"APLICACIÓN DE USUARIO PARA CONTROLADOR LÓGICO PROGRAMABLE BASADO EN HARDWARE RECONFIGURABLE"

TRABAJO DE DIPLOMA

PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR:

EDUARDO MILLÁN NÚÑEZ

TUTOR:

ING. MAIKEL RAMÍREZ DESPAINE

MAYO, 2011

AÑO 53 DE LA REVOLUCIÓN

El éxito no se logra sólo con cualidades especiales. Es sobre todo un trabajo de constancia, de método y de organización.

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.
Para que así conste firmo la presente a los _16_ días del mes de _mayo_ del año _2011_.

_____ _____

Firma del Autor **Firma del Tutor**

"…Ningún hombre ha experimentado nunca tantas dificultades para expresarse, como el hombre agradecido…". Jamás imaginé que fuera tan complicado agradecer, no por la simple acción (porque es algo que sale del corazón y da un inmenso placer hacerlo y uno mayor recibirlo), sino por el temor de que en mi calidad de ser humano, cometa el error de olvidar a alguna de las tantas personas que han tenido que ver con este logro.

Gracias infinitamente:

A mi mamá, por su amor, su confianza sin límites, por su entrega y sacrificio incondicional. Gracias por todo, por inspirarme, por apoyarme, por estar a mi lado en cada una de mis metas, por ser mi sol. Te amo mamá...

A mi papá, por ser mi ejemplo, mi guía brillante, mi héroe, mi mejor amigo. No hay palabras que resuman mi admiración, por todo eso y más, Gracias!

A mi niña, por endulzarme la vida, por obligarme a ser cada día mejor, solo para marcarle el camino a seguir.

A mi hermanita, por ser mi otra niña, a ti que me tienes como guía, dedico con mucho amor mis esfuerzos.

A mis abuelos, por todo su amor.

A mi esposa, por su infinita ternura, su paciencia y su manera especial de presenciar todos los buenos y malos momentos, por comprenderme y aceptarme sin reservas.

A mi familia, porque de una forma u otra, siempre están a mi lado, por su cariño incondicional.

A mis suegros, por su cariño, por aceptarme en su familia incondicionalmente, por su ayuda en todos estos años.

A los verdaderos amigos, los que no necesitan nombres, gracias por levantarme en mis tropiezos, por quererme incondicionalmente y permitirme entrar en sus vidas.

A mis compañeros de estos cinco años, a quienes quiero y admiro, gracias por los momentos de alegría, por enseñarme, por compartir esta etapa inolvidable.

A mi tutor, por la ayuda prestada en la culminación de este trabajo.

A todos, gracias.

A mis padres, por su amor y dedicación en todo momento

A mi niña por ser el regalo más grande que la vida me ha dado

A mi hermanita por su amor y ternura

A mi esposa por todo su apoyo

Los constantes cambios en la ciencia y la tecnología demandan métodos y técnicas eficientes que impulsen el desarrollo de estos campos a nivel mundial. La automática no es la excepción y por ese motivo se ve enfrascada en una constante batalla de innovación y desarrollo.

El presente trabajo se basa en el desarrollo de una aplicación de usuario que permita programar un controlador lógico programable (PLC) basado en hardware reconfigurable.

El desarrollo de la aplicación está sustentado por una de las metodologías ágiles descrita por XP, utilizando las técnicas de modelación establecidas por el Lenguaje Unificado de Modelado (UML) y haciendo uso de una potente herramienta de programación y diseño de interfaces gráficas como es Qt. La programación del PLC se realiza mediante el lenguaje de escalera, atendiendo la norma IEC 61131-3.

PALABRAS CLAVE: controlador lógico programable, lenguaje escalera, procesador microblaze, traducir.

Índice

INTRODUCCION.

Los PLCs son importantes dispositivos de control industrial basados en tecnología microprocesador. Características del PLC tales como alta generalidad y flexibilidad, fácil de programar y usar, lo hacen ampliamente utilizado en la industria.

Existen en el mundo importantes fabricantes de PLC: Siemens, Allen Bradley, Omron, entre otros. Todos ellos brindan además, la aplicación que permite programarlo utilizando uno o varios lenguajes de programación de PLC establecidos en la norma IEC 61131-3.

Las técnicas de manufacturas han evolucionado con el tiempo, desde la producción masiva de ayer, a la automatización flexible de hoy y a la automatización reconfigurable de mañana.

El crecimiento de esquemas complejos de control obliga a ingenieros e investigadores a explorar nuevas arquitecturas. El hardware reconfigurable, específicamente los arreglos de compuertas programables por campos (FPGA), han surgido como una alternativa a la demanda de aplicaciones.

En investigaciones realizadas en años anteriores, se desarrolló un controlador lógico programable basado en hardware reconfigurable, con un procesador Softcore embebido, reutilizando módulos IP para hardware reconfigurable, y con las funcionalidades necesarias en el control industrial actual (Ramírez Despaigne, M. 2010), sin embargo se hace necesario desarrollar una aplicación de usuario que permita programarlo mediante un lenguaje de programación de PLC.

Luego de un análisis de la situación actual, descrita anteriormente, se identificó el siguiente ***problema científico***: ¿Cómo lograr mediante una aplicación de usuario programar el controlador lógico programable basado en hardware reconfigurable?

La solución a este problema está enmarcada en el ***objeto de estudio*****:** Las aplicaciones de programación de los controladores lógicos programables mediante lenguajes de la norma IEC 61131-3, centrando su ***campo de acción*** en el desarrollo de una aplicación para programar el controlador lógico programable -PLC.

El ***objetivo general*** de este trabajo es: Desarrollar la aplicación de escritorio que permita programar el controlador lógico programable basado en hardware reconfigurable mediante lenguajes de la norma IEC 61131-3.

A partir del cual se definieron los siguientes ***objetivos específicos*****:**

- ✓ Conceptualizar el controlador lógico programable-reconfigurable.

- ✓ Definir una arquitectura general para la aplicación.
- ✓ Implementar la aplicación para programar al controlador lógico programable basado en hardware reconfigurable.

Para dar respuesta a estos objetivos, se establecieron las siguientes ***tareas de investigación:***

- ✓ Estudio de la arquitectura, hardware y funcionalidades de los controladores lógicos programables actuales.
- ✓ Estudio de las aplicaciones comerciales de programación de PLC.
- ✓ Revisión bibliográfica del estado del arte de los controladores lógicos programables reconfigurables.
- ✓ Estudio de los lenguajes de programación de PLC.
- ✓ Procesamiento y evaluación de la información obtenida.
- ✓ Diseño teórico de la aplicación.
- ✓ Selección del lenguaje de programación de PLC.
- ✓ Selección de las herramientas para el desarrollo de la aplicación.
- ✓ Edición de un programa en lenguaje PLC.
- ✓ Traducción de un programa en lenguaje PLC al firmware del controlador lógico programable basado en hardware reconfigurable.

Surge entonces, como ***idea a defender:*** Una aplicación de escritorio que permita editar un programa en lenguaje de PLC, chequear su sintaxis y traducirlo al firmware del controlador lógico programable basado en hardware reconfigurable puede utilizarse para programar dicho PLC.

Se emplean los siguientes métodos de investigación científica:

Métodos Teóricos:

- ✓ **Análisis histórico-lógico:** Sobre los conocimientos teóricos existentes para la determinación de los momentos más trascendentales en la evolución histórica del desarrollo de los PLC y sus tendencias.
- ✓ **Analítico-Sintético**: Análisis documental de la bibliografía que recoge los elementos del estado del arte de los controladores lógicos programables y el hardware reconfigurable en el control industrial.
- ✓ **Inductivo-Deductivo*:*** A partir de la interpretación de la realidad se establecen posibles situaciones o resultados para llegar a conclusiones.

Métodos Empíricos:

- ✓ **Consulta a especialistas*:*** Se empleó para comprobar la necesidad y la funcionalidad práctica de la aplicación de escritorio que se propone en la presente investigación.

- ✓ **La observación:** Mediante guías de observación se le dará seguimiento al desarrollo de la aplicación.

Los aportes esperados de esta investigación son:

Aporte Teórico*:* Diseño de la aplicación para programar el controlador lógico programable basado en hardware reconfigurable.

Aporte Práctico: Desarrollo de la aplicación para programar el controlador lógico programable basado en hardware reconfigurable.

CAPÍTULO 1: Fundamentación Teórica.

Introducción.

El presente capítulo tiene como objetivo abordar elementos teóricos en lo concerniente al uso de herramientas de programación para la creación de una aplicación capaz de editar un programa en lenguaje de PLC, analizar su sintaxis y traducirlo al firmware del controlador lógico programable basado en hardware reconfigurable. Se abordan características de los principales lenguajes de programación de PLC, así como los principios de funcionamiento de un traductor, mostrando las diferentes fases contenidas dentro del proceso, además de otros aspectos como las herramientas, sistemas, lenguajes y metodologías utilizadas en la solución.

1.1 Lenguaje de programación.

Un **lenguaje de programación** es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado por un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se aprueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación. [1]

1.1.1 Clasificación de los lenguajes de programación para PLC.

Los lenguajes de programación para PLC son de dos tipos, visuales y escritos. Los visuales admiten estructurar el programa por medio de símbolos gráficos, similares a los que se utilizan para describir los sistemas de automatización, planos esquemáticos y diagramas de bloques. Los escritos son listados de sentencias que describen las funciones a ejecutar. [2]

Los programadores de PLC poseen formación en múltiples disciplinas y esto determina que exista diversidad de lenguajes. Los programadores de aplicaciones familiarizados con el área industrial prefieren lenguajes visuales, por su parte quienes tienen formación en electrónica e informática optan, inicialmente por los lenguajes escritos.

- ✓ Lenguajes visuales :

Utilizan los símbolos de planos esquemáticos y diagramas de bloques. El acceso a los recursos se ve restringido a los símbolos que proporciona el lenguaje. En este grupo se destacan los lenguajes LD, SFC y FBD.

- ✓ Lenguajes escritos:

Utilizan sentencias similares a las de programación de computadoras. Brindan acceso total a los recursos de programación. En este grupo se destacan los lenguajes LI y ST.

Todos ellos tienen la finalidad de generar código objeto para que sea ejecutado en la CPU del PLC.

1.2 Características de los lenguajes de programación de PLC.

1.2.1 Listado de Instrucciones (LI).

- ✓ Es un tipo de lenguaje ensamblador con un repertorio muy reducido de instrucciones.
- ✓ Los programas utilizan un estilo muy similar al empleado por los lenguajes de ensamblador.
- ✓ Es una transcripción elemental e inmediata de las instrucciones del lenguaje máquina que están representadas por expresiones nemotécnicas.
- ✓ Se suele aplicar para pequeñas aplicaciones y para optimizar partes de una aplicación.

1.2.2 Texto Estructurado (ST).

- ✓ Facilitan la programación de procesos que requieren instrucciones complejas y cálculos muy grandes.
- ✓ Es un lenguaje de alto nivel.

1.2.3 Esquema Básico de Funciones (FBD).

- ✓ Es un lenguaje gráfico.
- ✓ Los programas son bloques de funciones entre sí de forma análoga al esquema de un circuito.
- ✓ Tiene una interface de E/S bien definida y además posee un código interno oculto.

1.2.4 Esquema de Contactos (LD).

- ✓ La lógica de escalera o ladder es el lenguaje más usado para la programación de PLCs.

- ✓ Fue el primero con el que se comenzó a programar, de ahí que presente grandes semejanzas con los diagramas eléctricos de escalera utilizados por los técnicos anteriormente a la aparición del autómata.
- ✓ Está especialmente indicado para facilitar el cambio de un sistema de control realizado con relés por un PLC.

1.2.5 Esquema Secuencial de Funciones (SFC).

- ✓ En sus orígenes fue GRAFCET (Gráfico Funcional de Control Etapa Transición).
- ✓ Es una técnica eficaz para describir el comportamiento secuencial de un proceso y de un programa.
- ✓ Se usa para distribuir un problema de control.
- ✓ Permite un rápido diagnóstico.
- ✓ Pueden darse esquemas menos lineales.

De los lenguajes antes descritos, LD es específicamente el cual usaremos para editar los programas PLCs de la aplicación en cuestión. Esto se debe en gran medida a que como bien mencionábamos es el lenguaje más usado actualmente en la programación de estos dispositivos, además de que permite una representación general de circuitos de control que facilitan su análisis mediante el uso de contactos N.A y N.C, Temporizadores, Contadores de eventos, Registros de corrimiento y otros elementos de control.

