del autómata y la interpretación de las instrucciones del programa. A partir de las instrucciones almacenadas en la memoria y de los datos que recibe de las entradas, genera las señales de las salidas. La memoria se divide en dos bloques, la memoria de solo lectura o ROM (*Read Only Memory*) y la memoria de lectura y escritura o RAM (*Random Access Memory*).

En la memoria ROM se almacenan programas para el correcto funcionamiento del sistema, como el programa de comprobación de la puesta en marcha y el programa de exploración de la memoria RAM (Balcells y Romeral, 1998).

La memoria RAM a su vez puede dividirse en dos áreas:

- Memoria de datos, en la que se almacena la información de los estados de las entradas y salidas y de variables internas del controlador lógico programables.
- Memoria de usuario, en la que se almacena el programa con el que trabajará el autómata.

El sistema de entradas y salidas recoge la información del proceso controlado (entradas) y envía las acciones de control del mismo (salidas). Los dispositivos de entrada pueden ser pulsadores, interruptores, termostatos, presostatos, detectores de nivel, detectores de proximidad, contactos auxiliares, etc.

Por su parte, los dispositivos de salida son también muy variados: Pilotos indicadores, relés, contactores, arrancadores de motores, válvulas, entre otras.

Sistema de entradas y salidas. En general, las entradas y salidas (E/S) de un autómata pueden ser discretas, analógicas, numéricas o especiales.

Las E/S discretas se caracterizan por presentar dos estados diferenciados: presencia o ausencia de tensión, relé abierto o cerrado. Su estado se puede visualizar mediante indicadores tipo LED que se iluminan cuando hay señal en la entrada o cuando se activa la salida. Los dispositivos de salida más frecuentes son relés y transistores (Balcells y Romeral, 1998).

Las E/S analógicas tienen como función la conversión de una magnitud analógica (tensión o corriente) equivalente a una magnitud física (temperatura, presión, grado de acidez, etc.) en una expresión binaria de 11, 12 o más bits, dependiendo de la precisión deseada. Esto se realiza mediante conversores analógico-digitales (ADC's).

Las E/S numéricas permiten la adquisición o generación de información a nivel numérico, en códigos, La información numérica puede ser entrada mediante dispositivos electrónicos digitales apropiados. Por su parte, las salidas numéricas suministran información para ser utilizada en dispositivos visualizadores (de 7 segmentos) u otros equipos digitales.

Por último, las E/S especiales se utilizan en procesos en los que con las anteriores E/S vistas son poco efectivas, bien porque es necesario un gran número de elementos adicionales, bien porque el programa necesita de muchas instrucciones. Entre las más importantes están:

- Entradas para termopar y termorresistencia: Para el control de temperaturas.
- Salidas de trenes de impulso: Para el control de motores paso a paso (PAP).

- Entradas y salidas de regulación P+I+D (Proporcional + Integral + Derivativo): Para procesos de regulación de alta precisión (Balcells y Romeral, 1998).
- Salidas ASCII: Para la comunicación con periféricos inteligentes (equipo de programación, impresora, PC, etc.).

2.1.8.1. IEC

Comisión electrotécnica Internacional (CEI o *IEC*, por sus siglas del idioma inglés *International Electrotechnical Commission*). Es una organización de normalización en los campos eléctrico, electrónico y tecnologías relacionadas. Numerosas normas se desarrollan conjuntamente con la ISO (normas ISO/IEC).

La IEC, fundada en 1906 y cuyo primer presidente fue Lord Kelvin, tenía su sede en Londres hasta que en 1948 se trasladó a Ginebra. Integrada por los organismos nacionales de normalización, en las áreas indicadas, de los países miembros (Tiegelkamp y Heinz 1995).

A la IEC se le debe el desarrollo y difusión de los estándares para algunas unidades de medida, particularmente el *gauss*, *hercio* y *weber*; así como la primera propuesta de un sistema de unidades estándar, el sistema *Giorgi*, que con el tiempo se convertiría en el sistema internacional de unidades.

En 1938, el organismo publicó el primer diccionario internacional (*International Electrotechnical Vocabulary*) con el propósito de unificar la terminología eléctrica, esfuerzo que se ha mantenido durante el transcurso del tiempo, siendo el vocabulario electrotécnico internacional un importante referente para las empresas del sector (Montejo, 2005).

2.1.8.2. IEC 61131-3: Un recurso de programación estándar

En la actualidad, aún siguen persistiendo sistemas de control específicos del fabricante, con programación dependiente y conexión compleja entre distintos sistemas de control. Esto significa para el usuario costos elevados, escasa flexibilidad y falta de normalización en las soluciones al control industrial.

IEC 61131, es el primer paso en la estandarización de los autómatas programables y sus periféricos, incluyendo los lenguajes de programación que se deben utilizar. Esta norma se divide en cinco partes:

Parte 1: Vista general.
Parte 2: *Hardware*.
Parte 3: Lenguaje de programación.
Parte 4: Guías de usuario.
Parte 5: Comunicación.

IEC 61131-3, pretende ser la base real para estandarizar los lenguajes de programación en la automatización industrial, haciendo el trabajo independiente de cualquier compañía (Tiegelkamp y Heinz 1995).

IEC 61131-3, es el resultado del gran esfuerzo realizado por 7 multinacionales a los que se añaden muchos años de experiencia en el campo de la automatización industrial. Incluye 200 páginas de texto aproximadamente, con más de 60 tablas. IEC 61131-3, son las especificaciones de la sintáxis y semántica de los lenguajes de programación, incluyendo el modelo de *software* y la estructura del lenguaje.

Otra visión distinta es dividir el estándar en dos partes (Figura 1):
Elementos comunes.

Lenguajes de programación.

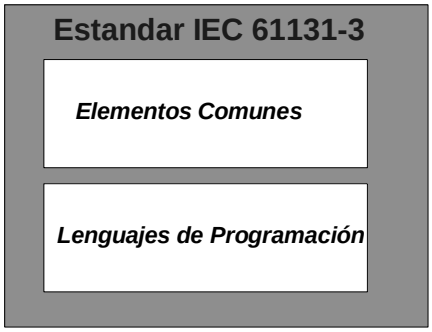


Figura 1.División del estándar IEC61131-3.

2.1.8.3. Elementos comunes

2.1.8.3.1. Tipos de datos

Dentro de los elementos comunes, se definen los tipos de datos. Los tipos de datos previenen de errores en una fase inicial, como por ejemplo, la división de un dato tipo fecha por un número entero. Los tipos comunes de datos son: variables *booleanas*, números enteros, números reales, *byte* y palabras, pero también los derivados: fechas, horas del día y cadenas (*strings*).

2.1.8.3.2. Variables

Las variables permiten identificar los objetos de datos cuyos contenidos pueden cambiar, por ejemplo, los datos asociados a entradas, salidas o a la memoria del autómata programable. Una variable se puede declarar, como uno de los tipos de datos elementales definidos o como uno de los tipos de datos derivados. De este modo, se crea un alto nivel de independencia con el hardware, favoreciendo la reusabilidad del *software*. La extensión de las variables está normalmente limitada a la unidad de organización en la cual han sido declaradas como locales. Esto significa,

que sus nombres pueden ser reutilizados en otras partes sin conflictos, eliminando una frecuente fuente de errores.

2.1.8.3.3. Configuración, recursos y tareas

Para entender esto mejor, vamos a ver el modelo de *software*, que define IEC 61131-3 (Figura 2):

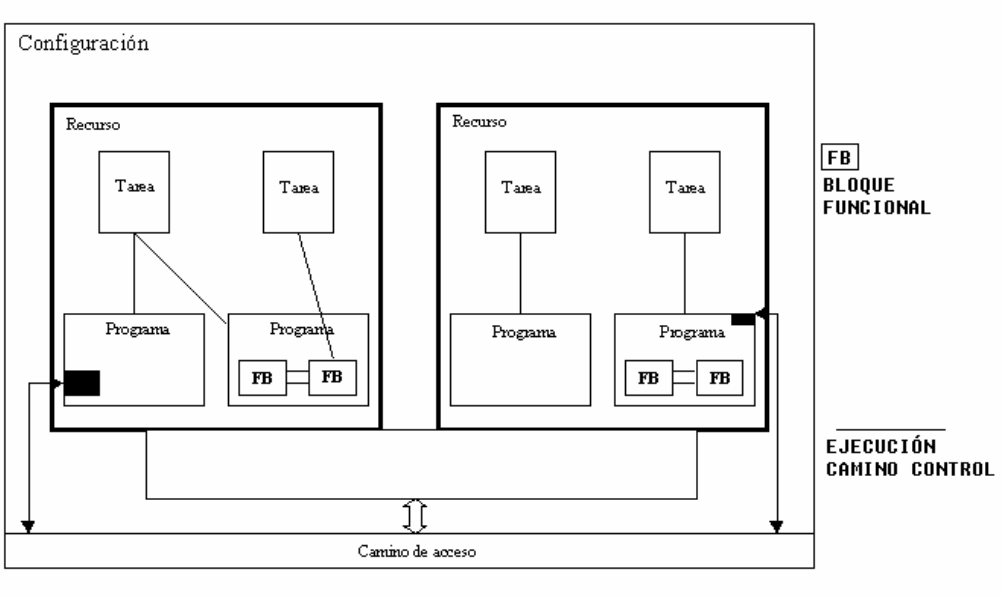


Figura 2. Configuración, recursos y tareas.

Al más alto nivel, el elemento *software* requerido para solucionar un problema de control particular puede ser formulado como una 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 I/O y otras capacidades del sistema.

Dentro de una configuración, se pueden definir uno o más recursos. Se puede entender el recurso como un procesador capaz de ejecutar programas IEC.

Con un recurso, pueden estar definidas una o más tareas. Las tareas controlan la ejecución de un conjunto de programas y/o bloques de función. Cada una de ellas puede ser ejecutada periódicamente o por una señal de disparo especificada, como el cambio de estado de una variable.

Los programas, están diseñados a partir de un diferente número de elementos de *software* escrito en algunos de los distintos lenguajes definidos en IEC 61131-3. Típicamente, un programa es una interacción de funciones y bloques funcionales, con capacidad para intercambiar datos. Funciones y bloques funcionales son las partes básicas de construcción de un programa, que contienen una declaración de datos y variables y un conjunto de instrucciones.

Comparado esto con un PLC convencional, éste contiene un solo recurso, ejecutando una tarea que controla un único programa de manera cíclica. IEC 61131-3 incluye la posibilidad de disponer de estructuras más complejas. El futuro que incluye multiprocesamiento y gestión de programas por eventos. Observar simplemente las características de los sistemas distribuidos o los sistemas de control de tiempo real. IEC 61131-3 está disponible para un amplio rango de aplicaciones, sin tener que conocer otros lenguajes de programación adicionales.

2.1.8.3.4. Unidades de organización de programa

Dentro de IEC 61131-3, los programas, bloques funcionales y funciones se denominan Unidades de Organización de Programas, UOP.

2.1.9. Funciones

IEC 61131-3 especifica funciones estándar y funciones definidas por usuario.

Las funciones estándar son por ejemplo ADD (suma), ABS (valor absoluto), Las funciones definidas por usuario, una vez implementadas pueden ser usadas indefinidamente en cualquier UOP.

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).

2.1.9.1. Bloques funcionales (BF)

Los bloques funcionales son los equivalentes de los circuitos integrados, que representan funciones de control especializadas.

Los BF contienen tanto datos como instrucciones, y además pueden guardar los valores de las variables (que es una de las diferencias con las funciones). Tienen un interfaz de entradas y salidas bien definido y un código interno oculto, como un circuito integrado o una caja negra. De este modo, establecen una clara separación entre los diferentes niveles de programadores, o el personal de mantenimiento. Un lazo de control de temperatura, es un excelente ejemplo de bloque funcional. Una vez definido, puede ser usado una y otra vez, en el mismo programa, en diferentes programas o en distintos proyectos. Esto lo hace altamente reutilizable. Los bloques funcionales, pueden ser escritos por el usuario en alguno de los lenguajes de la norma IEC, pero también existen FB estándar (biestables, detección de flancos, contadores, temporizadores, etc.). Existe la posibilidad de ser llamados múltiples veces creando copias del bloque funcional que se denominan instancias. Cada instancia llevará asociado un identificador y una estructura de datos que contenga sus variables de salida.

2.1.9.2. Programas

Los programas, son un conjunto lógico de todos los elementos y construcciones del lenguaje de programación que son necesarios para el tratamiento de señal previsto que se requiere para el control de una máquina o proceso mediante el sistema de autómata programable. Un programa puede contener, aparte de la declaración de tipos de datos, variables y su código interno, distintas instancias de funciones y bloques funcionales.

2.1.9.3. Gráfico funcional secuencial (GFS)

GFS describe gráficamente el comportamiento secuencial de un programa de control. Esta definición deriva de las Redes de *Petri* y *Grafcet* (IEC 848), con las modificaciones adecuadas para convertir las representaciones de una norma de documentación en un conjunto de elementos de control de ejecución para una UOP de un autómata programable (Figura 3).

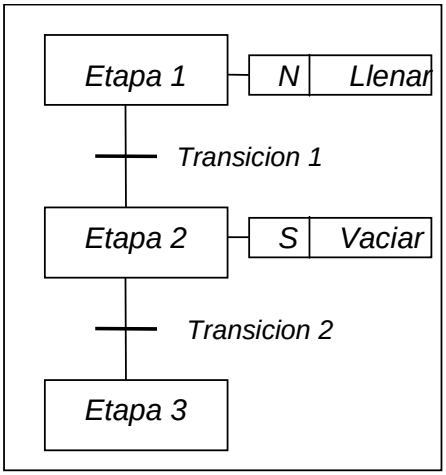


Figura 3. Etapas de gráfico funcional secuencial.

GFS ayuda a estructurar la organización interna de un programa, y a

descomponer un problema en partes manejables, manteniendo simultáneamente una visión global.

Los elementos del GFS proporcionan un medio para subdividir una POU de un autómata programable en un conjunto de etapas y transiciones interconectadas por medio de enlaces directos.

Cada etapa lleva asociados, un conjunto de bloques de acción y a cada transición va asociada una condición de transición que cuando 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 GFS. Dado que los elementos del GFS requieren almacenar información, las únicas POU que se pueden estructurar utilizando estos elementos son los bloques funcionales y los programas.

Se pueden usar secuencias alternativas y paralelas, comúnmente utilizadas en muchas aplicaciones. Debido a su estructura general, de sencilla comprensión, GFS permite la transmisión de información entre distintas personas con distintos niveles de preparación y responsabilidad dentro de la empresa.

2.1.9.4. Lenguajes de Programación del IEC61131-3

Se definen cuatro lenguajes de programación normalizados. Esto significa que su sintáxis y semántica ha sido definida, no permitiendo particularidades distintivas (dialectos). Una vez aprendidos se podrá usar una amplia variedad de sistemas basados en esta norma.

Los lenguajes consisten en dos de tipo literal y dos de tipo gráfico (figura 4).

Literales:

Lista de instrucciones (LI).

Texto estructurado (TE).

Gráficos:

Diagrama de contactos (DC).

Diagrama de bloques funcionales (DBF).

En la siguiente figura, los cuatro programas describen la misma acción. La elección del lenguaje de programación depende de: los conocimientos del programador, el problema a tratar, el nivel de descripción del proceso, la estructura del sistema de control.

Los cuatros lenguajes están interrelacionados y permiten su empleo para resolver conjuntamente un problema común según la experiencia del usuario.

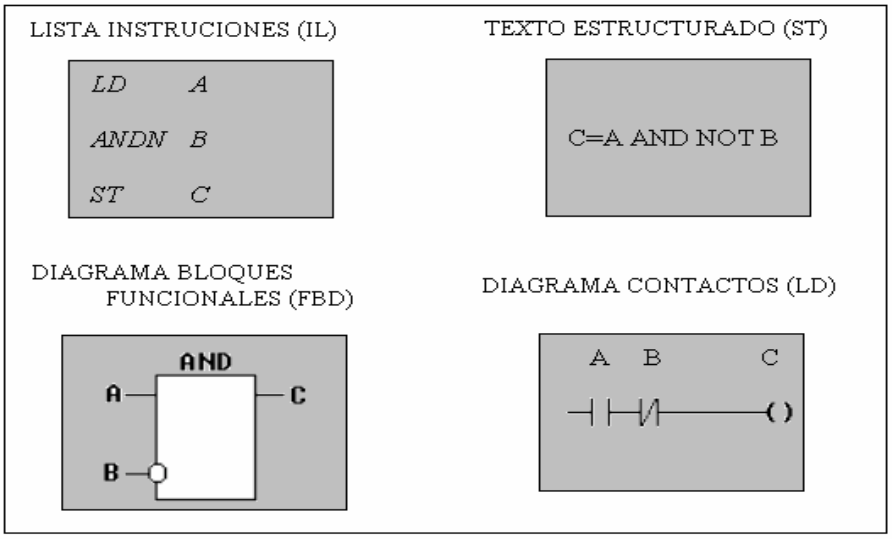


Figura 4. Lenguajes de programación gráficos y literales.

2.1.9.5. Lista de instrucciones (LI)

Es el modelo de lenguaje ensamblador basado un acumulador simple; procede del alemán *Anweisungliste*, AWL.

2.1.9.6. El diagramas de bloques funcionales (DBF)

Es muy común en aplicaciones que implican flujo de información o datos entre componentes de control. Las funciones y bloques funcionales aparecen como circuitos integrados y es ampliamente utilizado en Europa.

2.1.9.7. Texto estructurado (TE)

Es un lenguaje de alto nivel con orígenes en *Ada*, *Pascal* y C; puede ser utilizado para codificar expresiones complejas e instrucciones anidadas; este lenguaje dispone de estructuras para bucles.

El lenguaje *Ladder* (DL), también denominado lenguaje de contactos o en escalera, es un lenguaje de programación gráfico muy popular dentro de los autómatas programables debido a que está basado en los esquemas eléctricos de control clásicos. De este modo, con los conocimientos que todo técnico eléctrico posee, es muy fácil adaptarse a la programación en este tipo de lenguaje.

2.1.9.8. Elementos de programación del lenguaje escalera

Para programar un autómata con *ladder*, además de estar familiarizado con las reglas de los circuitos de conmutación, es necesario conocer cada uno de los elementos de que consta este lenguaje (Tiegelkamp y Heinz 1995).

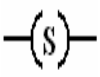
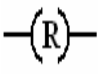
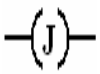
A continuación se describen de modo general los más comunes.

2.1.9.9. Elementos básicos

Tabla 1. Símbolos de los elementos básicos del lenguaje escalera.

Símbolo	Nombre	Descripción
	Contacto NA	Se activa cuando hay un uno lógico en el elemento que representa, esto es, una entrada (para captar información del proceso a controlar), una variable interna o un *bit* de sistema.
	Contacto NC	Su función es similar al contacto NA anterior, pero en este caso se activa cuando hay un cero lógico, cosa que deberá de tenerse muy en cuenta a la hora de su utilización.

Continuación Tabla 1. Símbolos de los elementos básicos del lenguaje escalera.

Símbolo	Nombre	Descripción
	Bobina NA	Se activa cuando la combinación que hay a su entrada (izquierda) da un uno lógico. Su activación equivale a decir que tiene un uno lógico. Suele representar elementos de salida, aunque a veces puede hacer el papel de variable interna.
	Bobina NC	Se activa cuando la combinación que hay a su entrada (izquierda) da un cero lógico. Su activación equivale a decir que tiene un cero lógico. Su comportamiento es complementario al de la bobina NA.
	Bobina *SET*	Una vez activa (puesta a 1) no se puede desactivar (puesta a 0) si no es por su correspondiente bobina en *RESET*. Sirve para memorizar bits y usada junto con la bobina *RESET* dan una enorme potencia.
	Bobina *RESET*	Permite desactivar una bobina *SET* previamente activada.
	Bobina *JUMP*	Permite saltarse instrucciones del programa e ir directamente a la etiqueta que se desee. Sirve para realizar subprogramas.

2.1.10. Variables internas y *bits* de sistema

Las variables internas son bits auxiliares que pueden ser usados según convenga sin necesidad de que representen ningún elemento del autómata. Se suele indicar mediante los caracteres B ó M y tienen tanto bobinas como contactos asociados a las mismas de los tipos vistos en el punto anterior. Su número de identificación suele oscilar, en general, entre 0 y 255. Su utilidad fundamental es la de almacenar información intermedia para simplificar esquemas y programación. Los *bits* de sistema, son contactos que el propio autómata activa cuando conviene o cuando se dan unas circunstancias determinadas. Existe una gran variedad, siendo los más importantes los de arranque y los de reloj, que permiten que empiece la ejecución desde un sitio en concreto y formar una base de tiempos respectivamente. Su nomenclatura es muy diversa, dependiendo siempre del tipo de autómata y fabricante (Montejo, 2005).

2.1.10.1. Software libre

Software libre (en inglés *free software*) es el *software* que, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente. Este suele estar disponible gratuitamente, pero no hay que asociar el término libre con gratuito; sin embargo no es obligatorio que sea así y, aunque conserve su carácter de libre, puede ser vendido comercialmente. Análogamente, el *software* gratis o gratuito (denominado usualmente *freeware*) incluye en algunas ocasiones el código fuente; sin embargo, este tipo de *software* no es libre en el mismo sentido que el *software* libre, al menos que se garanticen los derechos de modificación y redistribución de dichas versiones modificadas del programa.

No debe confundirse "*software* libre" con el de dominio público. Éste último, es

aquél por el que no es necesario solicitar ninguna licencia y cuyos derechos de explotación son para toda la humanidad, porque pertenece a todos por igual. Cualquiera puede hacer uso de él, siempre con fines legales y consignando su autoría original. Este *software* sería aquél cuyo autor lo dona a la humanidad o cuyos derechos de autor han expirado. Si un autor condiciona su uso bajo una licencia, por muy débil que sea, ya no es dominio público. En resumen, un programa de dominio público es la pura definición de la libertad de usufructo de una propiedad intelectual que tiene la humanidad porque así lo ha decidido su autor o la ley tras un plazo contado desde la muerte de éste, habitualmente 70 años (Stallman, 2002).

2.1.10.2. Decreto 3390

Firmado por el presidente Hugo Chávez el 23 de Diciembre de 2004 y publicado en Gaceta Oficial de la República Bolivariana de Venezuela N°. 38.095 de fecha 28 de Diciembre de 2004, el cual dispone que, la Administración Pública Nacional empleará prioritariamente *software* libre desarrollado con estándares abiertos en sus sistemas, proyectos y servicios informáticos (Córdova, 2004).

2.1.10.3. Multiplataforma

Es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de *software*, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en *Windows* en un procesador x86, en *GNU/Linux* en un procesador x86, en *Mac OS X* en uno x86, en un *Sun SPARC* con Solaris o en un *PowerPC*.

Una plataforma es una combinación de *hardware* y *software* usada para ejecutar aplicaciones; en su forma más simple consiste únicamente de un sistema operativo, una arquitectura, o una combinación de ambos. La plataforma más

conocida es probablemente *Microsoft Windows* en una arquitectura x86; otras plataformas conocidas son *GNU/Linux* y *Mac OS X* (que ya de por sí son multiplataforma). Hay, por otro lado, aparatos como celulares que, a pesar de ser plataformas informáticas, no se consideran usualmente como tales (Wyke y Thomas, 2001).

2.1.10.4. Perl

Lenguaje práctico para la extracción de reportes, es un lenguaje de programación diseñado por Larry Wall creado en 1987. *Perl* toma características del C, del lenguaje interpretado *Shell*, *AWK*, *sed*, *Lisp* y, en un grado inferior de muchos otros lenguajes de programación.

Estructuralmente, *Perl* está basado en un estilo de bloques como los del C o *AWK*, y fue ampliamente adoptado por su destreza en el procesador de texto y por no tener ninguna de las limitaciones de los otros lenguajes de *script* además este lenguaje corre en múltiples plataformas (Wyke y Thomas, 2001).

Larry Wall comenzó a trabajar en *Perl* en 1987 mientras trabajaba como programador en *Unisys* y anunció la versión 1.0 en el grupo de noticias *comp.sources.misc* el 18 de Diciembre de 1987. El lenguaje se expandió rápidamente en los siguientes años. *Perl* 2, liberado en 1988, aportó un mejor motor de expresiones regulares. *Perl* 3, liberado en 1989, añadió soporte para datos binarios.

Hasta 1991 la única documentación de Perl era una simple (y cada vez más larga) página de manual *Unix*. En 1991 se publicó *Programming Perl* (el libro del dromedario) y se convirtió en la referencia de facto del lenguaje. Al mismo tiempo, el número de versión de *Perl* saltó a 4, no por marcar un gran cambio en el lenguaje, sino por identificar a la versión que estaba documentada en el libro.

Perl 4 trajo consigo una serie de lanzamientos de mantenimiento, culminando en *Perl* 4.036 en 1993. En este punto, Larry Wall abandonó *Perl* 4 para comenzar a trabajar en *Perl* 5. *Perl* 4 se quedaría en esa versión hasta hoy. El desarrollo de *Perl* 5 continuó en 1994. La lista de correo *perl5-porters* se estableció en Mayo de 1994 para coordinar el trabajo de adaptación de *Perl* 5 a diferentes plataformas. Es el primer foro para desarrollo, mantenimiento y adaptación de *Perl* 5 (Wyke y Thomas, 2001).

Perl 5, fue liberado el 17 de Octubre de 1994. Fue casi una completa reescritura del intérprete y añadió muchas nuevas características al lenguaje, incluyendo objetos, referencias, paquetes y módulos.

A destacar, los módulos proveen de un mecanismo para extender el lenguaje sin modificar el intérprete. Esto permitió estabilizar su núcleo principal, además de permitir a los programadores de *Perl* añadirle nuevas características.

El 26 de Octubre de 1995, se creó el *Comprehensive Perl Archive Network* (CPAN). CPAN es una colección de sitios *Web* que almacenan y distribuyen fuentes en *Perl*, binarios, documentación, *scripts* y módulos. Originalmente, cada sitio CPAN debía ser accedido a través de su propio *URL*; hoy en día, http://www.cpan.org redirecciona automáticamente a uno de los cientos de repositorios espejo de CPAN.

En 2007, *Perl* 5 continua siendo mantenido. Características importantes y algunas construcciones esenciales han sido añadidas, incluyendo soporte *Unicode*, Hilos (*threads*), un soporte importante para la programación orientada a objetos y otras mejoras. La última versión estable liberada en los repositorios a nivel mundial es Perl 5.8.8.

2.1.10.5. Perl/Tk

Es una herramienta GUI desarrollada por John Ousterhout, inicialmente como una extensión de Tcl (*Tool Command Language*) para facilitar la creación de interfaces gráficas de usuario. Desde que fue creado ha crecido en popularidad, y como resultado ha sido portado a muchos lenguajes diferentes. Nick Simmons portó *Tk* a *Perl*, resultándonos *Perl/Tk* (Steve y Walsh, 2003).

2.1.10.6. Pilas

Una pila (*stack*) es una colección de elementos a los que sólo se puede acceder por un único lugar o extremo de la pila. Los elementos de la pila se añaden o se quitan solo por su parte superior o cima, las pilas son estructuras de datos tipo LIFO (*Last-in, First-out)* Primero en entrar Último en salir (Joyanes, 1999).

2.1.11. Programación orientada a objeto

La Programación Orientada a Objetos (POO), se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución de los problemas se realizan en términos de objetos (Joyanes, 1999).

La POO es el método de implementación en el cual los programas se organizan como colecciones cooperantes de objetos, cada uno de los cuales representa un ejemplo de alguna clase, y cuyas clases son miembros de una jerarquía de clases unidas por relaciones (herencia) (Schmuller, 2000). Los mecanismos básicos de la POO son los objetos, los mensajes, los métodos las clases y las subclases (Schmuller, 2000).

Entre las características que se encuentran en la programación orientada a

objetos tenemos la abstracción, el encapsulamiento, la herencia y el polimorfismo (Joyanes, 1999).

2.1.11.1 Lenguaje de programación orientado a objetos

Es aquel que soporta objetos como una característica fundamental del mismo, así como los conceptos de encapsulamiento, herencia y polimorfismo, los cuales pueden ser aplicados a los objetos de trabajo. Los lenguajes de programación *Java* y *C#* son ejemplos de lenguajes orientados a objetos (Joyanes, 1999).

2.1.11.2. Análisis orientado a objetos

El análisis orientado a objeto realiza la explicación del sistema o dominio de problemas a partir de los conceptos de dominio, como tipos de objetos, asociaciones y cambios de estados, modelándolos como un grupo de objetos iterativos. Un objeto se define por su clase, elementos de datos y comportamiento. Por ejemplo, en un sistema de procesamiento de pedidos, una factura es una clase, e impresión, visualización, y totalización son ejemplos de este comportamiento. Los objetos (facturas individuales) heredan este comportamiento y lo combinan con sus propios elementos de datos (Kroll y Kruchten, 2003).

2.1.11.3. Diseño orientado a objetos

El diseño orientado a objeto transforma el modelo obtenido en la fase de análisis en las especificaciones requeridas para una solución lógica de *software* que no es más que crear un sistema.

El análisis orientado a objetos se inclina al diseño orientado a objeto mediante

la expansión de un modelo más detallado de la arquitectura del *software* (Pressman, 2002).