El diagrama de escalera le facilita al programador, entender el funcionamiento del programa, pero no son instrucciones que el PLC directamente ejecute para el caso de algunos equipos comerciales, por lo cual es necesario codificar. El diagrama de escalera se convierte a lista de mnemónicos, en la cual el PLC sí ejecuta en particular modelo y marca; esta tarea de conversión es propia del programador, para lo cual deberá dedicar tiempo para estudiar la parte técnica y características del PLC a usar. [3]

Aunque presente mayor dificultad a los desarrolladores a la hora de representar gráficamente estos diagramas, es mucho más cómoda su traducción a cualquier otro lenguaje, ya que durante el barrido del editor podríamos asignarle la función correspondiente para realizar dicha conversión una vez identificado el componente presente en el programa.

A continuación se describe un pequeño ejemplo de un programa en lenguaje escalera:

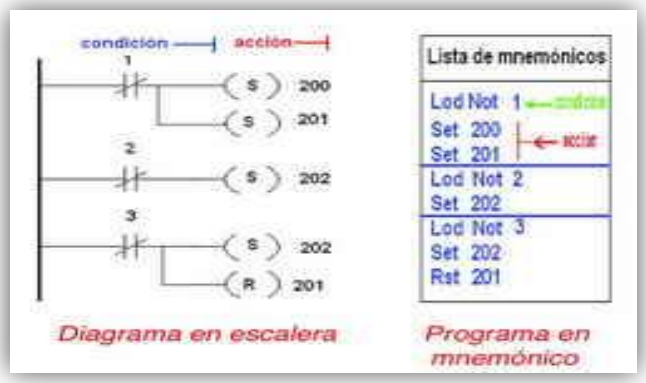

Figura 1.1 Programa en lenguaje escalera.

LOD es un mnemónico o instrucción usado para unir cada bloque o inicio de condiciones, en general conexión e interconexión con otro bloque a diferentes niveles, indicando después de LOD el elemento que lo antecede seguido del que lo sucede:

(LD) ½ Condición (s) ½ Acción (s)

Un diagrama de escalera tiene su equivalente en lista de mnemónicos. En la Figura 1.1 vemos un equivalente para estas dos formas de expresar un programa. En esta figura se especifica, mediante dos lenguajes o formas diferentes, un programa que, al ser grabado en un PLC hará que cuando esté abierto el interruptor (1) (entrada 1) se activen las salidas (200) y (201); el mismo programa establece otra condición que especifica que cuando esté desconectada la entrada (2) se activará la salida (202) y una tercer condición que especifica que si la entrada (3) está en estado bajo, entonces se activarán las salidas (202) y (201).

Es claro que el PLC sólo puede procesar o ejecutar la lista de mnemónico, si nuestro circuito solución es un diagrama de componentes lógico (parte izquierda de la Figura 1.1), podemos convertirlo a lista de mnemónicos (parte derecha de la Figura 1.1) y cargarla al PLC de manera local o remota.

LOD es una función muy importante en los diagramas de escalera, esta permite cargar “alambrar” elementos de control o bloques.

Por lo que se observa que la función *LOD* es equivalente a un cable o alambre que sirve para conectar elementos de control o bloques. Siempre se debe alambrar de izquierda a derecha (entradas y salidas) y además de arriba para abajo, “el orden si importa”.

1.3 Proceso de compilación.

Entre el lenguaje natural de los seres humanos y las computadoras existen muchísimas diferencias, debido a que los seres humanos utilizan lenguajes basados en un conjunto de símbolos muy diversos y complejos en el cual intervienen disímiles componentes de la lengua (emisor-receptor-mensaje-codificador-descodificador-ruidos-canales, y otros que influyen en la comunicación: su cantidad y su calidad). Las computadoras aunque utilizan la mayoría de estos componentes del lenguaje, solamente en esencia son un medio tecnológico que utilizan el lenguaje binario: ceros y unos, dejando fuera componentes de carácter subjetivo que tienen que ver con el hombre: sus intereses, motivaciones, necesidades, sentimientos que influyen en la comunicación y en la gama de su interpretación de los objetos y fenómenos.

Pero ¿cómo es posible entonces que se comuniquen las computadoras y los hombres?

Cada acción que realiza el hombre sobre una computadora se traduce a una secuencia de comandos, que son ejecutados por programas para dar la respuesta solicitada por el usuario. A los programas que realizan la traducción del lenguaje humano al lenguaje binario se les llama traductores, Figura 1.2, y más específicamente compiladores.

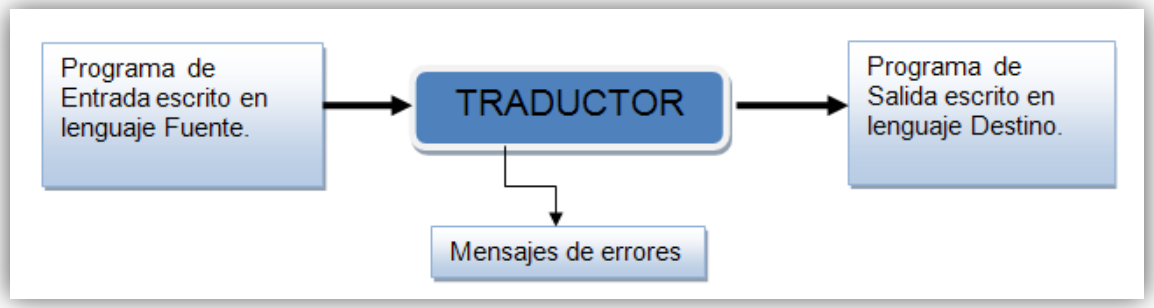

Figura 1.2 Esquema de un Traductor.

El término de traductor engloba tanto a los compiladores como a los intérpretes, Figura 1.3.

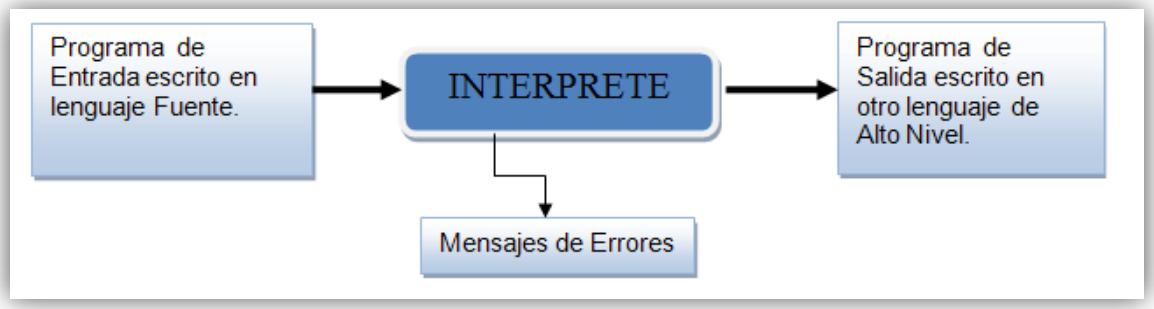

Figura 1.3 Esquema de un Intérprete.

En la década de los 50, se consideraba a los traductores como programas muy difíciles de escribir. Un ejemplo de la anterior afirmación se refleja en la creación del primer compilador, Figura 1.4, de Fortran (Formula Translator), que necesitó para su implementación el equivalente a 18 años de trabajo individual lo cual realmente no se tardó tanto puesto que el trabajo se desarrolló en equipo.

Objetivamente la aplicación de la teoría de autómatas y lenguajes formales son significativos avances al desarrollo de traductores, erradicando problemas existentes antes de su aplicación. Sin embargo, hoy día un compilador básico puede ser el proyecto fin de carrera de cualquier estudiante universitario de Informática.

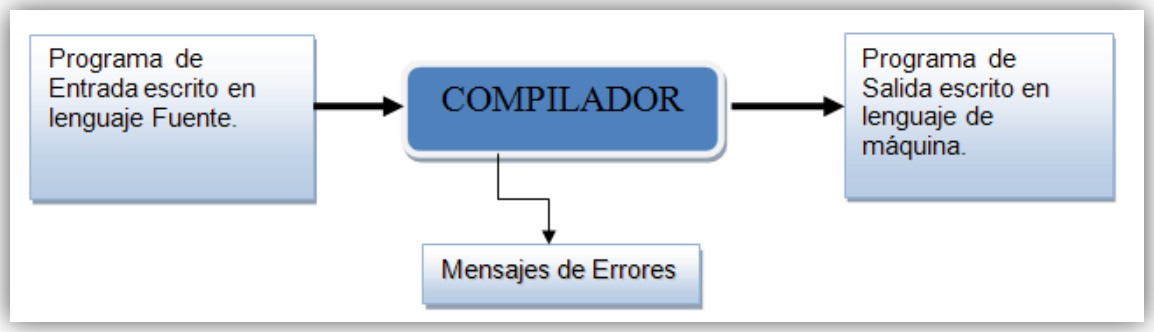

Figura 1.4 Esquema de un Compilador.

1.4 Fases del proceso de compilación.

El proceso de compilación, Figura 1.5, se divide en seis fases fundamentales.

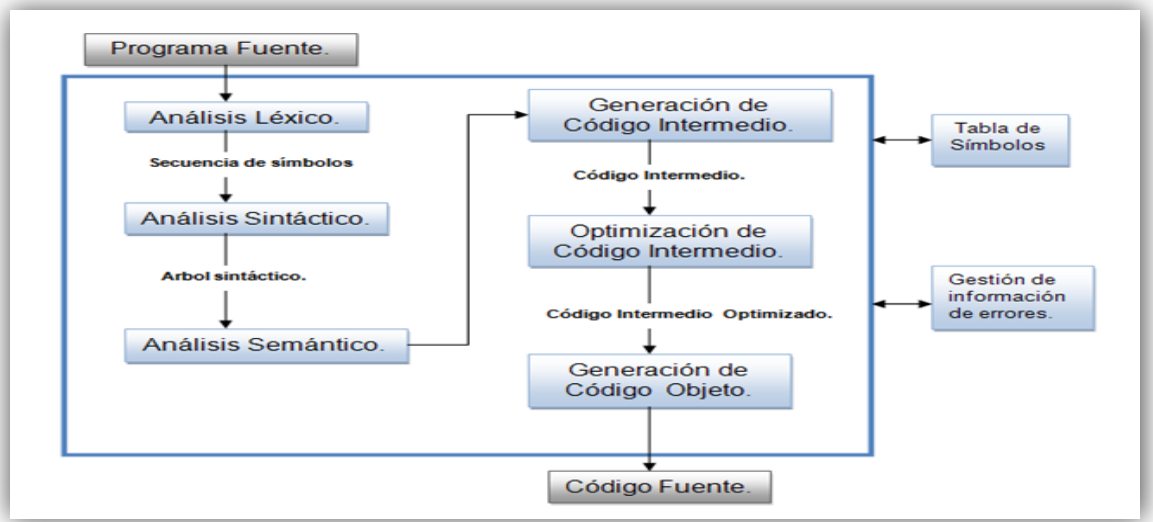

Figura 1.5 Fases del proceso de compilación.

1.4.1 Análisis lexicológico.

La entrada de un compilador es el código de un programa escrito en un lenguaje de programación. Dicho código no es más que una secuencia de símbolos pertenecientes al alfabeto de un determinado lenguaje. El analizador lexicológico o scanner se encarga de tomarlos y agruparlos en entidades sintácticas simples o elementales denominadas tokens o lexemas. [4]

Los tokens pueden ser de diferentes categorías pero las más elementales son:

- ✓ Palabras reservadas.
- ✓ Identificadores.
- ✓ Constantes numéricas y literales.
- ✓ Operadores .

Se considera una buena práctica a cada token asignarle una estructura lexicológica, consistente en un par de la forma <tipo del token, info>, donde “tipo del token” almacena la categoría lexicológica del token, e “info” el valor del token en particular (ejemplo: el valor de la constante, nombre del identificador, etc.). Por tanto, un scanner no es más que un traductor cuya entrada es una cadena de símbolos escritas en lenguaje natural (programa fuente) y cuya salida es una secuencia de estructuras lexicológicas o tokens.