2.1.11.4. Objeto

Es una unidad que puede representar una entidad física o abstracta. Además de poseer atributos y comportamientos particulares, posee métodos (funciones) que han sido definidos para interactuar en operaciones comunes con dicho objeto (Joyanes, 1999).

Un objeto puede considerarse como una especie de cápsula dividida en tres partes: relaciones, propiedades, métodos. Cada uno de estos componentes desempeña un papel totalmente independiente:

Las relaciones permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos. Las propiedades distinguen un objeto determinado de los restantes que forman parte de la misma organización y tienen valores que dependen de la propiedad de éste. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.

Los métodos son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporadas en él, las cuales el objeto es capaz de ejecutar y poner a disposición de sus descendientes a través de la herencia (Fowler, 2003).

2.1.11.5. Herencia

Es el mecanismo para compartir automáticamente métodos y atributos entre objetos, la herencia permite crear un nuevo tipo de objeto (descendiente) de otro objeto base existente (antecedente).

Un objeto descendiente hereda todas las propiedades y métodos públicos y protegidos definidos por su clase antecedente, pero con la libertad de deshacerse de éstos o añadir nuevos sin alterar el objeto padre (Joyanes, 1999).

2.1.11.6. Abstracción

Representación de las características esenciales o generales de cosas similares. También define las características esenciales resultantes de algo, sin incluir antecedentes o detalles irrelevantes (Schmuller, 2000).

2.1.11.7. Abstracción de datos

En programación orientada a objetos, crear tipos de datos definidos por el usuario, los cuales contienen sus propios datos y procesamiento. Tales estructuras de datos u objetos, no son consistentes de los detalles físicos de los otros usuarios, y sólo conocen los servicios que cada uno realiza. Esta es la base del polimorfismo y de la ocultación de la información (Schmuller, 2000)

2.1.11.8. Polimorfismo

Estas características permiten implementar múltiples formas de un mismo método, dependiendo cada una de ellas de la clase sobre la que realice la implantación, es decir, consiste en que dos o más tipos de objetos pueden responder a un mismo mensaje de formas diferentes (Joyanes, 1999).

2.1.11.9. Encapsulamiento

Se refiere a la práctica de incluir dentro de un objeto todo lo que necesita de tal

forma que ningún otro objeto necesite conocer nunca su estructura interna, es decir, un objeto encapsula sus datos y sus métodos con la finalidad de ocultar los detalles de su implantación. La interacción con un objeto se realiza a través de una interfaz pública de las operaciones (Larman, 2004).

2.1.11.10. Método

Son funciones destinadas a manipular elementos que son definidos en conjunto con el objeto, esto implica que un objeto sólo puede manipular elementos que estén contenidos en el (Efecto "Caja Negra"). Por ejemplo en la programación orientada a objeto, en un objeto lista, la función buscar sólo lo hace en los ítems del objeto lista (Larman, 2004).

2.1.11.11. Eventos

Cada acción que se puede realizar sobre un objeto es un evento. Por ejemplo, en la programación orientada a objetos, hacer clic sobre un objeto botón, presionar una tecla al escribir en un objeto caja de texto, etc. Cada una de estas acciones es independiente una de otras, pero no necesariamente son excluyentes (Joyanes, 1999).

2.1.11.12. Mensaje

Un mensaje es la especificación de un estímulo, es decir la unidad de comunicación entre los objetos (Joyanes, 1999).

2.1.11.13. Modelo

Un modelo es una abstracción de la realidad. Especificando el sistema modelado desde un cierto punto de vista y en determinado nivel de abstracción. Un

modelo es el punto inicial de partido en la construcción de sistemas (Fowler, 2003).

2.1.11.14. UML

El Lenguaje Unificado de Modelado (LUM), prescribe un conjunto de notaciones y diagramas estandarizados para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan.

La utilización de UML es independiente del lenguaje de programación y de las características de los proyectos, ya que UML ha sido diseñado para modelar cualquier tipo de proyectos, tanto informáticos como de arquitectura, o de cualquier otro ramo.

El UML permite la modificación de todos sus miembros mediante estereotipos y restricciones. Un estereotipo nos permite indicar especificaciones del lenguaje al que se refiere el diagrama de UML. Una restricción identifica un comportamiento forzado de una clase o relación, es decir mediante la restricción estamos forzando el comportamiento que debe tener el objeto al que se le aplica (Larman, 2004).

UML dispone de dos tipos diferentes de diagramas los que dan una vista estática del sistema y los que dan una visión dinámica.

2.1.11.15. Estereotipo

Es una extensión del vocabulario de UML, que permite la creación de nuevos tipos de bloques de construcción que se derivan de otros existentes pero que son específicos a un problema en particular (Fowler, 2003).

2.1.11.16. Casos de uso

Los casos de uso, son un importante artefacto del análisis de requerimientos pero realmente no están orientados a objetos, es decir un caso de uso es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso.

Los casos de uso, son historias o casos de utilización de un sistema; no son exactamente los requerimientos ni las especificaciones funcionales, sino que ejemplifican e incluyen tácticamente los requerimientos en las historias que narran.

Los casos de uso describen un proceso de principio a fin, descripción que suele abarcar muchos pasos o transacciones.

Un caso de uso, no es un paso ni una actividad individual de un proceso (Fowler, 2003). Un caso de uso se modela para todos los procesos que el sistema debe llevar a cabo. Los procesos se describen dentro del caso de uso por una descripción textual o una secuencia de pasos ejecutados. Los casos de uso se utilizan también para probar el sistema y ver si satisface los requisitos iníciales.

Cada caso de uso se documenta por una descripción del escenario. La descripción puede ser escrita en modo de texto o en un formato paso a paso. Cada caso de uso puede ser también definido por otras propiedades, como las condiciones *pre* y *post* del escenario, condiciones que existen antes de que el escenario comience, y condiciones que existen después de que el escenario se completa (Schmuller, 2000).

En UML un caso de uso es representado por un ovalo, el cual encierra su nombre (Figura 5).

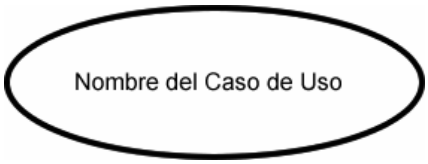


Figura 5. Icono para el modelaje de un caso de uso en UML.

2.1.11.17. Actores

Es un rol que un usuario juega con respecto al sistema. Es importante destacar el uso de la palabra rol, pues con esto se especifica que un Actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema (Fowler, 2003).

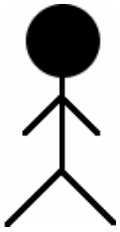


Figura 6. Icono para el modelaje de un actor en UML.

2.1.11.18. Diagramas de casos de uso

Se emplean para visualizar el comportamiento del sistema, una parte de él o de una sola clase. De forma que se pueda conocer cómo responde esa parte del sistema. El diagrama de casos de uso es muy útil para definir como debería ser el comportamiento de una parte del sistema, ya que sólo especifica como deben comportarse y no como están implementadas las partes que define. Por ello es un

buen sistema de documentos para las partes del código que deban ser reutilizables por otros desarrolladores. El diagrama también puede ser utilizado para que los expertos de dominio se comuniquen con los informáticos sin llegar a niveles de complejidad. (Fowler, 2003).

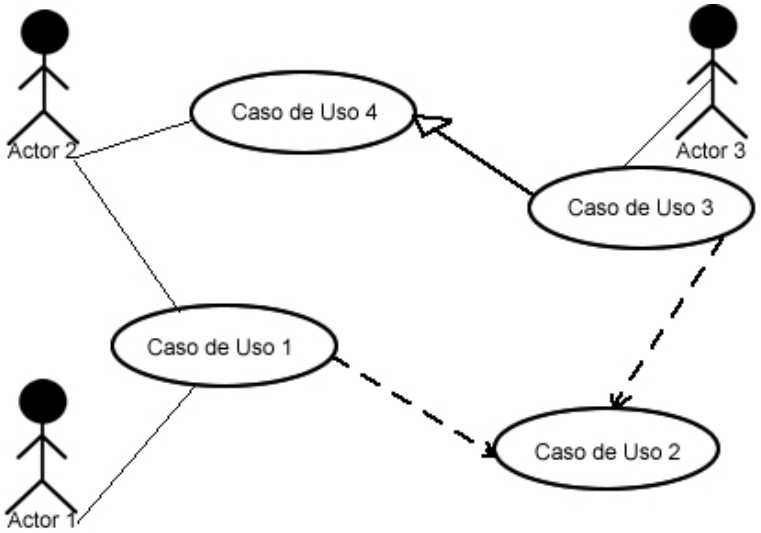


Figura 7. Diagrama de casos de uso.

Para el modelaje de diagramas de casos de uso en UML se utilizan diversos tipos de relaciones para representar su funcionalidad. A continuación se presentan los tipos de relaciones:

Asociación: es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos. La agregación es un tipo especial de asociación, que representa una relación estructural entre un todo y sus partes (Fowler, 2003).

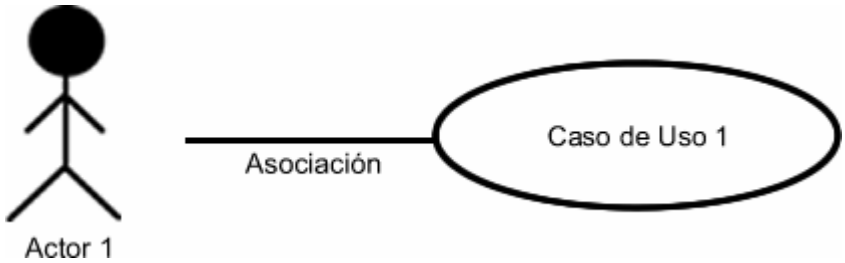


Figura 8. Asociación entre el actor y el caso de uso.

Inclusión (*include*): es una relación mediante la cual se re-usa un caso de uso encapsulado en distintos contextos a través de su invocación desde otros Casos de Uso (Larman, 2004).

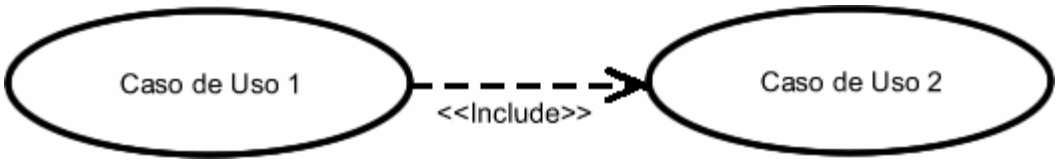


Figura 9. Relación de inclusión entre casos de uso.

Extensión (*extend*): es una relación que amplía la funcionalidad de un caso de uso mediante la extensión de sus secuencias de acciones.

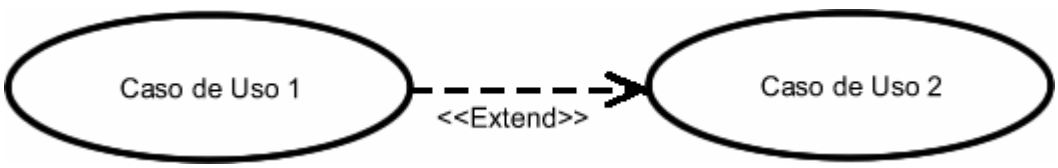


Figura 10. Relación de extensión entre casos de uso.

Generalización (*generalization*): es una relación que amplía la funcionalidad de un Caso de Uso o refina su funcionalidad original mediante el agregado de nuevas operaciones y/o atributos y/o secuencias de acciones.

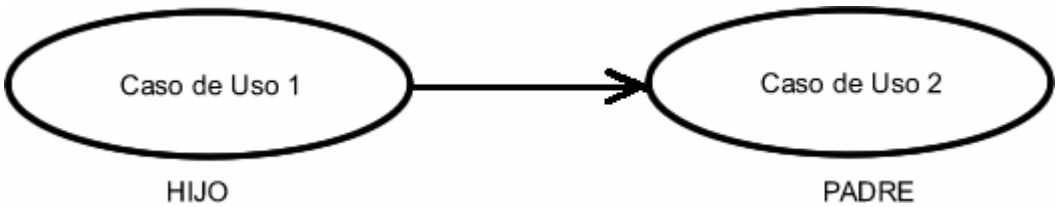


Figura 11. Relación de generalización entre casos de uso.

2.1.11.19. Clase

Una clase es una descripción de un grupo de objetos con propiedades (atributos), comportamiento (operaciones), relaciones y semántica común. Por tanto, una clase es una plantilla (*template*) para la creación de objetos, donde cada objeto de dicha clase será una instancia de ella (Fowler, 2003).

Cuando se escribe un programa en lenguaje orientado a objetos, no se definen objetos verdaderos sino se definen clases de objetos las cuales están compuestas por tres partes: nombre de la clase, atributos de la clase y operaciones o métodos de la clase.

Figura 12. Icono que representa una clase en UML.

Atributo (*Attributes*): Representa alguna propiedad de la clase que se encuentra en todas las instancias de la clase. Los atributos pueden representarse solo mostrando su nombre, mostrando su nombre y su tipo, e incluso su valor por defecto.

Método (*Method)*: Es la implementación de un servicio de la clase, que muestra un comportamiento común a todos los objetos. En resumen es una función que le indica a las instancias de la clase que realicen alguna acción. También se le conoce como operación (*Operation*).

2.1.11.19.1. Diagramas de clase

Los diagramas de clases son diagramas de estructura estática que muestran las clases del sistema y sus interrelaciones (incluyendo herencia, agregación, asociación, entre otros) (Fowler, 2003).

El diagrama de clase puede ser dividido en capas: aplicación, y datos, las cuales muestran las clases que intervienen con la interfaz de usuario, la lógica del software de la aplicación, y el almacenamiento de datos respectivamente por esta razón resulta muy útil cuando se desea realizar un análisis de dominio.

Este presenta un mecanismo de implementación neutral para modelar los aspectos de almacenado de datos del sistema. Las clases persistentes, sus atributos, y sus relaciones pueden ser implementados directamente en una base de datos orientada a objetos. La cardinalidad de las relaciones indica el grado y nivel de dependencia, es decir especifica cuantas instancias de una clase se pueden relacionar a una sola instancia de otra clase (Larman, 2004).

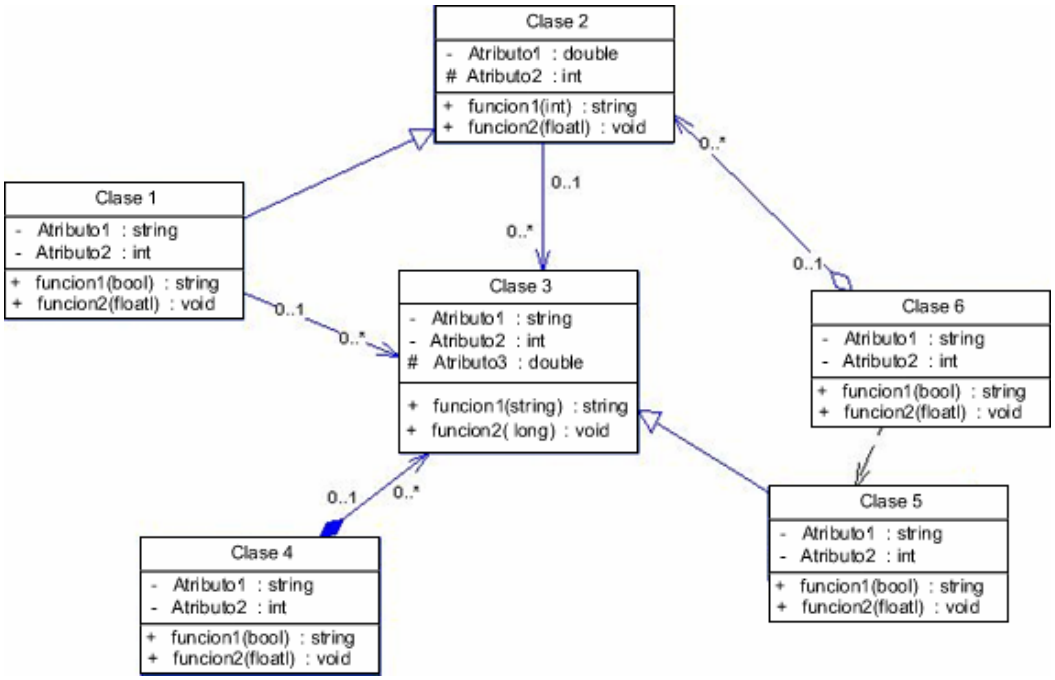


Figura 13. Diagrama de clases.

Las relaciones entre las clases se realizan a través de diferentes simbologías que tienen un significado propio. A continuación se presenta los distintos tipos de relaciones:

2.1.11.19.2. Asociación (*Association*)

Una asociación es una abstracción de la relación existente en los enlaces entre

los objetos y expresa una conexión bidireccional entre los objetos (Larman, 2004). Se representa mediante una línea continua, que une las dos clases.

Figura 14. Relación de asociación entre clases.

2.1.11.19.3. Composición (*composition*)

Son asociaciones que representan uniones muy fuertes. Esto significa que las composiciones también forman relaciones completas, pero dichas relaciones son tan fuertes que las partes no pueden existir por sí mismas. Únicamente existen como parte del conjunto, y si este es destruido las partes también lo son.

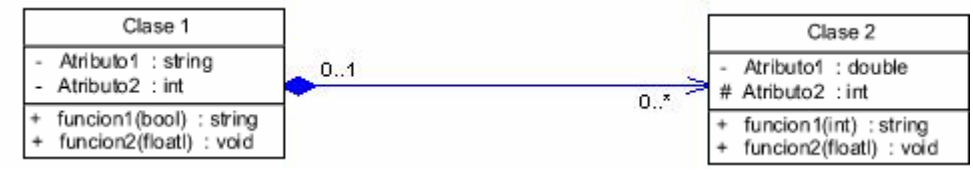


Figura 15. Relación de composición entre clases.

2.1.11.19.4. Generalización (Generalitation)

Es cuando una clase comparte estructura y/o comportamiento con una o más clases. El término superclase se refiere a la clase que guarda la información común, mientras que el término subclase se refiere a cada uno de los descendientes de la

superclase. A partir de las relaciones de generalización se crea una jerarquía de abstracción en la cual una subclase puede heredar de una o más superclases (Larman, 2004).

Figura 16. Relación de generalización entre clases.

2.1.11.19.5. Agregación (Agregation)

Una asociación normal representa una relación estructural entre iguales, es decir, las clases asociadas están conceptualmente en el mismo nivel, sin ser ninguna más importante que la otra. Por el contrario, una agregación, es una asociación especializada en la cual un todo se relaciona con sus partes. También se le suele denominar como la relación “parte de” (Fowler, 2003).

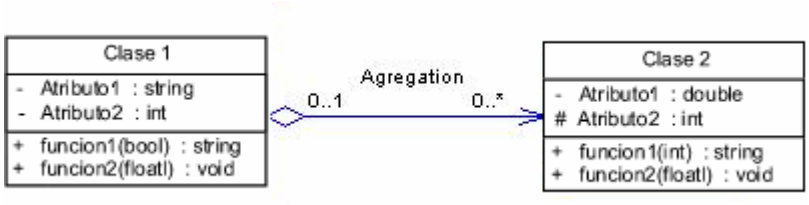


Figura 17. Relación de agregación entre clases.

2.1.11.19.6. Dependencia (Dependency)

Es una relación semántica entre dos elementos, en la cual un cambio a un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (el dependiente). Las dependencias generalmente representan relaciones de uso que declara que un cambio en la especificación de un elemento puede afectar a otro elemento que la utiliza, pero no necesariamente a la inversa (Larman, 2004).

Figura 18. Relación de dependencia entre clases.

Especificación de multiplicidad (mínima...máxima)

1	Uno y sólo uno.
0..1	Cero o uno.
M..N	De M hasta N (enteros naturales).
*	Muchos.
0..*	Cero o muchos.
1..*	Uno o muchos (al menos uno).

2.1.11.19.7. Diagramas de secuencia

El diagrama de secuencia es uno de los diagramas más efectivos para modelar interacción entre objetos en un sistema. Un diagrama de secuencia se modela para cada caso de uso. Mientras que el diagrama de caso de uso permite el modelado de

una vista del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes pasados entre los objetos (Larman, 2004). El diagrama de secuencia forma parte del modelado dinámico del sistema. Se modelan las llamadas entre clases desde un punto concreto del sistema. Es útil para observar la vida de los objetos en el sistema, identificar llamadas a realizar o posibles errores del modelado estático, que imposibiliten el flujo de información o de llamadas entre los componentes del sistema (Fowler, 2003).

El diagrama de la figura 19, está conformado por los siguientes elementos: actores, objetos, eventos, operaciones, destructores, activación y líneas que muestran el tiempo de vida de un objeto.

Actor: es aquel que se relaciona con los objetos de la aplicación. Puede ser un humano u otro sistema (Larman, 2004).

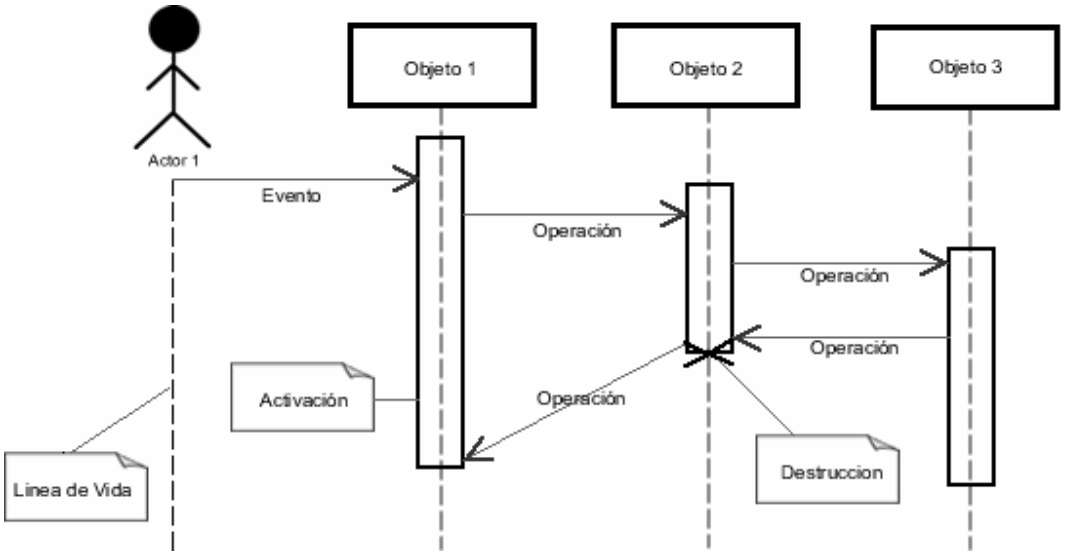


Figura 19. Diagrama de secuencia.

Objeto: es una entidad discreta con límites bien definidos y con identidad, es

una unidad atómica que encapsula el estado y su comportamiento. La encapsulación en un objeto permite una alta cohesión y un bajo acoplamiento. El objeto es reconocido también como una instancia de la clase a la cual pertenece (Schmuller, 2000).

Eventos: es una ocurrencia que puede causar la transición de un estado a otro en un objeto (Larman, 2004).

Operación: es una acción que se ejecuta en respuesta a un evento del sistema.

Destructor: operación que destruye un objeto o su estado.

Mensaje: es el soporte de una comunicación que vincula dinámicamente los objetos que fueron separados previamente en el proceso de descomposición. Adquiere toda su fuerza cuando se asocia al polimorfismo y al enlace dinámico. Un estímulo causará la invocación de una operación, la creación o destrucción de un objeto o la aparición de una señal. Un mensaje se muestra con una flecha sólida desde la línea de vida de un objeto a la línea de vida de otro objeto (Fowler, 2003).

Línea de vida de un objeto: simboliza la existencia de éste en un cierto periodo de tiempo. Se representa mediante una línea discontinua vertical que va desde su creación hasta la destrucción. Si un objeto es destruido, se le coloca una "X" grande al final y se envía un mensaje (Fowler, 2003).

Activación: muestra el período de tiempo en el cual el objeto se encuentra desarrollando alguna operación, bien sea por sí mismo o por medio de delegación a alguno de sus atributos. Se denota como un rectángulo delgado sobre la línea de vida del objeto (Fowler, 2003).

2.2. Marco Metodológico

2.2.1. Metodología de la investigación

2.2.1.2. Metodología del área de aplicación

La metodología seguida para el desarrollo del trabajo fue el Proceso Unificado para el Desarrollo de *Software* (PUDS), propuesta por Booch, Jacobson y Rumbaugh en el año 1998 dentro de la corporación *Rational Software.*

El PUDS es un proceso de ingeniería de *software*. Provee un esquema disciplinado para asignar tareas y responsabilidades en una organización de desarrollo. Su objetivo en el proyecto fue asegurar la producción de *software* de alta calidad que reúna las necesidades de sus usuarios dentro de los límites presupuestarios y de calendario. Ésta metodología constó de una serie de ciclos que se ejecutaron repetidamente. Cada ciclo o iteración estuvo dividido por de las fases de concepción, elaboración, construcción y transición y culminó con una versión del producto de software (Figura 20). Estas iteraciones estaban constituidas de la misma forma que un proyecto de *software*, como lo es la planificación y el desarrollo de una serie de flujo de trabajos, los cuales se conformaron por captura de requisitos, análisis, diseño, implementación y pruebas. El contenido de una iteración cambiaba para acomodarse a los objetivos de cada fase.

Figura 20. Ciclo Iterativo de PUDS.

A continuación se describen cada uno de los flujos que contribuyeron a la construcción del sistema:

2.2.2. Modelado del negocio

En este flujo se entendieron los problemas que la organización deseaba solucionar y se identificaron las mejoras potenciales, luego se midió el impacto del cambio organizacional y la forma asegurar que los clientes, usuarios finales y los otros participantes tuviesen un entendimiento compartido del problema.

2.2.3. Requerimientos

Aquí se estableció y se mantuvo un acuerdo con los clientes y los otros interesados acerca de que se debía hacer el sistema para proveerme a mi como desarrollador un mejor entendimiento de los requerimientos del mismo, definiendo su

alcance y un sustento para la planeación de los contenidos técnicos de las iteraciones y la definición de una interfaz de usuario para el sistema enfocada en sus necesidades y objetivos.

También se estableció una base para la estimación de costo y tiempo necesarios para desarrollar el sistema.

2.2.4. Análisis

En este flujo de trabajo se analizó las solicitudes descritas en la captura de requisitos, mediante su refinamiento y estructuración.

Con esto se logró una comprensión más precisa de los requerimientos, y se obtuvo una descripción de las peticiones lo que hizo más fácil de mantener y fue lo que ayudó a dar estructura al sistema en su conjunto; incluyendo su arquitectura.

2.2.5. Diseño

Se logró formular los modelos que se centran en los requisitos no funcionales y el dominio de la solución, y se prepararon para la implementación y pruebas del sistema.

2.2.6. Implementación

Flujo de trabajo fundamental donde se obtuvo el propósito esencial que fue implementar el sistema en términos de componentes, es decir, ficheros de código fuente, *scripts*, ficheros de módulos, ejecutable y librerías.

2.2.7. Prueba

Aquí se pudo comprobar el resultado de la implementación mediante las pruebas de cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema que fueron entregadas a terceros.

2.2.8. Administración y configuración de cambios

Este flujo permitió controlar los cambios y mantener la integridad del producto desarrollado mediante la identificación y restricción de los cambios en los elementos configurables, auditoría de los cambios hechos a estos elementos y definición de los métodos, procesos y herramientas usadas para proveer la administración y configuración.

2.2.9. Ambiente

Dentro del ambiente enfocamos las actividades necesarias para configurar el proceso al trabajo. Se pudo describir las actividades requeridas para generar las líneas guías de apoyo al trabajo.

Todo esto con el fin de proveer a la organización, el ambiente necesario (herramientas y procesos) que me daban el soporte como equipo de desarrollo.

2.2.10. Evaluación

Sustentó un marco de trabajo, que permitió administrar el proyecto de *software*. Proporcionándome guías prácticas para la planeación, soporte, ejecución y

monitoreo del trabajo, además de una base para la administración del riesgo.

Tabla 2. Relación de hitos por fases.

Fase	Hitos
Inicio	Objetivos y ámbitos
Elaboración	Arquitectura
Construcción	Capacidad operacional inicial
Transición	Liberación del producto

Cada hito se determinó por la disponibilidad de un conjunto de artefactos; es decir, ciertos modelos o documentos que fueron desarrollados hasta alcanzar un estado predefinido. Conjuntamente, con el PUDS utilicé el Lenguaje Unificado de Modelado (UML) para preparar y describir todos los esquemas y modelos (artefactos) en cada una de las fases. La figura 21, muestra gráficamente las fases e iteraciones del PUDS; es decir, la relación entre cada fase conjuntamente con el contenido de las iteraciones representadas por flujos de trabajo (*workflows*).

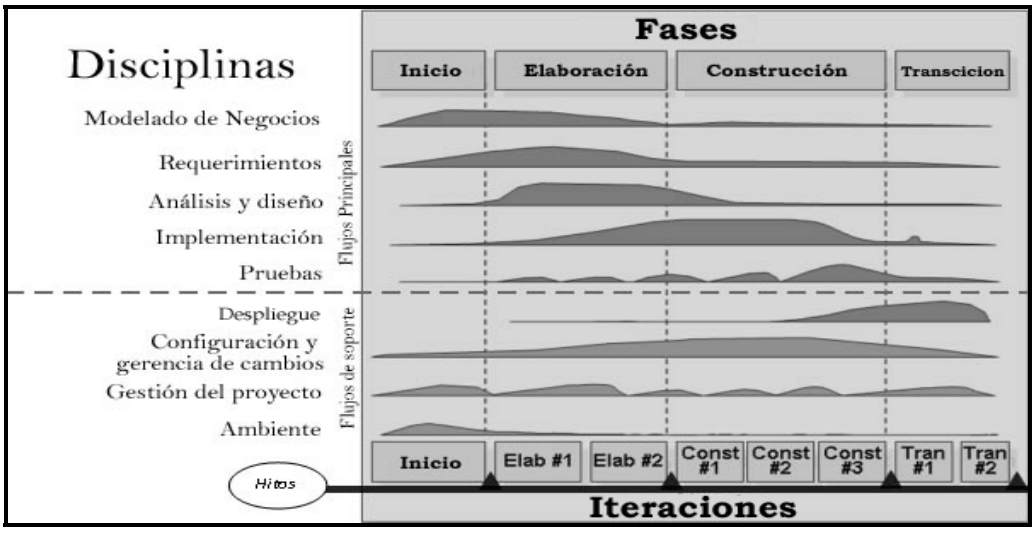


Figura 21. Representación gráfica del PUDS.
A continuación se describirán las fases de la metodología:

2.2.11. Fase de Inicio

Se desarrolló una descripción del producto final a partir de una buena idea y se presentó el análisis del negocio para la herramienta. Esencialmente, en esta fase se respondieron a las siguientes preguntas:

¿Cuáles son las principales funciones del sistema para los usuarios más importantes? ¿Cómo debe ser la arquitectura del sistema? ¿Cuál es el plan de proyecto? ¿Cuánto costará desarrollar el producto?

2.2.12. Fase de elaboración

Aquí se especificó en detalle la mayoría de los casos de uso del producto y se diseñó la arquitectura del sistema. La relación entre la arquitectura del sistema y el propio sistema fue primordial.

La arquitectura se expresó en forma de vistas de todos los modelos del sistema, los cuales juntos representaron al sistema entero. Esto implica que hay vistas arquitectónicas de los modelos de casos de uso, análisis y diseño.

La vista del modelo de implementación incluye componentes para probar que la arquitectura es ejecutable.

Durante esta fase del desarrollo, se realizaron los casos de uso más críticos identificados en la fase de inicio. El resultado de esta fase fue una línea base de la arquitectura.

Al final de la fase de elaboración, como director de proyecto planifique las

actividades y estime los recursos necesarios para terminar el proyecto. Aquí lo fundamental fue; confirmar si eran lo suficientemente estables los casos de uso, validar la arquitectura y el plan, medir los riesgos de continuar con el desarrollo.

2.2.13. Fase de construcción

Aquí se creó el producto; añadiendo el *software* terminado a la arquitectura. En esta fase, la línea de la arquitectura creció hasta convertirse en el sistema completo. La descripción evolucionó hasta transformarse en un producto preparado que fue entregado a la comunidad de usuarios. El grueso de los recursos requeridos se utilizó durante esta fase del desarrollo. Sin embargo, la arquitectura del sistema ya es estable, aunque se descubrieron formas mejores de estructurar el sistema, pues se recibieron sugerencias de cambios arquitectónicos de menor importancia y que la dirección y los clientes acordaron para el desarrollo de la primera versión.

CAPÍTULO III
DESARROLLO

El *software* PLCLAB, en su primera etapa, fue desarrollado cumpliendo con las directrices del PUDS.

Cada una de las fases que conforman el desarrollo del sistema, fue ejecutada mediante ciclos de iteraciones y de forma incremental se generaron los diagramas finales de casos de uso, clases y secuencia.

En otras palabras, se inició desde un conjunto de requerimientos iníciales que conformaron el dominio preliminar del sistema, y luego cada uno de estos productos fue madurando a través de cada iteración. De esta forma se logró modelar, una arquitectura sólida basada en los requerimientos de los usuarios.

Se utilizó la herramienta de diseño Umbrello 2.51, para el modelado del sistema empleando (UML) como lenguaje de referencia para la notación, descripción y organización del comportamiento del PLCLAB.

A la hora de comenzar se tenía el cronograma establecido para desarrollar el sistema basado PUDS.

Según el cronograma se tenían dos iteraciones para la fase de inicio, cuatro iteraciones para la fase de elaboración y seis iteraciones para la fase de construcción, esto se visualiza en la figura 22.

3.1. Fase de inicio

Tomando en cuenta los objetivos de la aplicación en esta fase se comenzó en la primera iteración con las entrevistas a los operadores de SUTRONIX a fin de entender el comportamiento que se deseaba para la aplicación, a través de la recolección de datos se ubicó el único actor del sistema que es el operador o programador de autómatas, también se generaron los diagramas de casos de uso más críticos (Figura 23), la imagen ampliada y la descripción aparece en el (Apéndice A).

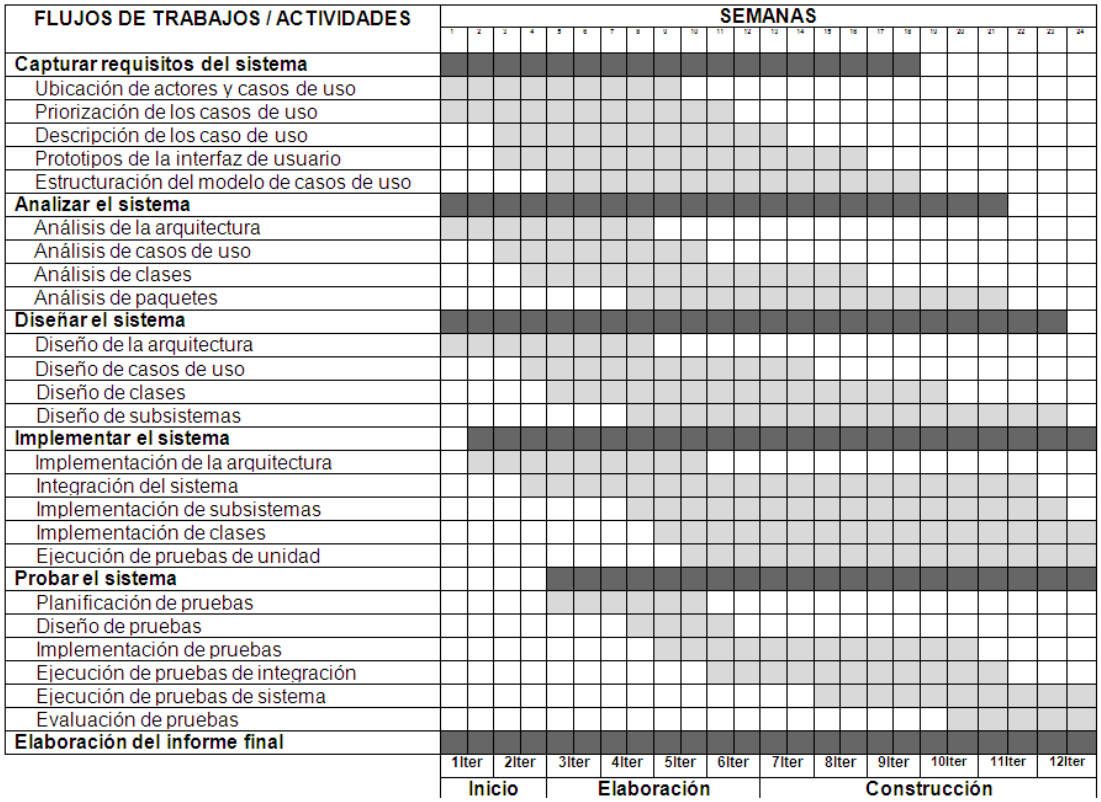


Figura 22. Cronograma para el desarrollo del sistema.

A continuación se describen cada una de las actividades o flujos de trabajos realizados en la primera iteración de la fase de inicio.

3.1.1. Modelado del negocio

En esta primera iteración el modelo de negocio fue una de las actividades más preponderante, ya que se pudo definir exactamente lo que SUTRONIX esperaba de la aplicación tomando en cuenta el tiempo que teníamos para la investigación, también se concientizó a los analista de la empresa sobre el apoyo y el constante acompañamiento que deberían tener con el proyecto para así asegurar el éxito del mismo.

Figura 23. Diagrama de casos de uso preliminar.

Aquí, se establecieron los documentos que se deberían generar, por ejemplo los planos arquitectónicos del sistema requerido por la metodología y regido por las necesidades de SUTRONIX. Durante la segunda iteración y con la información obtenida surgieron nuevas interrogantes acerca de lo que se perseguía realmente a nivel de negocio del software.

3.1.2. Requerimientos

Aquí se comenzó de manera formal con el ciclo de entrevistas y consultas a los técnicos de la empresa; con lo que se determinó que el sistema tendría un único actor, al cual se le denominó programador ladder, este se encargaba de todos los procesos en el sistema, con esto, se construyeron los casos de uso iniciales del sistema, generando así los primeros artefactos o documentos del software.

Durante la segunda iteración se mantuvieron las entrevistas y las jornadas de formación y uso de otras aplicaciones de fabricantes reconocidos en el área de *software* para autómatas. En esta iteración se refinaron y extendieron los casos de uso del sistema, figura24 (Apéndice A).

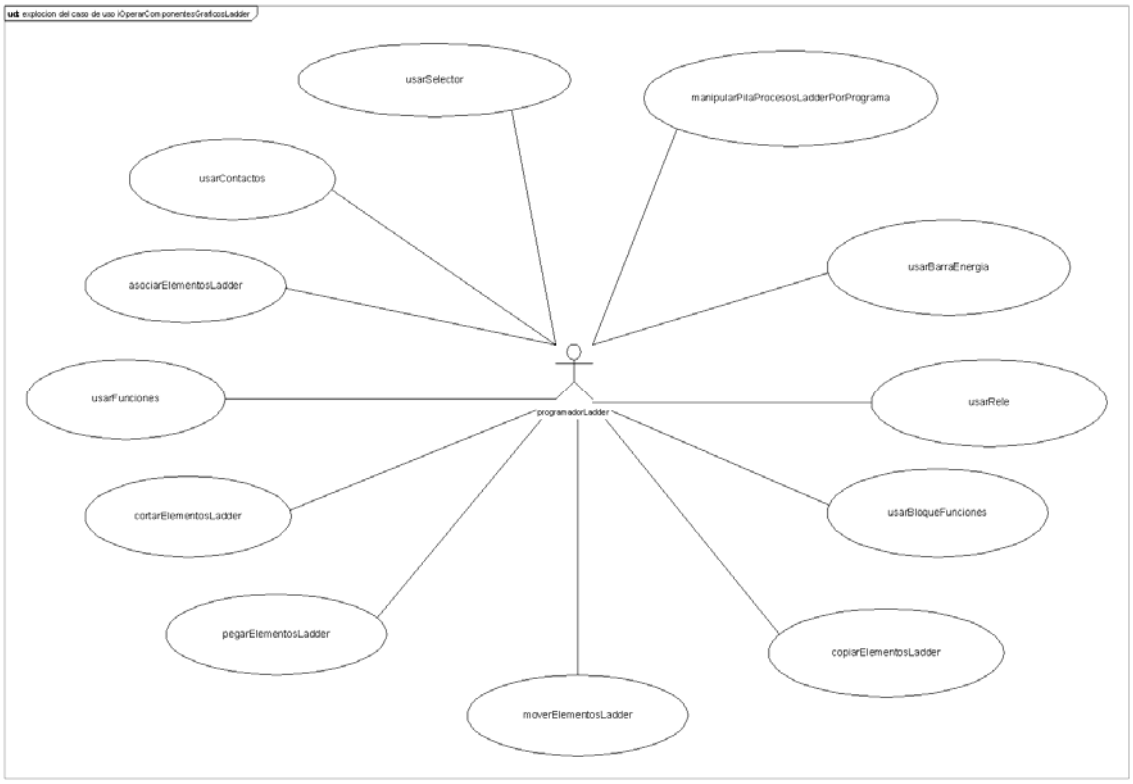


Figura 24. Explosión del caso de uso del sistema.

3.1.3. Análisis y diseño

Una vez definido los casos de uso preliminares, se comenzaron a descubrir las posibles clases del sistema y se generó el primer bosquejo de objetos permitiendo así construir el diagrama de clase de dominio inicial.

Durante la iteración dos (2), existía una visión más general del sistema lo que permitió actualizar el diagrama de clase de dominio

3.1.4. Implementación

Siendo la parte fundamental de esta actividad la programación se utilizó el este

tiempo para realizar una investigación y varias pruebas programáticas con el lenguaje seleccionado Perl; para ver si todas las características del sistema serian soportadas por el lenguaje de igual forma en las distintas plataformas. En la segunda iteración se crearon prototipos de interfaz que incluyeron el diseño de los posibles iconos finales para la aplicación (Figura 25, Apéndice I).

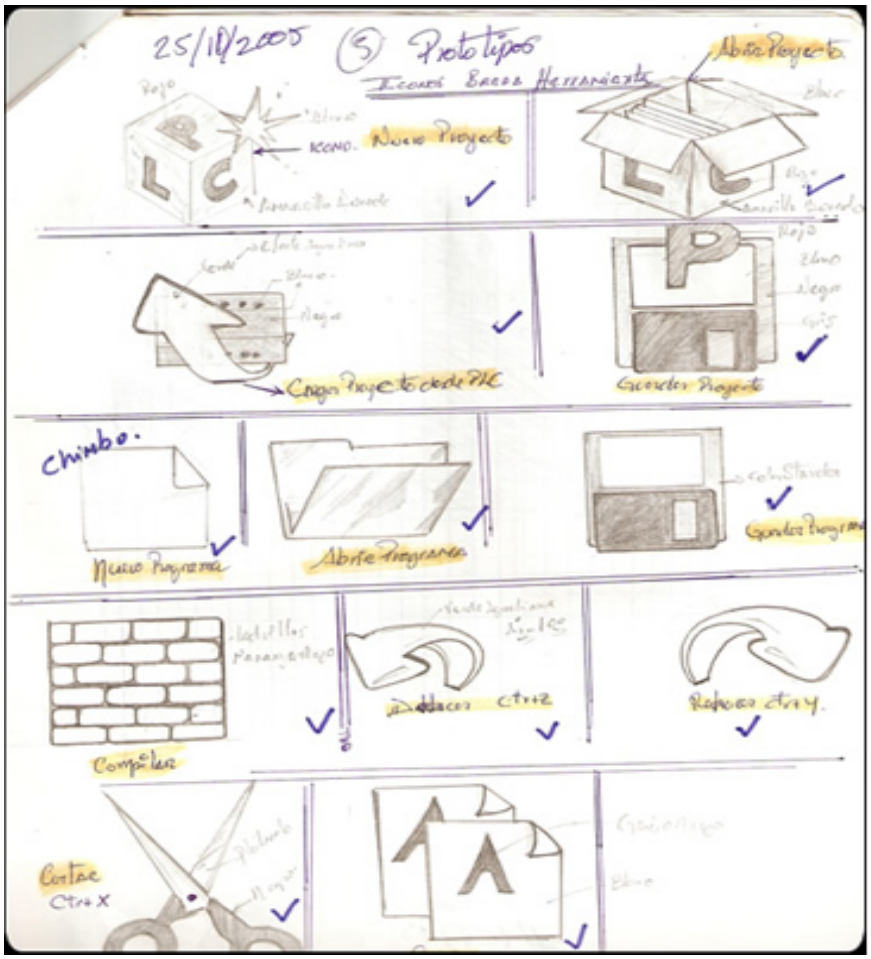


Figura 25. Primer prototipo de iconos del sistema.

3.1.5. Pruebas

Para el flujo de pruebas en la primera iteración se mantuvieron las

investigaciones sobre el lenguaje Perl aplicando pruebas de mayor rigurosidad sobre todo para la librería grafica seleccionada Tk. Durante la segunda iteración se comenzó a investigar como soportar un comportamiento multiplataforma del sistema bajo la librería gráfica de Perl figura 26 (Apéndice L).

```
sub cargarInformacionSistemaOperativo
{ #.....Inicio del Metodo: cargarInformacionSistemaOperativo
    my$this = shift;
    #procedemos a determinar en que sistema operativo estamos trabajando

    if($^O eq "MSWin32")
     { #_____"MSWin32"
          $this::imgBarram= $PAHT."IMAGENES/barra_programa.gif";
          $this::imgCerrarp = $PAHT."IMAGENES/botones/bt_cerrar_herramienta.gif";
          $this::centM = 0;
          $this::imgCarpeta = $PAHT."IMAGENES/botones/bt_carpeta_azul.gif";
          $this::colorExp = "black" ;
          $this::fuenteExp = ['Century Gothic','10','bold'];
          $this::nF = 15;
          $this::nC = 17;
          $this->{-nLista} = 0;
     } #_____"MSWin32"
    else
     {
       if($^O eq "linux")
       { #_____"linux"
          $this::imgBarram= $PAHT."IMAGENES/barra_programa.gif";
          $this::imgCerrarp = $PAHT."IMAGENES/botones/bt_cerrar_herramienta.gif";
          $this::centM = 0;
          $this::imgCarpeta = $PAHT."IMAGENES/botones/bt_carpeta_azul.gif";
          $this::colorExp = "black" ;
          $this::fuenteExp = ['Century Gothic','10','bold'];
          $this::nF = 15;
          $this::nC = 17;
          $this->{-nLista} = 0;
       } #_____"linux"
       else
       {
          if($^O eq "MacOS")
            { #_____"MacOS"
            $this::imgBarram= $PAHT."IMAGENES/barra_programa.gif";
            $this::imgCerrarp = $PAHT."IMAGENES/botones/bt_cerrar_herramienta.gif";
            $this::centM = 0;
            $this::imgCarpeta = $PAHT."IMAGENES/botones/bt_carpeta_azul.gif";
            $this::colorExp = "black" ;
            $this::fuenteExp = ['Century Gothic','10','bold'];
            $this::nF = 15;
            $this::nC = 17;
            $this->{-nLista} = 0;

            } #_____"MacOS"
          else
            {
                if($^O eq "solaris")
                { #_____"solaris"
                   $this::imgBarram= $PAHT."IMAGENES/barra_programa.gif";
                   $this::imgCerrarp = $PAHT."IMAGENES/botones/bt_cerrar_herramienta.gif";
                   $this::centM = 0;
                   $this::imgCarpeta = $PAHT."IMAGENES/botones/bt_carpeta_azul.gif";
                   $this::colorExp = "black" ;
                   $this::fuenteExp = ['Century Gothic','10','bold'];
                   $this::nF = 15;
                   $this::nC = 17;
                   $this->{-nLista} = 0;
                } #_____"solaris"
            }
       }
     }
} #.....Fin del Metodo: cargarInformacionSistemaOperativo
```

Figura 26. Primeras pruebas para el ajuste multiplataforma.

3.1.6. Evaluación

Luego del comportamiento y de los resultados obtenidos durante las actividades anteriores, se confirmó que la selección del lenguaje de desarrollo era correcta.

Con la intensión de mantener un orden interno para el desarrollo fue necesario definir en cada iteración un cronograma, en el cual se asignaron un número de días para cada flujo de trabajo, lo que permitió reflejar a corto plazo pequeñas metas que acumuladas permitieron alcanzar el hito de la fase pues se definieron los objetivos y el ámbito de la aplicación (Figura 27, Apéndice J).

Figura 27. Cronograma interno para las iteraciones en PUDS.

3.2. Fase de elaboración

En base a la planificación reflejada en el proyecto, en esta fase se tenía estimado cuatro iteraciones de dos semanas, las cuales se emplearon de la siguiente forma:

En la primera iteración, se refinaron los diagramas de caso de uso críticos desarrollados en la fase de inicio resultando una explosión de los diagramas, lo que facilitó el entendimiento del dominio real del *software*. (Apéndice A).

En la segunda iteración, se construyeron los distintos diagramas de clases, primeramente el diagrama de clases de dominio que permitió identificar las clases más importantes reflejadas en los casos de usos y que en operaciones futuras de SUTRONIX le permita identificar el desarrollo del software con cualquier herramienta y con cualquier lenguaje de la norma IEC-61131-3. (Apéndice B).

El diagrama de clases de dominio, se explotó definiendo de forma mucho más expedita los métodos y atributos de las clases existentes y generando nuevas entidades para así culminar el diagrama de clase de diseño, que me permitió obtener un plano más detallado de la aplicación en términos de programación pero aun sin amarrarlo al lenguaje Perl, situación que aprovechará SUTRONIX a su conveniencia en aplicaciones futuras. (Apéndice C).

Una vez concluido el diagrama de diseño, se generó el diagrama de clases de implementación, lo que me permitió obtener de forma exacta la traducción de todo el dominio expresado en los artefactos anteriores y perfectamente adherido con el lenguaje de desarrollo Perl. (Apéndice D).

Las siguientes dos iteraciones se usaron para generar aquellos diagramas de secuencia más importantes y con ello definir el flujo de eventos en el sistema (Apéndice E), de esta forma se reforzaron los atributos y métodos del diagrama de clases de implementación además se obtuvo la arquitectura del sistema que era el hito que perseguía en esta fase.

3.3. Fase de construcción

En esta fase, se comenzó de forma más formal con la programación. En la primera iteración se realizó el mapeo de los artefactos de clases a el lenguaje de

programación Perl (Apéndice K). Lo que representó un reto ya que el soporte que brinda el lenguaje Perl en la programación orientada objeto tiende a ser un poco más complejo, pero en el que igual se logro con todo éxito la interconexión entre los distintos objetos del sistema.

Luego, se diseñaron las distintas interfaces definitivas (Apéndice F) y se realizaron las primeras pruebas multiplataforma para ajustar el código a las exigencias del decreto 3390.

Durante esta fase, se hicieron pequeños ajustes a los diagrama de secuencia ya que se observó más claramente la comunicación entre los objetos.

También, se ajustaron los diagramas de clase de diseño y de implementación debido a nuevos atributos y métodos generados por la programación.

En esta fase, los flujos de trabajo de modelado de negocio, requerimiento y análisis disminuyeron en gran medida pero igual generaron algunos resultados expresados en los artefactos del sistema.

El resto de las cinco iteraciones se basaron en largas jornadas de programación que permitieron mejorar el código creado, iteración por iteración, reflejando esto en los diagramas.

Para esta fase, se intensificaron los flujos de trabajo primordiales en construcción como lo es la implantación, las pruebas, y la Integración; sin descuidar las pequeñas y mínimas alteraciones que surgieron en los artefactos, con lo que al final consiguió alcanzar el hito esperado que fue la capacidad operativa inicial del *software* multiplataforma en su beta 0.1 cumpliendo con los objetivos establecidos, figura 28 (Apéndice G).

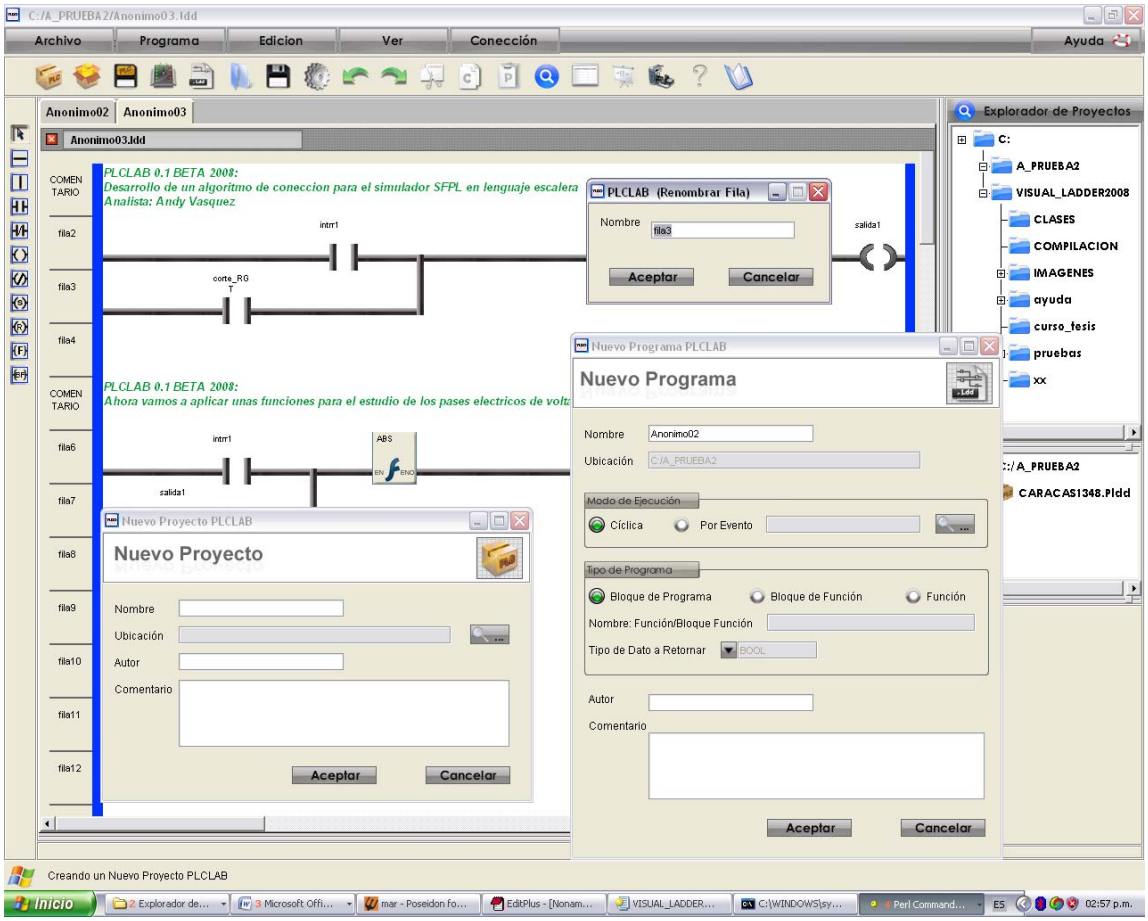


Figura 28. Interfaz final del primer beta del sistema.

CONCLUSIONES

Las fases contempladas por el PUDS aseguraron la construcción de un sistema de manera ordenada lo que permitió esculpir la lógica de los programas con el lenguaje escalera, pero con un software.

Con esta aplicación, se fomenta el uso de las nuevas tecnologías, además brinda la posibilidad de ser difundido y utilizado en la universidad y su código debe servir como guía para el desarrollo de nuevas aplicaciones de propósito general en la UDO-Sucre.

En este primer beta, se cumplieron todas las expectativas pero queda abierto el camino a través del código fuente existente para darle mayor potencialidad al *software* pues sería muy sencillo y beneficioso guardar el espacio de trabajo del programa y conservar viva la pila de procesos una vez que se cierre la aplicación.

RECOMENDACIONES

Incorporar a otros tesistas, para la integración de los restantes lenguajes de la norma al *software* actual, establecer una vez concluido el *hardware* del PLC de SUTRONIX, la conexión y registro físico en línea del *software* con el equipo industrial, la generación del simulador para el uso académico y la mayor difusión dentro de la universidad de las bondades de la informática en el área de automatismo industrial.

En proyectos modalidad pasantía es recomendable utilizar, sólo aquellos diagramas que permiten definir la arquitectura del sistema, sin caer en la tentación de generar documentos que ocupen horas de trabajo y que luego solo redunden en información ya expresada en otros componentes, reduciendo así horas de programación. Hay que recordar que el propósito de la ingeniería de *software*, más que cumplir estrictamente con una metodología, busca desarrollar aplicaciones que cumplan con lo requerido en el tiempo estipulado.

También se recomienda que se emprenda una campaña de información sobre el *software* para así difundir el uso de los autómatas dentro de las universidades, liceos y empresas del estado. Es muy importante de igual forma recomendar la publicación del sistema como un proyecto software libre con una licencia GPL.

BIBLIOGRAFÍA

Balcells, J. & Romeral, J.L.1998.Autómatas programables. Alfa Omega grupo de editores S.A. de C.V., México

Córdova, M. 2004. Libro Amarillo del Software Libre (Uso y Desarrollo en la Administración Pública). Ministerio de Ciencia y Tecnología, Venezuela.

Fowler, M. 2003. UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition. Corporation Addison Wesley, E.E.U.U.

Gates, B. 2000. Los negocios en la era digital. Editorial Plaza & Janes Editores S.A., México.

Jacobson, I., Booch, G. y Rumbaugh, J. 1999. The Unified Software Ddevelopment Process Rational Software. Corporation Addison Wesley, E.E.U.U.

Joyanes, L., 1999. Programación Orientada a Objeto. McGraw-Hill. España.

Kroll, P., Kruchten, P. 2003. Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Corporation Addison Wesley, E.E.U.U.

Larman, G., 2004. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Unified Process. Editorial Prentice Hall, E.E.U.U

Montejo, M. 2005."Introducción al estándar IEC 1131-3". "Autómatas". <http://www.automatas.org/software.htm > (16/08/2005).

Pressman, R. 2002. Ingeniería del Software. McGraw-Hill. España.

Serrano, E. y Luján, E. 1996.Implementación de la técnica ladder del PLC de la telemecanique en el taller de control y autómatas no industrial del IUT-Cumaná. Trabajo de grado del departamento de electrónica, Instituto Universitario de Tecnología-Cumaná.