1.4.2 Análisis sintáctico.

El análisis sintáctico es un proceso en el cual se examina la secuencia de tokens para determinar si el orden de la secuencia es correcto de acuerdo a ciertas convenciones estructurales (reglas) de la definición sintáctica del lenguaje. La entrada del analizador sintáctico o parser es la secuencia de tokens generada por el scanner. El parser analiza solamente la primera componente de cada tokens; la segunda componente se utiliza en otros pasos. [4]

1.4.3 Análisis semántico.

En la fase de Análisis Semántico el compilador adiciona información al Árbol de Sintaxis Abstracta generado en la fase de Análisis Sintáctico. Esta operación está relacionada con la segunda componente de la tupla devuelta en el análisis léxico (<tipo del token, info>). Al comprobar la validez semántica de un programa se pudiera decir que se realiza un reconocimiento sintáctico-semántico, pero los errores semánticos no pueden ser detectados por el analizador sintáctico, puesto que se relacionan con

interdependencias entre las diferentes partes de un programa que no son reflejadas en un análisis gramatical. El analizador semántico revisa el programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. En ella se utiliza la estructura jerárquica determinada por la fase de análisis sintáctico para identificador los operadores y operandos de expresiones y proposiciones. Un componente importante del análisis semántico es la verificación de tipos. Aquí, el compilador verifica si cada operador tiene operandos permitidos por la especificación del lenguaje fuente. Por ejemplo, las definiciones de muchos lenguajes de programación requieren que el compilador indique un error cada vez que se use un número real como índice de una matriz. Sin embargo, la especificación del lenguaje puede imponer restricciones a los operandos, por ejemplo, cuando un operador aritmético binario se aplica a un número entero y una cadena de caracteres.

1.4.4 Generación de código intermedio.

Después del análisis sintáctico un compilador puede generar una o varias representaciones explícitas intermedias del código fuente. Dicha representación puede servir para la realización de un análisis semántico del código fuente o para la optimización del código. Ejemplo de estos formas intermedias son los códigos: MSIL, Java bytecode, Notación de Cuádruplos, Notación Polaca. Las formas intermedias deben tener dos características muy importantes: debe ser fácil de producir y fácil de traducir al programa objeto. Las mismas se clasifican en:

Formas Intermedias de Alto Nivel: son aquellas que se suelen emplear en las primeras fases de análisis.

Formas Intermedias de Nivel Medio: son válidas para representar un conjunto amplio de lenguajes fuente, no siendo dependientes de uno en concreto. Válidas para representar un conjunto extenso de arquitecturas de hardware.

Formas Intermedias de Bajo Nivel: permiten traducir a distintos micros de una misma arquitectura, creando una dependencia a ésta.

Formas Intermedias Multinivel: Aquéllas que conjugan varias de las características anteriores.

1.4.5 Optimización del código intermedio.

La fase de optimización se encarga de transformar el código intermedio en un nuevo código de función equivalente pero de menor tamaño o de menor tiempo de ejecución.
Algunas de las transformaciones que puede llevar a cabo la fase de optimización son:

- ✓ Eliminar el cálculo de expresiones cuyo valor no se usa.
- ✓ Fundir en uno, el cálculo repetido de la misma expresión.
- ✓ Sacar de los lazos, las expresiones cuyo valor no cambia en ellos.
- ✓ Reducir el uso de memoria local reutilizando el espacio de una variable muerta.

1.4.6 Generación del código objeto.

La fase de generación de código objeto se encarga de generar el programa nativo usando el juego de instrucciones específico de la máquina o CPU objeto, y el formato para archivos ejecutables del sistema operativo. Entre otras cosas, también se le asignan direcciones definitivas a las rutinas y variables que componen el programa.

1.4.7 Gestión de información de errores.

Si los compiladores tuvieran que procesar solamente programas correctos, su diseño e implementación se simplificaría en buena medida. Pero los programadores escriben programas incorrectos frecuentemente, y un buen compilador debe ayudar al programador a localizar e identificar los errores. [4]
Los errores en un programa pueden clasificarse en 4 grandes grupos:

- ✓ Lexicológicos.
- ✓ Sintácticos.
- ✓ Semánticos.
- ✓ Lógicos o de programación.

Un traductor eficiente debe ser capaz de detectar los errores y tratarlos de manera que pueda continuar, permitiendo así que se puedan detectar errores posteriores. Un traductor que se detenga ante el primer error que se encuentre no es muy eficaz.

El tratamiento de los errores en cualquiera de las fases debe cumplir con los siguientes requisitos:

- ✓ Reportar la presencia de los errores de forma clara y precisa.
- ✓ Recuperarse de los errores rápido y ser capaz de continuar parar detectar los errores siguientes.
- ✓ No demorar significativamente el procesamiento de los programas correctos.

1.5 Diferentes tipos de aplicaciones para programar PLCs.

En la actualidad cada fabricante de PLC brinda junto con el hardware que pone a la venta una serie de software que son usados para la configuración del mismo, entre los cuales se encuentran las herramientas necesarias para programar las rutinas que se desean ejecutar con este dispositivo. En muchos de los casos los PLC traen en su propio mecanismo una consola en la cual se realizan estas tareas y se descargan de forma automática en la memoria del equipo, por otra parte, existen aplicaciones que se encargan de editar, compilar y traducir a lenguaje máquina las instrucciones propiciadas por los programadores, para luego ser descargadas estas rutinas en el PLC. Entre algunas de estas aplicaciones podemos mencionar:

- ✓ SimuPLC 3.1.0*.*

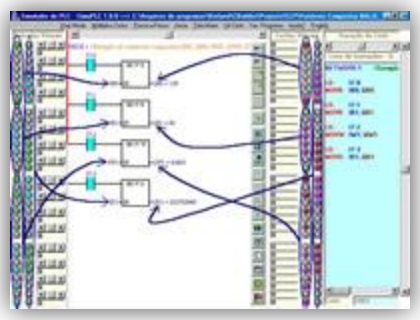

Figura 1.6 SimuPLC 3.1.0.

Simulador de autómatas programables. Permite la programación en lista de instrucciones y en lenguaje de contactos. Dispone de ejemplos y plantas para la simulación de procesos de forma gráfica.

- ✓ LAV 1.9.

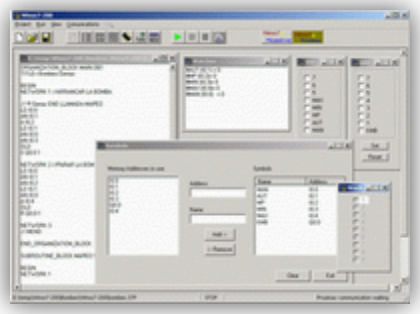

Figura 1.7 LAV-1.9.

Software de simulación para los autómatas S7-200 y S5 de Siemens.

- ✓ PC_Simu.

Figura 1.8 PC_Simu.

Permite simular procesos automáticos de forma gráfica intercambiando las entradas salidas, evitando de esta forma el tener que activar los interruptores de entrada o visualizando los led de salida del PLC.

- ✓ ZelioSoft 2.4.1.

Figura 1.9 ZelioSoft-2.4.1.

Software de programación para los autómatas Zelio de Schneider Electric. Permite simular el funcionamiento de los programas sin necesidad de disponer del PLC. La Programación se puede hacer en modo de contactos (LD) o funciones lógicas (FBD). Además permite introducir los programas dibujándolos, mediante el editor, o a través de un interfaz que simula el aspecto físico de la controladora Zelio elegida.

- ✓ RSLogix 500

Figura 1.10 RSLogix-500.

Es la herramienta ideal para aprender los fundamentos de la programación lógica de la escala de RSLogix. Incluye un editor de ladder y verificador de proyectos (creación de una lista de errores) entre otras opciones.

1.6 Metodologías y herramientas de desarrollo.

1.6.1 El Lenguaje Unificado de Modelado (UML) como soporte a la programación orientada a objetos.

"El UML se ha vuelto el estándar de facto (impuesto por la industria y los usuarios) para el modelado de aplicaciones de software. En los últimos años, su popularidad trascendió al desarrollo de software y, en la actualidad, el UML es utilizado para modelar muchos otros dominios, como por ejemplo el modelado de procesos de negocios". [5]

Lenguaje: Implica que este cuenta con una sintaxis y una semántica. Por lo tanto, al modelar un concepto en UML, existen reglas sobre cómo deben agruparse los elementos del lenguaje y el significado de esta agrupación.

Modelado: El UML es visual. Mediante su sintaxis se modelan distintos aspectos del mundo real, que permiten una mejor interpretación y entendimiento de éste.

Unificado: Unifica varias técnicas de modelado en una única. El UML proviene de técnicas orientadas a objetos, se crea con la fuerte intención de que este permita un correcto modelado orientado a objetos. Es un lenguaje que no describe métodos o procesos, sino que los especifica. [5]

En la actualidad UML es el más conocido y utilizado como lenguaje de modelado de sistema de software, utilizado para modelar muchos otros dominios, como por ejemplo el modelado de procesos de negocios, así como para documentar, construir y detallar artefactos en el sistema, o sea, en el que está descrito el modelo.

UML ofrece un estándar para describir un "plano" del sistema (modelo), incluye aspectos conceptuales (procesos de negocios y funciones del sistema) y aspectos concretos (expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables).

La ventaja principal que presenta UML es la combinación de diferentes notaciones, como se muestra en la Figura 1.11.

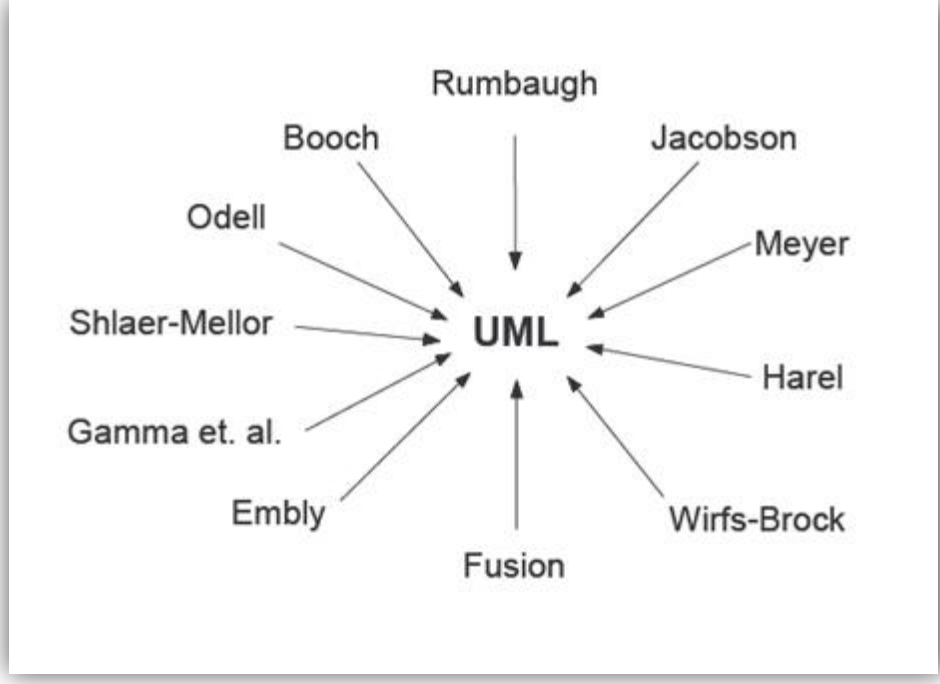

Figura 1.11 Integración de notación en UML.

Características del UML

- ✓ UML es un lenguaje de modelado de propósito general que pueden usar todos los modeladores. Está basado en el común acuerdo de gran parte de la comunidad informática.
- ✓ UML no pretende ser un método de desarrollo completo. No incluye un proceso de desarrollo paso a paso. UML incluye todos los conceptos que se consideran necesarios para utilizar un proceso moderno iterativo, basado en construir una sólida arquitectura para resolver requisitos dirigidos por casos de uso.

- ✓ Debe ser tan simple como sea posible pero manteniendo la capacidad de modelar toda la gama de sistemas que se necesita construir. UML necesita ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la concurrencia y distribución, así como también los mecanismos de la ingeniería de software, como son la encapsulación y componentes.
- ✓ Debe ser un lenguaje universal, como cualquier lenguaje de propósito general.
- ✓ Pretende imponer un estándar mundial.
- ✓ Es orientado a objetos, permitiéndole al programador que organice su programa de acuerdo con abstracciones de más alto nivel, siendo estas más cercanas a la forma de pensar de las personas [5].

Los principales beneficios de UML son:

- ✓ Mejores tiempos totales de desarrollo (de 50 % o más).
- ✓ Modelar sistemas (y no sólo de software) utilizando conceptos orientados a objetos.
- ✓ Establecer conceptos y artefactos ejecutables.
- ✓ Encaminar el desarrollo del escalamiento en sistemas complejos de misión crítica.
- ✓ Crear un lenguaje de modelado utilizado tanto por humanos como por máquinas.
- ✓ Mejor soporte a la planeación y al control de proyectos.
- ✓ Alta reutilización y minimización de costos [6].

1.6.2 ¿Por qué C++ como lenguaje de programación?

El lenguaje C++ como lenguaje de programación orientado a objetos en sí tiene múltiples ventajas en las que se encuentran:

- ✓ Eficiencia.
- ✓ Uniformidad.
- ✓ Comprensión: Los datos componen los objetos y los procedimientos que los manipulan están agrupados en clases que se corresponden con las estructuras de información que el programa trata.
- ✓ Flexibilidad: Al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan.