Tiegelkamp, K. y Heinz, J. 1995. IEC 61131-3 Programming Industrial Automation System. Editorial Springer. E.E.U.U.

Steve, L. y Walsh, N. 2003. Mastering Perl/T
k. Editorial Oreally. E.E.U.U

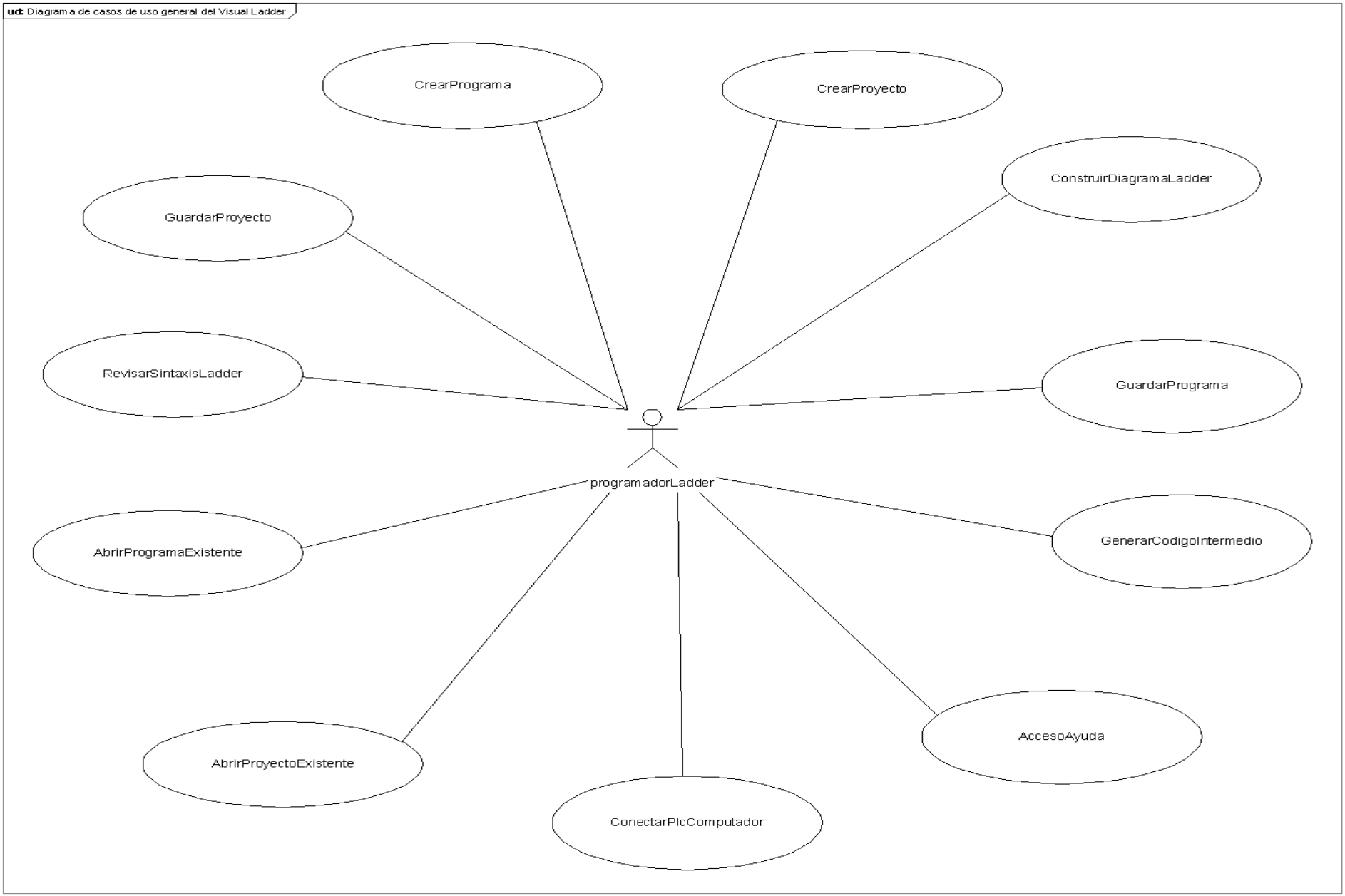
Stallman, R. 2002. Free Software, Free Society. Editorial Joshua Gay. E.E.U.U.

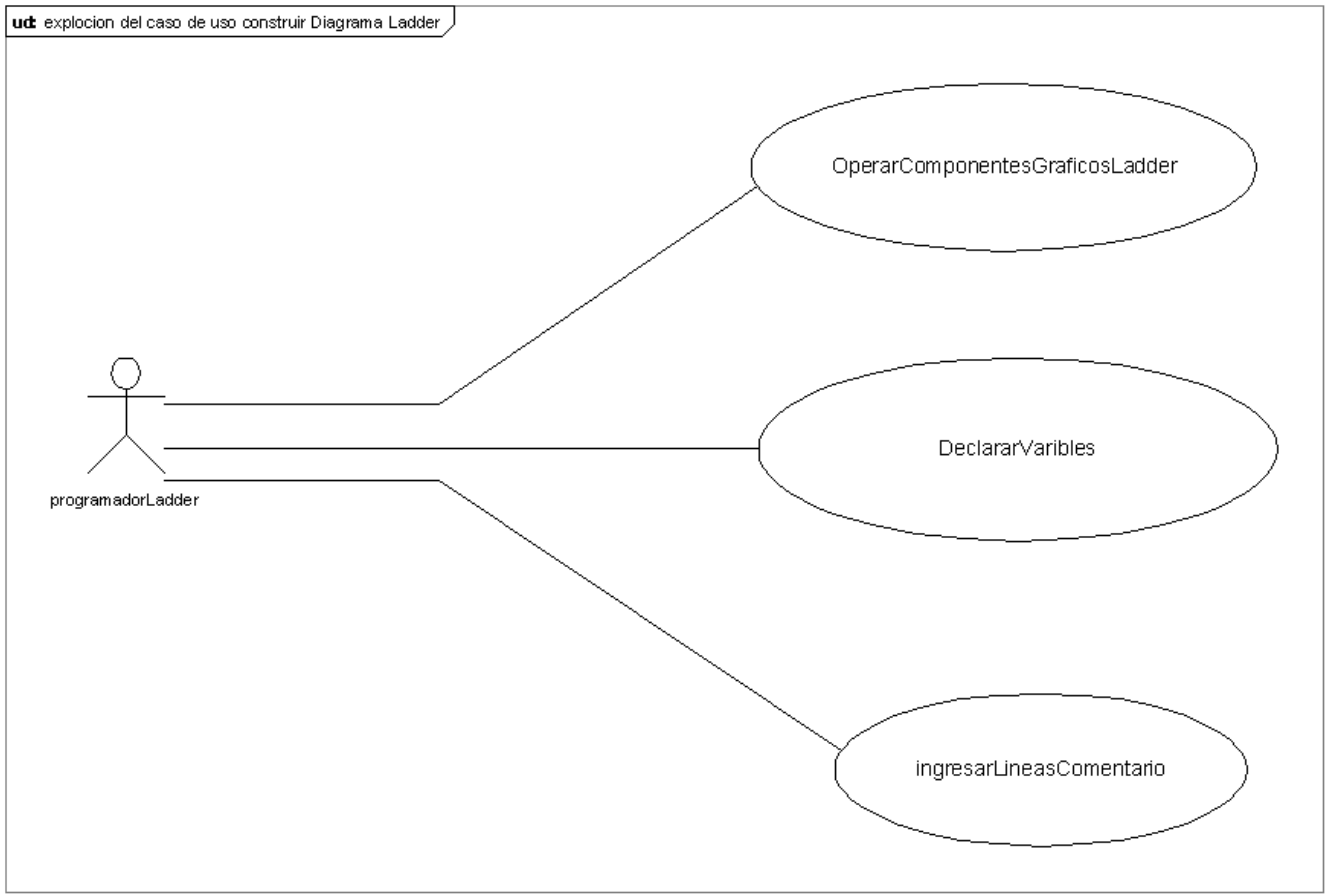
Schmuller, J. 2000. Aprendiendo UML en 24 Horas. Editorial Prentice Hall, Mexico.

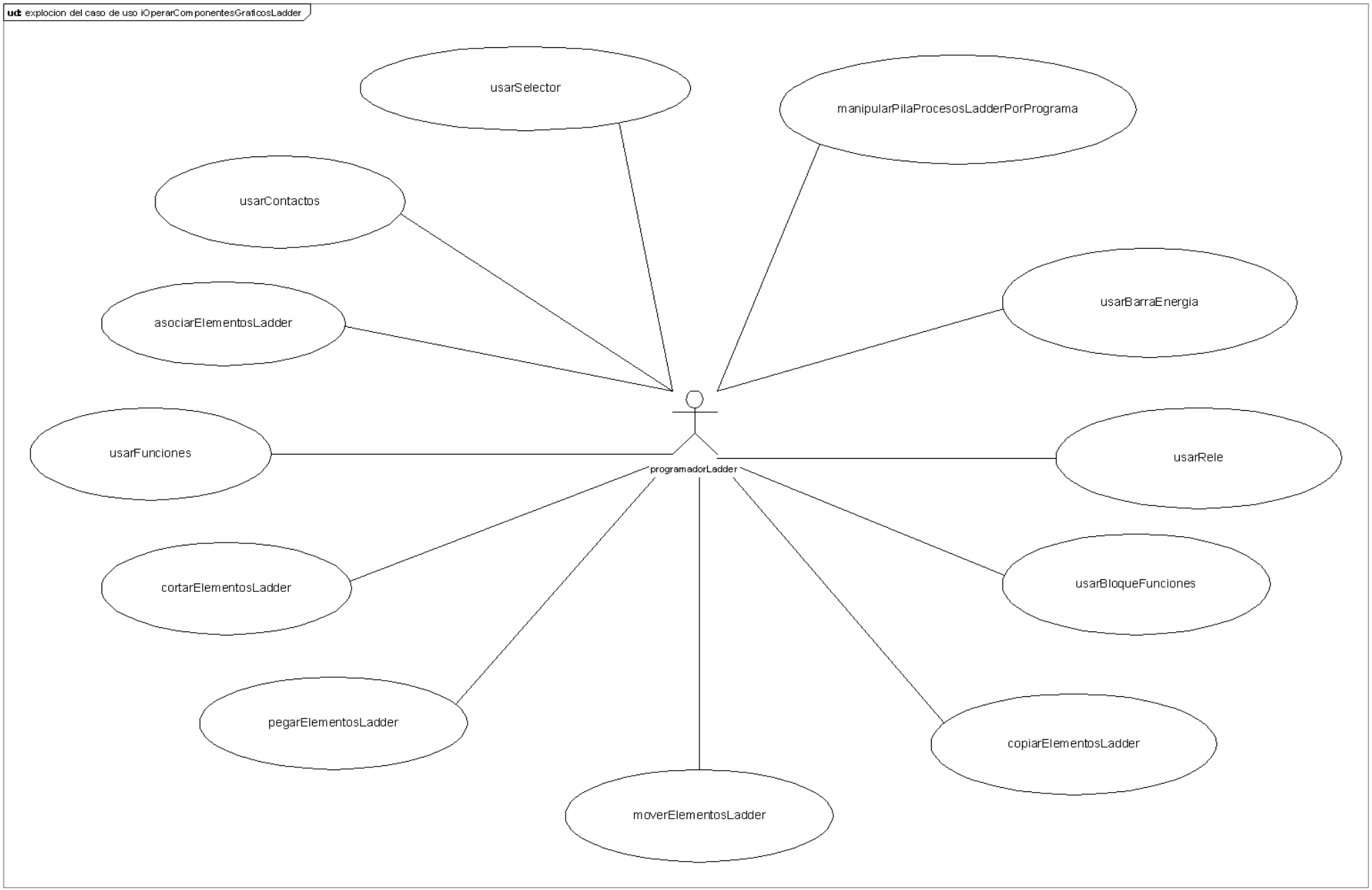
Wyke, A. y Thomas, D.2001. Fundamentos de programación en Perl. Editorial McGraw-Hill. Bogotá, Colombia.

APÉNDICES

APÉNDICE A: Diagramas y documentación de los casos de uso del sistema


ud: Diagrama de casos de uso general del Visual Ladder
CrearPrograma
CrearProyecto
ConstruirDiagramaLadder
GuardarProyecto
RevisarSintaxisLadder
GuardarPrograma
programadorLadder
AbrirProgramaExistente
GenerarCodigoIntermedio
AccesoAyuda
AbrirProyectoExistente
ConectarPlcComputador


uc: explocion del caso de uso construir Diagrama Ladder
OperarComponentesGraficosLadder
DeclararVaribles
ingresarLineasComentario
programadorLadder


uc: explocion del caso de uso iOperarComponentesGraficosLadder
usarSelector
manipularPilaProcesosLadderPorPrograma
usarContactos
usarBarraEnergia
asociarElementosLadder
usarFunciones
programadorLadder
usarRele
cortarElementosLadder
usarBloqueFunciones
pegarElementosLadder
copiarElementosLadder
moverElementosLadder

ACTOR(ES)

Programador Ladder: Es la persona que se encargada de manipular, el PLCLAB Visual ladder, además de crear proyectos y programas, revisar su sintaxis generar los diagramas, guardarlos etc.

CASO(S) DE USO

CASO DE USO: crearProyecto

ACTOR: Programador Ladder.

PROPÓSITO: Gestionar todas las operaciones referentes para la creación de proyectos en el visual ladder.

PRE-CONDICIÓN: Programador tiene la necesidad de crear un proyecto de forma local.

POST-CONDICIONES:

-Nuevo Proyecto creado.

-Archivo de Proyecto generado.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona la opción para crear un nuevo proyecto.
2. El sistema solicita la información del proyecto a crear.
3. El programador registra la información del proyecto.
4. El sistema genera el archivo físico y crea el proyecto.

CASO DE USO: crearPrograma

ACTOR: Programador Ladder.

PROPÓSITO: Gestionar todas las operaciones referentes para la creación de programas asociados a un proyecto en el visual ladder.

PRE-CONDICIÓN: Programador tiene la necesidad de crear un programa.
POST-CONDICIONES:

-Nuevo Programa creado.

-Archivo de Programa generado asociado al proyecto.
DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona la opción para crear un nuevo programa.
2. El sistema verifica si existe un proyecto abierto al cual asociar el programa.
3. El sistema en caso de estar abierto asocia el programa al proyecto actual, en caso de no existir le permite crear un nuevo proyecto.
4. El programador inserta los datos del programa.
5. El sistema crea el archivo físico del programa asociado a su respectivo proyecto.
6. El sistema informa sobre la ejecución exitosa de creación del programa y expone la interfaz para la programación.

CASO DE USO: construirDiagramaLadder

ACTOR: Programador Ladder.

PROPÓSITO: Gestionar todas las operaciones referentes para la creación de algoritmos a través del los elementos gráficos del lenguaje escalera, así como la declaración de variables y la manipulación de las operaciones de edición.

PRE-CONDICIÓN: El programador tiene la necesidad de generar la sintaxis gráfica de sus programas a través de los elementos del lenguaje escalera.

POST-CONDICIONES:

-Generación de algoritmos a través del lenguaje ladder.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador utiliza cualquiera de los elementos gráficos del lenguaje por medio de la barra de herramienta y los combina en el lienzo del programa actual.
2. El sistema coloca y asocia en el lienzo del programa los elementos seleccionados por el programador.
3. El sistema verifica que la asociación léxica sintáctica y semántica entre los elementos sea correcta.
4. El sistema almacena cada movimiento en la pila de procesos del programa.
5. El sistema genera el código intermedio asociado al programa en la memoria RAM de forma temporal.
6. El programador obtiene el diagrama deseado.

CASO DE USO: OperarComponentesGraficosLadder

ACTOR: Programador Ladder.

PROPÓSITO: Manipular todos los componentes gráficos del lenguaje escalera.

PRE-CONDICIÓN: El programador tiene la necesidad de usar algún elemento grafico en un lienzo de programa.

POST-CONDICIONES:

-Implementación en el lienzo de programa de los elementos gráficos ladder.

-Registro dentro de la pila de procesos de cada programa.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona el elemento ladder a usar.
2. El programador despliega y asocia el elemento ladder.
3. El sistema registra y publica la asociación gráfica y la instrucción en la pila de procesos del programa
4. El sistema guarda las instrucciones en la memoria del programa.

CASO DE USO: DeclararVaribles

ACTOR: Programador Ladder.

PROPÓSITO: manejar la declaración y asignación de variables locales y globales en los componentes ladder.

PRE-CONDICIÓN: El programador tiene la necesidad de asignar a cualquier elemento una variable.

POST-CONDICIONES:

-Relación entre el componente gráfico seleccionado y la variable.

-Asociación del tipo de dato para la variable.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona el elemento ladder a usar.
2. El sistema despliega la interfaz necesaria para que el programador asocie un nombre y un tipo de dato.
3. El sistema verifica las reglas sintácticas y semánticas cumpliendo con la asignación.

CASO DE USO: IngresarLineasComentario

ACTOR: Programador Ladder.

PROPÓSITO: Poder manejar las líneas de comentario en los programas ladder.

PRE-CONDICIÓN: El programador tiene la necesidad de escribir líneas de comentario para entender su codificación.

POST-CONDICIONES:

- Líneas de comentario reflejadas en la codificación ladder.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona la opción para colocar líneas de comentario en el programa.
2. El sistema despliega la interfaz necesaria para que el programador digite los comentarios necesarios en su programa.
3. El sistema refleja en la interfaz del programa los comentarios.

CASO DE USO: usarSelector

ACTOR: Programador Ladder.

PROPÓSITO: Manipular el selector de elementos ladder del programa.

PRE-CONDICIÓN: El programador tiene la necesidad de seleccionar cualquier elemento ladder a través del selector.

POST-CONDICIONES:

- Activación del selector del programa en el cursor de acciones graficas.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador accede al selector de elementos ladder.
2. El sistema activa las opciones de edición interna sobre los elementos del diagrama por medio del selector.

CASO DE USO: usarContactos

ACTOR: Programador Ladder

PROPÓSITO: Operar los distintos tipos de contactos: normalmente abiertos, normalmente cerrado.

PRE-CONDICIÓN: El programador tiene la necesidad de utilizar los contactos en la lógica de su programación.

POST-CONDICIONES:

- Creación edición y eliminación de contactos ladder en el lienzo del programa.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador accede a través del selector al cualquiera de los contactos disponibles.
2. El programador despliega por el lienzo los contactos asociándolos con los otros elementos ladder.
3. El programa valida la asociación efectiva de los elementos.
4. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método reciproco de respuesta inversa.
5. El programa refleja gráficamente la asociación y publicación de los contactos en el lienzo.

CASO DE USO: asociarElementosLadder

ACTOR: Programador Ladder.

PROPÓSITO: Manipular las acciones generadas en la asociación de elementos del lenguaje ladder.

PRE-CONDICIÓN: El programador tiene la necesidad de asociar elementos gráficos del lenguaje para esculpir la lógica de sus programas.
POST-CONDICIONES:

- Una asociación correcta y validada por las reglas sintácticas del lenguaje escalera.

-La representación grafica en el lienzo de la asociación establecida por el programador.
DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programa determina los elementos a asociar en el lienzo.
2. El programa revisa las reglas sintácticas y semánticas de dichos elementos.
3. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método recíproco de respuesta inversa.

CASO DE USO: manipularFunciones

ACTOR: Programador Ladder.

PROPÓSITO: contemplar todas las acciones y disposiciones de las funciones en el lenguaje escalera a través del programa.

PRE-CONDICIÓN: El programador tiene la necesidad de crear borrar o editar cualquier función tanto interna del lenguaje o propia de su lógica en el sistema.
POST-CONDICIONES:

- Una asociación correcta y validada por las reglas sintácticas del lenguaje escalera.

-La representación grafica en el lienzo de la asociación establecida por el programador.

DESCRIPCIÓN:

El caso de uso comienza cuando

1. El programa determina la función y los elementos con los que se va asociar en el lienzo.
2. El programa revisa las reglas sintácticas y semánticas de dichos elementos.
3. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método reciproco de respuesta inversa.

CASO DE USO: cortarElementoLadder

ACTOR: Programador Ladder.

PROPÓSITO: Contemplar todas las acciones y disposiciones del sistema a la hora de cortar cualquier elemento gráfico del lenguaje escalera.

PRE-CONDICIÓN: El programador tiene la necesidad de cortar un elemento ladder.

POST-CONDICIONES:

- Eliminación gráfica del elemento y asignación del mismo en la pila de procesos de edición del sistema.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona el los elementos a cortar.
2. El programa verifica si la selección es correcta y procede a eliminarlo gráficamente del lienzo.

3. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método reciproco de respuesta inversa.

CASO DE USO: pegarElementoLadder

ACTOR: Programador Ladder.

PROPÓSITO: contemplar todas las acciones y disposiciones del sistema a la hora de pegar cualquier elemento grafico del lenguaje escalera.

PRE-CONDICIÓN: El programador tiene la necesidad de pegar elementos ladder.

POST-CONDICIONES:

- Una asociación correcta y validada por las reglas sintácticas del lenguaje escalera.

-La representación gráfica en el lienzo de la asociación establecida por el programador.

DESCRIPCIÓN:

El caso de uso comienza cuando.

1. El programador selecciona el lugar del lienzo donde desea pegar los elementos.
2. El programa verifica si la selección y asociación es correcta y procede a pegarlo gráficamente del lienzo.
3. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método reciproco de respuesta inversa.

CASO DE USO: moverElementoLadder

ACTOR: Programador Ladder.

PROPÓSITO: contemplar todas las acciones y disposiciones del sistema a la hora de mover cualquier elemento grafico del lenguaje escalera.

PRE-CONDICIÓN: El programador tiene la necesidad de mover elementos ladder.
POST-CONDICIONES:

- Una asociación correcta y validada por las reglas sintácticas del lenguaje escalera.

-La representación grafica en el lienzo de la asociación establecida por el programador.
DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona a través del selector del lenguaje el lugar del lienzo donde desea mover el elemento.
2. El programa verifica si la selección y asociación es correcta y procede a moverlo gráficamente en el lienzo.
3. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método reciproco de respuesta inversa.

CASO DE USO: copiarElementoLadder

ACTOR: Programador Ladder.

PROPÓSITO: Contemplar todas las acciones y disposiciones del sistema a la hora de copiar cualquier elemento gráfico del lenguaje escalera.

PRE-CONDICIÓN: El programador tiene la necesidad de copiar elementos ladder.
POST-CONDICIONES:

- Ajuste y operación de la pila de procesos para la solicitud de copia.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador indica a través del selector el o los elementos a copiar.
2. El programa verifica si la es correcta.
3. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método reciproco de respuesta inversa.

CASO DE USO: usarBloqueFunciones

ACTOR: Programador Ladder.

PROPÓSITO: Contemplar todas las acciones y disposiciones del bloque de funciones en el lenguaje escalera a través del programa.

PRE-CONDICIÓN: El programador tiene la necesidad de crear borrar o editar cualquier bloque de función tanto interna del lenguaje o propia de su lógica en el sistema.

POST-CONDICIONES:

- Una asociación correcta y validada por las reglas sintácticas del lenguaje escalera.

-La representación gráfica en el lienzo de la asociación establecida por el programador.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador determina las entradas y salidas del bloque.
2. El sistema chuequea las reglas sintácticas y semánticas de asociación del bloque de funciones en el lenguaje ladder.
3. El sistema refleja gráficamente en el lienzo del programa la asociación del bloque de función.
4. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método reciproco de respuesta inversa.

CASO DE USO: usarRele

ACTOR: Programador Ladder.

PROPÓSITO: Contemplar todas las acciones y disposiciones de los relés en el lenguaje escalera a través del programa.

PRE-CONDICIÓN: El programador tiene la necesidad de crear borrar o editar cualquier relé en el sistema.

POST-CONDICIONES:

- Una asociación correcta y validada por las reglas sintácticas del lenguaje escalera del relé.

-La representación gráfica en el lienzo de la asociación establecida por el programador.
DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador determina si desea crear un relé en el lienzo o si desea editar o eliminar alguno existente.
2. El sistema valida las reglas sintácticas y semánticas de la operación de relé.
3. El sistema ejecuta gráficamente en el lienzo del programa la operación de relé.
4. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método reciproco de respuesta inversa.

CASO DE USO: usarBarraEnergia

ACTOR: Programador Ladder.

PROPÓSITO: Contemplar todas las acciones y disposiciones de las barras de energía vertical y horizontal en el lenguaje escalera.

PRE-CONDICIÓN: El programador tiene la necesidad de crear borrar o editar cualquier barra energética en el sistema.
POST-CONDICIONES:

- Una asociación correcta y validada por las reglas sintácticas del lenguaje escalera de las barras energéticas.

-La representación gráfica en el lienzo de la asociación establecida por el programador.
DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador determina si desea crear una barra en el lienzo o si desea editar o eliminar alguna existente.

2. El sistema se encarga de validar las reglas sintácticas y semánticas de la operación sobre las barras energéticas.

3. El sistema ejecuta gráficamente en el lienzo del programa la operación de barra.

4. El sistema carga la forma asociativa a la pila de procesos del programa y ajusta su método reciproco de respuesta inversa.

CASO DE USO: manipularPilaProcesosLadderPorPrograma

ACTOR: Programador Ladder.

PROPÓSITO: Contemplar todas las acciones de manipulación de la pila del proceso por programa.

PRE-CONDICIÓN: El programador tiene la necesidad de operar la pila de procesos del programa.
POST-CONDICIONES:

- Manipulación de la pila de procesos del programa.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador determina a través de operaciones gráficas la pila de procesos afectada.
2. El sistema ingresa por la cabeza los procesos generados y ajusta el método recíproco de la operación ingresada.
3. El sistema elimina por la cabeza de la pila el proceso ya sea por desbordamiento o por operaciones de edición, de igual forma desaparece el método reciproco de la cabeza de la pila.

CASO DE USO: GuardarPrograma

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora de guardar un programa.

PRE-CONDICIÓN: Programador tiene la necesidad de guardar un programa.

POST-CONDICIONES:

-Programa guardado.

-Archivo de Programa modificado generado o actualizado.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona la opción para guardar un programa
2. El sistema verifica la pila de procesos del programa y reajusta el archivo de programa.
3. El sistema informa sobre el éxito o fracaso de la operación de guardado.

CASO DE USO: GuardarProyecto

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora de guardar un proyecto.

PRE-CONDICIÓN: Programador tiene la necesidad de guardar un proyecto.
POST-CONDICIONES:

-Proyecto guardado.

- Listado de programas asociados al proyecto guardados si y sólo estaban editados a la hora de la petición de guardado.

-Archivo de Proyectos y programas modificados o generados.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona la opción para guardar un proyecto.
2. El sistema verifica la pila de procesos de los programas asociados y reajusta el archivo de los mismos.
3. El sistema verifica la pila de procesos del proyecto y reajusta el archivo asociado a él.
4. El sistema informa sobre el éxito o fracaso de la operación de guardado.

CASO DE USO: RevisarSintaxisLadder

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora revisar la sintaxis del programa.

PRE-CONDICIÓN: Programador tiene la necesidad de validar sus ordenes graficas en el

lienzo.

POST-CONDICIONES:

-Asociación correcta de elementos gráficos en el lienzo de programa.

- Manipulación de la pila de procesos de programa.

- notificación adecuada de los errores de sintaxis en el sistema.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El sistema verifica cualquier operación grafica o externa relacionada al estándar IEC-61131-3.
2. El sistema ejecuta un análisis léxico.
3. El sistema ejecuta un análisis sintáctico.
4. El sistema ejecuta un análisis semántico.
5. El sistema reporta los errores en caso de existir, de lo contrario permite la ejecución grafica de las instrucciones del programador.

CASO DE USO: GenerarCodigoIntermedio

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora de generar un código intermedio que refleje la representación grafica escalera.

PRE-CONDICIÓN: Programador tiene la necesidad de generar un código intermedio de la lógica de sus programas.

POST-CONDICIONES:

-Archivo de código intermedio generado por cada programa.

- Manipulación de la pila de procesos de programa.

- Notificación adecuada de los errores de generación.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador accede a la opción de generar el código intermedio de sus programas.
2. El sistema realiza un mapeo de la representación grafica hacia el código intermedio en forma de expresiones lógicas.
3. El sistema ejecuta un análisis sintáctico.
4. El sistema ejecuta un análisis semántico.
5. El sistema reporta los errores en caso de existir, de lo contrario permite la ejecución grafica de las instrucciones del programador.

CASO DE USO: EliminarProgramaExistente

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora de eliminar un programa ladder.

PRE-CONDICIÓN: Programador tiene la necesidad de eliminar un programa.

POST-CONDICIONES:

-Archivo de código eliminado.

- Removida la asociación del archivo del proyecto a que perteneció.

- Eliminación de todas sus variables y funciones.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador selecciona la opción de eliminar un programa.
2. El sistema verifica a que proyecto pertenece y elimina la relación.
3. El sistema elimina el archivo físicamente.
4. El sistema limpia las variables y funciones que estuvieron asociadas al programa.
5. El sistema reporta los errores en caso de existir, de lo contrario permite la ejecución gráfica de las instrucciones de eliminación.

CASO DE USO: AbrirProgramaExistente

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora abrir un programa existente.

PRE-CONDICIÓN: Programador tiene la necesidad de visualizar algún programa existente.

POST-CONDICIONES:

-Archivo de programa visualizado.

- Actualización y activación de la nueva pila de procesos del programa.

- Carga efectiva del proyecto padre del programa.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador presiona la opción para buscar un programa existente.
2. El sistema muestra de forma gráfica el selector de directorio para buscar el programa.
3. El programador selecciona el programa a visualizar.
4. El sistema recibe la ruta y el nombre del programa.

5. El sistema determina su proyecto padre al cual procede a abrir.
6. El sistema chequea si no existe ningún proyecto o programa cargado actualmente, de existir solicitad guardar los cambios en dichos archivos.
7. El sistema procede a cargar el proyecto y abrir el programa.
8. El programa despliega en el lienzo la codificación gráfica en ladder del programa.

CASO DE USO: AbrirProyectoExistente

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora abrir un proyecto existente.

PRE-CONDICIÓN: Programador tiene la necesidad de visualizar algún proyecto existente.

POST-CONDICIONES:

- El proyecto y todas sus opciones disponibles gráficamente para ser usadas.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador presiona la opción para buscar un proyecto existente.
2. El sistema muestra de forma grafica el selector de directorio para buscar el proyecto.
3. El programador selecciona el proyecto a visualizar.
4. El sistema recibe la ruta y el nombre del proyecto.
5. El sistema chequea si no existe ningún proyecto o programa cargado actualmente, de existir solicitad guardar los cambios en dichos archivos.

6. El sistema procede a cargar el proyecto.
7. El programa despliega en el lienzo la codificación gráfica en ladder del programa.

CASO DE USO: EliminarProyectoExistente

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora de eliminar un proyecto ladder.

PRE-CONDICIÓN: Programador tiene la necesidad de eliminar un proyecto.
POST-CONDICIONES:

-Archivo de código eliminado.

- Eliminación de todas sus variables y funciones asociadas al proyecto.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador presiona la opción de eliminar proyecto en el sistema.
2. El sistema elimina el archivo físicamente.
3. El sistema limpia las variables y funciones que estuvieron asociadas al proyecto.
4. El sistema reporta los errores en caso de existir, de lo contrario permite la ejecución grafica de las instrucciones de eliminación.

CASO DE USO: ConectarPlcComputador

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora definir las entradas de conexión del PLC con el computador.

PRE-CONDICIÓN: Programador tiene la necesidad de definir los parámetros de

conexión con el plc.

POST-CONDICIONES:

- Parámetros de conexión establecidos.

-Archivo de comunicación del PLC y el computador establecida.

DESCRIPCIÓN:

El caso de uso comienza cuando:

1. El programador presiona la opción de establecer comunicación PC PLC.
2. El sistema presenta una interfaz solicitando los parámetros de conexión.
3. El programador suministra los parámetros.
4. El sistema reporta los errores en caso de existir, establece la conexión del PLC con él computador.

CASO DE USO: AccesoAyuda

ACTOR: Programador Ladder.

PROPÓSITO: Precisar todas las operaciones y acciones del sistema a la hora a la ayuda del sistema.

PRE-CONDICIÓN: Programador tiene la necesidad de usar la ayuda en sus distintas opciones.

POST-CONDICIONES:

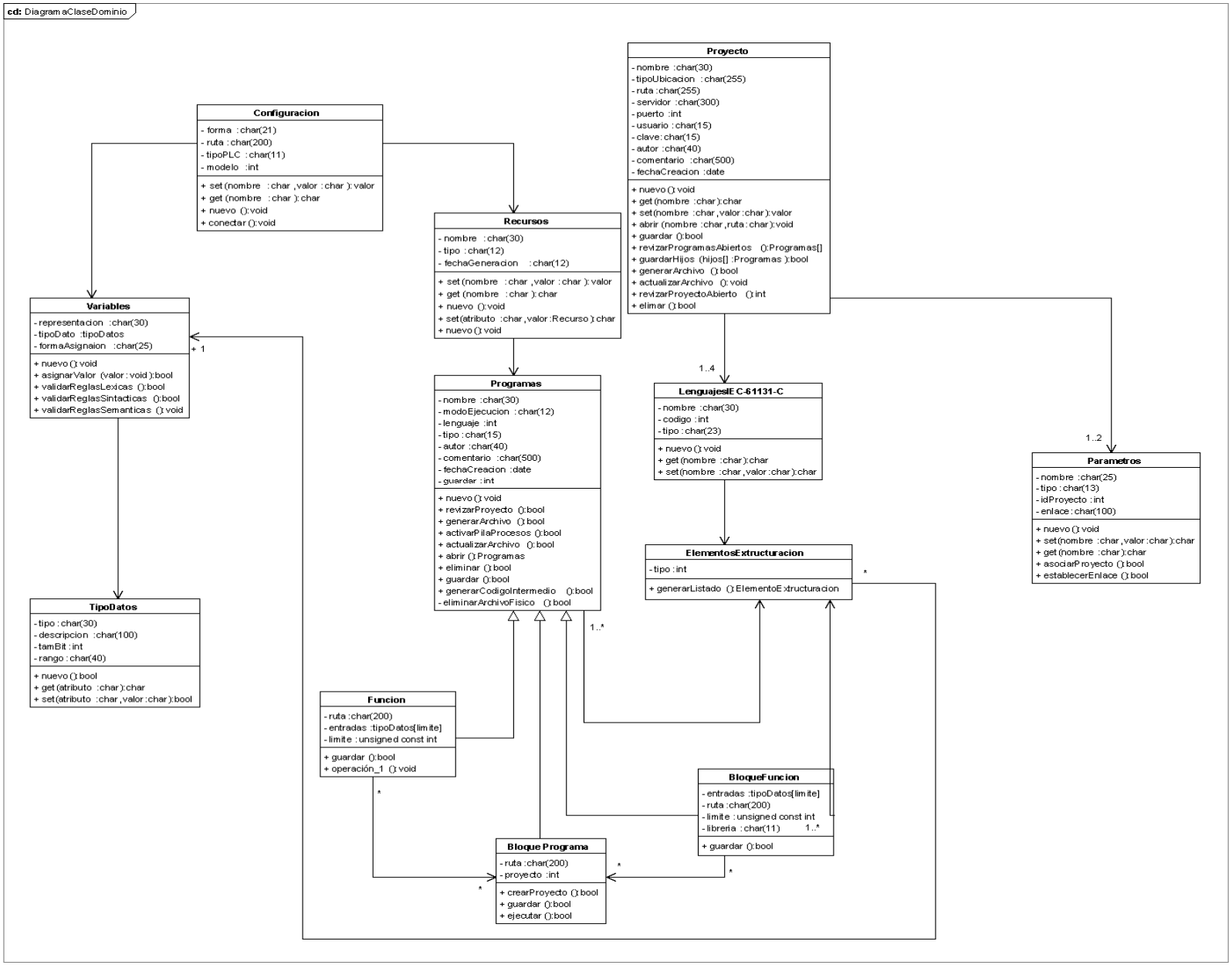
- Ayuda desplegada.

DESCRIPCIÓN:

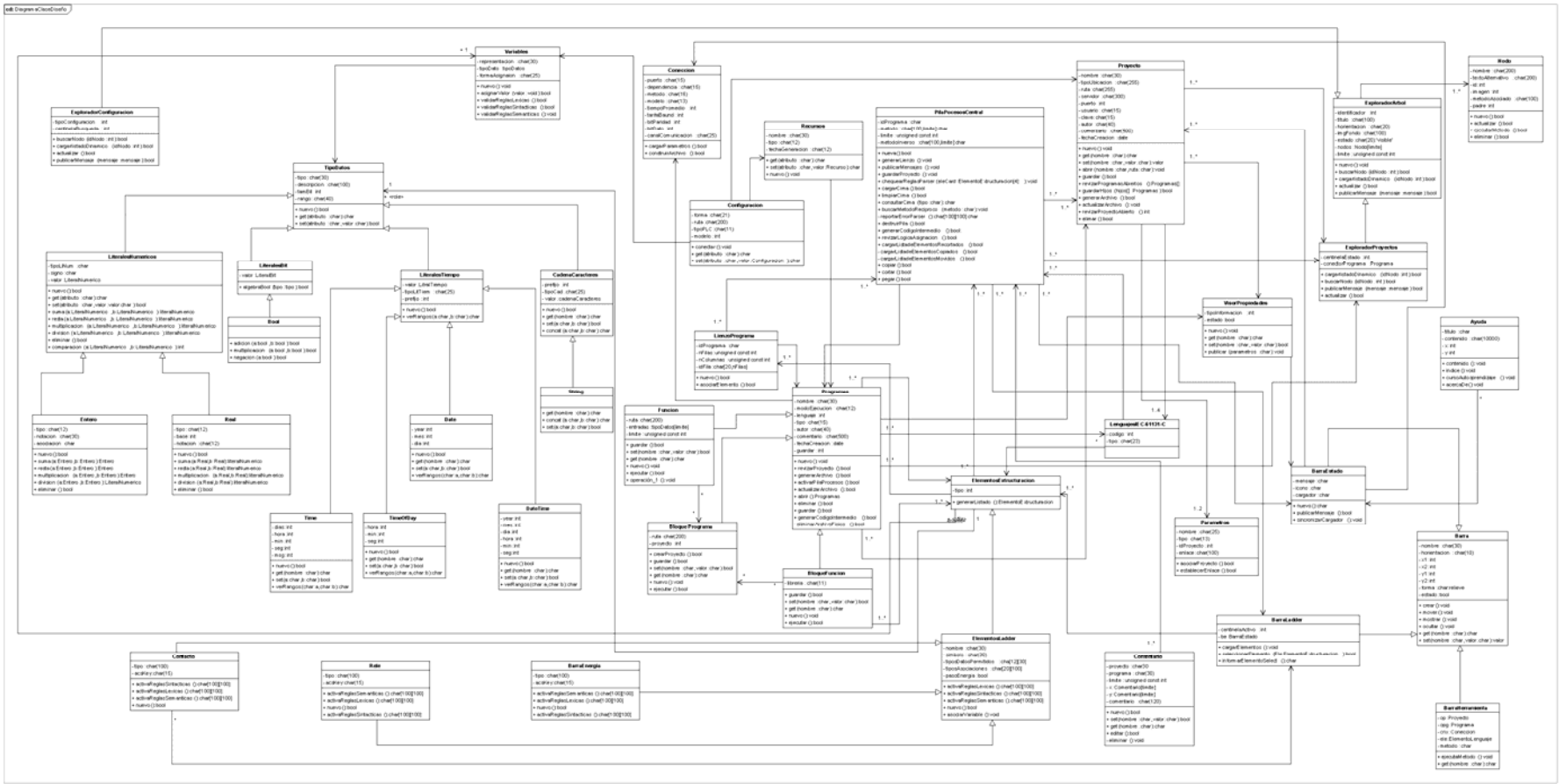
El caso de uso comienza cuando:

1. El programador presiona la opción de ayuda.
2. El sistema presenta una interfaz para la manipulación de la ayuda del sistema.

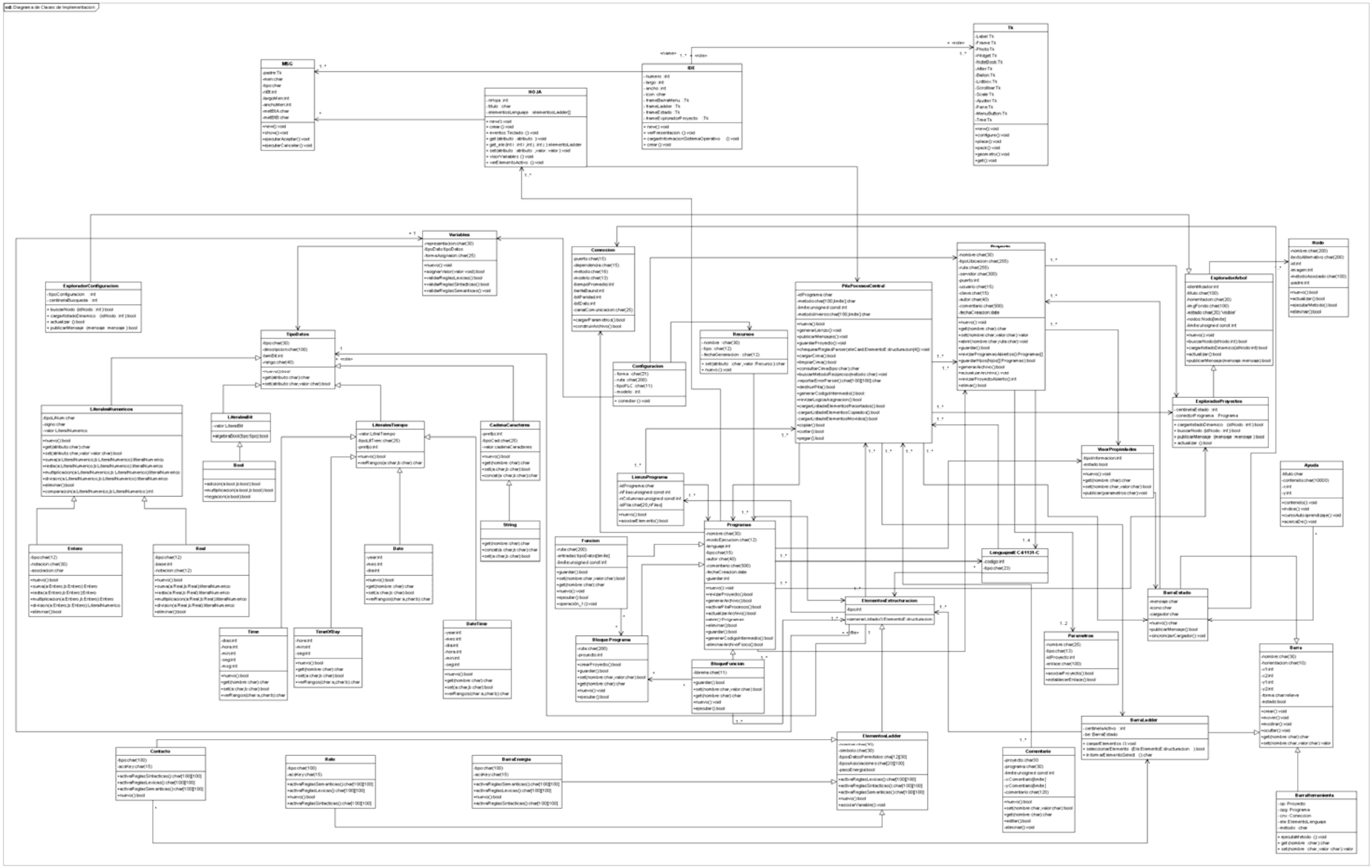
APÉNDICE B: Diagrama de clase de dominio del sistema


cd: DiagramaClaseDominio
Proyecto
- nombre :char(30)
- tipoUbicacion :char(255)
- ruta :char(255)
- servidor :char(300)
- puerto :int
- usuario :char(15)
- clave: char(15)
- autor :char(40)
- comentario :char(500)
- fechaCreacion :date
+ nuevo ():void
+ get (nombre :char):char
+ set(nombre :char ,valor :char):valor
+ abrir (nombre :char ,ruta: char):void
+ guardar ():bool
+ revizarProgramasAbiertos ():Programas[]
+ guardarHijos (hijos[] :Programas):bool
+ generarArchivo ():bool
+ actualizarArchivo ():void
+ revizarProyectoAbierto ():int
+ elimar ():bool
Configuracion
- forma :char(21)
- ruta :char(200)
- tipoPLC :char(11)
- modelo :int
+ set (nombre :char ,valor :char): valor
+ get (nombre :char):char
+ nuevo ():void
+ conectar ():void
Recursos
- nombre :char(30)
- tipo :char(12)
- fechaGeneracion :char(12)
+ set (nombre :char ,valor :char): valor
+ get (nombre :char):char
+ nuevo ():void
+ set(atributo :char ,valor :Recurso):char
+ nuevo ():void
Variables
- representacion :char(30)
- tipoDato :tipoDatos
- formaAsignaion :char(25)
+ nuevo ():void
+ asignarValor (valor : void):bool
+ validarReglasLexicas ():bool
+ validarReglasSintacticas ():bool
+ validarReglasSemanticas ():void
+ 1
Programas
- nombre :char(30)
- modoEjecucion :char(12)
- lenguaje :int
- tipo :char(15)
- autor :char(40)
- comentario :char(500)
- fechaCreacion :date
- guardar :int
+ nuevo ():void
+ revizarProyecto ():bool
+ generarArchivo ():bool
+ activarPilaProcesos ():bool
+ actualizarArchivo ():bool
+ abrir ():Programas
+ eliminar ():bool
+ guardar ():bool
+ generarCodigoIntermedio ():bool
- eliminarArchivoFisico ():bool
1..4
LenguajesIEC-61131-C
- nombre :char(30)
- codigo :int
- tipo :char(23)
+ nuevo ():void
+ get (nombre :char):char
+ set (nombre :char ,valor :char):char
1..2
Parametros
- nombre :char(25)
- tipo :char(13)
- idProyecto :int
- enlace :char(100)
+ nuevo ():void
+ set(nombre :char ,valor :char):char
+ get (nombre :char):char
+ asociarProyecto ():bool
+ establecerEnlace ():bool
ElementosExtructuracion
- tipo :int
+ generarListado ():ElementoExtructuracion
*
1..*
TipoDatos
- tipo : char(30)
- descripcion :char(100)
- tamBit : int
- rango : char(40)
+ nuevo ():bool
+ get (atributo :char):char
+ set(atributo :char ,valor :char):bool
Funcion
- ruta :char(200)
- entradas :tipoDatos[limite]
- limite : unsigned const int
+ guardar ():bool
+ operación_1 ():void
*
BloqueFuncion
- entradas :tipoDatos[limite]
- ruta :char(200)
- limite : unsigned const int
- libreria : char(11)
1..*
+ guardar ():bool
*
Bloque Programa
- ruta :char(200)
- proyecto :int
+ crearProyecto ():bool
+ guardar ():bool
+ ejecutar ():bool
*
*

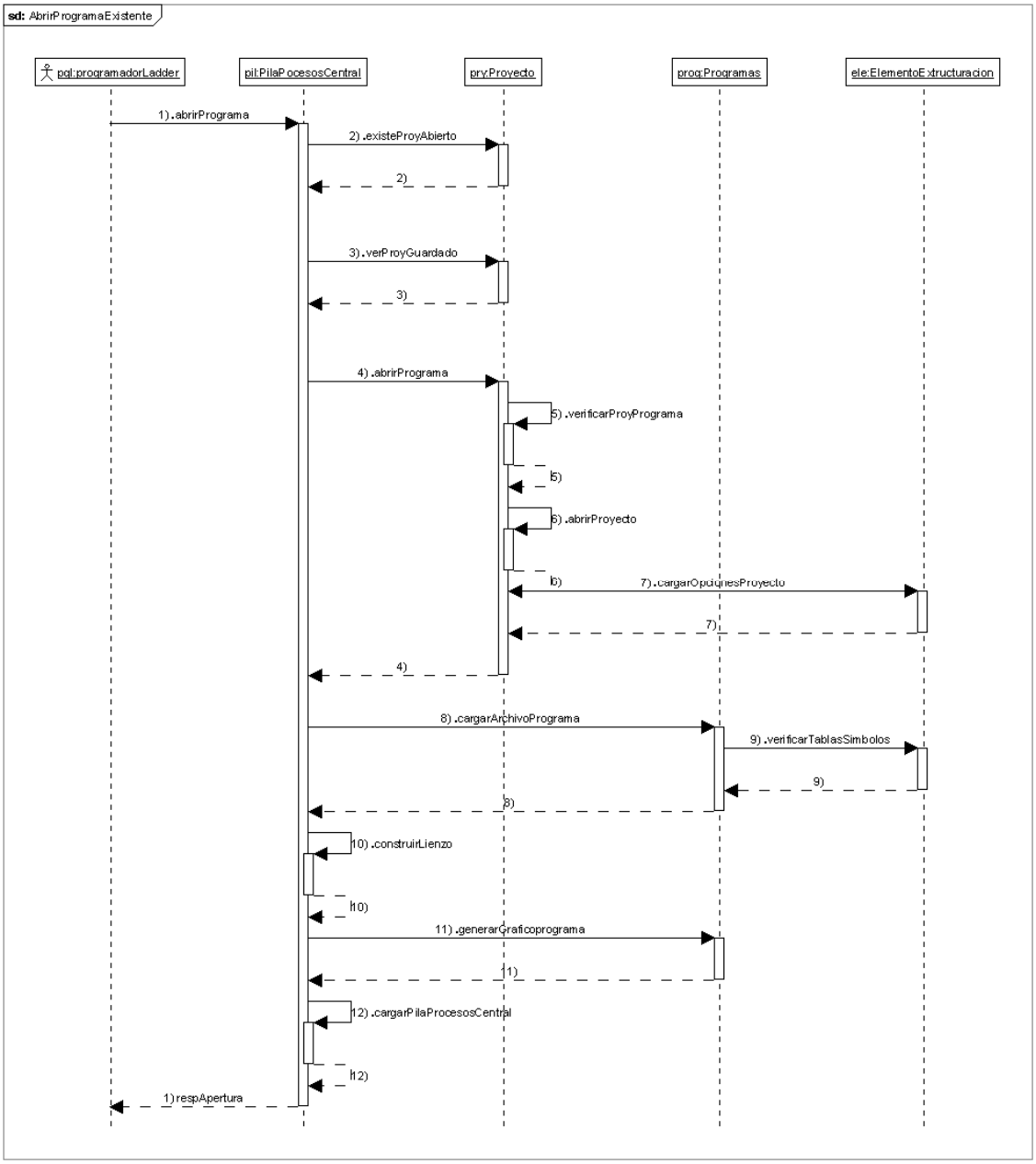
APÉNDICE C: Diagrama de clase de diseño del sistema

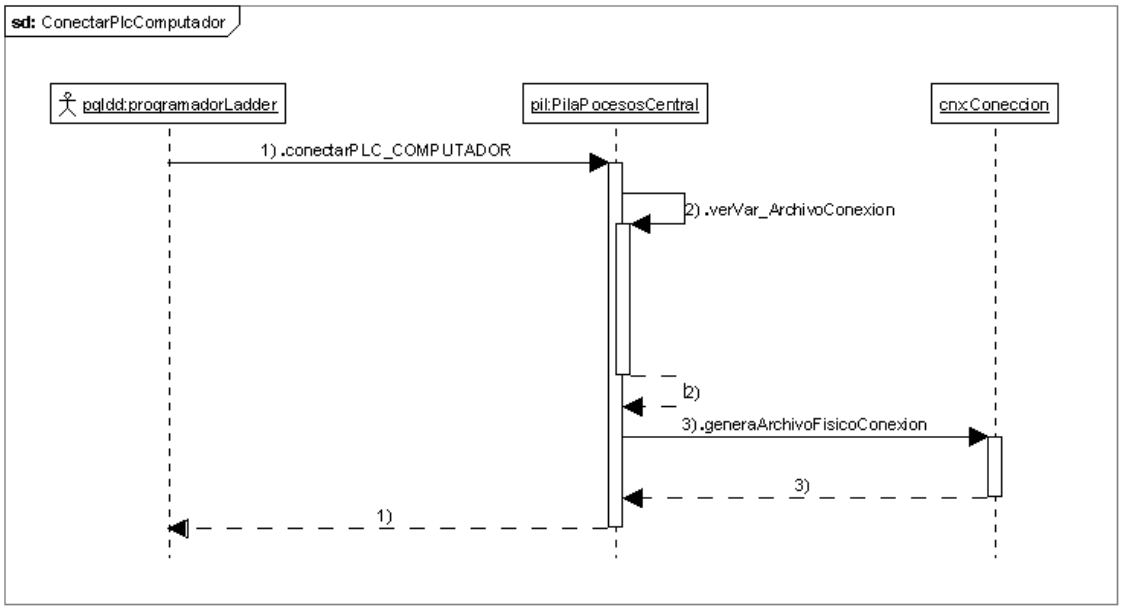


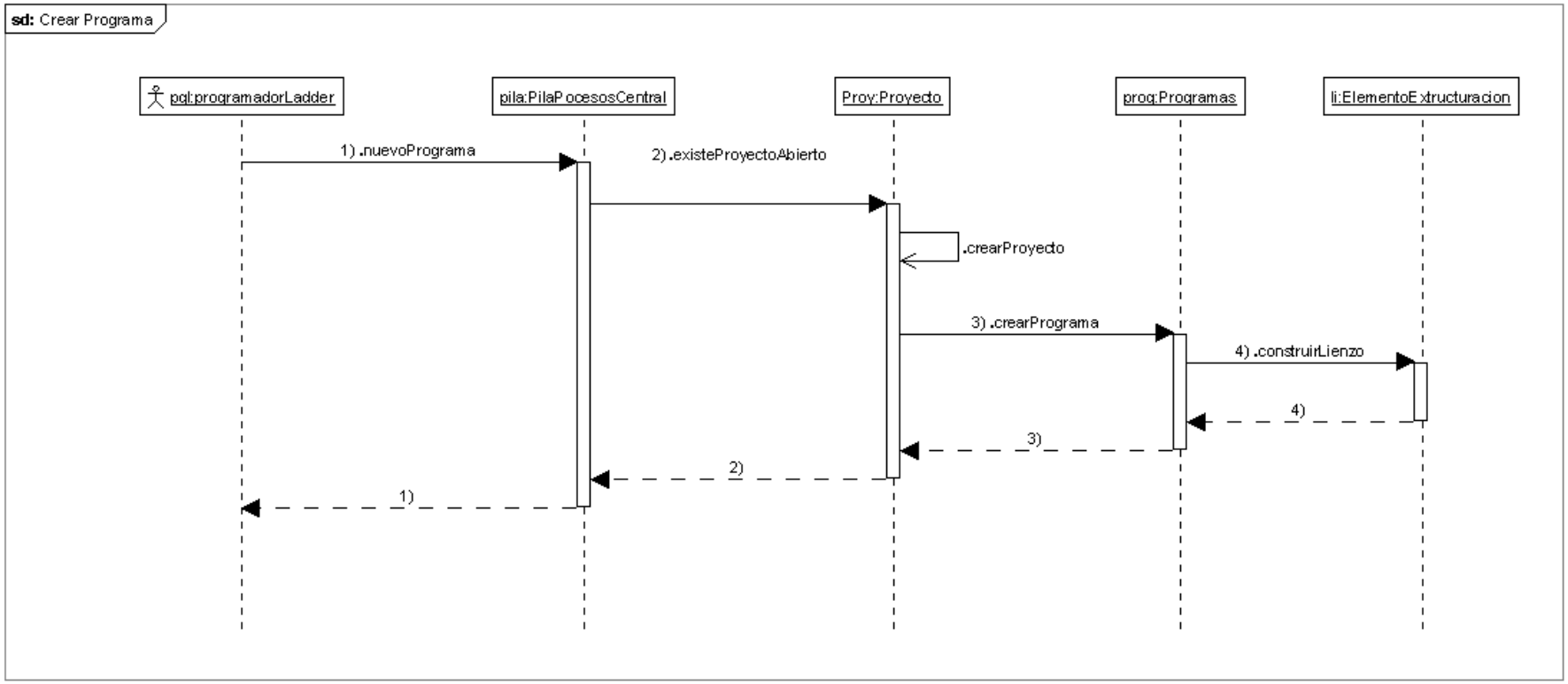
APÉNDICE D: Diagrama de clase de implementación del sistema