- ✓ Estabilidad: Dado que permite un tratamiento diferenciado de aquellos objetos que permanecen constantes en el tiempo sobre aquellos que cambian con frecuencia permite aislar las partes del programa que permanecen inalterables en el tiempo.
- ✓ Reusabilidad: La noción de objeto permite que programas que traten las mismas estructuras de información reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de objetos ya definidos donde estos están relacionados de una manera particular.
- ✓ Otra de las razones por las cuales se utilizó C++ como lenguaje de programación es debido a su increíble versatilidad. Con él pueden programarse desde los programas más simples hasta los programas más complicados como son los sistemas operativos.

Es además portable, es decir, un programa con el código escrito en C++, se podrá compilar en cualquier sistema operativo o sistema informático sin necesidad de cambiar casi el código fuente. Este es por ejemplo uno de los grandes secretos de Linux, al estar el código escrito en este lenguaje(al menos en su concepción original), es más fácil portarlo a diferentes ordenadores como PC´s, Macintosh, incluso superordenadores. Otras de las grandes ventajas del C++, es que es un lenguaje multi-nivel, es decir, puedes usarlo tanto para programar directamente el hardware (dependiendo del sistema operativo, eso sí), como para crear aplicaciones tipo Windows definidas todas por poseer una misma interfaz. [7]

1.6.3 Qt v4.6.1.

Qt es una amplia plataforma de desarrollo que dispone de tres grandes ventajas ante las bibliotecas rivales.

- ✓ Es completamente gratuito para aplicaciones de código abierto.
- ✓ Las herramientas, bibliotecas y clases están disponibles para casi todas las plataformas Unix y sus derivados (como Linux, MacOS X, Solaris, etc) como también para la familia Windows, por lo que una aplicación puede ser compilada y utilizada en cualquier plataforma sin necesidad de cambiar el código y la aplicación se verá y actuará mejor que una aplicación nativa.
- ✓ Qt tiene extensas bibliotecas con clases y herramientas para la creación de ricas aplicaciones. Estas bibliotecas y clases están bien documentadas, son muy fáciles de usar y tienen una gran herencia de programación orientada a objetos lo cual hace de la programación de interfaces gráficas una aventura placentera. [8]

1.6.4 Visual Paradigm.

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Además proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Esta herramienta tiene las siguientes características:

- ✓ Soporte de UML versión 2.1.
- ✓ Modelado colaborativo con CVS y Subversion (nueva característica).
- ✓ Interoperabilidad con modelos UML 2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI (nueva característica).
- ✓ Ingeniería inversa - Código a modelo, código a diagrama.
- ✓ Ingeniería inversa Java, C++, Esquemas XML, XML, NET exe/dll, CORBA IDL.
- ✓ Generación de código - Modelo a código, diagrama a código.
- ✓ Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- ✓ Diagramas EJB - Visualización de sistemas EJB.
- ✓ Generación de código y despliegue de EJB´s - Generación de beans para el desarrollo y despliegue de aplicaciones.
- ✓ Diagramas de flujo de datos.
- ✓ Soporte ORM - Generación de objetos Java desde la base de datos.
- ✓ Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- ✓ Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.

- ✓ Generador de informes para generación de documentación, Distribución automática de diagramas, Reorganización de las figuras y conectores de los diagramas UML.
- ✓ Importación y exportación de ficheros XML.
- ✓ Integración con Visio - Dibujo de diagramas UML con plantillas (stencils) de MS Visio (14).

Visual Paradigm es la herramienta case que nos permite llevar una buena organización y planificación de los diagramas y modelados de la aplicación a desarrollar.

1.6.5 Programación Extrema (XP).

Es el más destacado de los procesos ágiles de desarrollo de software. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad; considerándose que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Siendo capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto, es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos. [9]

Las características fundamentales del método son:

- ✓ Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- ✓ Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión.
- ✓ Programación en parejas.
- ✓ Frecuente integración del equipo de programación con el cliente o usuario.
- ✓ Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- ✓ Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento.
- ✓ Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto.
- ✓ Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo.

Conclusiones parciales.

En este capítulo se han abordado elementos teóricos concernientes al uso de herramientas de programación para la creación de una aplicación capaz de editar un programa en lenguaje PLC, analizar su sintaxis y traducirlo al firmware del controlador lógico programable basado en hardware reconfigurable. De igual forma se trataron características de los principales lenguajes de programación para estos dispositivos, así como los principios de funcionamiento de un traductor, mostrando las diferentes fases contenidas dentro del proceso, además de otros aspectos como las herramientas, sistemas, lenguajes y metodologías utilizadas para el desarrollo de la aplicación.

CAPÍTULO 2: Características del Sistema. Exploración y Planificación

Introducción.

El presente capítulo tiene como objetivo hacer una valoración de las principales características de la aplicación a desarrollar. Se hace mención a las fases de exploración y planificación, propias de la metodología de desarrollo utilizada para la implementación de la aplicación que se propone y se exponen los artefactos generados durante el transcurso de las mismas.

2.1 Propuesta del sistema.

Para cumplimentar los objetivos propuestos al inicio de este trabajo, y teniendo en cuenta todos los requerimientos planteados, el sistema que se propone debe tener un único módulo: el traductor. Se considera la existencia de un rol, o sea, el del usuario que interactuará con el software.

El módulo tiene como objetivo resolver el código del usuario y brindarle los resultados que se obtendrían mediante una herramienta de traducción del mismo tipo.

En resumen, el sistema brindará facilidades para implementar diagramas de tipo escalera, permitiendo la ejecución de los mismos y realizando un chequeo de la sintaxis, con el fin de generar una lista de errores en caso de que existan, ayudando al usuario en su identificación y corrección. El código generado a partir de la generación del código intermedio será probado en el PLC en desarrollo, para constatar que la aplicación realiza las operaciones para la cual fue creada.

2.2 Modelo de dominio.

Un Modelo de Dominio, es una representación visual estática del entorno real del proyecto. Puede utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis, como paso previo al diseño de un sistema, ya sea de software o de otro tipo. El objetivo del modelado del dominio es comprender y describir las clases más importantes dentro del contexto del sistema, y ayudar a los usuarios, clientes, desarrolladores y otros interesados a utilizar un vocabulario común [6].

El objetivo que perseguimos con la utilización de este modelo es capturar los elementos necesarios para comprender como funcionará el sistema que estamos diseñando, debido a que el mismo ocupa un rol protagónico en el desarrollo moderno de software.

Para esto se identifica los conceptos presentes en el modelo de dominio mediante un glosario de términos:

DEFINICIÓN 2.2.1

Se considera un Diagrama a una secuencia de símbolos de cierto alfabeto colocados uno a continuación del otro.

DEFINICIÓN 2.2.2

Un Usuario es la persona que interactúa con la aplicación.

DEFINICIÓN 2.2.3

El Código Intermedio es una interpretación que genera un traductor o compilador, de las instrucciones entradas, que sirve para ser trasladado o interpretado por otros sistemas.

DEFINICIÓN 2.2.4

Una Lista de Variables es el conjunto de varias variables declaradas y en uso del código fuente escrito por el usuario.

DEFINICIÓN 2.2.5

Una Lista de Componentes es el conjunto de varios componentes declarados y en uso, del código fuente escrito por el usuario.

DEFINICIÓN 2.2.6

Se considera un Componente a todo aquel elemento que sea capaz de guardar una información para ser utilizada o modificada en alguna instrucción, o que a su vez sea una instrucción en si mismo.

DEFINICIÓN 2.2.7

Se considera un Intérprete al componente del sistema o sistema externo capaz de reconocer y ejecutar el código intermedio generado por una aplicación.

DEFINICIÓN 2.2.8

Se considera como Resultado al conjunto de valores provenientes de la ejecución de un programa.

DEFINICIÓN 2.2.9

Se considera un Lenguaje al conjunto de cadenas (componentes) sobre un alfabeto, que obedecen reglas gramaticales.

El modelo del dominio que se describe en la Figura 2.1, utilizando el Lenguaje Unificado de Modelado (UML). Representa un concepto significativo para el dominio del problema mediante el uso de clases conceptuales.

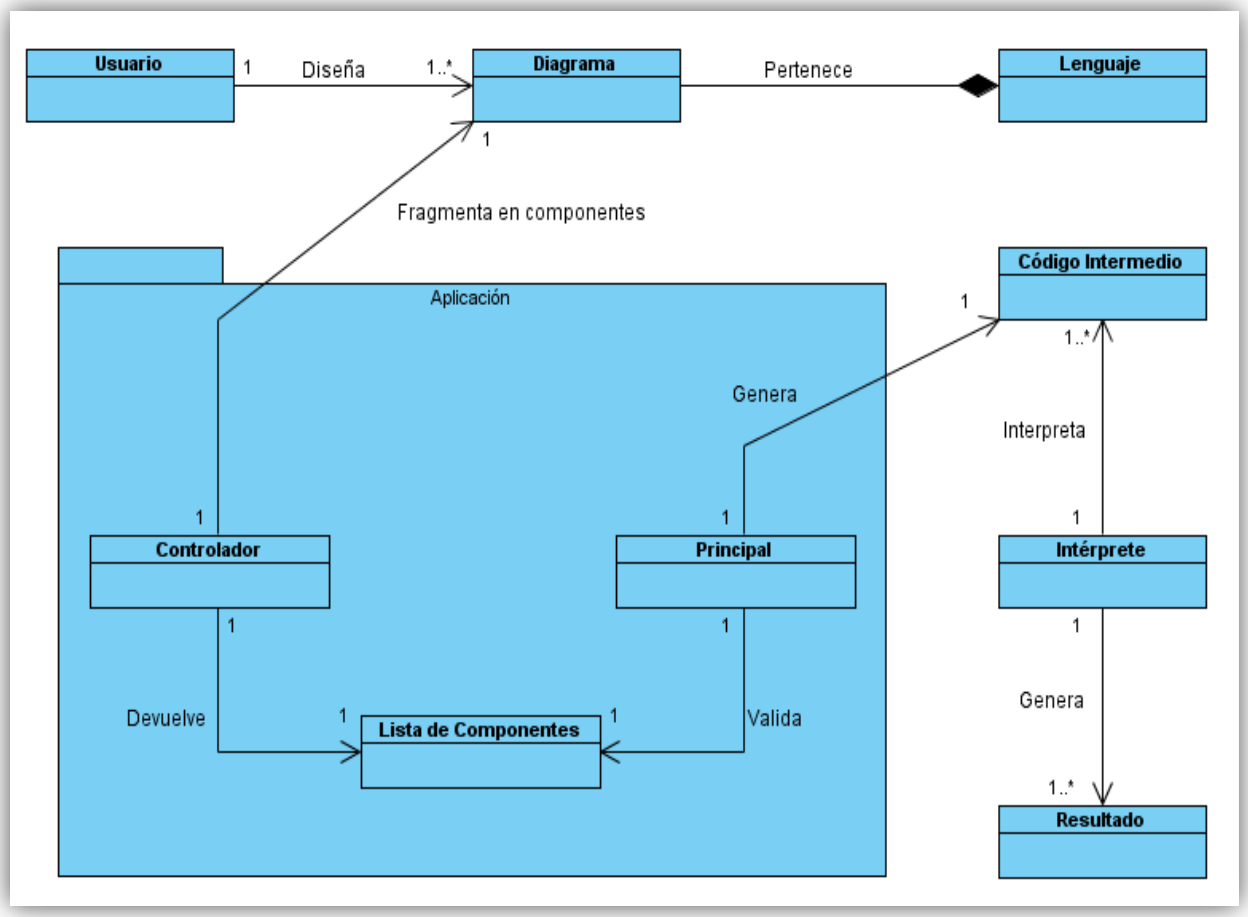

Figura 2.1 Modelo de dominio.

2.3 Requerimientos del sistema.

Un requerimiento: es una condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema, para satisfacer un contrato, estándar, u otro documento impuesto formalmente [10].

2.3.1 Requerimientos funcionales.

Un requerimiento funcional define el comportamiento interno del software: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran cómo los historias de usuario serán llevados a la práctica. Los requerimientos funcionales se mantienen invariables sin importar con qué cualidades o propiedades se relacionen.

A continuación se muestran los requerimientos funcionales:

- ✓ R.1 El sistema debe brindar un mecanismo de almacenamiento para los valores de los componentes del programa en desarrollo.
- ✓ R.2 El sistema debe ser capaz de guardar ficheros con los datos del programa en desarrollo.
- ✓ R.3 El sistema debe ser capaz de cargar ficheros con datos de programas que hayan sido guardados con anterioridad.
- ✓ R.4 El sistema debe ser capaz de informar los diferentes errores encontrados durante el chequeo del programa en desarrollo.
- ✓ R.5 El sistema debe ser capaz de reconocer e interpretar las diferentes instrucciones creadas por el usuario.
- ✓ R.6 El sistema debe ser capaz de exportar y traducir los datos del programa desarrollado a funciones en lenguaje C.
- ✓ R.7 El sistema debe brindar la posibilidad de crear nuevos programas.
- ✓ R.8 El sistema debe brindar la posibilidad de introducir, cambiar y consultar los datos del programa en desarrollo en cualquier instante de tiempo.

2.3.2 Requerimientos no funcionales.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. En muchos casos, estos requerimientos son fundamentales en el éxito del producto y generalmente están vinculados a requerimientos funcionales. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable.

Requerimientos de Usabilidad.

- ✓ El sistema deberá tener una interfaz amigable con una buena utilización de los elementos de diseño, con adecuada combinación de colores.
- ✓ El sistema deberá ser usado por usuarios capacitados para el uso de la herramienta y que hayan leído previamente el manual de ayuda.

Requerimientos de Soporte

- ✓ El sistema debe ser capaz de dar las mismas salidas para diferentes ambientes.
- ✓ El sistema debe ser flexible ante la necesidad de cambios de sus salidas.
- ✓ El sistema debe tener alguna documentación o ayuda.
- ✓ Los errores del sistema no pueden afectar a los sistemas clientes.

Requerimientos de portabilidad y operatividad.

- ✓ El sistema debe ser compatible con las plataformas Windows y Linux.

Requerimientos Ambientales.

Requerimientos de Hardware del Sistema.

- ✓ Las computadoras que utilizarán el software a desarrollar deberán tener 128 MB de Memoria tipo RAM como mínimo.

Requerimientos de Software del Sistema.

- ✓ Las computadoras que utilizarán el software deben tener instalado: Windows XP Professional, Windows 7 ó alguna distribución GNU/Linux.
- ✓ Qt 4.7.0 o superior.

2.4 Fase de exploración. Definición.

El ciclo de vida de XP enfatiza en el carácter interactivo e incremental del desarrollo. Una iteración de desarrollo es un período de tiempo en el que se realiza un conjunto de funcionalidades determinadas, que en el caso de XP corresponden a un conjunto de historias de usuarios [10]

La metodología de desarrollo XP comienza con la fase de exploración, en esta fase los clientes plantean a grandes rasgos las historias de usuario mediante un proceso de identificación que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

2.4.1 Actores del sistema.

Tabla 1 Actores del sistema.

Actor	Descripción
Usuario	Representa a una persona capaz de desarrollar un programa en lenguaje escalera, accediendo a todas las funcionalidades que brinde la aplicación.

2.4.2 Definición de historias de usuario.

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.

El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Las HU conducen al proceso de creación de los test de aceptación, los cuales servirán para verificar que estas historias se han implementado correctamente. Otra de sus características es que solamente proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo conllevará su implementación.

Tabla 2 Plantilla Historia de Usuario.

Historia de Usuario	
No.: Número sucesivo a partir de 1.	**Nombre:** Identifica la historia de usuario en cuestión.
Usuario: Quien ejecuta la historia de usuario.	
Prioridad en el Negocio: Define la relevancia e impacto de la historia de usuario para el negocio de acuerdo con las necesidades del usuario.	**Nivel de Complejidad:** Define la dificultad técnica que supone desarrollar la historia de usuario desde el punto de vista del programador.
Puntos de Estimación: Permiten estimar duración de implementación.	**Iteración Asignada:** Precisa la iteración a la que pertenece la historia de usuario.
Descripción: Explica en qué consiste la historia de usuario, teniendo en cuenta las acciones realizadas por el usuario y la respuesta brindada por el sistema.	
Información adicional (Observaciones): Información extra que se estime agregar para hacer más comprensible la historia de usuario. Por ejemplo: conceptos, post-condiciones, relación con otros requisitos, etc.	

2.4.3 Descripción de las historias de usuario.

Tabla 3 HU Guardar programa.

Historia de Usuario	
Número: 1	**Nombre de la Historia de Usuario:** Guardar programa
Usuario: Desarrollador	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Puntos de Estimación: 1	**Iteración Asignada:** 2

Descripción: Permite crear un fichero XML que contiene la información del programa que se encuentra en desarrollo (imágenes y datos modificados por el usuario), el fichero es generado mediante la biblioteca QXMLStreamWriter().
Observaciones: La creación del fichero de tipo XML le permitirá al usuario contar con un respaldo en caso de cualquier situación extrema, además de acceder a los datos registrados de manera organizada en la estructura definida dentro del código de la aplicación.

Tabla 4 HU Abrir programa.

Historia de Usuario	
Número: 2	**Nombre de la Historia de Usuario:** Abrir programa
Usuario: Desarrollador	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Puntos de Estimación: 1	**Iteración Asignada:** 2
Descripción: Permite abrir un fichero XML que contiene información de un programa guardado previamente (imágenes y datos), el fichero es cargado mediante la biblioteca que se encarga de parsear el XML, QXMLStreamReader();	
Observaciones: Cualquier aplicación que trabaje con XML necesita un módulo de clases, su función es leer documentos y proporcionar acceso a su contenido y estructura. Para poder llevar a cabo esta función, la aplicación debe identificar la información XML y cómo se encuentra almacenada esta a través de un DTD (Document Type Declaration).	

Tabla 5 HU Analizar Sintaxis.

Historia de Usuario	
Número: 3	**Nombre de la Historia de Usuario:** Analizar Sintaxis
Usuario: Desarrollador	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Puntos de Estimación: 2	**Iteración Asignada:** 1
Descripción: Permite parsear el diagrama desarrollado por el usuario, ayudándolo a identificar los posibles errores cometidos durante la creación del mismo y habilita la opción de generar el código intermedio correspondiente al programa escrito, una vez corregidos todas las fallas.	
Observaciones: El proceso de análisis de la sintaxis, es el resultado del parseo del diagrama en cuestión. Permite la generación de una lista de errores en caso de que existan (HU. No.4). Una vez finalizada esta acción habilita la opción de generar el código intermedio, solo si la primera funcionalidad no arrojó fallo alguno. Este código intermedio será exportado y finalmente descargado en un dispositivo externo (HU. No.5).	

Tabla 6 HU Generar lista de errores.

Historia de Usuario	
Número: 4	**Nombre de la Historia de Usuario:** Generar lista de errores
Usuario: Desarrollador	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Puntos de Estimación: 1	**Iteración Asignada:** 1

Descripción: Permite mostrar al usuario los errores cometidos durante la creación del programa.
Observaciones: Posibilita el chequeo del diagrama mediante un algoritmo desarrollado, que tiene como fin arrojar un resultado (lista de errores) entendible para el usuario. Esta actividad forma parte de una de las principales funcionalidades de la aplicación (HU. No.3).

Tabla 7 HU Generar código intermedio.

Historia de Usuario	
Número: 5	**Nombre de la Historia de Usuario:** Generar código intermedio.
Usuario: Desarrollador	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Puntos de Estimación: 1	**Iteración Asignada:** 3
Descripción: Permite traducir el diagrama desarrollado en un lenguaje de alto nivel (C), con la finalidad de hacerlo entendible a dispositivos externos que manejen este tipo de información.	
Observaciones: Posibilita la generación de un código en otro lenguaje de mayor nivel (C), que facilite la comunicación entre el usuario y la maquina. Esta actividad forma parte de una de las principales funcionalidades de la aplicación (HU. No.3), y es dependiente de otra funcionalidad (HU. No.4).	

Tabla 8 HU Crear nuevo programa.

Historia de Usuario	
Número: 6	**Nombre de la Historia de Usuario:** Crear nuevo programa.

Usuario: Desarrollador	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Puntos de Estimación: 1	**Iteración Asignada:** 2
Descripción: Permite crear nuevos programas dentro de la aplicación.	
Observaciones: Habilita el editor para introducir nuevas sentencias con el fin de crear un nuevo diagrama, limpia todas las variables involucradas en el proceso y da la posibilidad de guardar los datos en caso de que se estuviese desarrollando un programa anterior.	

Tabla 9 Introducir, cambiar y consultar datos.

Historia de Usuario	
Número: 7	**Nombre de la Historia de Usuario:** Introducir, cambiar y consultar datos.
Usuario: Desarrollador	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Puntos de Estimación: 1	**Iteración Asignada:** 1
Descripción: Permite introducir, cambiar y consultar datos en la aplicación.	
Observaciones: Es una de las principales funciones que realiza el usuario en la aplicación, dándole características particulares al programa desarrollado, todas las acciones que encierra esta funcionalidad se realizan mediante un inspector de propiedades.	

2.5. Fase de Planificación.

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses.

Esta fase dura unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos.

Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración. La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

2.5.1. Estimación del esfuerzo por historia de usuario

Para el buen desarrollo del sistema propuesto, se realizó una estimación para cada una de las historias de usuario identificadas, llegando a los resultados que se muestran a continuación:

Tabla 10 Estimación del esfuerzo por HU.

Historia de usuario	Puntos estimados
Guardar programa	1 semana
Abrir programa	1 semana
Analizar sintaxis	2 semanas
Generar lista de errores	1 semana

Generar código intermedio	1 semana
Crear nuevo programa	1 semana
Introducir, cambiar y consultar datos	1 semana
Total	**8 semanas**

2.5.2. Plan de iteraciones.

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción. Los elementos que deben tomarse en cuenta durante la elaboración del plan de iteraciones son: historias de usuario. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

Una vez definidas las HU y estimado el esfuerzo propuesto para la realización de cada una de ellas, se decide realizar el sistema en 3 iteraciones, las cuales se describen detalladamente a continuación:

Iteración 1

Esta iteración tiene como objetivo darle cumplimiento a las HU que se consideraron de mayor importancia para el desarrollo de la aplicación. Al concluir dicha iteración se contará con todas las funcionalidades descritas en las HU 3, 4 y 7, las cuales hacen alusión al proceso de compilación del diagrama escalera y a la manipulación de los datos de cada componente insertado en el programa que el usuario se encuentra desarrollando.

Iteración 2

En la segunda iteración se realizará la implementación de las HU 1, 2 y 6, las cuales son las funcionalidades que engloban el proceso de salvar, cargar y crear un nuevo programa. Además, se corregirán errores o disconformidades del usuario con las HU implementadas en la iteración anterior. Esta

segunda versión será mostrada a los clientes con el único objetivo de realizar cambios en base a la aceptación del mismo.

Iteración 3

En la tercera iteración se realizará la implementación de la HU 5, la cual es la encargada de generar el código intermedio o código objeto que interpretará el agente externo. Esta HU es integrada con el resultado de las iteraciones anteriores y de esta manera se obtiene la versión 1.0 del producto final. A partir de este momento la aplicación será puesta en un proceso de prueba para evaluar su desempeño.

2.5.3. Plan de duración de las iteraciones.

Como parte del ciclo de vida de un proyecto guiado por la metodología de desarrollo de software XP, se crea el plan de duración de cada una de las iteraciones que se llevarán a cabo durante el desarrollo del proyecto. Este plan tiene como finalidad mostrar la duración de cada iteración, así como el orden en que serán implementadas las HU en cada una de las mismas.

Tabla 11 Plan de duración de las iteraciones.

Iteraciones	Orden de las HU a implementar	Duración total de la iteración
Iteración 1	Introducir, cambiar y consultar datos Generar lista de errores. Analizar sintaxis.	3 semanas
Iteración 2	Guardar programa. Abrir programa. Crear nuevo programa.	3 semanas
Iteración 3	Generar código intermedio.	1 semanas

Conclusiones parciales.

En este capítulo se inicia el desarrollo de la propuesta de solución que se desea implementar, se analizó la propuesta del sistema. Además se trata todo lo referente a las dos primeras fases de la metodología de desarrollo de software a utilizar, fase de exploración y planificación del sistema, donde se documentaron todos los artefactos generados en el transcurso de las mismas. Y quedó plasmado que el desarrollo del compilador se realizará en 3 iteraciones, como lo propone la metodología utilizada.

CAPÍTULO 3: Construcción de la Solución Propuesta.

Introducción

La Metodología XP plantea que la implementación de un software debe de realizarse de forma iterativa obteniendo al culminar cada iteración un producto funcional que debe de ser probado y mostrado al cliente para incrementar la visión de los desarrolladores con la opinión de éste.

En el presente capítulo se presenta el diagrama de clases de la arquitectura de la aplicación, así como el estándar de codificación utilizado para la implementación de las clases. Además se detallan las tres iteraciones llevadas a cabo durante la etapa de construcción del sistema, exponiéndose las tareas generadas por cada historia de usuario y las pruebas de aceptación efectuadas sobre el sistema.

3.1. Diseño de la Solución Propuesta

La Metodología de Desarrollo XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo. XP se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios.