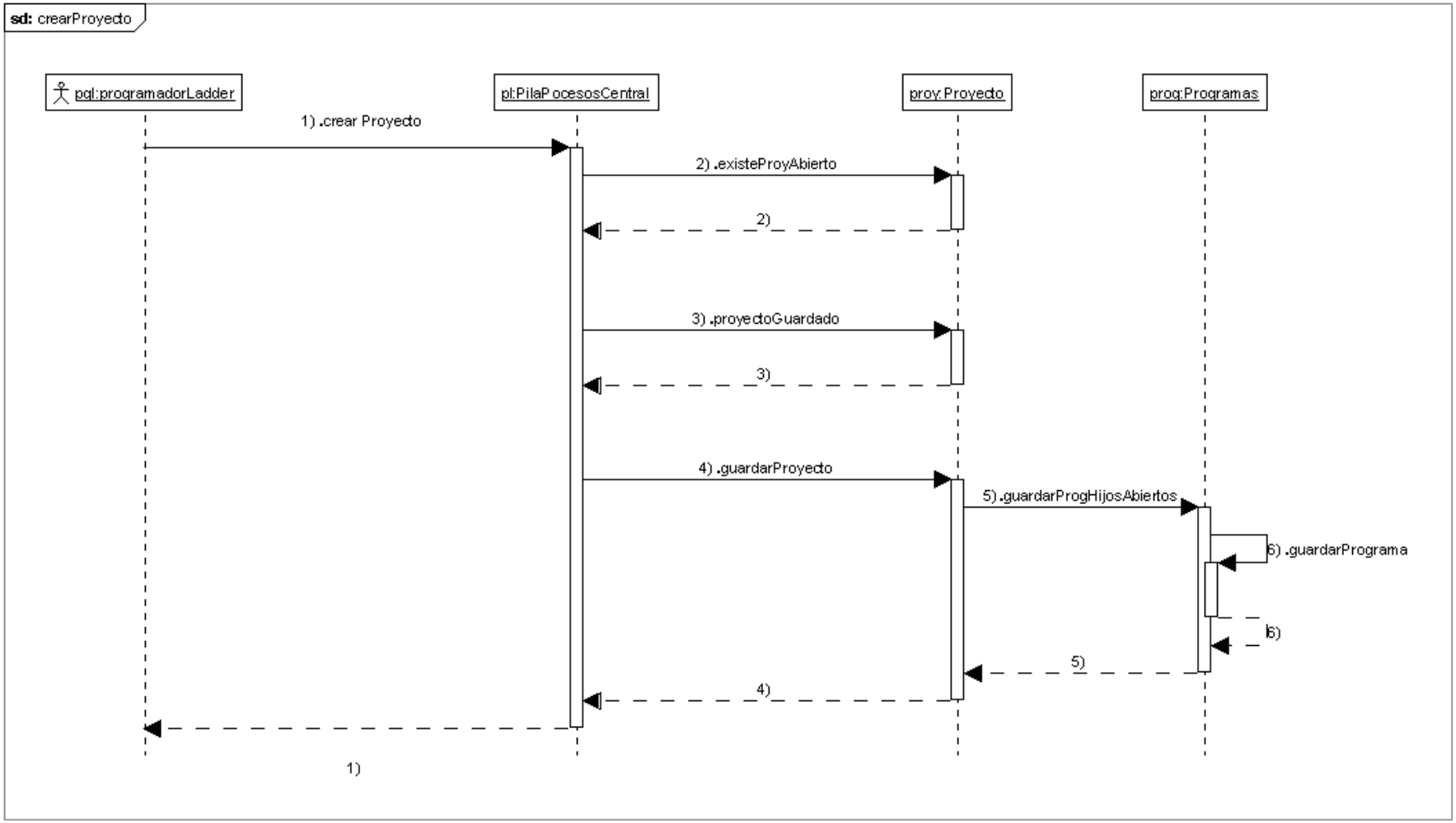

sd Diagrama de Clases de Implementacion
Tk
MSG
IDE
HOJA
Variables
Conexcion
Recursos
Configuracion
TipoDatos
ExploradorConfiguracion
PilaProcesosControl
Proyecto
Nodo
ExploradorArbol
ExploradorProyectos
LiteralesNumericos
LiteralesBit
LiteralesTiempo
CadenaCaracteres
Bool
String
Entero
Real
Date
Time
TimeOfDay
DateTime
LiteralesPrograma
Programas
Funcion
BloquePrograma
BloqueFuncion
ElementosEstructuracion
ElementosLadder
Contacto
Rele
BarraEnergia
Comentario
VisorPropiedades
LenguajeIEC 61131-3
Parametros
BarraEstado
Barra
BarraLadder
BarraHerramienta
Ayuda

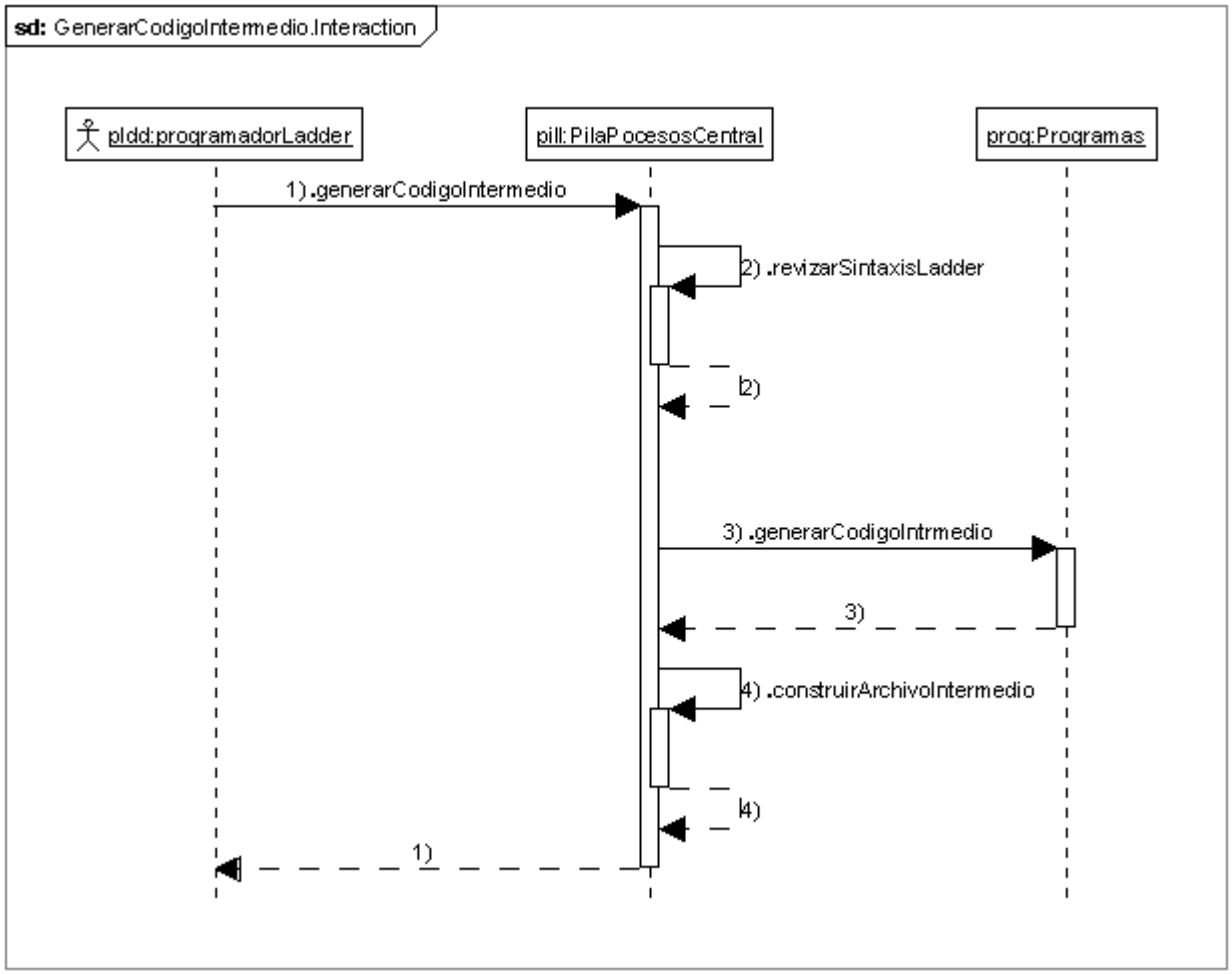
APÉNDICE E: Diagramas de Secuencia del sistema

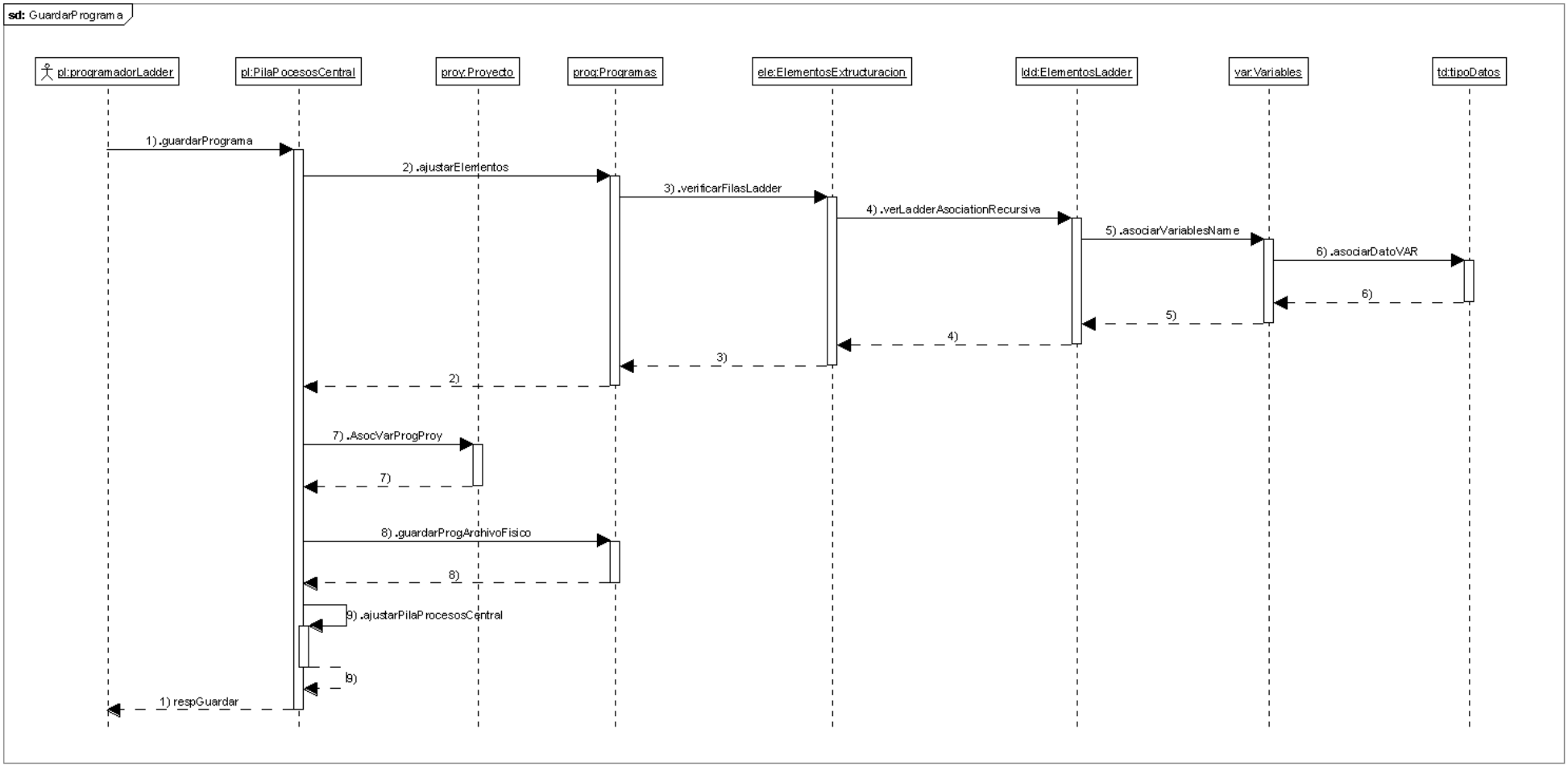


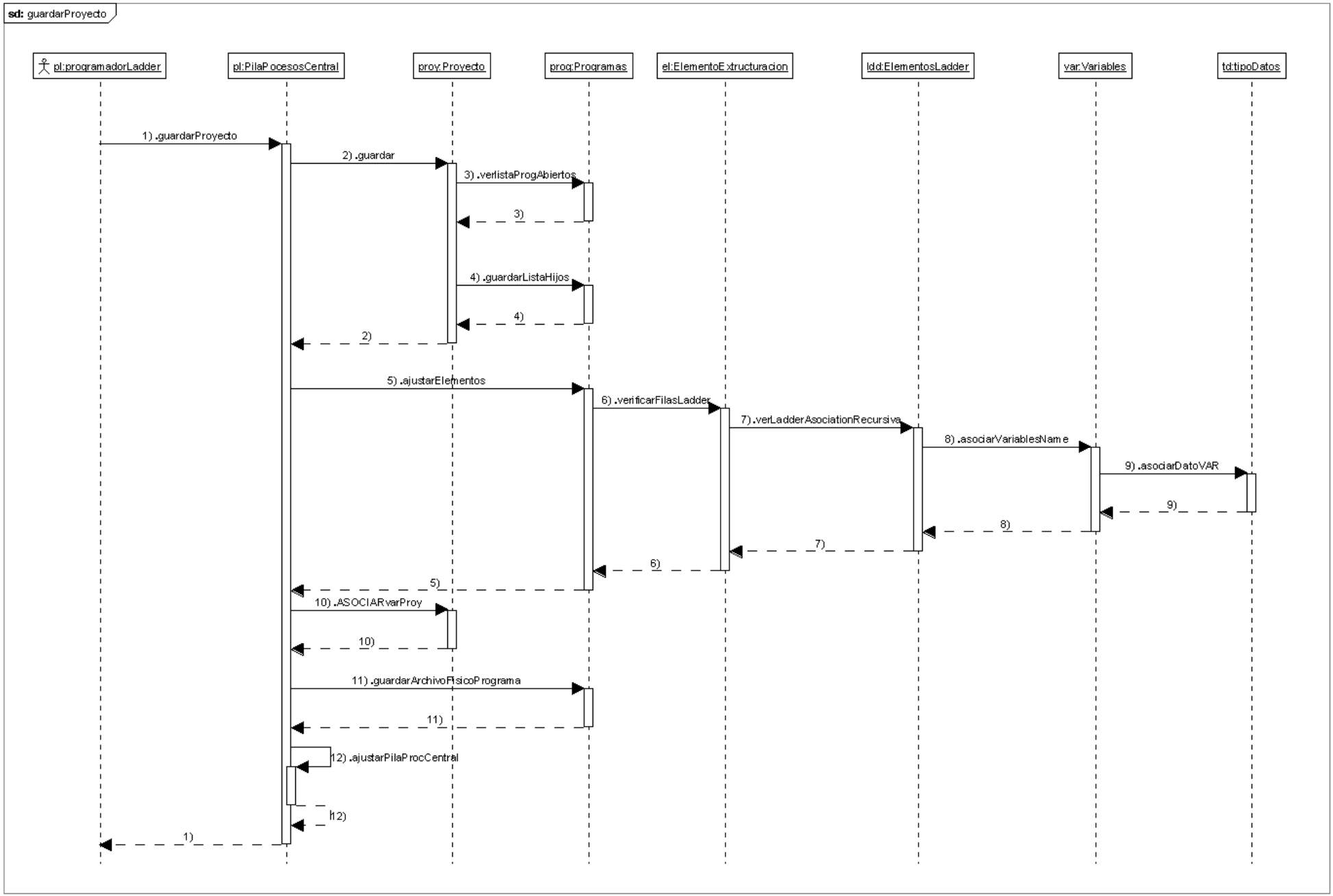

sd: ConectarPlcComputador
pgldd:programadorLadder
pil:PilaPocesosCentral
cnx:Coneccion
1) .conectarPLC_COMPUTADOR
2) .verVar_ArchivoConexion
b)
3) .generaArchivoFisicoConexion
3)
1)

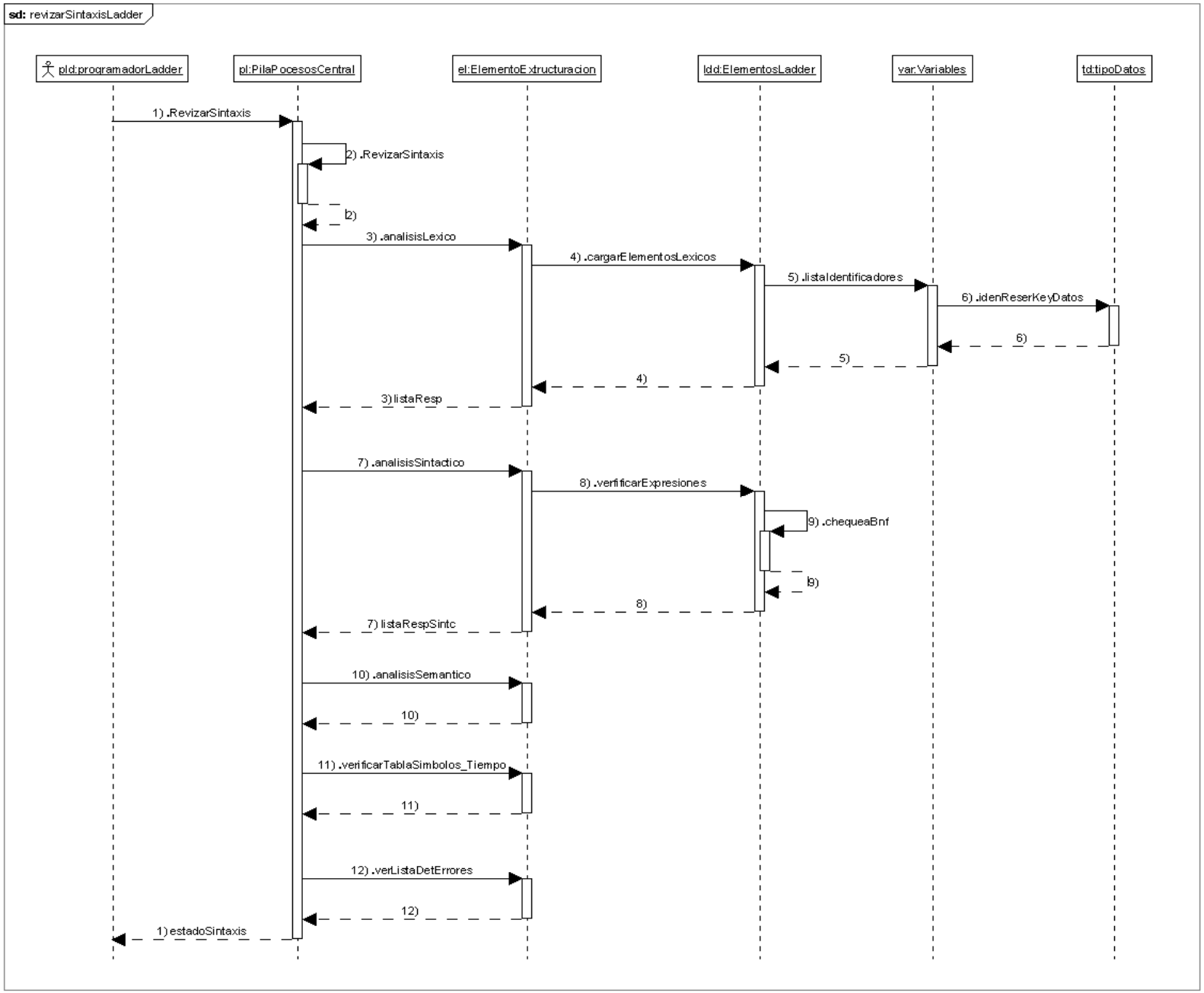

sd: Crear Programa
pql:programadorLadder
pila:PilaPocesosCentral
Proy:Proyecto
prog:Programas
li:ElementoExtructuracion
1) .nuevoPrograma
2) .existeProyectoAbierto
.crearProyecto
3) .crearPrograma
4) .construirLienzo
4)
3)
2)
1)


sd: crearProyecto
pql:programadorLadder
pl:PilaPocesosCentral
proy:Proyecto
prog:Programas
1) .crear Proyecto
2) .existeProyAbierto
2)
3) .proyectoGuardado
3)
4) .guardarProyecto
5) .guardarProgHijosAbiertos
6) .guardarPrograma
6)
5)
4)
1)

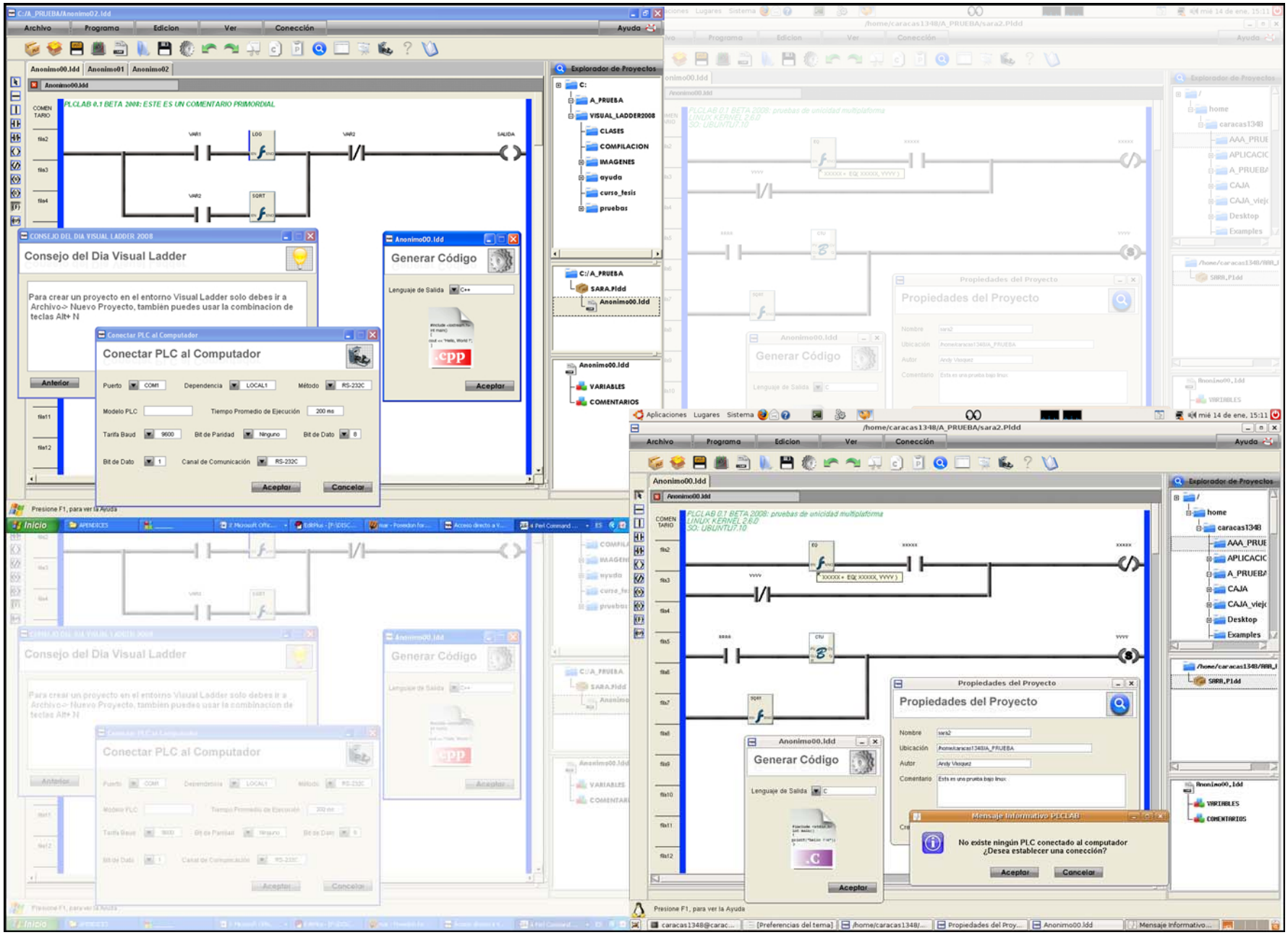

sd: GenerarCodigoIntermedio.Interaction
pldd:programadorLadder
pill:PilaProcesosCentral
prog:Programas
1).generarCodigoIntermedio
2).revizarSintaxisLadder
2)
3).generarCodigoIntrmedio
3)
4).construirArchivoIntermedio
4)
1)


sd: GuardarPrograma
pl:programadorLadder
pl:PilaPocesosCentral
proy:Proyecto
prog:Programas
ele:ElementosExtructuracion
ldd:ElementosLadder
var:Variables
td:tipoDatos
1) .guardarPrograma
2) .ajustarElementos
3) .verificarFilasLadder
4) .verLadderAsociationRecursiva
5) .asociarVariablesName
6) .asociarDatoVAR
6)
5)
4)
3)
2)
7) .AsocVarProgProy
7)
8) .guardarProgArchivoFisico
8)
9) .ajustarPilaProcesosCentral
9)
1) respGuardar

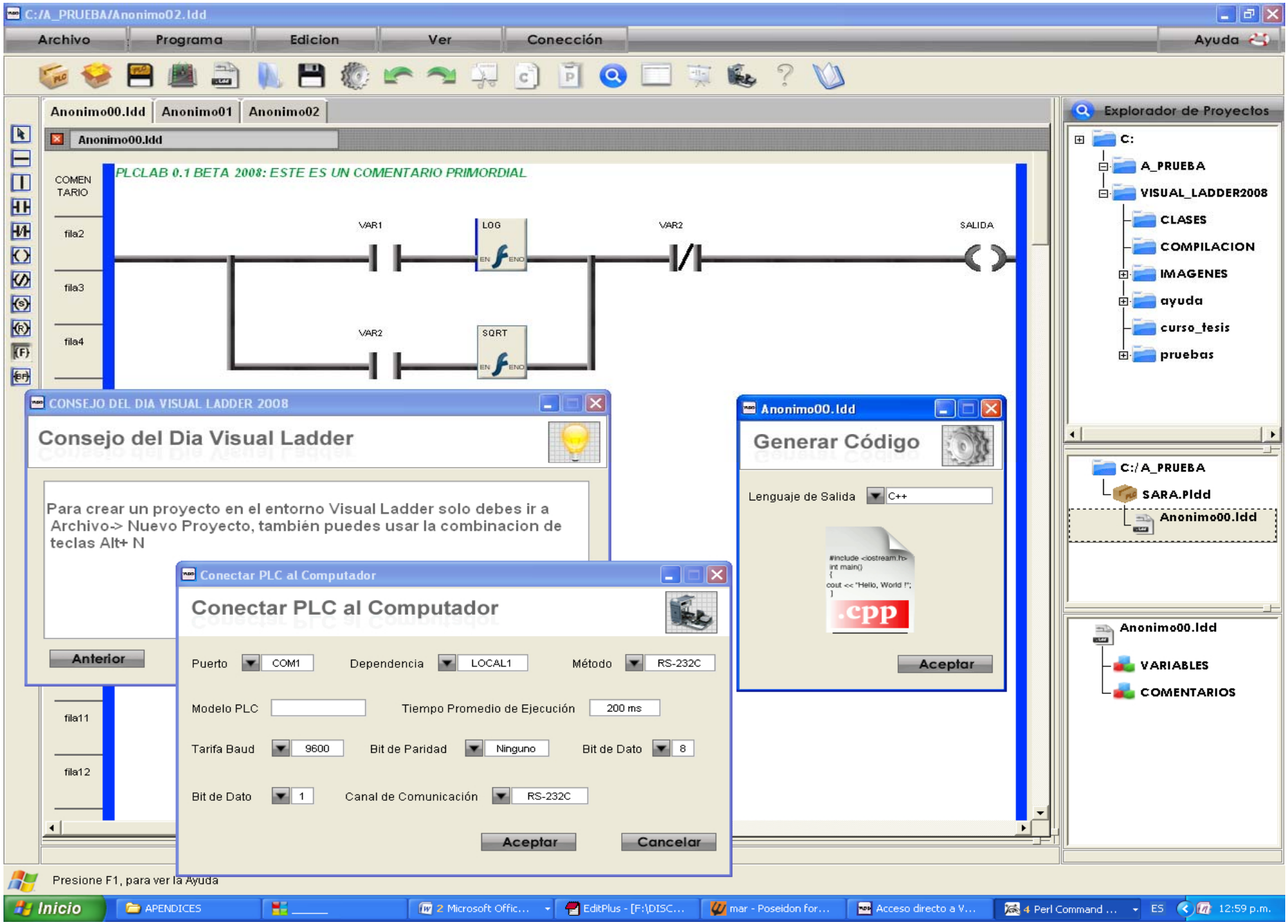

sd: guardarProyecto
pl:programadorLadder
pl:PilaPocesosCentral
proy:Proyecto
proy:Programas
el:ElementoExtructuracion
ldd:ElementosLadder
var:Variables
td:tipoDatos
1) .guardarProyecto
2) .guardar
3) .verlistaProgAbiertos
3)
4) .guardarListaHijos
4)
2)
5) .ajustarElementos
6) .verificarFilasLadder
7) .verLadderAsociationRecursiva
8) .asociarVariablesName
9) .asociarDatoVAR
9)
8)
7)
6)
5)
10) .ASOCIARvarProy
10)
11) .guardarArchivoFisicoPrograma
11)
12) .ajustarPilaProcCentral
12)
1)


sd: revizarSintaxisLadder
pld:programadorLadder
pl:PilaPocesosCentral
el:ElementoExtructuracion
ldd:ElementosLadder
var:Variables
td:tipoDatos
1) .RevizarSintaxis
2) .RevizarSintaxis
b)
3) .analisisLexico
4) .cargarElementosLexicos
5) .listaIdentificadores
6) .idenReserKeyDatos
6)
5)
4)
3) listaResp
7) .analisisSintactico
8) .verificarExpresiones
9) .chequeaBnf
b)
8)
7) listaRespSintc
10) .analisisSemantico
10)
11) .verificarTablaSimbolos_Tiempo
11)
12) .verListaDetErrores
12)
1) estadoSintaxis

APÉNDICE F: Interfaces del sistema


Archivo
Programa
Edicion
Ver
Conección
Ayuda
Explorador de Proyectos
Consejo del Dia Visual Ladder
Para crear un proyecto en el entorno Visual Ladder solo debes ir a Archivo-> Nuevo Proyecto, también puedes usar la combinacion de teclas Alt+ N
Anterior
Conectar PLC al Computador
Puerto
Dependencia
Método
Modelo PLC
Tiempo Promedio de Ejecución
Tarifa Baud
Bit de Paridad
Bit de Dato
Canal de Comunicación
Aceptar
Cancelar
Generar Código
Lenguaje de Salida
Propiedades del Proyecto
Nombre
Ubicación
Autor
Comentario
Mensaje Informativo PLCLab
No existe ningún PLC conectado al computador
¿Desea establecer una conección?
Presione F1, para ver la Ayuda

APÉNDICE G: Interfaz del primer beta del Visual Ladder


C:/A_PRUEBA/Anonimo02.ldd
Archivo
Programa
Edicion
Ver
Conección
Ayuda
Anonimo00.ldd
Anonimo01
Anonimo02
PLCLAB 0.1 BETA 2008: ESTE ES UN COMENTARIO PRIMORDIAL
COMENTARIO
fila2
fila3
fila4
fila11
fila12
VAR1
LOG
VAR2
SALIDA
SQRT
Explorador de Proyectos
C:
A_PRUEBA
VISUAL_LADDER2008
CLASES
COMPILACION
IMAGENES
ayuda
curso_tesis
pruebas
C:/A_PRUEBA
SARA.Pldd
Anonimo00.ldd
VARIABLES
COMENTARIOS
CONSEJO DEL DIA VISUAL LADDER 2008
Consejo del Dia Visual Ladder
Para crear un proyecto en el entorno Visual Ladder solo debes ir a Archivo-> Nuevo Proyecto, también puedes usar la combinacion de teclas Alt+ N
Anterior
Anonimo00.ldd
Generar Código
Lenguaje de Salida
C++
.cpp
Aceptar
Conectar PLC al Computador
Conectar PLC al Computador
Puerto
COM1
Dependencia
LOCAL1
Método
RS-232C
Modelo PLC
Tiempo Promedio de Ejecución
200 ms
Tarifa Baud
9600
Bit de Paridad
Ninguno
Bit de Dato
8
Bit de Dato
1
Canal de Comunicación
RS-232C
Aceptar
Cancelar
Presione F1, para ver la Ayuda
Inicio
APENDICES
2 Microsoft Offic...
EditPlus - [F:\DISC...
mar - Poseidon for...
Acceso directo a V...
4 Perl Command ...
ES
12:59 p.m.