Para el diseño de las aplicaciones, XP no requiere la presentación del sistema mediante diagramas de clases utilizando notación UML, en su lugar se usan otras técnicas como las tarjetas CRC (Contenido, Responsabilidad y Colaboración). No obstante el uso de estos diagramas puede aplicarse siempre y cuando influyan en el mejoramiento de la comunicación entre el equipo de desarrollo, no sea un peso su mantenimiento, no sean extensos y se enfoquen en la información importante.

3.1.1. Tarjetas CRC

Para poder diseñar el sistema como un equipo, se debe cumplir con tres principios: Cargo o Clase, Responsabilidad y Colaboración (CRC). Las tarjetas CRC permiten desprenderse del método de trabajo basado

en procedimientos y trabajar con una metodología basada en objetos. Además permiten que el equipo completo contribuya en la tarea del diseño.

Debido a la facilidad de uso y entendimiento, que propician dichas tarjetas, el equipo de desarrollo, decidió utilizarlas para diseñar la aplicación que se desea desarrollar.

Tabla 12 Plantilla para las Tarjetas CRC.

Tarjeta CRC	
Clase: Nombre de la clase que se está modelando.	
Súper Clase: Nombre de la clase padre en la herencia.	
Sub Clase(s): Nombre de la(s) clase(s) hija en la herencia.	
Responsabilidades: Es una descripción de alto nivel del propósito de la clase.	**Colaboraciones:** Indica con cuáles otras clases se requiere relación para cumplir la responsabilidad.

Tabla 13 Tarjeta CRC Componente

Tarjeta CRC	
Clase: Componente	
Súper Clase: -	
Sub Clases: ContactoA, ContactoC, Contador, División, Suma, Resta, Multiplicacion, Timer, Bobina, BobinaR, BobinaS, BobinaC, Lineas.	
Responsabilidades: Crear una plantilla que contenga elementos comunes entre los componentes a utilizar en el editor. Es la clase padre de los componentes.	**Colaboraciones:** Clase Tipo.

Tabla 14 Tarjeta CRC ContactoA.

Tarjeta CRC	
Clase: ContactoA	
Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	**Colaboraciones:** Clase Componente.

Tabla 15 Tarjeta CRC ContactoC

Tarjeta CRC	
Clase: ContactoC	
Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y	**Colaboraciones:** Clase Componente.

modificar estos datos.	

Tabla 16 Tarjeta CRC División.

Tarjeta CRC	
Clase: Division	
Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	**Colaboraciones:** Clase Componente.

Tabla 17 Tarjeta CRC Multiplicación.

Tarjeta CRC
Clase: Multiplicacion
Súper Clase: Componente
Sub Clases: -

Responsabilidades:	**Colaboraciones:**
Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	Clase Componente.

Tabla 18 Tarjeta CRC Suma.

Tarjeta CRC	
Clase: Suma	
Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	**Colaboraciones:** Clase Componente.

Tabla 19 Tarjeta CRC Resta.

Tarjeta CRC
Clase: Resta
Súper Clase: Componente
Sub Clases: -

Responsabilidades:	**Colaboraciones:**
Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	Clase Componente.

Tabla 20 Tarjeta CRC Contador.

Tarjeta CRC	
Clase: Contador	
Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	**Colaboraciones:** Clase Componente.

Tabla 21 Tarjeta CRC Timer.

Tarjeta CRC
Clase: Timer
Súper Clase: Componente

Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	**Colaboraciones:** Clase Componente.

Tabla 22 Tarjeta CRC Bobina.

Tarjeta CRC	
Clase: Bobina	
Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	**Colaboraciones:** Clase Componente.

Tabla 23 Tarjeta CRC BobinaS.

Tarjeta CRC
Clase: BobinaS

Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	**Colaboraciones:** Clase Componente.

Tabla 24 Tarjeta CRC BobinaR.

Tarjeta CRC	
Clase: BobinaR	
Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	**Colaboraciones:** Clase Componente.

Tabla 25 Tarjeta CRC BobinaC.

Tarjeta CRC

Clase: BobinaC	
Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación, así como permitir consultar y modificar estos datos.	**Colaboraciones:** Clase Componente.

Tabla 26 Tarjeta CRC Líneas.

Tarjeta CRC	
Clase: Lineas	
Súper Clase: Componente	
Sub Clases: -	
Responsabilidades: Establecer los atributos particulares del componente que representará en la aplicación.	**Colaboraciones:** Clase Componente.

Tabla 27 Tarjeta CRC Editor.

Tarjeta CRC	
Clase: Editor	
Súper Clase: QWidget	
Sub Clases: -	
Responsabilidades: Permitir realizar las operaciones necesarias para que el usuario pueda desarrollar su programa y manipularlo según necesite, atendiendo a las facilidades que brinde la aplicación.	**Colaboraciones:** Clase Temporal. Clase Controladora.

Tabla 28 Tarjeta CRC Temporal.

Tarjeta CRC	
Clase: Temporal	
Súper Clase: -	
Sub Clases: -	
Responsabilidades: Encargada de conformar la plantilla general que contendrá los datos de todos los componentes, permitiendo utilizarla para las	**Colaboraciones:** Clase Tipo.

acciones de guardar y abrir ficheros.	

Tabla 29 Tarjeta CRC Panel.

Tarjeta CRC	
Clase: Panel	
Súper Clase: QDockWidget	
Sub Clases: -	
Responsabilidades: Permite agrupar de forma organizada todos los componentes a utilizar en la aplicación, dándoles un orden y una distribución capaces de otorgarle funcionalidad, rapidez y sencillez al diseño del editor.	**Colaboraciones:**

Tabla 30 Tarjeta CRC Controladora.

Tarjeta CRC
Clase: Controladora
Súper Clase: -

Sub Clases: -	
Responsabilidades: Permite controlar cualquier operación de tipo general o independiente que se realicen sobre los componentes brindados por la aplicación, ya que agrupa mediante estructuras de datos particulares cualquier acción que realice el usuario.	**Colaboraciones:** Clase Componente Clase Timer Clase Contador Clase ContactoA Clase ContactoC Clase Temporal Clase Suma Clase Resta Clase Multiplicacion Clase Division Clase Bobina Clase BobinaR Clase BobinaS Clase BobinaC Clase Tipo

Tabla 31 Tarjeta CRC codeeditor.

Tarjeta CRC
Clase: codeeditor
Súper Clase: QPlainTextEdit
Sub Clases: -

Responsabilidades:	**Colaboraciones:**
Informa los errores encontrados durante el proceso de análisis del diagrama al usuario.	

Tabla 32 Tarjeta CRC mainwindow.

Tarjeta CRC	
Clase: mainwindow	
Súper Clase: QMainWindow	
Sub Clases: -	
Responsabilidades: Encargada de construir la ventana principal de la aplicación, así como de contener varios de las principales funcionalidades del software.	**Colaboraciones:** Clase Componente Clase Timer Clase Contador Clase ContactoA Clase ContactoC Clase Temporal Clase Suma Clase Resta Clase Multiplicacion Clase Division Clase Bobina Clase BobinaR Clase BobinaS Clase BobinaC Clase Tipo

	Clase Controladora

3.1.2.1. Diagrama de Componentes

Un diagrama de componentes muestra las dependencias lógicas entre componentes software, sean éstos componentes fuentes, binarios o ejecutables, ilustran las piezas del software, controladores embebidos, etc. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema, es decir, para describir la vista de implementación estática de un sistema. Los diagramas de componentes se relacionan con los diagramas de clases, ya que un componente normalmente se corresponde con una o más clases, interfaces o colaboraciones pero un diagrama de componentes tiene un nivel más alto de abstracción que un diagrama de clases, usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución [11]

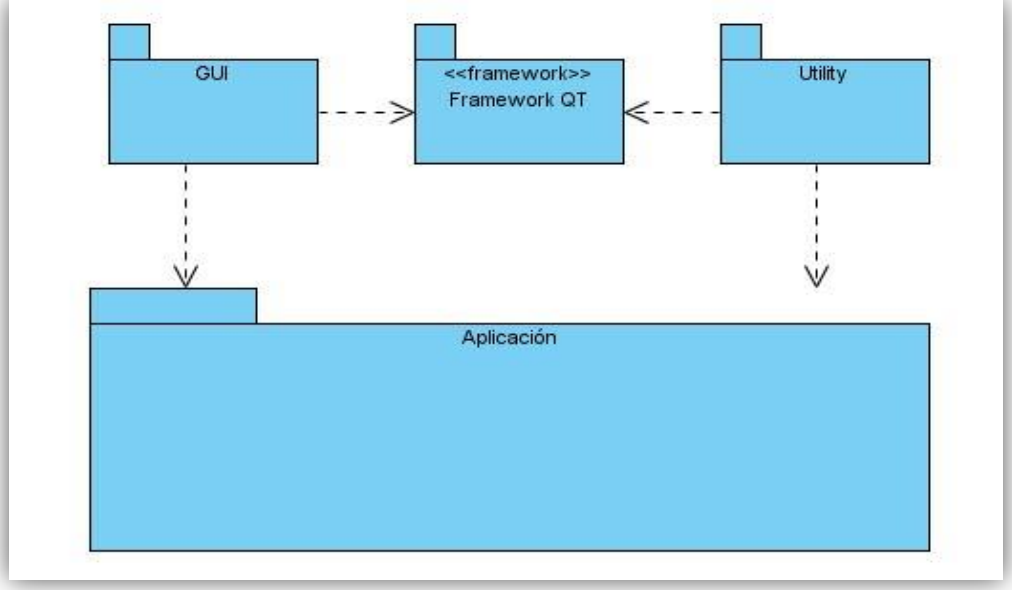

Figura 3.0.1 Diagrama de Paquete de Componentes.

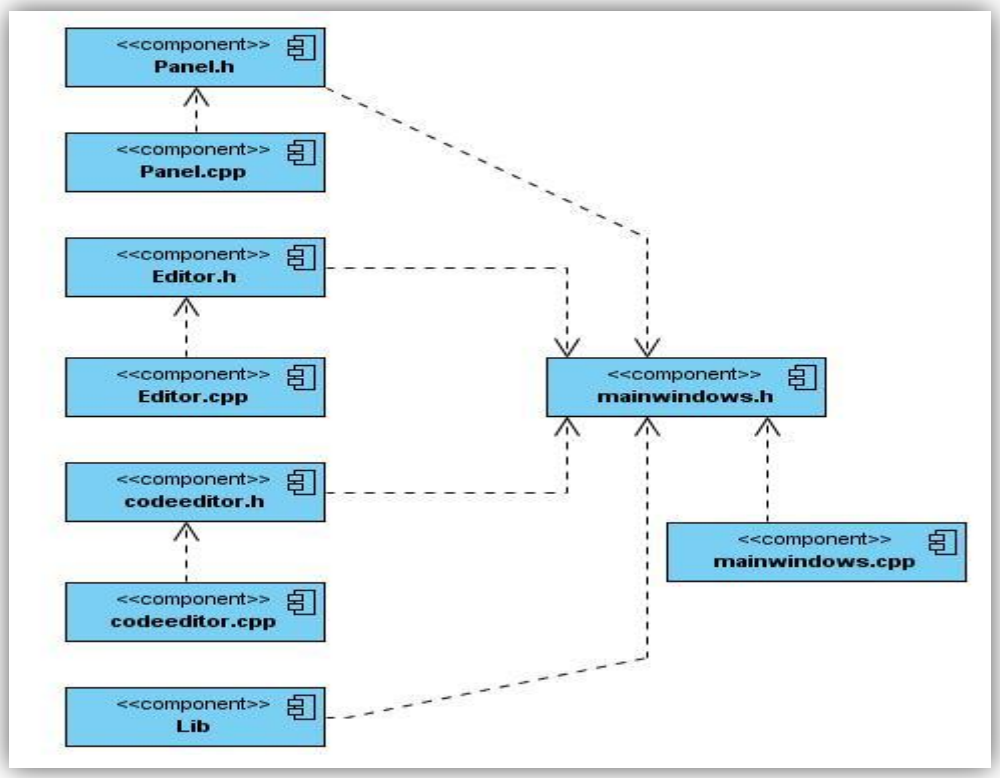

Figura 3.0.2 Diagrama de componentes. Paquete Interface.

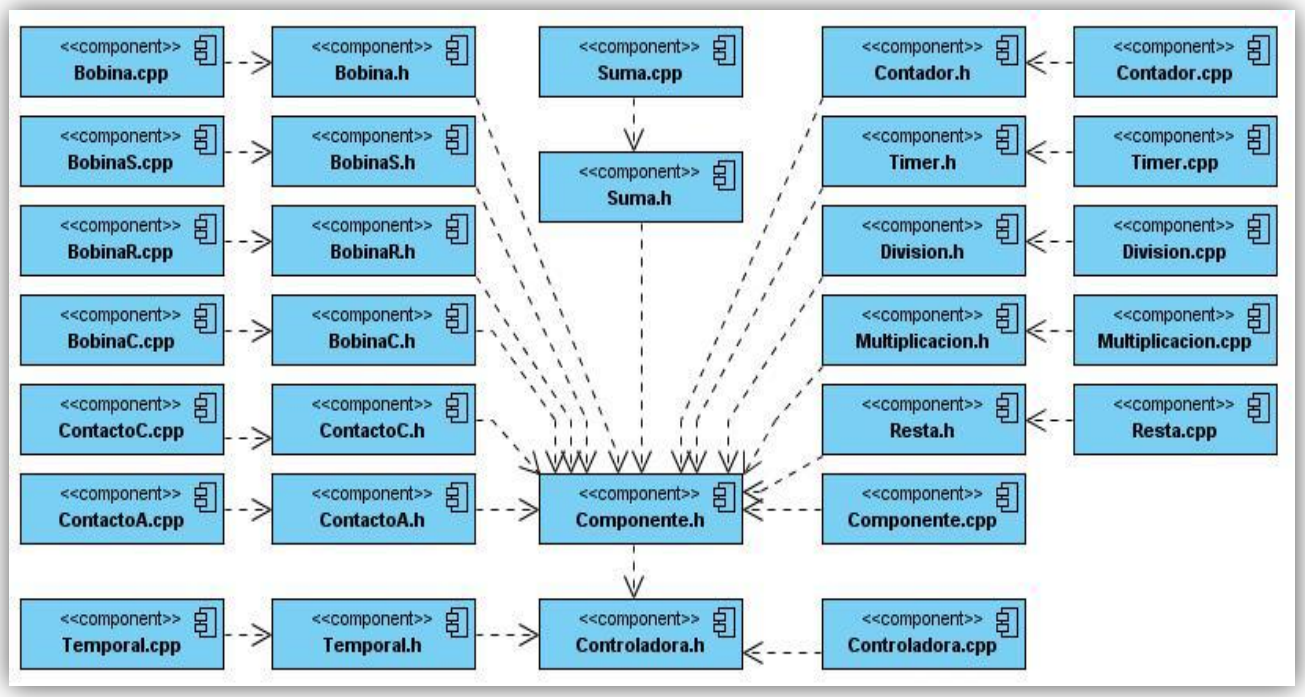

Figura 3.0.3 Diagrama de componentes. Paquete Utility.

3.1.4. Patrón de Diseño.

Los patrones de diseño, también llamados estilos de programación o convenciones de código, no son más que convenios para escribir código fuente en ciertos lenguajes de programación. Estos estándares elevan la mantenibilidad del código, sirven como punto de referencia para los programadores, mantienen un estilo de programación y ayudan a mejorar el proceso de codificación, haciéndolo, entre otras cosas, mucho más eficiente.

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan.

Para la implementación de la aplicación se decidió utilizar el patrón Decorator.

Patrón Decorator:

Es un tipo de patrón estructural, que añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender funcionalidad. Permite introducir o eliminar funcionalidades de un componente sin modificar su apariencia externa o su función principal.

En este patrón existen varios participantes, entre ellos encontramos:

- ✓ **Component:** Define la interface de los objetos a los que se le puede adicionar responsabilidades dinámicamente.
- ✓ **Concrete Component:** Define el objeto al que se le puede adicionar una responsabilidad.
- ✓ **Decorator:** Mantiene una referencia al objeto Component y define una interface de acuerdo con la interface de Component.
- ✓ **Concrete Decorator:** Adiciona la responsabilidad al componente.

3.2. Descripción del XML

Como se expresó anteriormente el lenguaje XML fue el escogido para almacenar la información ofrecida por la aplicación, debido a las ventajas que brinda el mismo y a las facilidades que ofrece para el trabajo con imágenes, así como a la existencia de una biblioteca lo bastante amplia en Qt que permite su fácil manipulación. Para el desarrollo del sistema propuesto se confeccionó un estándar XML para los componentes que conforman el editor, permitiéndoles a los usuarios que interactúen con la aplicación chequear los datos manipulados de una forma rápida y sencilla.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Programa>
  - <Componente>
      <Id>1</Id>
      <Imagen>:/imagenes/TimenOD.PNG</Imagen>
      <LocalizacionX>32</LocalizacionX>
      <LocalizacionY>40</LocalizacionY>
      <CuadranteX>720</CuadranteX>
      <CuadranteY>80</CuadranteY>
      <A>0</A>
      <B>0</B>
      <Archivo>0</Archivo>
      <Bit>0</Bit>
    </Componente>
  - <Componente>
      <Id>2</Id>
      <Imagen>:/imagenes/ContactoA.PNG</Imagen>
      <LocalizacionX>33</LocalizacionX>
      <LocalizacionY>39</LocalizacionY>
      <CuadranteX>0</CuadranteX>
      <CuadranteY>0</CuadranteY>
      <A>0</A>
      <B>0</B>
      <Archivo>1</Archivo>
      <Bit>0</Bit>
    </Componente>
  - <Componente>
```

3.3. Desarrollo de las Iteraciones

En la fase de Planificación se detallaron las HU correspondientes a cada una de las iteraciones a desarrollar, teniendo en cuenta las necesidades requeridas por el cliente. Durante el transcurso de las iteraciones se lleva a cabo una revisión del plan de iteraciones y se modifica en caso de ser necesario. Como parte de este plan, se descomponen las HU en tareas de programación o ingeniería, asignando a un equipo de desarrollo (o una persona), responsable de su implementación, aplicando la práctica de la programación en parejas. Estas tareas no tienen que necesariamente ser entendidas por el cliente, pueden ser escritas en lenguaje técnico y son para el uso estricto de los programadores.

Teniendo en cuenta la planificación realizada con anterioridad, se llevó a cabo el desarrollo del sistema en tres iteraciones, obteniéndose como finalidad un producto con todas las restricciones y características deseadas por el cliente. A continuación se detallan cada una de las iteraciones.

3.3.1. Iteración 1

En esta iteración se le da cumplimiento a la implementación de las HU que corresponden a los números 1 y 2, consideradas de mayor importancia para el desarrollo de la aplicación, con el fin de obtener una versión del producto con algunas de las funcionalidades críticas como: Compilar, generar lista de errores e introducir, cambiar y consultar datos.

Tabla 33 HU abordadas en la primera iteración.

Historias de Usuario	Tiempo de Implementación (semanas)	
	Estimación	Real
Analizar sintaxis	2	2
Generar lista de errores	1	1
Introducir, cambiar y consultar datos.	1	1

Tabla 34 Tarea de la HU #7.

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 7
Nombre de la Tarea: Crear mecanismos que permitan manipular los datos de los componentes.	
Tipo de Tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 30/01/2011	**Fecha fin:** 06/02/2011
Programador responsable: Eduardo Millán Núñez	

Descripción: Tiene como fin, dotar a cada componente de valores válidos, que permitan el funcionamiento del programa.

Tabla 35 Tarea de la HU #4.

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 4
Nombre de la Tarea: Mostrar los errores cometidos durante la construcción del programa.	
Tipo de Tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 07/02/2011	**Fecha fin:** 13/02/2011
Programador responsable: Eduardo Millán Núñez	
Descripción: Orienta al usuario de los errores cometidos en la construcción del programa para guiarlo en la corrección de los mismos.	

Tabla 36 Tarea de la HU #3.

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 3
Nombre de la Tarea: Dotar al editor de un algoritmo de chequeo de errores y generación de código intermedio.	
Tipo de Tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 14/02/2011	**Fecha fin:** 28/02/2011
Programador responsable: Eduardo Millán Núñez	
Descripción: Es una de las principales funcionalidades que debe de tener la aplicación. Se implementa	

con el fin de agilizar el proceso de corrección de errores y generación de código intermedio, solo realizándose esta última actividad, si el programa desarrollado está completamente correcto.

3.3.2. Iteración 2

En esta iteración se implementaron las HU que corresponden a los números 1, 2 y 6. Dichas HU son las que brindan las funcionalidades de guardar, abrir y crear un nuevo programa.

Tabla 37 HU abordadas en la segunda iteración.

Historias de Usuario	Tiempo de Implementación (semanas)	
	Estimación	Real
Guardar programa	1	1
Abrir programa	1	1
Nuevo programa	1	1

Tabla 38 Tarea de la HU #1.

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 1
Nombre de la Tarea: Crear un estándar XML para el almacenamiento de la información de cualquier componente.	
Tipo de Tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 01/03/2011	**Fecha fin:** 08/03/2011
Programador responsable: Eduardo Millán Núñez	
Descripción: Funcionalidad que se encarga de almacenar todo el programa desarrollado por el usuario	

en una plantilla XML conformada por el desarrollador.

Tabla 39 Tarea de la HU #2.

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 2
Nombre de la Tarea: Desarrollar un método de carga de ficheros XML para mostrar su contenido en el editor.	
Tipo de Tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 09/03/2011	**Fecha fin:** 16/03/2011
Programador responsable: Eduardo Millán Núñez	
Descripción: Funcionalidad que se encarga de posicionar y establecer los datos de los componentes utilizados en programas desarrollados anteriormente y guardados en ficheros de tipo XML.	

Tabla 40 Tarea de la HU #6.

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 6
Nombre de la Tarea: Permitir desarrollar nuevos programas.	
Tipo de Tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 17/03/2011	**Fecha fin:** 24/03/2011
Programador responsable: Eduardo Millán Núñez	
Descripción: Mecanismos de limpieza de todas las estructuras de almacenamiento, permitiendo comenzar desde cero cualquier tipo de actividad.	

3.3.3. Iteración 3

En esta última iteración, se le dio cumplimiento a la HU 5. Dicha HU desempeña la funcionalidad generar el código intermedio que se obtiene como resultado de la traducción del lenguaje escalera a el lenguaje C.

Tabla 41 HU abordadas en la tercera iteración.

Historias de Usuario	Tiempo de Implementación (semanas)	
	Estimación	Real
Generar código intermedio	1	1

Tabla 42 Tarea de la HU #5.

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 5
Nombre de la Tarea: Generar código intermedio a partir del programa en lenguaje escalera.	
Tipo de Tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 25/03/2011	**Fecha fin:** 02/04/2011
Programador responsable: Eduardo Millán Núñez	
Descripción: Algoritmo de traducción encargado de conformar un fichero con el código en lenguaje C del programa desarrollado en lenguaje escalera.	

3.4 Pruebas

Uno de los pilares de la metodología XP es el uso de las pruebas para comprobar el funcionamiento de los códigos que se vayan implementando. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones [9]

La metodología ágil XP divide las pruebas en dos grupos: pruebas unitarias y pruebas de aceptación. Las pruebas unitarias son desarrolladas por los programadores y se encargan de verificar el código automáticamente y las pruebas de aceptación están destinadas a verificar que al final de cada iteración las Historias de Usuario cumplen con la funcionalidad asignada y satisfagan las necesidades del cliente.

Las pruebas de aceptación son más importantes que las pruebas unitarias dado que significan la satisfacción del cliente con el producto desarrollado y el final de una iteración y el comienzo de la siguiente, por esto el cliente es la persona adecuada para diseñar las pruebas de aceptación.

3.4.1 Desarrollo dirigido por pruebas.

El desarrollo dirigido por pruebas (TDD), es una práctica de programación que involucra otras dos prácticas: Escribir las pruebas primero (Test First Development) y Refactorización (Refactoring). Para escribir las pruebas generalmente se utiliza la Prueba Unitaria (unit test).

Primeramente se escribe una prueba y se verifica que las pruebas fallen, luego se implementa el código que haga que la prueba pase satisfactoriamente y seguidamente se refactoriza el código escrito. El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione (Del inglés: Clean code that works). La idea es que los requerimientos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que los requerimientos se hayan implementado correctamente.

TDD permite un diseño más robusto, tanto es así que a menudo se piensa TDD como Diseño manejado por las pruebas (Test Driven Design). En consecuencia, TDD facilita un diseño más mantenible a través de la noción de pruebas. Estas pruebas obligan a reflexionar sobre el comportamiento del código y la forma de garantizar que funciona según lo previsto. La mayoría de las veces, el código influenciado por TDD es relativamente seguro, y sin duda, es bastante simple [9].

La práctica de TDD puede resumirse en 3 simples reglas:

- ✓ No se puede escribir código productivo, a menos que sea para hacer pasar un test fallido.
- ✓ No se puede escribir más que lo necesario para que falle un test unitario; los errores de compilación se consideran fallos.
- ✓ No se puede escribir más código productivo del estrictamente necesario para hacer pasar un test [10]

Ciclo de desarrollo TDD

1. **Escribir la prueba.** Para escribir la prueba, el desarrollador debe entender claramente las especificaciones y los requisitos. El diseño del documento deberá cubrir todos los escenarios de prueba y condición de excepciones.
2. **Escribir el código haciendo que pase la prueba.** Este paso fuerza al programador a tomar la perspectiva de un cliente considerando el código a través de sus interfaces. Ésta es la parte conducida por el diseño, del TDD. Como parte de la calibración de la prueba, el código debe fallar la prueba significativamente las primeras veces.
3. **Ejecutar las pruebas automatizadas.** Si pasan, el programador puede garantizar que el código resuelve los casos de prueba escritos. Si hay fallos, el código no resolvió los casos de prueba.
4. **Refactorización y limpieza en el código.** Después se vuelven a efectuar los casos de prueba y se observan los resultados.
5. **Repetición.** Después se repetirá el ciclo y se comenzará a agregar las funcionalidades adicionales o a arreglar cualquier error [9].

3.4.2 Pruebas de aceptación

Las pruebas de aceptación son creadas en base a las HU, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada.

Las pruebas de aceptación son consideradas como “pruebas de caja negra”. Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Así mismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación.

Dado que la responsabilidad es grupal, es recomendable publicar los resultados de las pruebas de aceptación, de manera que todo el equipo esté al tanto de esta información.

Para la realización de cada una de las pruebas de aceptación se siguieron una serie de pasos que se muestran a continuación:

- ✓ Identificar todas las acciones en la historia de usuario.
- ✓ Para cada acción escribir al menos una prueba.
- ✓ Para algunos datos, reemplazar las entradas que hacen que la acción ocurra y llenar en la casilla resultados esperados los resultados obtenidos.
- ✓ Para otros datos, reemplazar las entradas que hacen que la acción falle, y registrar los resultados.

Tabla 43 Prueba de aceptación para la HU “Introducir, cambiar y consultar datos”.

Caso de Prueba de Aceptación	
Código: HU7_P1	**Historia de Usuario:** 7
Nombre: Comprobar la gestión de datos.	
Descripción: Verificar el correcto funcionamiento de los algoritmos de almacenamiento y modificación de los datos de los componentes.	
Condiciones de Ejecución: El cliente debe comprobar que se puede acceder a los datos de los componentes así como modificar los mismos.	
Entrada / Pasos de Ejecución: Probar la gestión de datos de cada uno de los componentes.	
Resultado Esperado: Se accede y se manipulan de forma satisfactoria todos los datos de todos	

los componentes.
Evaluación de la Prueba: -

Tabla 44 Prueba de aceptación para la HU "Generar lista de errores".

Caso de Prueba de Aceptación	
Código: HU4_P1	**Historia de Usuario:** 4
Nombre: Comprobar lista de errores.	
Descripción: Verificar el correcto funcionamiento de los algoritmos de chequeo de errores.	
Condiciones de Ejecución: El cliente debe comprobar que se generan los errores cometidos durante la construcción del programa en lenguaje escalera, especificando con exactitud la falla encontrada.	
Entrada / Pasos de Ejecución: Generar listas de errores de varios programas con diferentes fallas con el fin de verificar la mayor cantidad posible de fallos.	
Resultado Esperado: Se genera de forma satisfactoria la lista de errores correspondiente a cada compilación de la aplicación.	
Evaluación de la Prueba: -	

Tabla 45 Prueba de aceptación para la HU "Analizar sintaxis".

Caso de Prueba de Aceptación	
Código: HU3_P1	**Historia de Usuario:** 3
Nombre: Analizar sintaxis del programa escrito.	
Descripción: Verificar el correcto funcionamiento de los métodos que conforman el proceso de	

análisis de la sintaxis. Entre los cuales se encuentra el proceso de generar lista de errores.
Condiciones de Ejecución: El cliente debe comprobar que el proceso de análisis de la sintaxis realiza todas las tareas que agrupa esta funcionalidad, entre las cuales se encuentra la generación de la lista de errores correspondiente al programa en construcción así como habilitar las funcionalidades de generación de código intermedio.
Entrada / Pasos de Ejecución: Verificar varios programas bajo diferentes condiciones, chequeando que se realicen todas las tareas que conlleva la realización de esta actividad.
Resultado Esperado: Todas las tareas comprendidas en el proceso de análisis de la sintaxis se desarrollan de forma satisfactoria.
Evaluación de la prueba: -

Tabla 46 Prueba de aceptación para la HU "Guardar programa".

Caso de Prueba de Aceptación	
Código: HU1_P1	**Historia de Usuario:** 1
Nombre: Comprobar datos del fichero creado.	
Descripción: Verificar que el fichero generado una vez guardado el programa contenga todos los datos establecidos en la plantilla de tipo XML.	
Condiciones de Ejecución: El cliente debe comprobar que el fichero creado se guardó en la dirección deseada, así como el contenido del mismo esté acorde con los datos introducidos en los componentes.	
Entrada / Pasos de Ejecución: Guardar varios programas bajo diferentes condiciones y chequear el fichero generado cada vez que se ejecute esta acción, para comprobar que se actualizaron los datos de forma satisfactoria.	
Resultado Esperado: El fichero generado tras cada operación contiene los datos establecidos en	

la construcción del programa.
Evaluación de la prueba: -

Tabla 47 Prueba de aceptación para la HU "Abrir programa".

Caso de Prueba de Aceptación	
Código: HU2_P1	**Historia de Usuario:** 2
Nombre: Cargar datos de ficheros de tipo XML generados por la aplicación.	
Descripción: Verificar que el programa cargado mantenga la misma arquitectura y formato del programa guardado, así como chequear que cada parámetro en cada componente sea el correcto.	
Condiciones de Ejecución: El cliente debe comprobar que el fichero cargado generó el programa que anteriormente se había guardado, sin sufrir ningún tipo de alteración.	
Entrada / Pasos de Ejecución: Cargar diferentes ficheros con diferentes datos para comprobar que el proceso se realiza de forma correcta.	
Resultado Esperado: El fichero cargado en la aplicación muestra los datos guardados con anterioridad sin sufrir ningún tipo de alteración.	
Evaluación de la prueba: -	

Tabla 48 Prueba de aceptación para la HU "Nuevo programa".

Caso de Prueba de Aceptación	
Código: HU6_P1	**Historia de Usuario:** 6
Nombre: Crear nuevos programas.	
Descripción: Verificar que el las estructuras de almacenamiento de datos son limpiadas y puestas	

nuevamente en estado disponible para almacenar datos de nuevos programas.
Condiciones de Ejecución: El cliente debe comprobar que todas las estructuras borran los datos que poseían con anterioridad con el fin de no introducir valores no deseados en nuevos programas.
Entrada / Pasos de Ejecución: Una vez creados diferentes programas, ejecutar el algoritmo de limpieza de datos y confirmar que las estructuras envueltas en este proceso no contengan ningún tipo de información.
Resultado Esperado: Todas las estructuras de almacenamiento borran su información.
Evaluación de la prueba: -

Tabla 49 Prueba de aceptación para la HU "Generar código intermedio".

Caso de Prueba de Aceptación	
Código: HU7_P1	**Historia de Usuario:** 7
Nombre: Chequeo de datos del código intermedio generado.	
Descripción: Verificar que el fichero generado una vez terminado un programa, contenga las funciones correspondientes de la traducción del lenguaje escalera a C, y que además el objetivo con el cual fue desarrollado el programa en la aplicación, no pierda su esencia, es decir, que no se altere el funcionamiento final del código.	
Condiciones de Ejecución: El cliente debe comprobar que el programa desarrollado no contiene errores, para poder realizar una correcta traducción de un lenguaje a otro.	
Entrada / Pasos de Ejecución: Generar varios programas con diferentes instrucciones con el fin de abarcar la mayor cantidad posible de situaciones y comprobar que el código generado funciona correctamente. Este último paso se comprobará en el PLC en desarrollo.	
Resultado Esperado: El código generado ejecuta satisfactoriamente las instrucciones escritas por	

el usuario en lenguaje escalera.
Evaluación de la prueba: -

Conclusiones parciales

En este capítulo se brindó información detallada de las clases que componen la aplicación, así como sus atributos y las relaciones entre ellas. Se elaboran las tarjetas CRC y se definen los estándares de codificación utilizados durante su desarrollo. De igual manera de desarrollan las tareas que dan solución a las historias de usuario. Se realizan las pruebas unitarias y se definen las pruebas de aceptación. Con la culminación de este capítulo se da por terminada la propuesta de solución de la aplicación a implementar.

Conclusiones Generales

El presente trabajo se enfocó en el estudio y desarrollo de una aplicación de usuario capaz de analizar sintacticamente y traducir programas desarrollados en lenguaje gráfico (LD) a lenguajes escritos de tipo C, para la generación de instrucciones al PLC que se encuentra en desarrollo en nuestra universidad.

Como resultado del trabajo realizado se logró la implementación de una aplicación con las siguientes ventajas:

- ✓ Ahorro en tiempo y recursos para las personas que posteriormente interactúen con el PLC en cuestión.
- ✓ Fácil mantenimiento, de modo que se le puedan adicionar nuevos módulos a la aplicación.
- ✓ Fácil uso debido al desarrollo de un conjunto de plantillas o contenidos prediseñados.
- ✓ Reutilización de código y componentes en proyectos con características similares.
- ✓ Utilización de la herramienta en diferentes sistemas operativos.

El sistema cuenta con la documentación necesaria para su entendimiento, mantenimiento y posterior escalabilidad debido al uso de la metodología de desarrollo ágil XP, que permitió construir los artefactos requeridos.

El profundo estudio de diferentes temáticas en el campo de los PLC, se ven reflejado en el resultado que finalmente se alcanzó, siendo este, un paso importante para lograr un objetivo mayor, el de propiciar la independencia tecnológica de nuestro país.

Otro factor de importancia se ve reflejado en el cumplimiento de los objetivos trazados al inicio de esta actividad científica, pues se le dieron solución atendiendo al orden de prioridad de cada uno, en diferentes etapas del desarrollo del trabajo.

Recomendaciones

Los objetivos planteados para el desarrollo de este trabajo fueron alcanzados en el mismo, aunque se considera necesario continuar su perfeccionamiento con vistas a incrementar las prestaciones de la herramienta. A continuación se exponen un conjunto de recomendaciones que consideramos importantes a tener en cuenta para futuras versiones:

- ✓ Continuar el perfeccionamiento del diseño del editor, con el fin de hacer más fácil su uso.
- ✓ Desarrollar nuevos módulos con el objetivo de generar códigos intermedios en diferentes lenguajes.
- ✓ Implementar nuevas versiones que sean capaces de ser cargadas en el propio PLC.
- ✓ Dotar la aplicación de un modulo de simulación que permita el chequeo del programa desde otra perspectiva.

Estas, junto a otras muchas, podrían ser parte de las futuras mejoras a realizar sobre la herramienta, teniendo en cuenta que su objetivo final siempre ha de ser el mismo, el de contribuir a desarrollar un software que impulse el campo de la automática y genere ingresos a nuestra economía desde nuestro campo de acción.

Bibliografía

1. **V. Ahao, Alfred, Sethi, Ravi and Dullman, Jeffrey.** *Compiladores: principios, tecnicas y herramientas.* s.l. : Pearson Educación, 1998. 9684443331.

2. **Bolton, W.** *Programmable Logic Controllers.* s.l. : British Library, 2006.

3. **Zaragoza, Jordi, González, David and Balcells, Josep.** *Sistemas Electrónicos y Automáticos.* Terrassa : UPC, 2004.

4. *Conferencia 1. Introducción al proceso de compilación. .* **Informáticas., Universidad de las Ciencias.** La Habana, Cuba. : s.n., 2005-2006.

5. *UML 2.0 in a Nutshell.* **Pilone, Dan and Pitman, Neil.** s.l. : O'Reilly, 2005.

6. *El Lenguaje Unificado de Modelado UML.* **Booch, Grady, Jacobson, Ivar and Rumbaugh, Jim.** 1999. 0201571684.

7. *Curso de C++.* **Pozo Coronado, Salvador.** México DF, México : s.n., 2003.

8. *Programación en C++ con QT.* **Sande, Martín.** Buenos Aires, Argentina : s.n., 2004. 281622.

9. **Ferrer, Jorge.** Metodologías Ágiles. [Online] http://libresoft.dat.escet.urjc.es/html/downloads/ferrer-20030312.pdf..

10. **Craig, Lerman.** *UML y Patrones Introducción al Análisis y Diseño Orientado a Objeto.* México : Person, 1999. 970-17-0261-1.

11. **Sanchez Mendoza, María.** *Metodologías de Desarrollo de Software.* 2004.