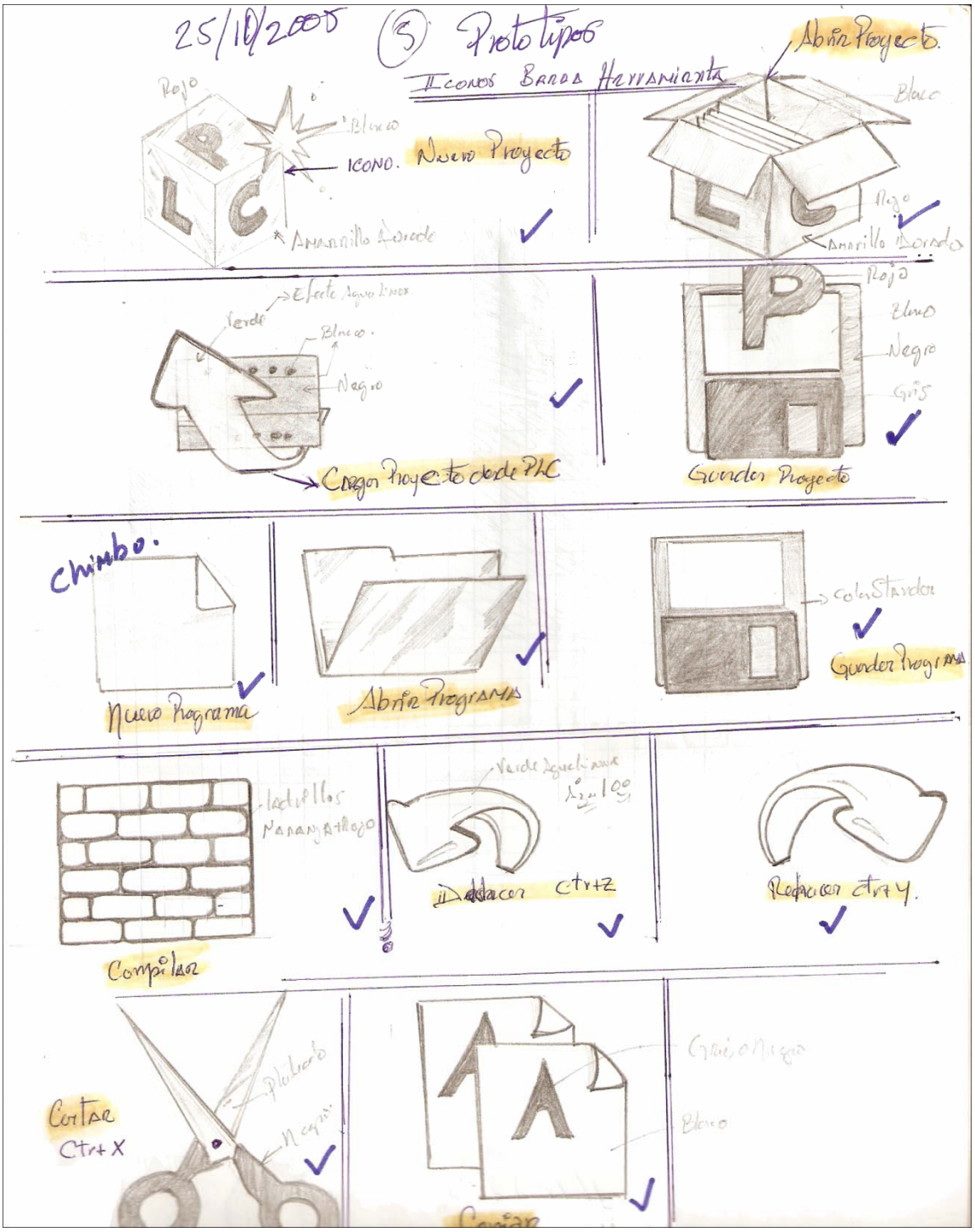
APÉNDICE H: Cuestionario de las entrevistas no estructuradas

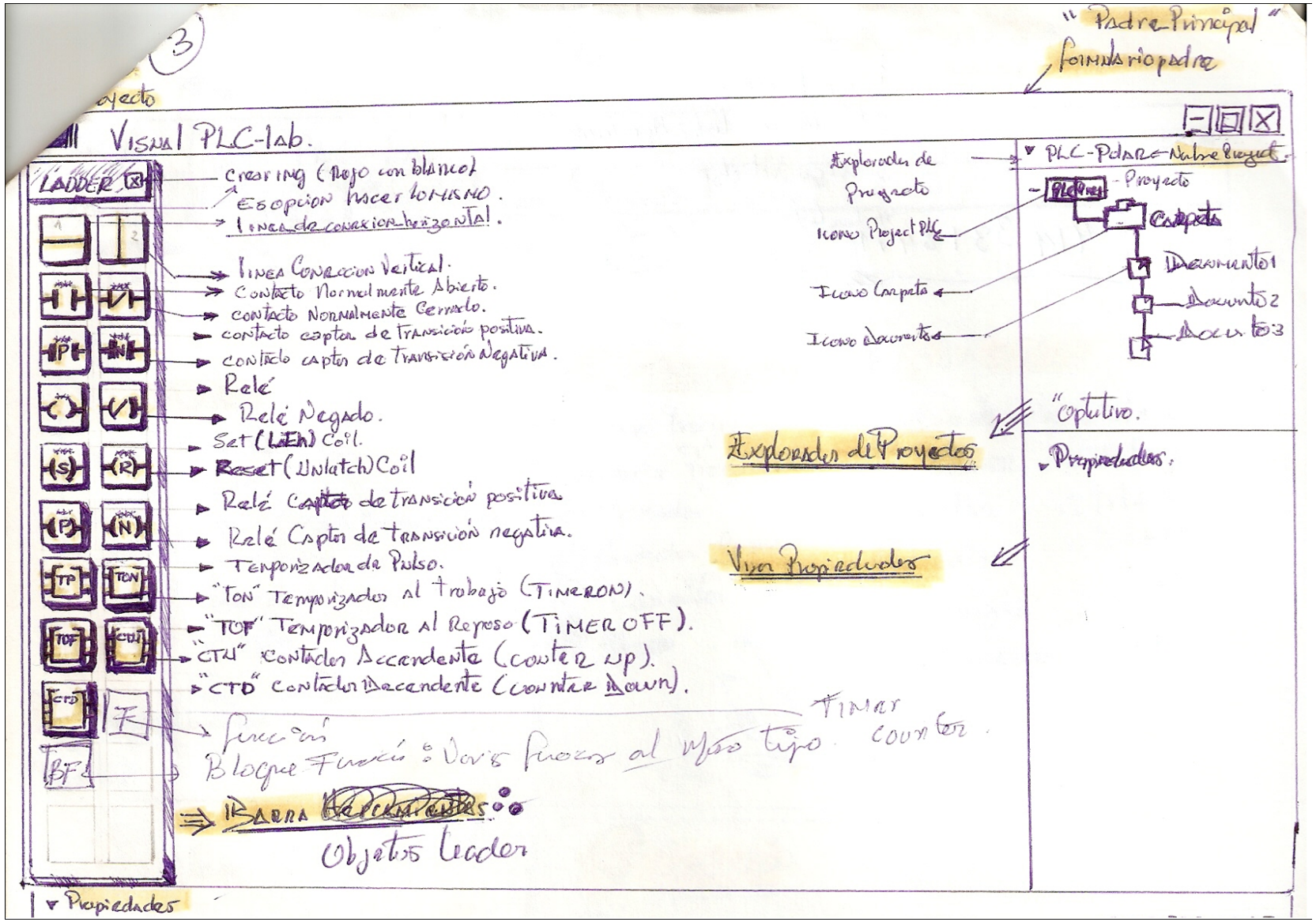
UNIVERSIDAD DE ORIENTE
NÚCLEO DE SUCRE
ESCUELA DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICAS
PROGRAMA DE LA LICENCIATURA EN INFORMÁTICA

Cuestionario para el levantamiento de información preliminar para el software VisualLadder

1. ¿Cómo debe interactuar el *software* con el PLC?
2. ¿Qué es el proyecto PLCLAB?
3. ¿En qué consiste la primera etapa del proyecto PLCLAB?
4. ¿Con qué lenguaje del estándar IEC 61131-3 se desea arrancar el proyecto?
5. ¿Cuál seria el alcance del *software* en la primera etapa del proyecto PLCLAB?
6. ¿De qué forma debe estar constituido el *software*?
7. ¿Qué filosofía de desarrollo se debe utilizar?
8. ¿Con qué herramienta se debe construir la aplicación?
9. ¿Por qué la aplicación debe ser multiplataforma?
10. ¿Cuáles artefactos proporcionados por UML necesita conservar SUTRONIX?
11. ¿El *software* debe generar proyectos y programas?
12. ¿Cuáles son las funciones del estándar que por defecto deben estar en el primer beta del *software*?
13. ¿Cuáles son los bloques de funciones del estándar que por defecto deben estar en el primer beta del *software*?
14. ¿Qué elementos del lenguaje serán implementados en el primer beta?
15. ¿Qué lenguaje de alto nivel desea utilizar como plantilla para generar el código intermedio en el *software*?

APÉNDICE I: Diseño de los prototipos de iconos finales




"Padre Principal"
Formulario padre
Visual PLC-lab.
LADDER
Creating (Hoja con blanco)
Escopion hacer lo mismo.
Linea de conexion horizontal.
Linea Conexion Vertical.
Contacto Normalmente Abierto.
Contacto Normalmente Cerrado.
contacto captor de transición positiva.
contacto captor de transición Negativa.
Relé
Relé Negado.
Set (Latch) Coil.
Reset (Unlatch) Coil
Relé Captor de transición positiva
Relé Captor de transición negativa.
Temporizador de Pulso.
"TON" Temporizador al trabajo (TIMERON).
"TOF" Temporizador al Reposo (TIMER OFF).
"CTU" Contador Ascendente (counter up).
"CTD" Contador Descendente (counter down).
Función
Bloque Función : Varios Inputs al mismo tipo
Timer
counter
Barra
Objetos Leader
Explorador de Proyecto
Icono Project PLC
Icono Carpeta
Icono Documento
PLC-Padre-Nuevo Proyect
Proyecto
Carpeta
Documento1
Documento2
Documento3
"Optativo.
Explorador de Proyectos
Propiedades.
Vista Propiedades
Propiedades

APÉNDICE J: Cronograma Interno dentro PUDS

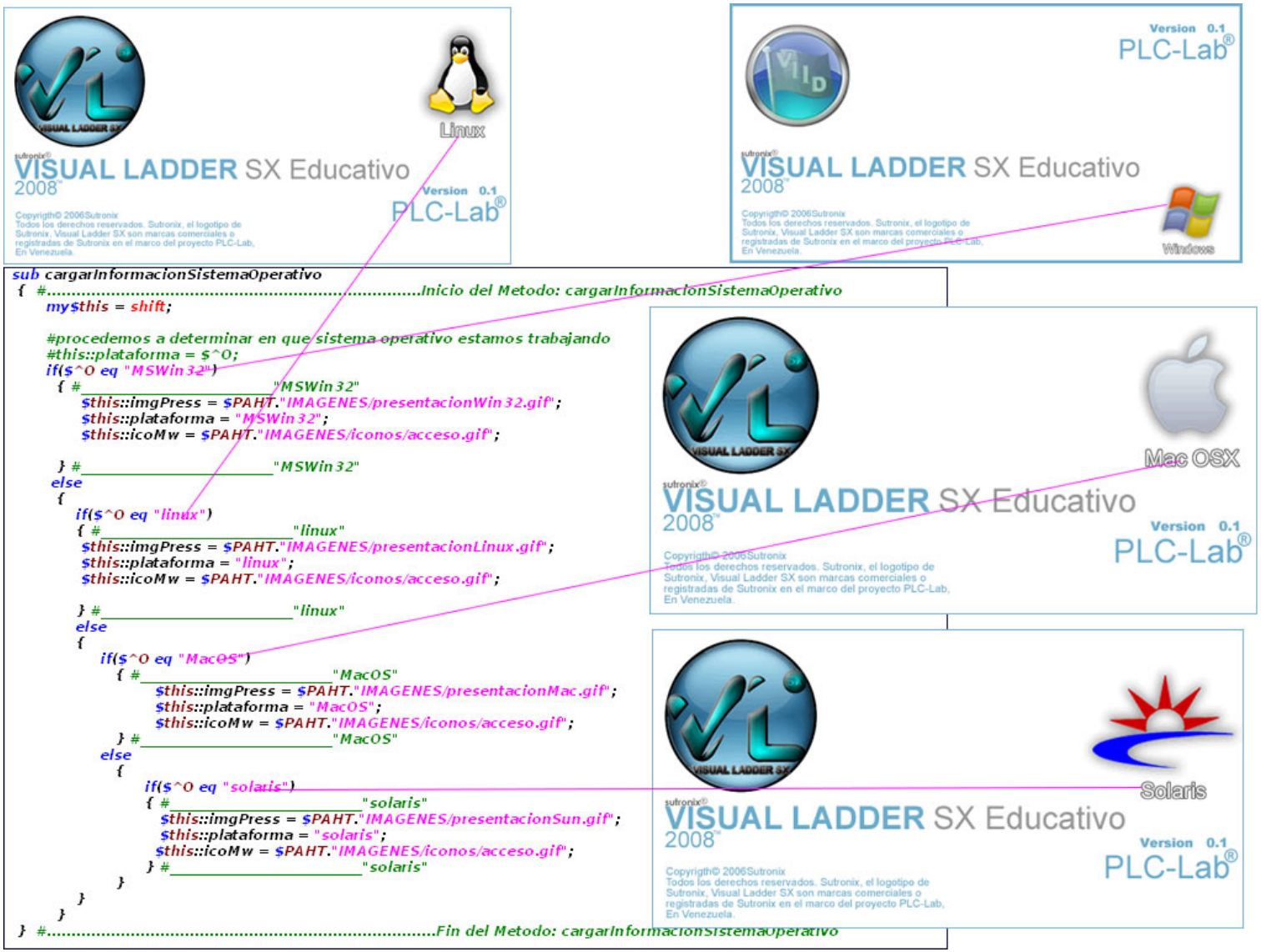
UNIVERSIDAD DE ORIENTE
NÚCLEO DE SUCRE
ESCUELA DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICAS
PROGRAMA DE LA LICENCIATURA EN INFORMÁTICA

Flujos de trabajados aplicados en las distintas fases del PUDS

APÉNDICE K: Mapeo de clases al lenguaje Perl

```
#!/usr/bin/perl
##################################################################################################################################################
#TRABAJO DE GRADO ::"SOFTWARE MULTIPLATAFORMA DE PROGRAMACION VISUAL PARA EL LENGUAJE EN ESCALERA
#              ::(LADDER) DE LA NORMA 61131-3, QUE SERVIRA PARA PROGRAMAR AUTOMATAS EN EL
#                PROYECTO PLCLAB DE SUCRE ELECTRONICA C.A "
#ANALISIS:: ANDY VASQUEZ
#DISEÑO:: ANDY VASQUEZ
#LOCACION::VENEZUELA-SUCRE-CUMANA
#TIME::15/02/2006
#ASUNTO::CLASE::PROYECTO;
##################################################################################################################################################

package PROYECTO;
 # {_____INICIO DE CLASE EXPLORADOR_PROYECTO_____
  # _____ATRIBUTOS DE LA CLASE_____
  # private :
      my $ide;
      my $mw; # ventana principal
      my $AbrirProy;
      my $plataforma;
      my $barraEst;#barra de estado
      my $expp; #mensajes del sistema
      my $colSt;#color de fondo estandar para la aplicacion
      my $cAbt; #color activo de los botones
      my $cAlt;#color de fondo de los textos alternativos

      #parametros de coneccion comun

     my $id;
     my $leng;
     my $nom;
     my $ubic ;
     my $autor;
     my $fechaC;
     my $fechaUltA;
     my $comentario;
     my $semaforo; # indica si existe alguin proyecto abierto



   #cajas de textos
      my $txt1;
      my $txt2;
      my $txt3;
      my $txt4;
 #botones
       my $bt1;
       my $bt2;
       my $bt3;

     my $largo;
     my $ancho;
     my $fuente;
     my $ventOp;
     my $bd;

     #imagenes
     my $ico1;
     my $img1;
     my $imgBtA;
     my $imgBtC;
     my $imgBtAD;
     my $imgBtCD;
     my $imgBanner;
```

APÉNDICE L: Pruebas y ajustes multiplataforma



APÉNDICE M: Manual de usuarios del sistema

PLCLAB VISUAL LADDER

Guía de Usuarios

Contenido

Introducción…… 1

Requerimientos…… 2

Instalación…… 3

Expresiones…….. 4

Sumario de Funciones…… 5

Sumario de Bloque de Funciones…….. 6

Manejo del Software…… 7

Introducción 1

Desde diciembre del año 2004 cuando el presidente Hugo Chávez formalizó, el decreto 3390, publicado en gaceta oficial de la República Bolivariana de Venezuela No. 38.095, se activó un plan a nivel nacional de información orientado a universidades y empresas, con la intensión de dar a conocer las nuevas directrices en cuanto a sistemas y servicios informáticos, para las empresas y organismos públicos venezolanos.

Gran cantidad de charlas, reuniones y eventos se efectuaron por parte de la gerencia de Automatización Informática y Telecomunicaciones AIT de PDVSA en la región oriental del país, específicamente en la ciudad de Cumaná, con la intención de sembrar una tendencia de investigación y desarrollo en software libre, es por ello que en el marco del proyecto PLCLAB se decidió crear el software de manipulación del plc con herramientas software libre para así estar acorde a la dinámica del país y que además fuese portable a cualquier sistema operativo, es allí cuando se comienza a trabajar en la elaboración de dicha aplicación con el fin de obtener un primer beta estable que permita una capacidad operativa inicial basada en el lenguaje escalera, con la intensión de presentarlo en PDVSA y el ministerio de ciencia y tecnología y así solicitar los recursos necesarios para apalancar el proyecto de creación de un plc venezolano.

El software PLCLAB VISUAL LADDER en su primer beta permite la creación y edición de proyectos la creación de bloques de programas, bloque de funciones , funciones la generación de algoritmos gráficos basado en el lenguaje escalera con una interfaz impecable, la asociación de variables locales y globales, operaciones de edición clásicas traducción de la lógica del algoritmo a otros lenguajes de alto nivel, la generación de archivos de extensión propia y además la portabilidad de la aplicación en varias plataformas.

Esta guía tiene como finalidad servir de apoyo para iniciar la programación en escalera de futuro PLC venezolano mediante el conocimiento de:

- Estructura de un proyecto.
- Expresión de los datos.
- Tipos de Datos.
- Declaración de variables.
- Tipos de Programas.
- Instrucciones básicas.
- Manejo del software PLCLAB VISUAL LADDER

Requerimientos 2

¿Qué necesita usted?

- El intérprete del lenguaje Perl en el sistema operativo que esté usando**.**
- Un procesador como mínimo Pentium III, AMD.
- Capacidad mínima en memoria RAM de 128 MB.
- Espacio en disco duro mayor de 16 MB.

Instalación 3

Lo primero que debe hacer es adquirir el instalador oficial del VISUAL LADDER, lo puede descargar desde:

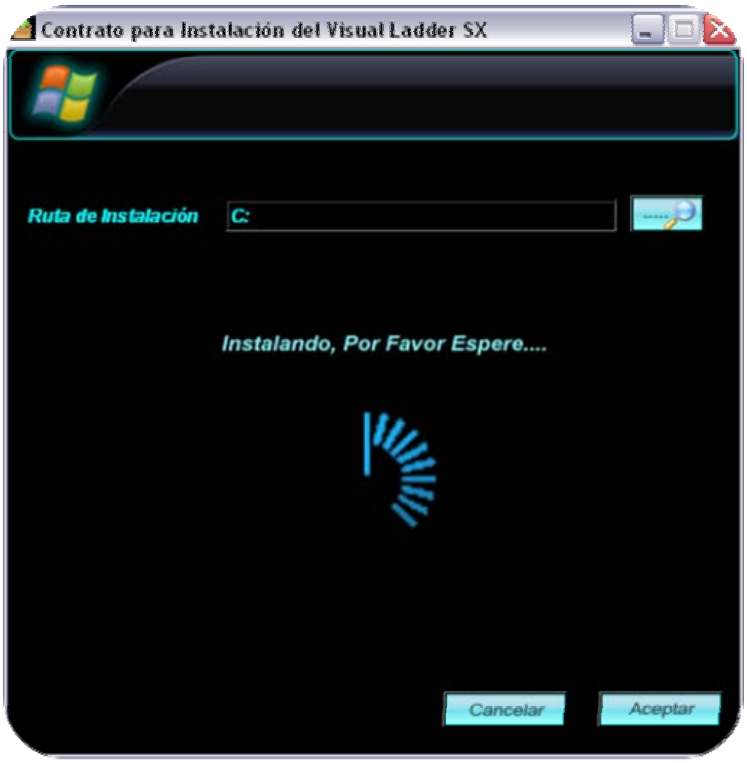
www.sutronix.com.ve/descargas/visualadder.html

Una vez adquirido el instalador simplemente presiona doble clic sobre él.

El instalador primeramente determinará el sistema operativo en el cual se va a instalar la aplicación, para ello usted observará una interfaz como la siguiente:

Cuando detecta la plataforma el instalador informa y presenta el siguiente cuadro de dialogo.

Al presionar aceptar el instalador procederá a copiar los archivos en el directorio definido para el sistema operativo residente.



Una vez instalado busque en el directorio el icono de acceso directo de la aplicación:

Expresiones 4

EL lenguaje escalera soportado por la aplicación posee elementos de estructuración como lo es la expresión de variables.

IDENTIFICADOR

EL identificador se entiende como el nombre designado para una variable del programa y puede ser una combinación de caracteres alfabéticos (reconocidos en el idioma inglés), números y caracteres de subrayado “_”.

Reglas:

- EL identificador no acepta el carácter “espacio”
- EL identificador acepta 16 caracteres alfabéticos máximo en caso de variables normales.
- EL identificador no puede ser igual al de una palabra reservada.

Ejemplos:

- Caso de letras y números: px123, px123y, QR68, SENSOR.
- Caso de letras números y carácter de subrayado insertos o al comienzo: sx_c_2, cap_we, _main, 5vt_7VT, _MOTOR.

EXPRESION DE LOS DATOS

Los distintos tipos de datos que se utilizan se pueden clasificar en tres grupos: literales numéricos, cadena de caracteres, y literales de tiempo.

Literales numéricos:

- Existen 2 clases de literales numéricos: enteros y reales.
- Los caracteres de subrayado insertados entre los dígitos de un literal numérico no tienen significado.
- Los símbolos “+” y “-” pueden ser usados por el exponente. La letra “E” identificará el exponente y no existe diferencia entre minúscula y mayúscula.

- Los decimales que tengan punto decimal serán reconocidos como literales reales.
- Los literales reales deben expresarse como sigue; Ej12.0E-5(12E-5 es incorrecto).
- Los símbolos "+" y "-" no deben ser usados en expresiones binarias.
- Los datos booleanos pueden ser expresados por los enteros 0 y 1.

Cadena de caracteres (STRING):

Los caracteres encerrados por el símbolo comilla simple (') serán reconocidos como una cadena de caracteres. Por ejemplo el dato '-25.36' no representa a un literal real dado que está encerrado entre comillas simples por lo cual -25.36 se interpreta como una cadena de caracteres.

La longitud está restringida a 16 caracteres para cadena de caracteres tipo constante y 30 caracteres para la inicialización de variables.

Literales de Tiempo:

Los literales de tiempo son clasificados en dos tipos: Duración ***(TIME)***, Hora ***(TIME_OF_DAY)***, Fecha ***(DATE)***, Fecha y Hora ***(DATE_AND_TIME).***

- ***Datos de Duración:***

  Son usados para la medición y asignación de tiempos de un evento.
  - El dato debe ser precedido por el caracteres T# o t#.
  - El dato está definido en función de días (**d**), horas (**h**), minutos (**m**), segundos (**s**) y milisegundos (**ms**) los cuales deben ser indicados en este orden jerárquico pudiendo comenzar desde cualquier unidad. No se permite exceptuar unidades intermedias.
  - El carácter de subrayado "_" no es usado.

- ***Fecha y Hora***

Tipo de Datos	Prefijo	Expresión	Ejemplo
Literales de fecha ***(DATE)***	***D#***	***[Prefijo][Año]-[Mes]-[Dia]***	***D#1999-10-04***
Literales de hora ***(TIME_OF_DAY)***	***#TOD***	***[Prefijo][h]:[m]:[s]***	***TOD#15:36:53.23***
Literales de fecha y hora ***(DATE_AND_TIME)***	***#DT***	***[Prefijo][Año]-[Mes]-[Dia]-][h]:[m]:[s]***	***DT#1999-10-04-15:36:53.23***

EXPRESION DE LOS TIPOS DE DATOS

Tipo	Descripción	Tamaño (en bits)	Rango
INT	***Entero***	***16***	*-32768 a 32767*
REAL	***Real***	***32***	*-3.402823E38 a – 1401298E-45 1.401298E-45 A 3.402823E38*
TIME	***Duración***	***32***	*T#0S a T#49D17H2M47S295MS*
DATE	***Fecha***	***16***	*D#1984-01-01 A D#2163-06-06*
TIME_OF_DAY	***Hora***	***32***	*TOD# 00:00:00 a TOD#23:59:59.999*
DATE_AND_TIME	***Fecha y Hora***	***64***	*DT#1984-01-01-00:00:00 a DT#2163-06-06-23:59:59.999*
STRING	***Cadena de caracteres***	***30*8***	-
BOOL	***Booleano***	***1***	*0,1*
BYTE	***Cadena de caracteres***	***8***	*16#0 a 16#FF*
WORD	***Cadena de caracteres***	***16***	*16#0 a 16#FFFF*
DWORD	***Cadena de caracteres***	***32***	*16#0 a 16#FFFFFFFF*
LWORD	***Cadena de caracteres***	***64***	*16#0 a 16#FFFFFFFFFFFFFFFF*

DECLARACIÓN DE VARIABLES

Al declarar una variable se debe especificar:

- El identificador (***Name***).
- El tipo de variable (***Variable Kind***).

Tipo de Variable	*Descripción*
VAR	*Variable general de lectura y escritura*
VAR_RETAIN	*Variable retentiva (Tiene la opción de conservar su valor actual después de desenergizar el PLC)*
VAR_CONSTANT	*Variable de solo lectura*
VAR_EXTERNAL	*Declaración de variable tipo global (VAR_GLOBAL)*

- El tipo de dato de la variable (***Data Type***) INT, REAL, BOOL, TIME, etc.
- Las variables pueden ser inicializadas en un valor definido. Las variables declaradas ***VAR_EXTERNAL*** y variables directas no pueden ser inicializadas.

TIPOS DE PROGRAMAS

El programa creado puede ser del tipo: función (***Function***), bloque de funciones (***Function Block***) o bloque de programa (***Program Block***).

- ***FUNCIÓN:***
  - La función es una operación o conjunto de operaciones que genera una salida.
  - La función no puede retener información de estados internamente, es decir no posee variables internas, por lo que la función genera siempre el mismo resultado (salida) para iguales argumentos (parámetros de entrada).
- ***BLOQUE DE FUNCIONES***
  - El bloque de funciones puede tener varias salidas.
  - El bloque de funciones puede almacenar datos internos.
- ***BLOQUE DE PROGRAMA***
  - Los bloques de programas o programas son declaraciones instancias y se le asigna un nombre.
  - Las variables directas pueden ser usadas en el programa.
  - Los programas pueden llamar a funciones y bloques de funciones.
  - Los programas no pueden llamarse así mismo.

Sumario de Funciones 5

FUNCIONES DE OPERACIONES NUMÉRICAS

Nombre	*Descripción*
ABS	*Operación Valor Absoluto*
SQRT	*Operación Raíz Cuadrada*
LN	*Operación Logaritmo Natural*
LOG	*Operación Logaritmo en base 10.*
EXP	*Operación Exponencial Natural*
SIN	*Operación Seno de la entrada en radianes*
COS	*Operación Coseno de la entrada en radianes*
TAN	*Operación Tangente de la entrada en radianes*

FUNCIONES DE OPERACIONES NUMÉRICAS BÁSICAS

Nombre	*Descripción*
ADD	*Operación que suma dos valores*
MULT	*Operación que multiplica dos valores*
SUB	*Operación que resta dos valores*
DIV	*Operación que divide dos valores*

FUNCIONES DE COMPARACION

Nombre	*Descripción*
GT	*Comparación "Mas grande que"*
GE	*Comparación "Mas grande o igual que"*
EQ	*Comparación "Igual que"*
LE	*Comparación "Menor o igual que"*
LT	*Comparación "Menor que"*
NE	*Comparación "Diferente que"*

Sumario de Bloques de Funciones 6

CONTADORES

Nombre	*Descripción*
CTU	*Contador ascendente.*
CTD	*Contador descendente.*
CTUD	*Contador ascendente/ descendente.*

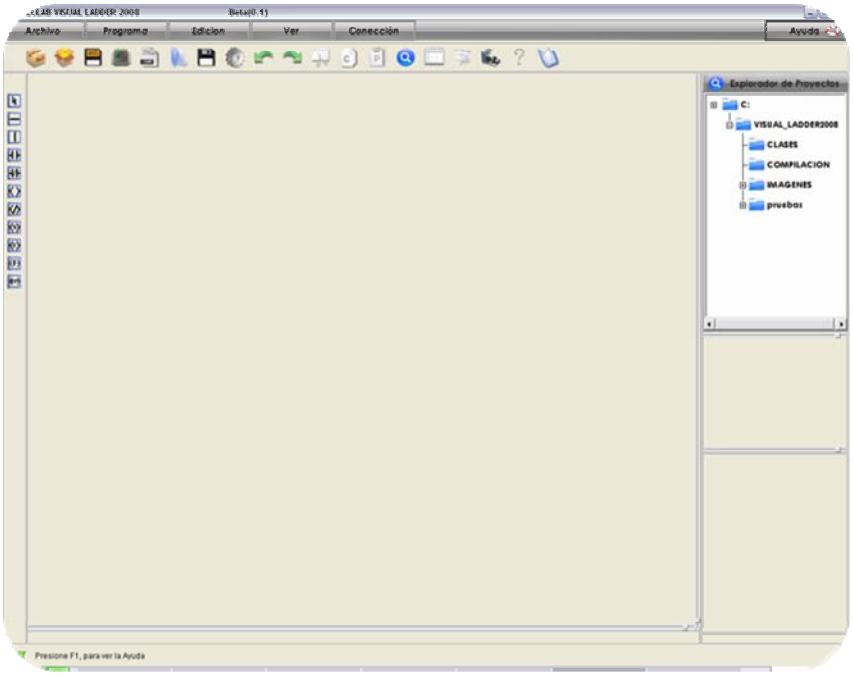
TEMPORIZADORES

Nombre	*Descripción*
TP	*Temporizador ONE SHOT.*
TON	*Temporizador ON DELAY*
TOF	*Temporizador OFF DELAY.*

Manejo del Software 7

Al momento de arrancar la aplicación observará la imagen de presentación del software.

Después de cargar se le presentará el entorno principal del programa:



El entorno de divide en cinco secciones:

MENU PRINCIPAL

Sección del entorno que le permite acceder de forma ordenada de a los eventos e información disponibles por la aplicación.

BARRA DE HERRAMIENTAS

Sección del entorno que le permite acceder directamente a los métodos por medio de los iconos representativos del software.

BARRA DE LADDER

Sección del entorno que permite acceder a los distintos tipos de elementos gráficos del lenguaje escalera, que permiten definir la lógica de los programas.

BARRA DE LADDER

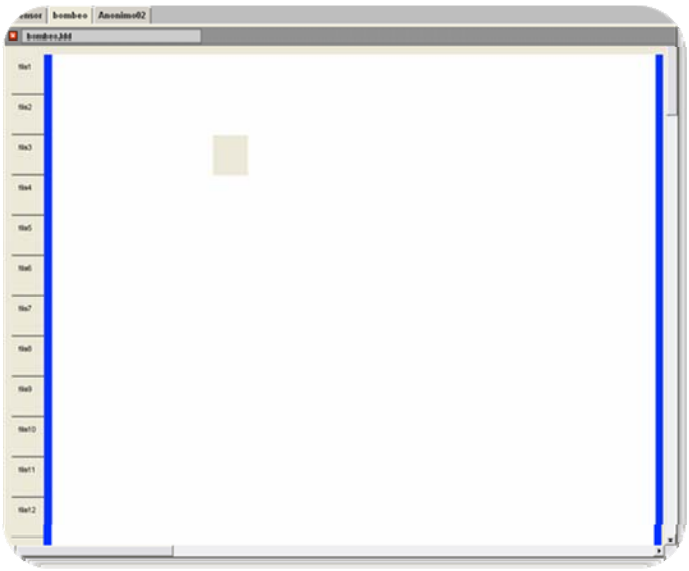
Sección del entorno que publica e informa todas las operaciones que se ejecutan en la aplicación.

EXPLORADOR DE PROYECTO

Sección del entorno que permite buscar a través de un árbol de carpetas del sistema operativo, los distintos proyectos y programas que genera el sistema, además de variables y funciones asociadas.

LIENZO DE PROGRAMAS

Sección del entorno donde se despliegan todos los programas.



FUNCIONALIDAD DEL SISTEMA

El sistema permite la ejecución de un conjunto de funcionalidades desplegadas en las distintas secciones de entorno.

Dentro de estas funcionalidades tenemos:

Nuevo Proyecto:

Esta opción, le permite la creación de proyectos nuevos y es accedida desde la barra de menú, en la sección **Archivo.**

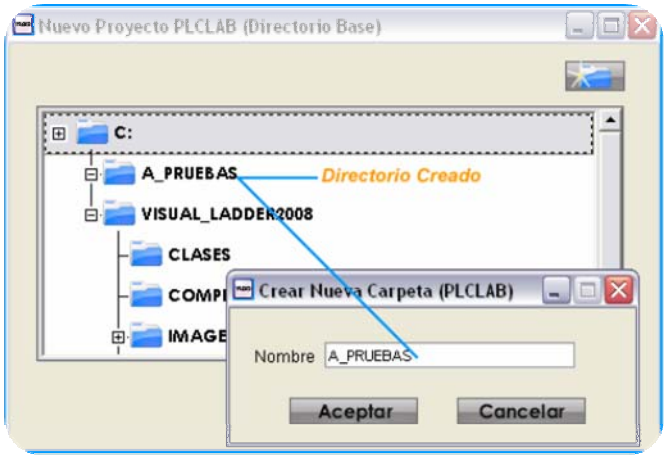
También es disparado éste evento usando la combinación de teclas **Crtl + P,** su icono se refleja en la barra de herramienta para un acceso por reconocimiento gráfico.

El sistema solicita los datos del nuevo proyecto a través de la siguiente interfaz.

Usted debe ingresar obligatoriamente el nombre del proyecto y la ubicación, para la ubicación puede seleccionar el directorio presionando el botón buscar directorio.

Dicho botón activara el explorador de ubicaciones que desplegará el árbol de directorios del sistema, además usted podrá también crear carpetas de manera dinámica.

Las opciones de carpeta permiten ingresar un nuevo directorio en el nodo seleccionado del árbol.



Una vez definido el directorio si lo desea puede ingresar el nombre del creador del proyecto y algún comentario asociado y se manda a crear el proyecto.

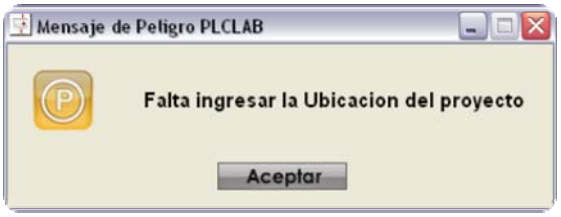
A la hora crear el nuevo proyecto son varios los mensajes que el sistema le puede presentar.

- Cuando se define el nombre del proyecto no se puede colocar espacios en blanco.

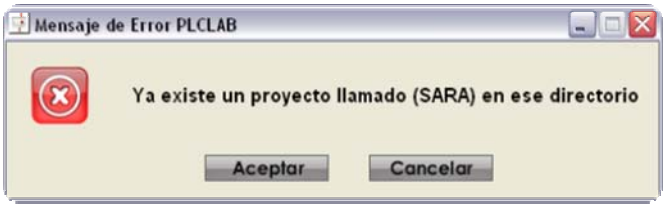
- El nombre del proyecto no puede tener más de 12 caracteres.

- Es obligatorio ingresar el nombre del proyecto.

- Es obligatorio definir la ruta donde se creará el proyecto.



- No se puede crear un proyecto con el mismo nombre a uno ya existente en un directorio específico.



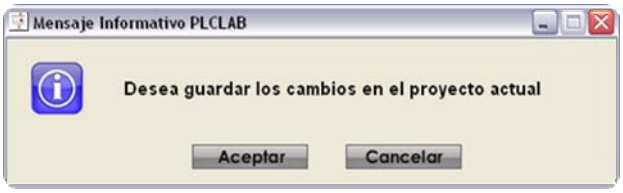
Una vez solventado cualquiera de estos mensajes el proyecto se creará satisfactoriamente en el directorio especificado.

Abrir Proyecto:

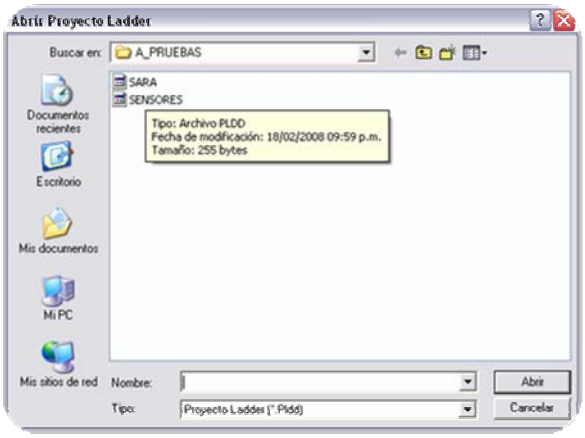
Está opción permite abrir cualquier proyecto existente.

También es disparado éste evento usando la combinación de teclas **Crtl + O,** su icono se refleja en la barra de herramienta para un acceso por reconocimiento gráfico.

Cuando se desea abrir un proyecto existente el sistema le presentará el siguiente mensaje en caso de que exista algún proyecto abierto.



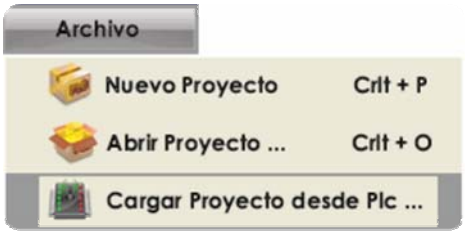
En caso de no existir ningún proyecto abierto observará en el explorador la opción de testear solo archivos '.Pldd', que es la extensión de archivo generado por el sistema.



Una vez seleccionado el proyecto simplemente se carga él y todas las variables globales así como los programas funciones y bloques de este.

Cargar Proyecto desde PLC:

Para este primer beta por falta de la existencia física del hardware sólo se definió el archivo de conexión.



También es disparado éste evento presionado en la barra de herramienta para el reconocimiento gráfico.

Cuando quieran establecer una conexión deben llenar los parámetros del siguiente formulario.

Aquí después de completar los campos se genera el archivo que en versiones futura establecerá la conexión con el PLC.

Guardar Proyecto:

También es disparado éste evento usando la combinación de teclas **Crtl + S,** su icono se refleja en la barra de herramienta para un acceso por reconocimiento gráfico.

Esta opción permite guardar cualquier cambio en el proyecto actual, lo cual repercute en todos su programas activos.

Guardar Proyecto Como:

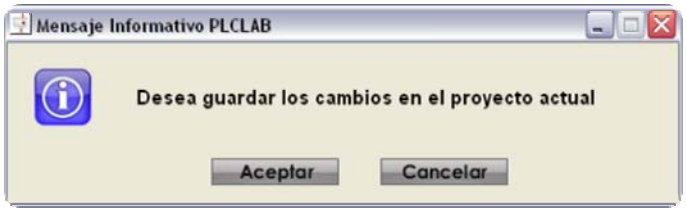
Esta le permite a guardar cualquier proyecto con otro nombre y en otra ruta, teniendo el sistema que generar todos los archivos y funciones necesarios para dicha clonación.

Salir:

Aquí está el método que permite salir del software PLCLAB siempre y cuando se cumplen todas las reglas.

También es disparado éste evento usando la combinación de teclas **Crtl + Q,** su icono se refleja en la barra de herramienta para un acceso por reconocimiento gráfico.

Después de ejecutar la orden de salir, es posible que software le muestre el siguiente mensaje.

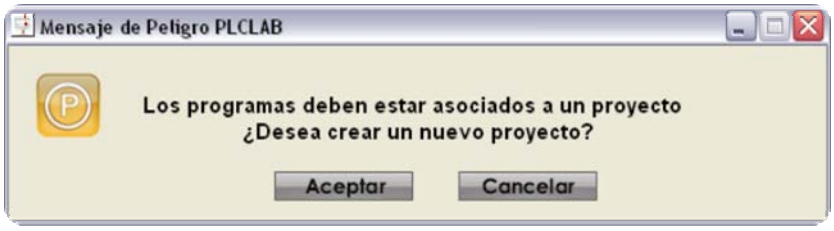


Nuevo Programa:

Para definir un nuevo programa debemos activar su llamado desde la barra de menú en la sección **Programa.**

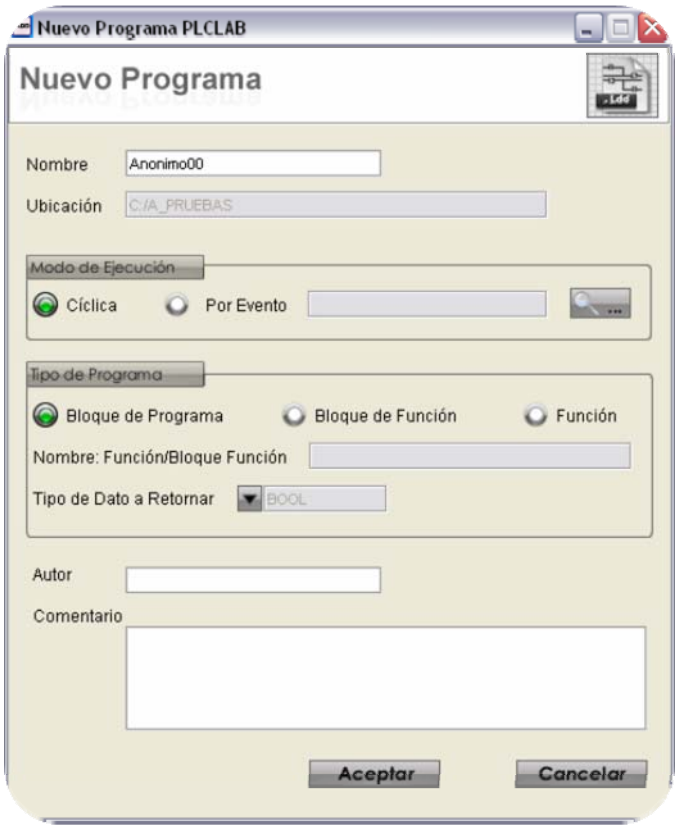
También es disparado éste evento usando la combinación de teclas **Crtl + N,** su icono se refleja en la barra de herramienta para un acceso por reconocimiento gráfico.

Antes de crear cualquier programa es necesario que exista algún proyecto abierto a cual se debe asociar el programa, en caso de existir el VISUAL LADDER emitirá un mensaje como el siguiente.



Una vez abierto o creado nuestro proyecto, sistema nos solicita toda la información necesaria para poder definir el nuevo programa, por defecto el sistema le asigna un nombre anónimo al archivo, el cual se puede modificar, la ubicación del programa es obligatoriamente en el directorio donde se generó el proyecto, en este primer beta

sólo se permitirá definir el modo de ejecución cíclica, también puede colocarle el autor y algún comentario si lo desea al programa.

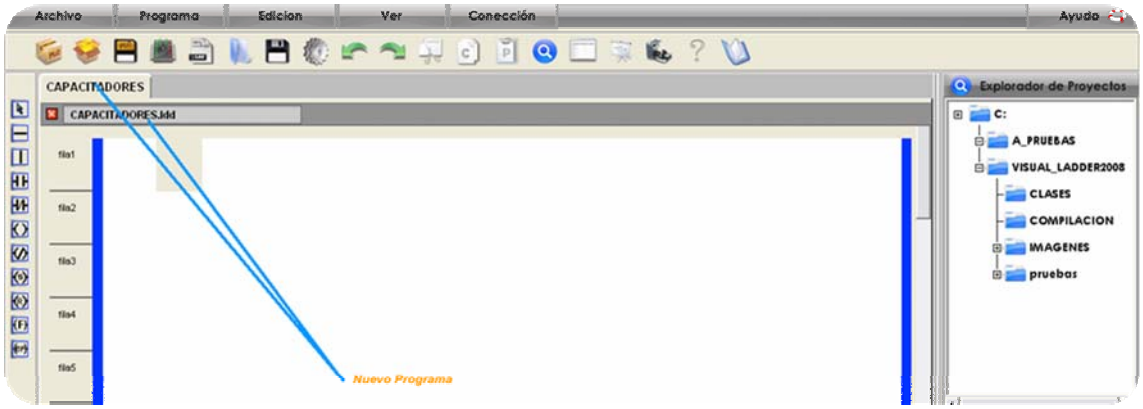


Al momento de crear un nuevo programa son varios los mensajes del software que pueden aparecer si las operaciones no se ejecutan correctamente.

El nombre de un programa no puede exceder de 20 caracteres.

No se permite la creación de programas con el nombre de otros aun si fuese de otro proyecto.

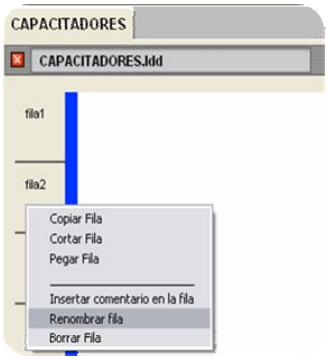
Si no ocurre ningún inconveniente el sistema genera el lienzo del nuevo programa. De forma correcta.



El programa esta divido por las barras de energía y el lienzo escalera y se permite un

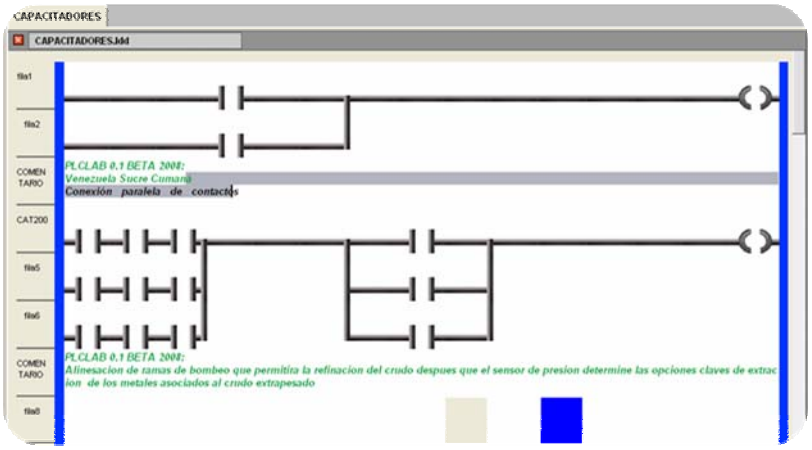
conjunto de operaciones para la ejecución gráfica de la lógica ladder. Entre estas operaciones tenemos:

- Se puede renombrar una fila para facilitar el entendimiento del programa, simplemente presionando clic derecho sobre fila deseada y accediendo a la opción **Renombrar fila** del sub menú desplegado.

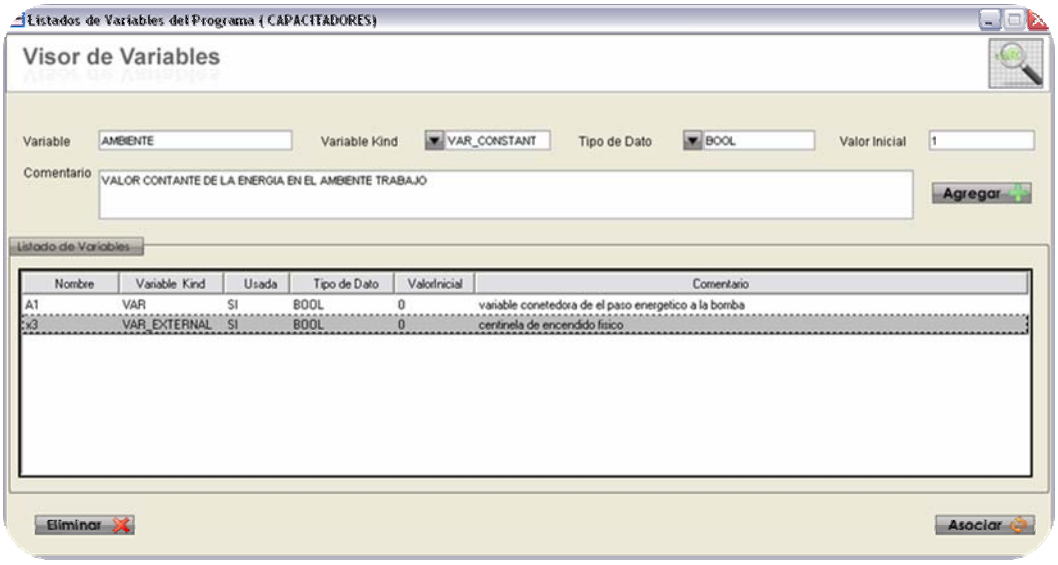


Cuando se desea renombrar la fila se muestra un formulario para escribir el nuevo nombre.

- También son posibles algunas operaciones de edición como **Copiar Fila, Cortar Fila** y **Pegar Fila,** estas opciones son validas siempre y cuando los espacios de conexión estén acordes a las entradas.
- La creación de comentarios como lo dice el estándar es parte fundamental del sistema pues le permitirá tener una lógica en los algoritmos escaleras mucho más fluida.



- Solo faltaría mencionar que se puede esculpir toda la programación gráfica en escalera simplemente presionado sobre los botones de la barra ladder y luego en los distintos espacios del lienzo, teniendo en cuenta que los contactos y las bobinas solo asumen valores booleanos y que la concepción de los otro tipos de datos se manejan en las funciones y bloques de funciones estándares anteriormente definidos.
- La asociación de variables se coordina usando las reglas de expresiones, ya antes expuestas.
- Dentro del lienzo y para cada elemento de lenguaje establecido están definidas las operaciones de **Edición** básicas, cortar, copiar, pegar, deshacer etc.



Al igual que en el proyecto es posible guardar los datos del programa creado también generar un código de intermedio de nuestra lógica ladder a un lenguaje de alta generación.

Hoja de Metadatos

Hoja de Metadatos para Tesis y Trabajos de Ascenso – 1/5

Título	**SOFTWARE MULTIPLATAFORMA DE PROGRAMACIÓN VISUAL BASADO EN EL LENGUAJE ESCALERA (LADDER) CUMPLIENDO CON LA NORMA IEC 61131-3, QUE SERVIRÁ PARA PROGRAMAR AUTÓMATAS, EN EL PROYECTO PLCLAB DE SUCRE ELECTRÓNICA C.A. (Modalidad: Pasantía)**
Subtítulo	

Autor(es)

Apellidos y Nombres	Código CVLAC / e-mail	
VÁSQUEZ M., ANDY D	CVLAC	
	e-mail	
	e-mail	
	CVLAC	
	e-mail	
	e-mail	
	CVLAC	
	e-mail	
	e-mail	
	CVLAC	
	e-mail	
	e-mail	

Palabras o frases claves:

Programación Objeto
Plataforma Linux
Plataforma Windows
PLC
Autómatas
IEC 61131-3
Multiplataforma
Perl
Entorno de desarrollo
Tecnología nacional software libre

Hoja de Metadatos para Tesis y Trabajos de Ascenso – 2/5

Líneas y sublíneas de investigación:

Área	Subárea
Ciencias	**Informática**
	Eléctronica

Resumen (abstract):

Se desarrolló, un software multiplataforma de programación visual basado en el lenguaje escalera (*ladder*) cumpliendo con la norma IEC 61131-3, el mismo, se realizó siguiendo las fases del Proceso Unificado de Desarrollo de *Software* (PUDS). Durante la aplicación metodológica se llevó a cabo todo el levantamiento de información necesario para recolectar, los requisitos del sistema. El PUDS, fue la base metodológica de ingeniería que permitió definir el esquema disciplinado para asignar las tareas y responsabilidades en el proceso de desarrollo. Por medio de la metodología, se trabajó con una serie de ciclos ejecutados repetidamente; cada ciclo o iteración estuvo divido en fases. El contenido de las iteraciones cambiaba para acomodarse a los objetivos de cada fase, adicionalmente se utilizó el lenguaje unificado de modelado (UML), para generar todos los planos arquitectónicos del sistema. En la etapa final, el volumen de la programación fue bajando a medida que se alcanzaban los objetivos y simplemente, se aumentaron las pruebas y la interacción con los técnicos de Sucre Electrónica C.A de forma mucho más permanente y así se cumplieron todas las fases de la metodología. El *software*, fue desarrollado con el lenguaje *Perl* en su versión 5.7.8, probado y ajustado todo su código fuente en plataformas *Windows* y *Linux* de forma estable y en *Solaris* y *Mac* de forma muy esporádica por no ser el objeto a corto plazo del PLCLAB, pero donde igual puede operar. El sistema se ajusta a lo solicitado por la empresa y permite crear la base del proyecto PLCLAB en su primera etapa, donde los analista podrán generar sus programas y proyectos en lenguaje ladder y en un futuro cuando se construya el PLC se podrá integrar el *hardware* con el *software*.

Hoja de Metadatos para Tesis y Trabajos de Ascenso – 3/5

Contribuidores:

Apellidos y Nombres	ROL / Código CVLAC / e-mail	
Martínez, Julio	**ROL**	**CA** [ ] **AS** [ ] **TU** [X] **JU** [ ]
	CVLAC	
	e-mail	
	e-mail	
Garcia, Hayed	**ROL**	**CA** [ ] **AS** [x] **TU** [ ] **JU** [ ]
	CVLAC	
	e-mail	
	e-mail	
Acosta, Leopoldo	**ROL**	**CA** [ ] **AS** [ ] **TU** [ ] **JU** [X]
	CVLAC	
	e-mail	
	e-mail	
Rodríguez, Maricarmen	**ROL**	**CA** [ ] **AS** [ ] **TU** [ ] **JU** [x]
	CVLAC	
	e-mail	
	e-mail	

Fecha de discusión y aprobación:

Año	Mes	Día
2008	11	20

Lenguaje: spa

Hoja de Metadatos para Tesis y Trabajos de Ascenso – 4/5

Archivo(s):

Nombre de archivo	Tipo MIME
Tesis-andyvasquez.doc	Aplication / Word

Alcance:

Espacial: **Venezuela** (Opcional)

Temporal: (Opcional)

Título o Grado asociado con el trabajo:
Licenciatura

Nivel Asociado con el Trabajo: **Licenciatura en Informática**

Área de Estudio:
Informática

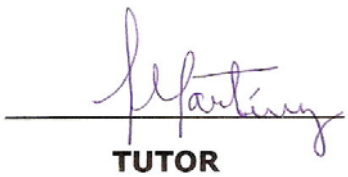
Institución(es) que garantiza(n) el Título o grado:

Universidad de Oriente

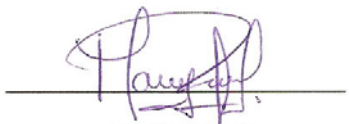
Hoja de Metadatos para Tesis y Trabajos de Ascenso – 5/5

Derechos:
Yo como autor intelectual de esta información le doy a la Universidad de Oriente el derecho de divulgar este conocimiento científico, siempre y cuando se me resguarde el derecho de patente industria y comercio.

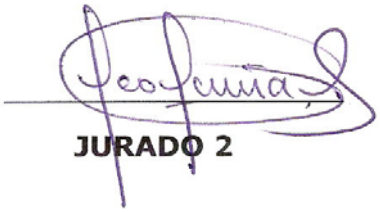
AUTOR



TUTOR



JURADO 1



JURADO 2

POR LA SUBCOMISIÓN DE TESIS: