Universidad de las Ciencias Informáticas
Facultad 5

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Título:

Compilador para el lenguaje Diagrama de Escalera

Autor:

Alex Fernández Castrillo

Tutor:

Ing. Yunior Peralta González

Ing. Julio Alberto Leyva Durán

Co-tutor:

Ing. Arianna Gómez Vargas

La Habana, 2014

Declaración de Autoría

Declaro ser autor del presente trabajo de diploma y concedo a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo. Para que así conste firmamos la presente a los ___ días del mes de _____ del año _____.

_____

Alex Fernández Castrillo

Autor

_____

Ing. Yunior Peralta González

Tutor

_____

Ing. Julio Alberto Leyva Durán

Tutor

_____

Arianna Gómez Vargas

Co-tutor

Datos de Contacto

Tutor:

Nombre y Apellidos: Yunior Peralta González.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: yperalta@uci.cu

Tutor:

Nombre y Apellidos: Julio Alberto Leyva Durán.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: jaleyva@uci.cu

Co-tutor:

Nombre y Apellidos: Arianna Gómez Vargas.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: agomezv@uci.cu

Agradecimientos

Agradecer a toda mi familia por el apoyo que me brindaron durante estos cinco largos años en mis estudios, además de saber aconsejarme de la mejor manera para lograr ser una mejor persona en la vida.

A mis padres y mi hermana por ser las personas que más me ayudaron para que yo lograra salir adelante, por estar pendientes de mis problemas y en todo momento ayudarme a buscar una solución, por ser comprensivos y brindarme todo su cariño.

Finalmente agradecer a todas aquellas personas que de una forma u otra estuvieron presentes y contribuyeron en mi formación como profesional.

Dedicatoria

Este trabajo es dedicado a toda mi familia por el apoyo que me brindaron en mi etapa de estudiante.

Resumen

Con el actual desarrollo de la industria es necesario tener un mayor control y automatización de los procesos industriales que surgen. Para dar respuesta a esto son utilizados determinados dispositivos de automatización, como los Controladores Lógicos Programables (PLC, por sus siglas en inglés). Por esta razón, la Universidad de las Ciencias Informáticas, específicamente la línea de Sistemas Embebidos del Centro de Informática Industrial crea el proyecto PLC Arex, cuyo propósito es el desarrollo de un dispositivo de control y adquisición de datos, haciendo uso de la tarjeta basada en microcontroladores llamada CID 300/9. Para lograr un correcto funcionamiento del mismo se tiene una interfaz, que permite la programación de rutinas de control para el PLC desde una terminal. Actualmente no se cuenta con un compilador que soporte la arquitectura de la tarjeta para los lenguajes que se utilizan en la programación de estos dispositivos. Pero se tiene un compilador GCC, para lenguaje C/C++, que soporta la arquitectura antes mencionada. El presente trabajo muestra la creación de un mecanismo que permite compilar rutinas de control para el PLC Arex. Como resultado se obtuvo un traductor del lenguaje Diagrama Escalera, definido en la norma IEC 61131-3, al lenguaje C, que unido a un *toolchain* de C/C++ permite compilar para la arquitectura de la tarjeta CID 300/9, facilitando la programación del PLC Arex.

Palabras clave: rutinas de control, lenguajes de programación, traductor.

Índice General

Índice General

Índice de Figuras

Índice de Tablas

Índice de Tablas

Introducción

El acelerado desarrollo de la ciencia y la tecnología ha revolucionado, en todos los sectores de la sociedad, la forma en que se gestionan las actividades diarias. El sector industrial no se ha quedado enajenado ante tan importante realidad, sino que ha ido evolucionando a la par con el desarrollo, de modo que los diversos procesos que se derivan en la actividad productiva han aumentado su grado de complejidad. Dichos procesos pueden ir desde el control de parámetros, como la temperatura y la presión, hasta el manejo de máquinas robotizadas, lo cual implica la utilización de gran cantidad de mano de obra, trayendo como consecuencia la ocurrencia de errores humanos y un aumento desmedido de los costos asociados a su ejecución.

Atendiendo a los inconvenientes anteriores y para disminuir la complejidad en el control y automatización de los procesos es que surgen diferentes dispositivos como: los Controladores Lógicos Programables (en lo adelante PLC); ”los cuales son aparatos electrónicos operados digitalmente, que utilizan una memoria para el almacenamiento de instrucciones, capaces de realizar diferentes operaciones tales como control de tiempos, lógica, secuenciación, aritméticas entre otras; para controlar a través de módulos de entrada/salida señales digitales y analógicas de varios tipos de procesos o máquinas” [1]. Sustituyendo así el trabajo de varios técnicos por el de una máquina que puede ser controlada por un especialista en automática, evitando en gran medida la ocurrencia de errores humanos y por consiguiente una disminución en los costos.

Una parte vital en el funcionamiento de los PLC es la rutina de control, por medio de la cual se controlan las diferentes variables asociadas al proceso, modificándolas de acuerdo a determinados valores prefijados por el operador del dispositivo.

Para la programación de dichas rutinas de control, están definidos en la norma internacional IEC[1] 61131-3 cuatro lenguajes: Lista de Instrucciones (IL por sus siglas en inglés), Texto Estructurado (ST por sus siglas en inglés), además de Diagrama de Escalera (LD por sus siglas en inglés) y Diagrama de Bloques Funcionales (FBD por sus siglas en inglés), siendo los dos primeros literales y los otros gráficos, respectivamente. En el marco de programación de un PLC los lenguajes gráficos son considerados de alto nivel, mientras que los literales son de bajo nivel. Los lenguajes definidos en el estándar son equivalentes entre sí, por lo que una rutina implementada en LD puede ser traducida a otro de los lenguajes definidos en el estándar sin ningún tipo de complicación.

En la Universidad de las Ciencias Informáticas (UCI) específicamente en la línea de Sistemas Embebidos del Centro de Informática Industrial, existe el proyecto PLC Arex, que tiene como

[1] Organización de normalización en los campos eléctrico, electrónico y tecnologías relacionadas.

objetivo desarrollar un dispositivo automático de adquisición y control de datos, como variante de automatización integral, haciendo uso de una tarjeta basada en micros controladores[2] denominada CID 300/9. Para la programación de las rutinas de control que ejecuta el PLC se utilizan los lenguajes establecidos por la norma IEC 61131-3. Para las rutinas de control implementadas en el lenguaje Diagrama de Escalera (LD) no se cuenta, en la línea Sistema Embebidos con un compilador que soporte este lenguaje para la arquitectura de la tarjeta CID 300/9. Sin embargo existe un *toolchain*[3] que permite compilar rutinas codificadas en el lenguaje C para la arquitectura de la tarjeta CID 300/9.

Ante lo descrito surge como **problema a resolver:** ¿Cómo facilitar la programación de rutinas de control en el lenguaje Diagrama de Escalera para el PLC Arex?

Se establece como **objeto de estudio** los procesos de compilación de lenguajes de programación para Controladores Lógicos Programables.

El **objetivo general** creación de un compilador para el lenguaje Diagrama de Escalera (LD).

El **objetivo específico es** desarrollar un traductor del lenguaje Diagrama de Escalera (LD) al lenguaje C.

Enmarcado en el **campo de acción** procesos de compilación del lenguaje Diagrama Escalera.

Tareas de investigación

- Caracterizar el lenguaje Diagrama de Escalera.
- Caracterizar el estándar OpenPLC.
- Caracterizar el lenguaje C.
- Definir las herramientas y tecnologías a utilizar para desarrollar la aplicación.
- Implementar el traductor del lenguaje diagrama de escalera al lenguaje C.
- Ejecutar pruebas funcionales a la aplicación desarrollada.

Los **métodos de investigación científica** empleados:

- **Análisis y síntesis** para la revisión de las normas, estándares, lenguajes y herramientas existentes que permiten llevar a cabo la traducción de un lenguaje al

[2] Micro controlador: Circuitos integrados programables, capaces de ejecutar las órdenes grabadas en su memoria.
[3] Toolchain: Conjunto de programas informáticos que se usan para crear un determinado producto, dígase un programa o sistema informático. Generalmente los distintos programas son usados en cadena.

otro y para establecer los niveles de relación entre los componentes que conforman toda la investigación alrededor del objetivo específico de desarrollar un traductor.

- **Histórico y lógico** para la organización de los conocimientos y el establecimiento de niveles de relación y entre los lenguajes a traducir con el formato que define el estándar OpenPLC. Así como el análisis de las herramientas existentes asociadas a la implementación del compilador para lenguaje Diagrama de Escalera (LD).

- **Genético** para el enfoque del objetivo específico de desarrollar un traductor y el análisis hecho de todos los factores, dígase normas, estándares, documentación sobre compiladores, lenguajes de programación, etcétera. Y su consiguiente estudio.

- **Dialéctico** para la identificación de posibles inconvenientes en el transcurso de la investigación así como de la posterior implementación y desarrollo del traductor.

Posible resultado:

- Traductor del lenguaje de Diagrama Escalera (LD) al lenguaje C.

El presente documento está estructurado en tres capítulos, a continuación se describe el contenido que se abordará en cada uno de ellos:

Capítulo 1: Abarca lo referente a la programación de los PLC mediante el lenguaje LD, sus características y generalidades. Se realiza un estudio de las normas para la programación de los PLC. Estudio de temas relacionados con los procesos de compilación y traducción. Además se caracterizan un conjunto de herramientas que realizan la traducción del lenguaje LD, y del estándar para el intercambio de información OpenPLC. Concluyendo con la definición de las herramientas y metodología de desarrollo a utilizar.

Capítulo 2: Se hace referencia a las fases principales que establece la metodología definida para el desarrollo del sistema. Se definen una serie de requisitos funcionales y no funcionales que debe cumplir dicho sistema, así como también se define una arquitectura que dará la línea base para su desarrollo. Por último se expondrá la propuesta de solución y un diseño de las clases a implementar mediante el uso de las tarjetas CRC.

Capítulo 3: Abarca lo referente a la fase de pruebas que se le realizarán al sistema desarrollado, serán definidas un conjunto de pruebas de aceptación. Con el objetivo de evaluar el desempeño de la aplicación en correspondencia con los requerimientos que fueron establecidos inicialmente. Además se brindará una información detallada de la respuesta del sistema en cada una de las pruebas que se le hagan.

1. Capítulo 1: Fundamentación Teórica

A continuación se dará un acercamiento a temas relacionados con la traducción y compilación de lenguajes de programación para PLC. Se hará referencia a la norma IEC 61131 en la que se establecen dichos lenguajes, igualmente se analizará el estándar para el intercambio de información PLCOpen. Se brindarán los principales elementos de los lenguajes diagrama de escalera (LD) y lenguaje C, analizándose también herramientas para realizar el proceso de compilación. Finalmente se definirán las herramientas de desarrollo, así como la metodología a utilizar.

1.1 Norma Internacional IEC 61131

Ante el acelerado desarrollo industrial y la amplia gama de fabricantes de autómatas[4] programables (en lo adelante AP) a nivel mundial, fue necesario elaborar un documento rector para la fabricación de estos dispositivos. Por lo que, en 1992, la Comisión Electrónica Internacional crea la norma internacional IEC 61131, la cual consta de ocho partes que hacen referencia a conceptos relacionados con el hardware, la programación y la comunicación con los PLC. La norma recoge todas las cuestiones vinculadas a la fabricación de los AP, brindando un mejor entendimiento entre los fabricantes y una mayor compatibilidad entre las herramientas de programación y configuración. [1]

En la investigación es necesario analizar la tercera parte del estándar, IEC 61131-3, la cual establece la sintaxis y semántica de cuatro lenguajes de programación para autómatas programables, los cuales son:

- ✓ Lista de Instrucciones (IL por sus siglas en inglés)
- ✓ Texto Estructurado (ST por sus siglas en inglés)
- ✓ Diagrama de Escalera (LD por sus siglas en inglés)
- ✓ Diagrama de Bloques Funcionales (FBD por sus siglas en inglés)

El estándar se divide en dos temas; el primer tema recoge los elementos comunes en la programación de los PLC y el segundo trata acerca de las particularidades de cada lenguaje. [1]

1.1.1 Generalidades de los lenguajes

En la Norma Internacional IEC 61131-3 se definen un grupo de elementos que son comunes para los lenguajes de programación de AP, tanto para los gráficos (Diagrama de Escalera y Diagrama

[4] Autómata: Dispositivo o conjunto de reglas que realizan un encadenamiento automático y continuo de operaciones capaces de procesar una información de entrada para producir otra salida.

de Bloques Funcionales) así como para los literales (Texto Estructurado y Lista de Instrucciones). A continuación se hará referencia a algunos de los elementos comunes más significativos.

Juego de caracteres: Se definen un conjunto de caracteres básicos basados en la norma ISO/CEI 646[5]. Los caracteres soportados son:

- ✓ Signo de número (#).
- ✓ Signo de dólar ($).
- ✓ Barra vertical (|) o signo de admiración (!).
- ✓ Corchetes izquierdo y derecho "[ ]", paréntesis izquierdo y derecho "( )". [1]

Identificadores: Un identificador es una cadena de letras, dígitos y caracteres de subrayado que debe comenzar por una letra o por un carácter de subrayado. No se permiten subrayados múltiples iniciales ni intercalados. Y soporta al menos un carácter de exclusividad. No debe tener caracteres de espacio intercalados. (Ver Ilustración 1) [1]

Características de identificadores

Nº	Descripción de la característica	Ejemplos
1	Mayúsculas y números	IW215 IW215Z QX75 IDENT
2	Mayúsculas y minúsculas, números, subrayados intercalados	Todos los anteriores y además: LIM_SW_5 LimSw5 abcd ab_Cd
3	Mayúsculas y minúsculas, números, subrayados iniciales o intercalados	Todos los anteriores y además: _MAIN _12V7

Ilustración 1. Ejemplos de Identificadores [1]

Representación de los datos: Consiste en literales numéricos, cadenas de caracteres y literales de tiempo. (Ver Ilustración 2).

[5] Estándar internacional para la codificación de caracteres, basada en el estándar estadounidense ASCII.

Literales numéricos

Nº	Descripción de la característica	Ejemplos
1	Literales enteros	-12 0 123_456 + 986
2	Literales reales	-12,0 0,0 0,4560 3.14159_26
3	Literales reales con exponentes	-1.34E-12 o -1.34e-12 -1.0E+ 6 o 1.0e+ 6 -1.234E6 o 1.234e6
4	Literales en base 2	2#1111_1111 (255 decimal) 2#1110_0000 (240 decimal)
5	Literales en base 8	8#377 (255 decimal) 8#340 (240 decimal)
6	Literales en base 16	16#FF o 16#ff (255 decimal) 16#E0 o 16#e0 (240 decimal)
7	Cero y uno booleanos	0 1
8	FALSE (falso) y TRUE (cierto) booleanos	FALSE TRUE

Ilustración 2. Ejemplos de literales numéricos [1]

Cadena de caracteres: Es una secuencia de uno o más caracteres precedidos y terminados por el carácter apóstrofe simple (‘). [1]

Literales de tiempo: Proporcionan una representación de dos tipos de datos relacionados con el tiempo: los datos de duración para medir o controlar el tiempo transcurrido de un acontecimiento de control y los datos de la hora del día, para sincronizar el comienzo o final de un acontecimiento de control. [1]

Tipos de datos: Define varios tipos de datos elementales con su correspondiente tamaño en bits. Entre los tipos comunes están: Booleano, Integer, Real, Byte, Word, Time_of_Day y String. Además el fabricante puede definir sus propios tipos de datos llamados “tipos de datos derivados”. [1]

Variables: Constituyen un medio de identificar los objetos de datos asociados a las entradas, salidas o a la memoria del AP. La visibilidad de las mismas esta normalmente limitada al unidad de organización en la cual fue declarada (esto sucede cuando son declaradas de forma local) si requieren ser usadas en otras partes del programa deberán ser declaradas como globales.

Su inicialización puede ser de tres formas diferentes, cuando un elemento de configuración se “arranca” cada una de las variables asociadas puede asumir los siguientes valores: [1]

- ✓ El valor que tenía la variable cuando se “detuvo” el elemento de configuración.
- ✓ Un valor inicial especificado por el usuario.

- ✓ El valor inicial por defecto correspondiente al tipo de datos asociados a la variable.

Unidades de organización del programa: Las unidades que definen son la función, el bloque funcional y el programa.

- ✓ **Función:** Se definen una serie de funciones estándar como SUB (resta), operaciones lógicas AND, OR y NOT, así como otras que pueden ser definidas por el programador, las cuales pueden ser reusadas [3].

- ✓ **Bloque Funcional:** Los bloques funcionales son los equivalentes de los circuitos integrados, que representan funciones de control especializadas. Contienen tanto datos como instrucciones, y además pueden guardar los valores de variables (siendo esta una de las diferencias con las funciones). Tienen una interfaz de entradas y salidas bien definidas y un código interno oculto, tal como un circuito integrado. Un lazo de control de temperatura es un ejemplo de un bloque funcional. Una vez definido, puede ser usado una y otra vez, en el mismo programa, en diferentes programas o en distintos proyectos. Esto lo hace altamente reutilizable. Los bloques funcionales pueden ser escrito por el usuario en cualquiera de los lenguajes de la norma IEC, pero también existen bloques de funciones estándar como los contadores y los temporizadores. Existe la posibilidad de ser llamados múltiples veces creando copias del bloque denominadas instancias. Cada instancia tendrá asociado un identificador con sus parámetros de entrada y variables de salida. [3]

- ✓ **Programa:** Según la norma IEC 61131-3 un programa es “conjunto lógico de todos los elementos y construcciones del lenguaje de programación que son necesarios para el tratamiento de señal previsto que se requiere para el control de una máquina o proceso mediante el sistema de AP”. Un programa puede contener aparte de las declaraciones de tipos de datos, variables y su código interno, distintas instancias de funciones y bloques funcionales. [3]

1.1.2 Lenguaje Diagrama de Escalera (LD)

Lenguaje basado en símbolos gráficos, organizados en redes semejantes a los escalones de un diagrama lógico de escalera de relés. Las redes están limitadas por el lado izquierdo y por el lado derecho por medio de barras de alimentación (Ver Ilustración 3).

Barras de alimentación

Nº	Símbolo/ejemplo	Descripción
1		Barra de alimentación izquierda (con enlace horizontal acoplado)
2		Barra de alimentación derecha (con enlace horizontal acoplado)

Ilustración 3. Barras de alimentación en LD [2]

El estado de la barra izquierda se considera en todo momento “on”, mientras que no se define ningún estado para la barra derecha. Los elementos pueden ser enlazados tanto verticales como horizontales. Un elemento de enlace horizontal estará representado por una línea horizontal desde la barra izquierda hasta el bloque funcional o el contacto de una bobina. El flujo de estado es desde el elemento inmediato a la izquierda hacia el elemento inmediato a la derecha.

El elemento de enlace vertical consistirá en una línea vertical que corte uno o más elementos de enlace horizontal a cada lado. El estado del enlace vertical representa el “or” inclusivo de los estados “on” de los estados horizontales situados a su izquierda, o sea que el estado del estado vertical del enlace será:

OFF, si los estados de todos los enlaces horizontales unidos a su izquierda son OFF.

ON si el estado de uno o más de los enlaces horizontales unidos a su izquierda es ON

El orden de ejecución de una red será de izquierda a derecha, mientras que dentro de una unidad de organización del programa será de arriba hacia abajo o respetando los elementos de control de ejecución definidos (Ver Ilustración 4). [2]

Ejemplo de código en LD

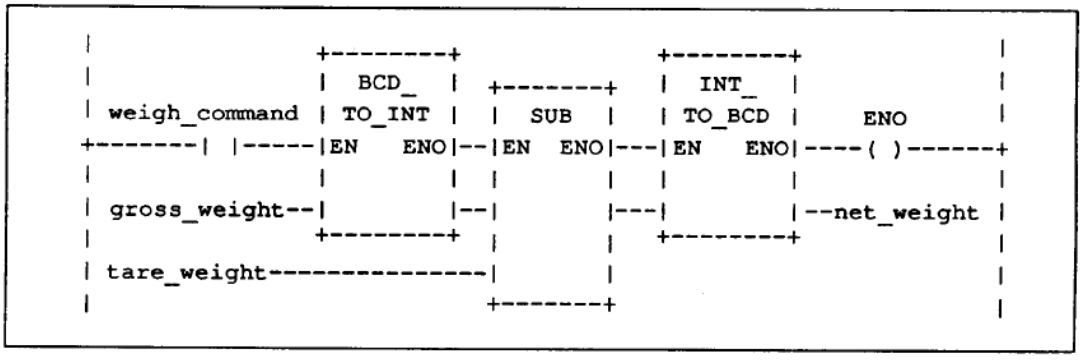

Ilustración 4. Código en lenguaje LD [2]

1.1.3 Lenguaje C/C++

Es un lenguaje orientado a la implementación de sistemas operativos. C es apreciado por la eficiencia del código que produce y es el lenguaje más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones. Se trata de un lenguaje de tipos de datos estáticos, débilmente tipificado, de *medio nivel*, pero con muchas características de *bajo nivel.* Dispone de las estructuras típicas de los lenguajes de *alto nivel* pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy *bajo nivel* (Ver Ilustración 5). [2]

```
function()
{
  bool weight_conmand;
  string gross_weight;
  int tare_weight;
  if(weight_conmand)
  {
      weigh = INT_TO_BCD(BCD_TO_INT(gross_weight)-tare_weight);
  }
}
```

Ilustración 5. Código en lenguaje C

Entre las principales características que presenta están:

- ✓ Un núcleo de lenguaje simple, con funcionalidades añadidas importantes, como definiciones matemáticas y de manejo de archivos, proporcionadas por bibliotecas.
- ✓ Es un lenguaje muy flexible que permite programar con múltiples estilos.
- ✓ Un sistema de tipos que impide operaciones sin sentido.
- ✓ Usa un lenguaje de pre procesado, el preprocesador de C, para tareas como definir macros e incluir múltiples archivos de código fuente.
- ✓ Acceso a memoria de bajo nivel mediante el uso de punteros.
- ✓ Interrupciones al procesador con uniones.
- ✓ Un conjunto reducido de palabras clave.
- ✓ Por defecto el paso de parámetros a una función se realiza por valor. El paso por referencia se consigue pasando explícitamente a las funciones las direcciones de memoria de dichos parámetros.
- ✓ Punteros a funciones y variables estáticas, que permiten una forma rudimentaria de encapsulado y polimorfismo.

- ✓ Tipos de datos agregados que permiten que datos relacionados se combinen y se manipulen como un todo.

1.2 Estándar PLCOpen

Desde el surgimiento de la Norma Internacional IEC 61131-3 que define los lenguajes para la programación de los PLC, los desarrolladores tienen la necesidad de intercambiar sus programas, bibliotecas y proyectos entre los diferentes entornos de desarrollo, así como lograr una mayor persistencia de la información. Es con este fin que la organización PLCOpen crea el grupo de trabajo TC6 para XML definiéndose de esta forma el estándar PLCOpen que define una interfaz abierta que soporta diferentes tipos de herramientas de software[6] y proporciona la capacidad de transferir la información entre los lenguajes establecidos en la norma, basándose para la transferencia en el lenguaje Extended Markup Language (XML). [4]

Definiciones: El estándar establece cuatro definiciones principales para estructurar la información.

- ✓ **Proyecto:** Un proyecto consta de las bibliotecas y configuraciones. Contiene una parte tipo y una parte instancia.
- ✓ **Biblioteca:** Una biblioteca es una colección de tipos de datos y tipos de punto de uso.
- ✓ **Elemento:** Entidades en sí que poseen contenido.
- ✓ **Objeto:** Un elemento que representa un objeto de un proyecto PLC. Todos los objetos tienen un número de identificación local para la generación del sistema. Se denomina localID y debe ser único dentro de una unidad de organización de programa (en lo adelante POU). [4]

Características

- ✓ Los documentos poseen una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Los elementos con contenido deben estar cerrados.
- ✓ Los documentos XML solo permiten un elemento raíz del que todos los demás sean parte, es decir, solo pueden tener un elemento inicial.
- ✓ Los valores de los atributos siempre deben estar encerrados entre comillas simples o

[6] Software: Equipamiento lógico o soporte lógico de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas.

dobles.

- ✓ Es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea).
- ✓ Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. Los nombres poseen características comunes.
- ✓ Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; representan partes del documento que el procesador debe entender. El resto del documento entre marcas son los datos “entendibles” por los usuarios (Ver Ilustración 6). [4]

```
    </connectionPointIn>
  </rightPowerRail>
- <block localId="3" width="120" height="60" typeName="BCD_TO_UINT">
    <position x="208" y="105" />
  - <inputVariables>
    - <variable formalParameter="EN">
      - <connectionPointIn>
          <relPosition x="0" y="30" />
        - <connection refLocalId="4">
            <position x="208" y="135" />
            <position x="175" y="135" />
          </connection>
        </connectionPointIn>
      </variable>
    - <variable formalParameter="IN">
      - <connectionPointIn>
          <relPosition x="0" y="50" />
```

Ilustración 6. Ejemplo de código XML [4]

1.2.1 Lenguaje Extended Markup Language (XML)

XML es un lenguaje etiquetado derivado de SGML. Se puede decir que es un metalenguaje, o sea un lenguaje para definir lenguajes. Proporciona un modo consistente y preciso de aplicar etiquetas para describir las partes que componen un documento, permitiendo el intercambio de información entre diferentes plataformas. [5]

1.3 Traductores y Compiladores

De la misma forma que un lenguaje natural (dígase inglés, español y francés), los lenguajes de computación definen una forma de ordenar palabras para construir oraciones que comunican información. Mientras que un lenguaje natural comunica sentimientos, hechos, preguntas, etc. Un lenguaje artificial se restringe a comandos que son seguidos por una máquina sin cuestionar. Cuando una persona que desconoce el francés con una que desconoce el español, se requieren los servicios de un traductor. Si se le proporciona una frase gramaticalmente incorrecta, el traductor explicará que no hay traducción para una frase sin sentido. Un compilador sustituye a un

traductor humano en los lenguajes artificiales y aplica ciertas reglas a las frases a traducir de forma que si tiene sentido entrega como resultado la frase equivalente en otro idioma. [6]

1.3.1 Traductores de lenguaje

Un traductor es un programa que recibe como entrada código[7] escrito en cierto lenguaje (lenguaje fuente) y produce como salida código en otro lenguaje (lenguaje objeto). Generalmente el lenguaje de entrada es de más alto nivel que el de salida. [6]

1.3.2 Compilador

Programa informático[8] que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, o también, genera aplicaciones que sean directamente utilizables en un ordenador o computadora. Un compilador lee el código fuente creado en un determinado lenguaje de programación, lo interpreta, comprueba su sintaxis y traduce a lenguaje o código máquina toda la serie de instrucciones, generando el archivo ejecutable final (7).

1.3.3 Estructura de un compilador

El trabajo de un compilador es tomar la cadena fuente del programa, determinar si es válida desde el punto de vista sintáctico y generar un programa equivalente en un lenguaje que la computadora entienda. Esto se puede dividir en partes, o fases del proceso de compilación (Ver Ilustración 7). El orden de los procesos puede variar en un compilador. En traductores reales el proceso no ocurre de forma lineal, normalmente el centro del proceso de compilación gira alrededor del Analizador Sintáctico el cual es el encargado de activar cada una de las restantes fases según sea necesario. [8]

[7] Código: Conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa.
[8] Programa informático: Es un conjunto de instrucciones que una vez ejecutadas realizaran una o varias tareas en una computadora.

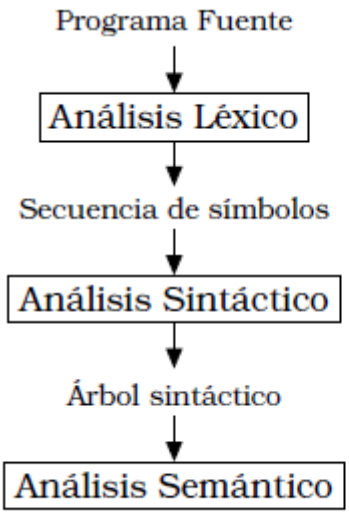

Ilustración 7. Procesos de traducción [9]

Analizador léxico

La función del analizador léxico es tomar cada uno de los símbolos del código fuente y agruparlos en entidades sintácticas simples o elementales denominadas tokens. [10]

Analizador sintáctico

Es el proceso en el cual se examina una secuencia de tokens para determinar si el orden de esa secuencia es correcto de acuerdo a ciertas convenciones estructurales de la definición sintáctica del lenguaje. Se genera el árbol de sintaxis abstracta, que no es más que una representación intermedia de los elementos significativos así como su relación sintáctica (10).

Analizador semántico

Recibe la información resultado del análisis sintáctico y detecta errores relacionados con la validez del programa desde el punto de vista de la semántica. [10]

1.4 Compiladores de lenguajes para Controladores Lógicos Programables

Existen numerosas herramientas que realizan las fases de compilación para los lenguajes establecidos en la norma IEC 61131-3. A continuación se analizarán un conjunto de compiladores que realizan dicha tarea.

1.4.1 Compilador STL para el PLC-UD

Es un compilador encargado de ejecutar código ejecutable a partir de rutinas de control implementadas en lenguaje Texto Estructurado para el PLC-UD. Basa su principio de funcionamiento en recibir un archivo de entrada llamado código fuente en lenguaje Texto

Estructurado y generar un archivo de salida denominado código objeto o código máquina. El proceso de compilación está estructurado en varias partes funcionales (Ver ilustración 8):

- ✓ **Análisis léxico.**
- ✓ **Análisis sintáctico.**
- ✓ **Análisis semántico.**
- ✓ **Generar código.**
- ✓ **Manejador de Errores.**

Estos procesos se dividen en dos partes fundamentales, primeramente se realizan los análisis léxico, sintáctico y semántico. Como resultado de esta primera fase se genera una representación intermedia que será utilizada en una segunda fase donde se genera el código objeto.

Para la implementación del compilador se utilizó el lenguaje C++ y la herramienta de desarrollo BuilderC++ 6.0. [11]

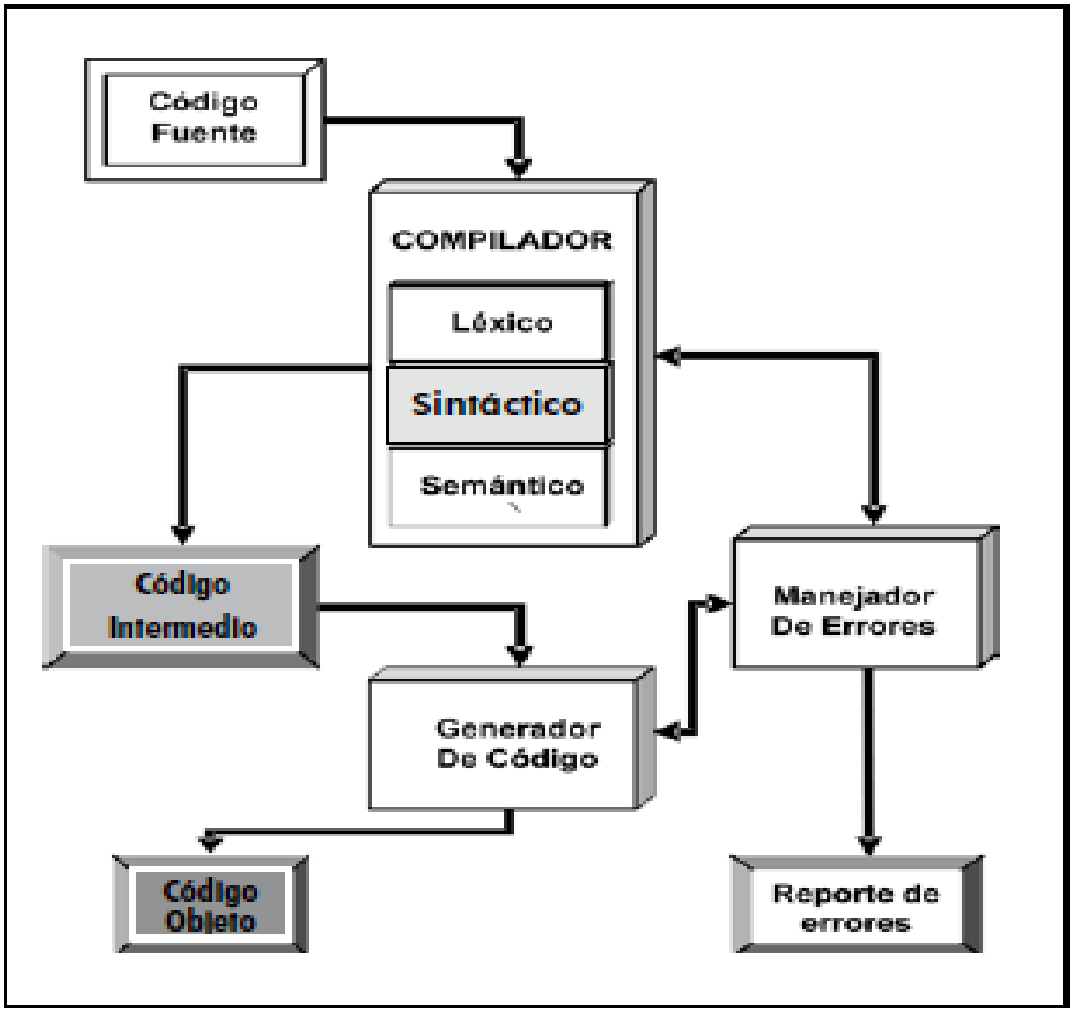

Ilustración 8. Procesos de compilación PLC-UD

1.4.2 Laboratorio de Emulación Remota de PLC

Es una herramienta de Emulación Remota de PLC, basada en una arquitectura Cliente-Servidor. Donde el servidor es el encargado de emular el funcionamiento de un PLC y los procesos de compilación. El mismo soporta todos los lenguajes establecidos en la norma IEC 61131-3. El emulador recibe los programas generados en la aplicación cliente compilándolos y ejecutándolos. El emulador se puede dividir en 4 bloques, el primero denominado bloque traductor se encarga de realizar un proceso de traducción a lenguaje Java, revisando si poseen errores. Si no contiene errores se procede al segundo bloque, donde toma los ficheros generados en Java y los compila utilizando el JavaCompiler del JDK de Java. El tercer módulo se encarga se encargar de ejecutar los programas generados. La cuarta fase es la encargada de generar los errores ocurridos. (Ver Ilustración 9) [12]

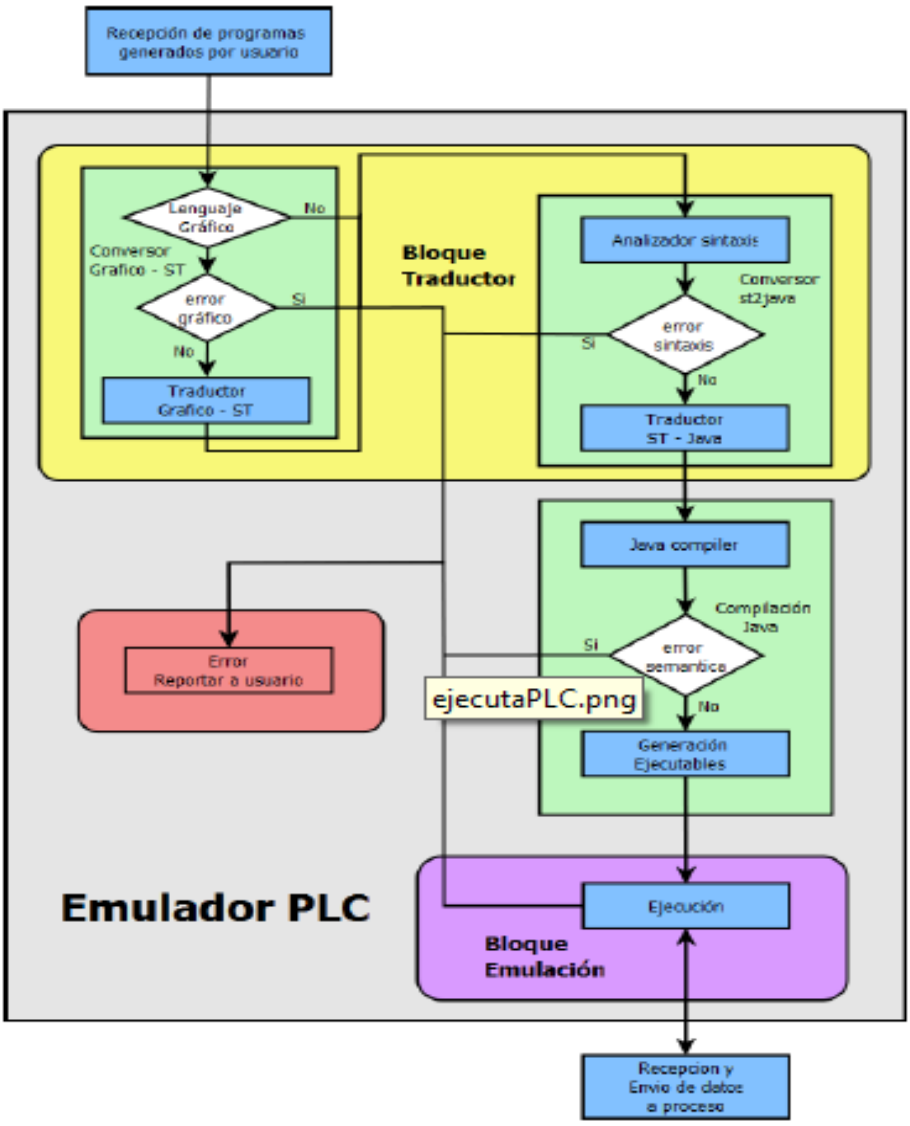

Ilustración 9. Fases del Emulador.

Específicamente en el bloque de traducción los lenguajes gráficos son llevados a lenguajes textuales, concretamente el lenguaje Texto Estructurado, el cual posteriormente es traducido al lenguaje Java. La implementación de esta fase se hizo en dos bloques. (Ver ilustración 10)

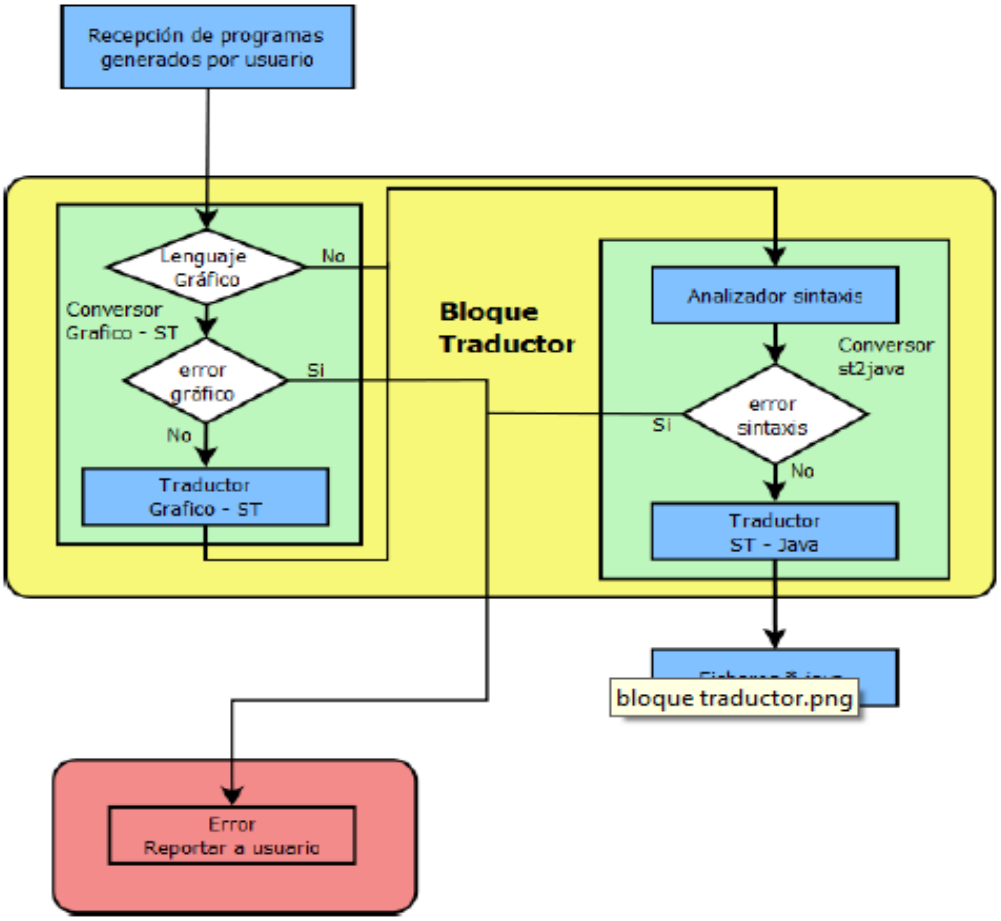

Ilustración 10. Flujo de Bloque Traducción.

1.4.3 Compilador para el PLC IDE4PLC

Es un compilador para los lenguajes de programación definidos en el estándar IEC 61131-3, el cual se encarga de generar archivos en lenguaje C y Lista de Instrucciones, con la implementación de las rutinas de control y los elementos de configuración, para su posterior compilación a código ejecutable. Para la compilación de los archivos generados utiliza un programa compilador de C desarrollado por terceros, a través de comandos enviados al Sistema Operativo. A continuación se muestra el modelo conceptual de dicho compilador. (Ver ilustración 11)

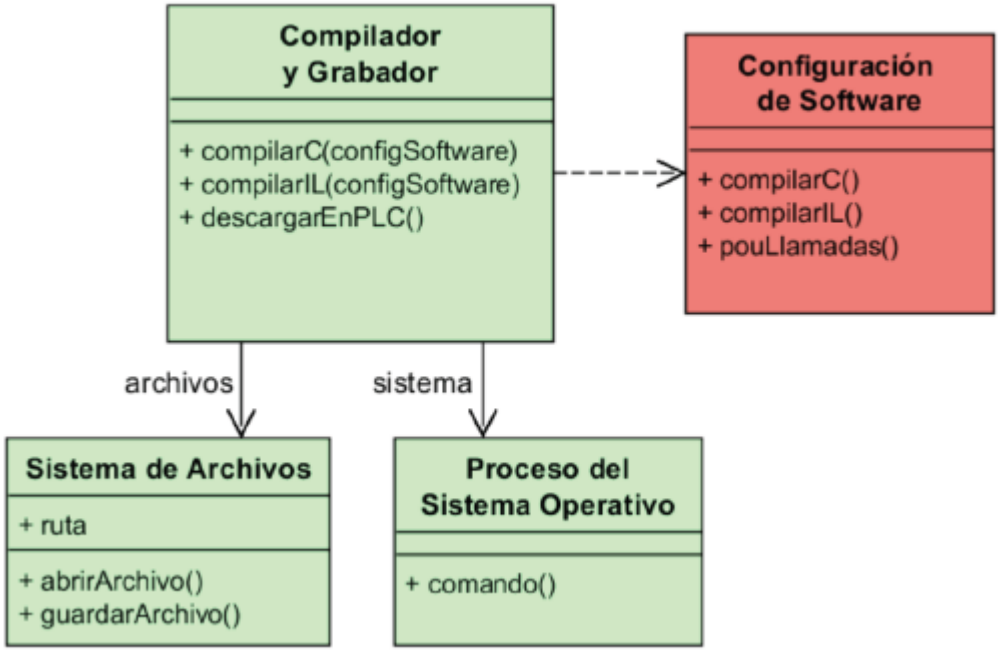

Ilustración 11. Modelo Conceptual del compilador para el ID4PLC

1.4.4 Beremiz

Beremiz es una herramienta para el desarrollo de aplicaciones para la automatización de procesos industriales. Combina los estándares IEC 61131, PLCOpen, CanOpen y SVGUI, mediante los cuales crea aplicaciones. Con PLCOpen es capaz de importar y exportar sus fuentes en XML. Esta aplicación dispone de una interfaz gráfica para el desarrollo de los sistemas. Se distribuye bajo la licencia GPL y existen versiones tanto para Linux como para Windows, aunque es recomendable utilizarlo en sistemas Linux. Su interfaz se muestra en la Ilustración 12. [13]

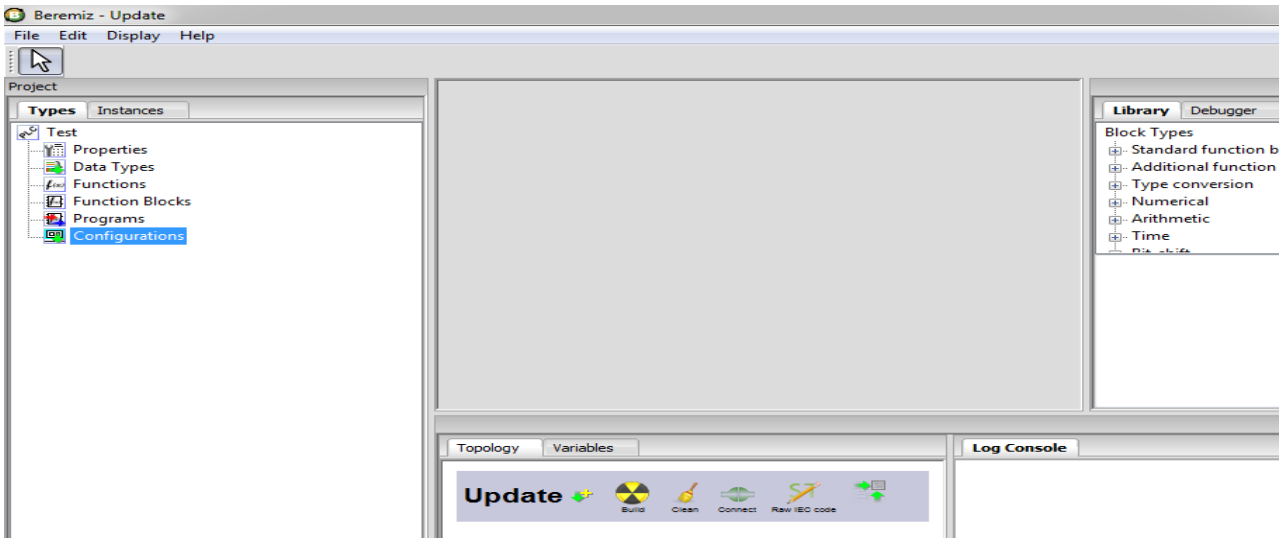

Ilustración 12. Interfaz de Beremiz [14]

1.4.5 Matiec

Es un proyecto que tiene como objetivo producir un compilador de código abierto para los lenguajes de programación definidos en la norma IEC 61131-3. El proyecto Matiec genera dos compiladores, ambos aceptan la misma entrada que es un archivo de texto con los lenguajes Texto Estructurado y Lista de Instrucciones. [15]

Se basa en 6 fases para realizar el proceso de compilación.

1. **Analizador Léxico**, implementado con la biblioteca Flex, donde el archivo que se genera se denomina iec_flex.ll.
2. **Analizador Sintáctico**, implementado con la biblioteca Bison, donde el archivo que se genera se denomina iec_bison.yy.
3. **Tabla de símbolos**, donde se generan todos los símbolos del árbol de sintaxis abstracta.
4. **Analizador Semántico**, donde se realiza el chequeo de tipos.
5. **Generación de código ANSI C**.
6. **Generación de código binario integrado con el compilador GCC.**

1.4.6 Observaciones sobre las herramientas analizadas

En total se analizaron cuatro herramientas de programación de PLC de distintos fabricantes. A continuación la Tabla 1 muestra una serie de características observadas en cada una:

Tabla 1. Resumen de observaciones

Herramienta	Lenguajes que soporta	Proceso de compilación	Propósito	Software libre
Compilador STL para el PLC-UD	ST	ST =>Arquitectura del microprocesador del PLC-UD	Industrial	No
Laboratorio de Emulación Remota de PLC	IL, ST, FBD, LD	FBD,LD => ST => Java => Arquitectura del microprocesador del PLC	Educativo	No
Compilador	IL, ST, FBD,	ST, FBD, LD => IL =>	Industrial	Si

para el PLC IDE4PLC	LD	C=> Arquitectura del microprocesador del PLC		
Matiec	IL, ST, FBD, LD	FBD, LD => ST => C => Arquitectura del microprocesador del PLC	Industrial, Educativo	Si

El análisis de la tabla anterior muestra que el proceso de compilación por lo general se realiza desde un lenguaje textual hasta la arquitectura del microprocesador del PLC, dado que estos lenguajes son compatibles entre sí, resulta conveniente traducir desde los lenguajes gráficos hasta uno de los textuales. Usualmente los microprocesadores cuentan con un toolchain para C hasta su arquitectura, por lo que se puede realizar el proceso de traducción hasta C y luego compilar específicamente para la arquitectura haciendo uso del toolchain antes mencionado. La naturaleza propietaria de las dos primeras herramientas analizadas hace que no sea posible considerarlas como posibles soluciones. Así también, en el caso de las herramientas de código abierto, el proyecto Matiec no se utiliza porque a pesar de ser una herramienta de carácter multiplataforma, en el sistema operativo Linux su funcionamiento no es correcto. Así mismo el compilador para el PLC IDE4PLC, pese a que cuenta con características muy similares al objetivo propuesto no constituyó una alternativa ya que es un proyecto que aún se encuentra en desarrollo. El estudio de todas estas herramientas contribuyó a la definición del flujo a seguir para lograr compilar rutinas de control desde los lenguajes gráficos hasta la arquitectura del PLC Arex.

1.5 Herramientas y metodología a utilizar

En la presente sección se exponen las herramientas y tecnologías que serán utilizadas para dar cumplimiento al objetivo general de la investigación. La elección de las mismas se basa en el hecho de ser pilares en el movimiento de Software Libre, además de poseer características que se ajustan a las necesidades del presente trabajo de diploma. Por otra parte son ampliamente utilizadas en la línea Sistemas Embebidos del centro CEDIN.

1.5.5 Entorno Integrado de desarrollo: QtCreator

El entorno de desarrollo a utilizar es QtCreator. Es un entorno potente, con una amplia gama de funcionalidades que facilitan su uso. Utiliza el lenguaje de programación orientado a objetos C++. Se basa en Qt una biblioteca multiplataforma y gratuita versión 5.2.0. La versión a utilizar será QtCreator 3.0.0. [16]

Principales características:

- ✓ Es un editor avanzado de C++, para escribir, editar y navegar por el código fuente sin utilizar el ratón (mouse).
- ✓ Interfaz gráfica para la depuración, que incrementa la percepción de la estructura de clases de Qt.
- ✓ Integración con QtDesigner para editar los archivos de interfaz gráfica de usuario.
- ✓ La herramienta localizador (Locator) para la navegación rápida por archivos de proyecto, funciones, clases e informaciones de ayuda.
- ✓ Integración con QtHelp, facilitando el acceso a la ayuda de la API Qt (tipos de datos, funciones) de Qt a través de una ayuda sensible al contexto.

1.5.6 Metodología eXtreme Programming

Para el desarrollo se utilizará la metodología Programación Extrema (en lo adelante eXtreme Programming o XP), es una metodología de procesos ágiles para el desarrollo de software. Es una de las metodologías más exitosas debido a que se centra en potenciar las relaciones entre los desarrolladores para lograr el éxito, promoviendo el trabajo en equipo, la comunicación con el cliente, lo cual propicia un buen entorno de trabajo. Algunas de las características que la distinguen son:

- ✓ La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
- ✓ Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
- ✓ Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
- ✓ Las personas del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
- ✓ Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
- ✓ El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
- ✓ El software que funciona es la medida principal de progreso.

- ✓ Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
- ✓ La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
- ✓ La simplicidad es esencial.
- ✓ Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
- ✓ En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento. [17]

1.5.7 Lenguaje de programación C++

El lenguaje utilizado será C++. Es el más popular en la implementación de compiladores. Es una extensión del lenguaje C, con mecanismos que permiten la manipulación de objetos. Contiene los paradigmas de programación estructurada y orientada a objetos, por lo que se suele decir que es un lenguaje multiparadigma.

Entre las principales ventajas de C++ como lenguaje de programación orientado a objetos se encuentran:

- ✓ C++ es un lenguaje de propósito general, o sea, que con él se puede programar cualquier cosa, desde sistemas operativos y compiladores hasta aplicaciones de bases de datos y procesadores de texto.
- ✓ Los programas escritos en C++ tienen la ventaja de que podrán ejecutarse en cualquier máquina y bajo cualquier sistema operativo.
- ✓ Es un lenguaje que permite programar en alto y bajo nivel.
- ✓ Es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. [18]

1.5.8 Lenguaje de modelado UML

Es un Lenguaje Unificado de Modelado (por sus siglas en inglés UML), “es un lenguaje estándar que permite escribir planos de software. Puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software”. [19]

UML permite la modelación de sistemas de software con tecnología orientada a objetos, siendo uno de los más conocidos y utilizados en la actualidad. Define los sistemas por cada una de las etapas por la que debe pasar, indica que es lo que supuestamente hará, pero no como lo hará.

Incluye aspectos conceptuales como procesos de negocio, funciones del sistema y aspectos concretos como expresiones de lenguaje de programación, esquemas de base datos y componentes reutilizables, permitiendo una mayor comprensión del sistema. [20]

Se utilizara UML 2.0, el cual define trece tipos de diagramas, divididos en tres categorías.

- ✓ Seis representan estructuras estáticas de la aplicación.
- ✓ Cuatro representan diferentes aspectos de interacción.
- ✓ Tres representan tipos generales de comportamiento.

1.5.9 Herramienta de modelado: Visual Paradigm

Las herramientas de Ingeniería de Software Asistida por Ordenador (por su siglas en inglés CASE) “son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas contribuyen de manera directa en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras”. [21]

Visual Paradigm puede manejar gran parte de los diagramas definidos en UML, ya sea cerrándolos manualmente importándolos a partir de código en C++, Java, Pascal entre otros, creando posibilidades de ingeniería inversa. Permite crear un modelo y automáticamente generar el código en varios lenguajes para ayudar a comenzar la implementación del proyecto.

Para el modelado de los artefactos y diagramas generados a lo largo del ciclo de vida del trabajo se decidió emplear Visual Paradigm en su versión 3.4, pues su uso está muy estandarizado a nivel mundial y constituye una herramienta multiplataforma muy madura y acabada.

Conclusiones parciales

Durante el desarrollo de este capítulo se analizaron los lenguajes de programación para PLC, haciendo mayor énfasis en el lenguaje Diagrama de Escalera, además se caracterizó el lenguaje C pues constituye una parte fundamental en el posterior desarrollo de la aplicación. También se identificaron conceptos relacionados con la traducción y compilación de lenguajes de programación y herramientas desarrolladas que realizan la compilación de los lenguajes para PLC. Se examinaron los métodos de transferencia de información entre diferentes plataformas y se definieron las herramientas y metodología para el posterior desarrollo del traductor. Estableciendo como entorno de desarrollo QtCreator, lenguaje de modelado UML en su versión 2.0 y por último la metodología a usar será eXtreming Programming.

2. Capítulo 2: Planificación y diseño

En el presente capítulo se puntualizarán cada uno de los aspectos fundamentales dentro de la planificación y el diseño del “Compilador para el lenguaje Diagrama de Escalera”. Presentando los aspectos relacionados a las actividades que propone la metodología XP, durante sus fases de planificación y diseño.

2.1 Fase de exploración

La metodología de desarrollo eXtreme Programming define la fase de exploración para su comienzo. Durante la misma se realiza el proceso de identificación de las historias de usuarios y características no funcionales del sistema, así como la familiarización de los equipos de trabajo con las tecnologías y herramientas seleccionadas para la construcción del proyecto.

2.1.1 Historias de usuario

La técnica utilizada por la metodología eXtreme Programming para especificar los requisitos del software son las historias de usuarios. No son más que tarjetas donde el cliente expone de manera breve las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. Esto hace más flexible y dinámico su tratamiento, pues estas pueden ser en cualquier momento reemplazadas por otras más generales o específicas, ser modificadas o añadir nuevas historias de usuario a las ya existentes. Cada una de estas historias de usuario son lo suficientemente comprensibles para que el programador pueda implementarla en unas pocas semanas. [17]

A continuación se definen un conjunto de historias de usuario que responden a las funcionalidades que el sistema debe contener.

Tabla 2. Historia de Usuario 1

Historia de Usuario	
Número: 1	**Usuario:** Línea Sistemas Embebidos
Nombre de la historia: Cargar y validar archivo XML.	
Prioridad en el negocio: Alta	**Riesgo en el desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: Alex Fernández Castrillo	

Descripción: Se llevará a cabo cuando se ejecute la aplicación, permitiendo cargar el archivo XML que contiene la información del código en lenguaje Diagrama de Escalera. Permitirá hacer la validación del documento desde el punto de vista del lenguaje XML.

Tabla 3. Historia de Usuario 2

Historia de Usuario	
Número: 2	**Usuario:** Línea Sistemas Embebidos
Nombre de la historia: Realizar análisis léxico al código del lenguaje Diagrama de Escalera en el archivo XML.	
Prioridad en el negocio: Alta	**Riesgo en el desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 2
Programador responsable: Alex Fernández Castrillo	
Descripción: Después de comprobar que la información del código en Diagrama de Escalera es correcta, realizar el análisis léxico, para identificar del código XML, la información referente a la programación salvada en el mismo y determinar si existe etiquetas sin contenido Si se detectan errores los mismos deben ser mostrados por la consola.	

Tabla 4. Historia de Usuario 3

Historia de Usuario	
Número: 3	**Usuario:** Línea Sistemas Embebidos
Nombre de la historia: Realizar análisis sintáctico al código del lenguaje Diagrama de Escalera.	
Prioridad en el negocio: Alta	**Riesgo en el desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 2
Programador responsable: Alex Fernández Castrillo	

Descripción: Después de realizar el análisis léxico si no existieron errores, se lleva a cabo el análisis sintáctico. Con el objetivo de construir las estructuras de datos necesarias para la posterior generación de código C/C++.

Tabla 5. Historia de Usuario 4

Historia de Usuario	
Número: 4	**Usuario:** Línea Sistemas Embebidos
Nombre de la historia: Generar código C/C++.	
Prioridad en el negocio: Alta	**Riesgo en el desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 3
Programador responsable: Alex Fernández Castrillo	
Descripción: Después de realizar el análisis léxico y sintáctico, si no existieron errores de ningún tipo, se generará un archivo con la traducción de las sentencias del lenguaje Diagrama de Escalera al lenguaje C/C++, que va a ser una representación intermedia para el posterior análisis semántico y generación de código objeto.	

Tabla 6. Historia de Usuario 5

Historia de Usuario	
Número: 5	**Usuario:** Línea Sistemas Embebidos
Nombre de la historia: Realizar análisis semántico a las sentencias en C/C++.	
Prioridad en el negocio: Alta	**Riesgo en el desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 3
Programador responsable: Alex Fernández Castrillo	
Descripción: Se realizará el análisis semántico, con el objetivo de identificar posibles errores en el código traducido a C/C++, los cuales deben ser mostrados por la consola.	

Tabla 7. Historia de Usuario 6

Historia de Usuario

Número: 6	**Usuario:** Línea Sistemas Embebidos
Nombre de la historia: Requisitos de software.	
Descripción: Debe estar disponible para las distribuciones del sistema operativo GNU/Linux.	

Tabla 8. Historia de Usuario 7

Historia de Usuario	
Número: 7	**Usuario:** Línea Sistemas Embebidos
Nombre de la historia: Requisitos de hardware.	
Descripción: Para lograr un correcto funcionamiento del programa se debe disponer de un ordenador que tenga como mínimo las prestaciones siguientes: 1) Procesador 400 MHz. 2) Memoria RAM 128 MB. **3)** Capacidad de almacenamiento 25 MB.	

Tabla 9. Historia de Usuario 8

Historia de Usuario	
Número: 8	**Usuario:** Línea Sistemas Embebidos
Nombre de la historia: Restricciones en el diseño y la implementación.	
Descripción: Para el desarrollo de la solución se definen una serie de restricciones tales como: 1) Lenguaje de programación C++. 2) Framework Qt 5.2.0. **3)** Entorno de desarrollo integrado QtCreator versión 3.0.0.	

2.2 Fase de planificación

Durante esta fase se lleva a cabo la estimación del esfuerzo que costará implementar cada historia de usuario, con una estimación de esfuerzo para cada funcionalidad, además se define el plan de iteraciones que explica que implementar en cada iteración hasta obtener la versión final del producto.

2.2.1 Estimación de esfuerzo por historia de usuario

La siguiente tabla muestra la estimación de esfuerzo por historia de usuario, según el orden a realizar. Las estimaciones incluyen el esfuerzo asociado a la historia de usuario. Esta expresada usando una medida de puntación directamente proporcional al esfuerzo. Un punto va a significar una semana en la que se trabajó sin ningún tipo de interrupción.

Tabla 10. Estimación de esfuerzo por HU

Historia de Usuario	**Puntos estimados**
Cargar y validar archivo XML.	3
Realizar análisis léxico al código del lenguaje Diagrama de Escalera en el archivo XML.	3
Realizar análisis sintáctico al código del lenguaje Diagrama de Escalera.	3
Generar código intermedio.	3
Realizar análisis semántico.	3

2.2.2 Tareas de ingeniería

Las historias de usuario están asociadas a un conjunto de tareas de ingeniería orientadas al desarrollo, que permiten la realización de la funcionalidad. En el siguiente apartado se recogen las tareas de ingeniería para cada historia de usuario identificada.

Tabla 11. Tarea de Ingeniería 1

Tarea	
Número de tarea: 1	**Numero de historia:** 1
Nombre de la tarea: Implementar en el archivo main.cpp el código para cargar el documento XML.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1/5
Fecha inicio: 13/01/2014	**Fecha fin:** 13/01/2014
Programador responsable: Alex Fernández Castrillo	

Descripción: El archivo main.cpp es donde se inicia la aplicación de consola, aquí se implementará el código para cargar el archivo XML a traducir. Así como permitirá validar que la dirección entrada por consola es válida y que el archivo especificado existe.

Tabla 12. Tarea de Ingeniería 2

Tarea	
Número de tarea: 2	**Numero de historia:** 1
Nombre de la tarea: Estudio del módulo QtXml.	
Tipo de tarea: Investigación	**Puntos estimados:** 3/5
Fecha inicio: 14/01/2014	**Fecha fin:** 18/01/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: El módulo de QtXml proporciona un lector de flujo, escritor de documentos y un Parser de documentos XML, además de implementaciones en C++ de DOM.	

Tabla 13. Tarea de Ingeniería 3

Tarea	
Número de tarea: 3	**Numero de historia:** 1
Nombre de la tarea: Diseñar e implementar la clase VerificarXML.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2/5
Fecha inicio: 19/01/2014	**Fecha fin:** 22/01/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: Implementación de la clase VerificarXML, la cual va permitir realizar una validación en busca de errores en la conformación del archivo de entrada.	

Tabla 14. Tarea de Ingeniería 4

Tarea	
Número de tarea: 4	**Numero de historia:** 1

Nombre de la tarea: Declaración de la interfaz CheckLD.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2/5
Fecha inicio: 23/01/2014	**Fecha fin:** 23/01/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: Declaración de la interfaz CheckLD, que permitirá la validación del fragmento de código salvado en el lenguaje Diagrama de Escalera, contenido en el archivo XML.	

Tabla 15. Tarea de Ingeniería 5

Tarea	
Número de tarea: 5	**Numero de historia:** 2
Nombre de la tarea: Estudio de la clase QXmlStreamReader.	
Tipo de tarea: Investigación	**Puntos estimados:** 3/5
Fecha inicio: 24/01/2014	**Fecha fin:** 31/01/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: La clase QXmlStreamReader proporciona una manera de manejar archivos XML, permitiendo moverse por todo el documento y extraer su contenido de manera rápida y sencilla.	

Tabla 16. Tarea de Ingeniería 6

Tarea	
Número de tarea: 6	**Numero de historia:** 2
Nombre de la tarea: Diseño e implementación de la clase AnalizadorLexico.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3/5
Fecha inicio: 1/02/2014	**Fecha fin**: 10/02/2014
Programador responsable: Alex Fernández Castrillo	

Descripción: Implementación de la clase Analizador Léxico con el objetivo de identificar los tokens del lenguaje en el archivo XML, haciendo uso para esto de la clase QXmlStreamReader.

Tabla 17. Tarea de Ingeniería 7

Tarea	
Número de tarea: 7	**Numero de historia:** 2
Nombre de la tarea: Diseño e implementación de la clase Token.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3/5
Fecha inicio: 11/02/2014	**Fecha fin:** 12/02/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: Implementación de la clase Token, la cual permitirá la representación de la información de los tokens identificados para el lenguaje Diagrama de Escalera.	

Tabla 18. Tarea de Ingeniería 8

Tarea	
Número de tarea: 8	**Numero de historia:** 2
Nombre de la tarea: Diseño e implementación de la clase TablaSimbolos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3/5
Fecha inicio: 13/02/2014	**Fecha fin:** 19/02/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: Implementación de la clase TablaSimbolos, la cual permitirá realizar una representación de algunas particularidades de los tokens creados que se identifiquen en la información del archivo XML.	

Tabla 19. Tarea de Ingeniería 9

Tarea	
Número de tarea: 9	**Numero de historia:** 3

Nombre de la tarea: Diseño e implementación de la clase Conexion.	
Tipo de tarea: Investigación	**Puntos estimados:** 3/5
Fecha inicio: 20/02/2014	**Fecha fin:** 25/02/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: Implementación de la clase Conexion para generar una representación intermedia de la relación entre las diferentes variables y bloques de funciones conectadas entre sí en el código contenido en el archivo XML.	

Tabla 20. Tarea de Ingeniería 10

Tarea	
Número de tarea: 10	**Numero de historia:** 3
Nombre de la tarea: Diseño e implementación de la clase AnalizadorSintactico.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3/5
Fecha inicio: 26/02/2014	**Fecha fin:** 10/03/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: Implementación de la clase AnalizadorSintactico, la que permitirá realizar una representación intermedia de las estructuras contenidas en el lenguaje LD.	

Tabla 21. Tarea de Ingeniería 11

Tarea	
Número de tarea: 11	**Numero de historia:** 4
Nombre de la tarea: Diseño e implementación de la clase POU.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3/5
Fecha inicio: 16/03/2014	**Fecha fin:** 21/03/2014
Programador responsable: Alex Fernández Castrillo	

Descripción: Implementación de la clase POU, con el objetivo de generar una representación intermedia de las estructuras reconocidas para su uso en fases posteriores.

Tabla 22. Tarea de Ingeniería 12

Tarea	
Número de tarea: 12	**Numero de historia:** 5
Nombre de la tarea: Diseño e implementación de la clase GenerarCodigo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3/5
Fecha inicio: 22/03/2014	**Fecha fin:** 4/04/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: Implementación de la clase GenerarCodigo, mediante la cual se realizará la traducción de las sentencias de código identificadas en el lenguaje Diagrama de Escalera.	

Tabla 23. Tarea de Ingeniería 13

Tarea	
Número de tarea: 13	**Numero de historia:** 6
Nombre de la tarea: Diseño e implementación de la clase AnalisisSemantico.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3/5
Fecha inicio: 4/04/2014	**Fecha fin:** 6/04/2014
Programador responsable: Alex Fernández Castrillo	
Descripción: Implementación de la clase AnalisisSemantico mediante la cual se podrá comprobar la validez semántica de las operaciones y expresiones traducidas a C/C++.	

2.3 Plan de iteraciones

Iteración 1

Esta iteración tiene como objetivo implementar algunas tareas de las historias de usuario que permiten la validación del código en lenguaje XML, contenido en el archivo cargado. Obteniéndose

una primera versión de la aplicación de consola donde solamente se trabaja la validación del archivo cargado e informando de los errores ocurridos.

Iteración 2

El objetivo de esta iteración es la implementación de las tareas de las historias de usuario que tienen relación con las funcionalidades de mayor peso dentro de la aplicación. Que son las relacionadas con la implementación de las fases principales que componen un traductor de lenguajes, análisis léxico y sintáctico. Se prevé como posible resultado una segunda versión de la aplicación con las principales funcionalidades de un traductor implementadas.

Iteración 3

Durante el transcurso de esta iteración se prevé la implementación de las tareas de historia de usuario que permiten la generación del código C/C++ y la realización del análisis semántico. Integrando de esta forma las anteriores iteraciones, para obtener como resultado la versión final del producto. Como resultado de la misma la aplicación es puesta a un periodo de tiempo de funcionamiento para evaluar su desempeño.

2.4 Propuesta del sistema

El sistema va estar compuesto por dos módulos, en un primer módulo donde se llevará a cabo el proceso de análisis léxico, sintáctico y la generación de un código intermedio en lenguaje C/C++. Siendo esta salida del primer módulo la entrada del segundo, donde se realizará el análisis semántico y la llamada al compilador GCC que permitirá finalmente obtener el código objeto correspondiente a la rutina implementada en lenguaje Diagrama de Escalera. En caso de ocurrir errores en algunas de las fases de análisis los mismos son reportados por consola.

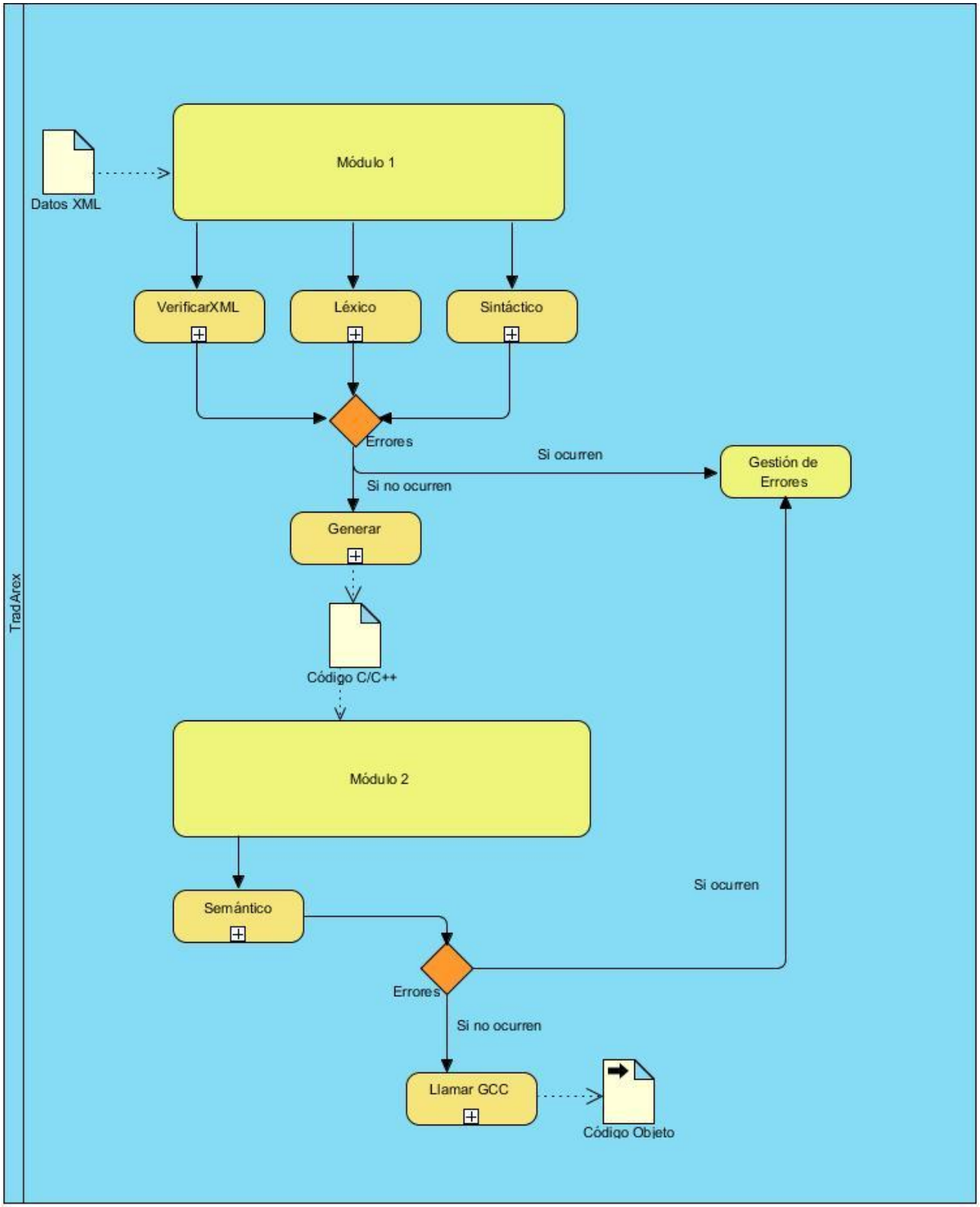

Ilustración 13. Propuesta del sistema

2.5 Diseño e implementación del sistema

En esta fase del diseño se presenta una descripción de algunos elementos, atendiendo a la metodología seleccionada, que facilitaran el posterior desarrollo de la herramienta. Entre los que se destacan la arquitectura que se propone y los patrones de diseño.

2.5.1 Arquitectura propuesta

Para el desarrollo de un software se utilizan de uno a varios estilos arquitectónicos. Cada estilo describe una categoría de sistemas que abarca:

- ✓ Un conjunto de componentes (por ejemplo, una base de datos, módulos computacionales) que realizan una función requerida por el sistema.
- ✓ Un conjunto de conectores que permiten la comunicación, coordinación y cooperación entre los componentes.
- ✓ Restricciones que definen como se integraran los componentes para formar el sistema.
- ✓ Modelos semánticos que permiten a un diseñador, mediante el análisis de las propiedades conocidas de las partes que lo integran, comprender las propiedades generales de un sistema.

Un estilo arquitectónico no es más que una transformación impuesta al diseño de todo un sistema. Que tiene como objetivo establecer una estructura para todos los componentes del sistema. [20]

Partiendo de lo antes expuesto, el estilo arquitectónico propuesto para el desarrollo del sistema es:

Arquitectura de flujo de datos: Esta arquitectura se aplica cuando los datos de entrada se habrán de trasformar en datos de salida mediante una serie de componentes para el cálculo o la manipulación. Se representa como una estructura de tuberías y filtros, donde el conjunto de componentes se denominan filtros, conectados mediante tuberías que transmiten datos de un componente al siguiente. Está diseñado para esperar la entrada de datos de cierta forma y producir una salida de una forma específica. (Ver Ilustración 11)

2.5.2 Patrones de diseño o patrones Grasp

Un patrón de diseño no es más que una solución repetible a un problema recurrente en el diseño del software. No constituye un diseño terminado que pueda traducirse a directamente a código, más bien representa una descripción sobre cómo resolver el problema.

Los patrones de diseño reflejan todo el rediseño y remodificación que los desarrolladores han ido haciendo a medida que intentaban conseguir mayor reutilización y flexibilidad en su software. Un patrón de diseño permite denominar, abstraer e identificar los aspectos clave de una estructura de diseño orientado a objetos reutilizables. El patrón de diseño identifica las clases e instancias participantes, sus roles y colaboraciones, y la distribución de sus responsabilidades. Cada patrón de diseño se centra en un problema concreto, describiendo cuando aplicarlo y si tiene sentido hacerlo teniendo en cuenta otras restricciones de diseño, así como las consecuencias, ventajas e inconvenientes de su uso. El diseño de todas las tarjetas de Clase-Responsabilidad-Colaboradores, hacen uso de las buenas prácticas propuestas por Craig Laman, referente a la asignación de responsabilidades, que constituyen un elemento primordial cuando se diseña e implementa un sistema orientado a objetos. El patrón experto se utiliza para asignar la responsabilidad que tendrá cada clase, en dependencia de la información que maneja, algunas de las clases que representan dicho patrón en el sistema son la clase Token, la cual contiene toda la información referente a los objetos de tipo token y POU que representa todo la información de un conjunto de estructuras que representan el lenguaje LD.

El patrón creador se utiliza para asignar a determinadas clases la responsabilidad de crear instancias de otras si cumple con determinadas condiciones, en el sistema el mismo se representa mediante las clases AnalisisLexico y AnalisisSemantico. Para asignar las responsabilidades se tuvo en cuenta mantener un bajo acoplamiento y alta cohesión entre cada clase que compone el sistema.

2.5.3 Patrones GOF

El patrón de comportamiento utilizado fue el Singleton. Pues garantiza que solamente se cree una instancia de una clase y provee un punto de acceso global al mismo. Todos los objetos que usan la misma instancia.

En la aplicación se hace uso del patrón Singleton en la clase Controladora, para acceder con una misma instancia a diferentes clases desde cualquier parte de la aplicación, como son el análisis léxico y sintáctico. También para almacenar variables en la tabla de símbolos y actualizaras en el proceso de análisis.

2.6 Tarjetas Clase-Responsabilidades-Colaboradores (CRC)

Las tarjetas CRC (Class, Responsabilities and Collaboration) sirven para diseñar el sistema en conjunto entre todo el equipo, permiten reducir el modo de pensar procedural y apreciar la tecnología de objetos.

Las clases que conforman el sistema están descritas a continuación en las tarjetas CRC.

Tabla 24. Tarjeta CRC 1

Clase: Controladora	
Descripción: Es la clase que se encarga de proporcionar una interfaz unificada de alto nivel. Conoce que clases del subsistema son responsables de cada petición, y delega esas peticiones de los clientes a los objetos apropiados del subsistema**.**	
Responsabilidades: ✓ Crear la lista de errores ✓ Crear la tabla de símbolos. ✓ Ejecutar los analizadores léxico, sintáctico y semántico. ✓ Informar si existieron errores en el proceso de compilación. ✓ Ejecutar el generador de código.	**Colaboradores:** ✓ AnalisisLexico ✓ AnalisisSintactico ✓ GenerarCodigo ✓ VerificarXML ✓ AnalisisSemantico ✓ Error ✓ TablaSimbolos

Tabla 25. Tarjeta CRC 2

Clase: Error	
Descripción: Es la clase que se encarga de manejar los errores encontrados durante el proceso de análisis y verificación.	
Responsabilidades: ✓ Manejar los errores de verificación del XML, además de léxicos, sintácticos y semánticos durante el proceso de	**Colaboradores:** ✓ AnalisisLexico ✓ AnalisisSintactico

construcción.	✓ AnalisisSemantico ✓ VerificarXML

Tabla 26. Tarjeta CRC 3

Clase: VerificarXML	
Descripción: Es la clase que se encarga de gestionar y verificar el documento XML.	
Responsabilidades: ✓ Verificar la validez del documento XML.	**Colaboradores:** ✓ Error

Tabla 27. Tarjeta CRC 4

Clase: AnalizadorLexico	
Descripción: Reconoce el conjunto de tokens y almacena en la tabla de símbolos los datos necesarios.	
Responsabilidades: ✓ Reconocer los tokens. ✓ Adicionar los identificadores a la tabla de símbolos. ✓ Reportar en caso de que ocurra algún error en el proceso.	**Colaboradores:** ✓ TablaSimbolos ✓ Error ✓ Conexion

Tabla 28. Tarjeta CRC 5

Clase: AnalizadorSintactico	
Descripción: Construye las estructuras de datos necesarias para facilitar generación de código intermedio.	
Responsabilidades: ✓ Se encarga de construir un conjunto de estructuras de datos, que constituyen una representación intermedia para ser utilizada en la generación de código	**Colaboradores:** ✓ Error ✓ TablaSimbolos

intermedio.	✓ POU

Tabla 29. Tarjeta CRC 6

Clase: Conexion	
Descripción: Brinda una representación del estado de las conexiones entre los diferentes componentes que conforman la rutina de control implementada en el lenguaje LD.	
Responsabilidades: ✓ Crear una representación de la relación existente entre los diferentes componentes que conforman el código en el lenguaje LD.	**Colaboradores:** ✓ AnalisisLexico ✓ AnalisisSintactico

Tabla 30. Tarjeta CRC 7

Clase: TablaSimbolos	
Descripción: Es la clase que se encarga de almacenar toda la información referente a los identificadores y variables**.**	
Responsabilidades: ✓ Almacenar los identificadores. ✓ Actualizar el estado de las variables.	**Colaboradores:** ✓ AnalisisLexico ✓ AnalisisSintactico

Tabla 31. Tarjeta CRC 8

Clase: POU	
Descripción: Es la clase que se encarga de almacenar toda la información referente a las estructuras reconocidas**.**	
Responsabilidades: ✓ Brindar una representación intermedia de las estructuras del código LD.	**Colaboradores:** ✓ AnalisisSintactico ✓ Conexion ✓ GenerarCodigo

Tabla 32. Tarjeta CRC 9

Clase: GenerarCodigo	
Descripción: Clase encargada de generar código intermedio**.**	
Responsabilidades: ✓ Crear código intermedio	**Colaboradores:** ✓ POU

Tabla 33. Tarjeta CRC 10

Clase: AnalisisSemantico	
Descripción: Clase encargada de comprobar la validez semántica de las instrucciones**.**	
Responsabilidades: ✓ Crear código objeto ✓ Analizar semánticamente	**Colaboradores:** ✓ POU ✓ GenerarCodigo

Conclusiones parciales

Para la implementación de sistema fueron definidos los requisitos funcionales y no funcionales que debe contener, obteniendo 8 historias de usuario. También se realizó una estimación de esfuerzo por historia de usuario. En cuanto a las actividades referentes al diseño, contenidas por la metodología XP a partir de la solución propuesta, se seleccionó el estilo arquitectónico Flujo de Datos. Definiéndose además los patrones de diseño GRASP y de comportamiento GOF, permitiendo crear la línea base de la arquitectura y una mejor organización y calidad de la solución, evidenciada a través de las tarjetas CRC.

3. Capítulo 3: Desarrollo y Pruebas

A continuación se muestra las actividades propuestas por la metodología XP dentro de las fases de Desarrollo y Pruebas. Por cada iteración se realiza una planificación que contiene las historias de usuario que son atendidas en cada iteración. Como también el conjunto de tareas que se ejecutan para darle cumplimiento a la historia de usuario. Finalmente cada iteración concluye con un conjunto de pruebas de aceptación que permiten validar el cumplimiento de los requisitos.

3.1 Iteraciones

A continuación se muestra el seguimiento del proyecto al pasar por cada una de las iteraciones definidas por el usuario en el plan de entregas inicial.

3.1.1 Primera iteración

Planificación de la iteración

Implementación de las tareas de las historias de usuario que contienen las funcionalidades iniciales de la aplicación, por lo que durante el transcurso de la misma se crea la base de la arquitectura del sistema con funcionalidades mínimas como cargar y validar el archivo.

Tabla 34. Planificación de la primera iteración

Historias de Usuario	Tareas de ingeniería	Puntos estimados
Cargar y validar archivo XML	Implementar en el archivo main.cpp el código para cargar el documento XML. (0.5) Estudio del módulo QtXml.(1) Diseñar e implementar la clase Verificar XML. (1) Declaración de la interfaz CheckLD. (0.5)	3

Versión de la aplicación obtenida.

Se muestra en el Anexo 1 Sección A.

Pruebas de aceptación

Historia de usuario 1

Tabla 35. Prueba de aceptación 1

Caso de prueba de aceptación 1	
Historia de usuario:	Cargar y validar archivo XML
Nombre:	Cargar archivo.
Responsable:	Alex Fernández Castrillo
Descripción:	Permitirá verificar que la dirección y el archivo que el usuario pasa por consola son válidos, de no serlos la aplicación debe dar la posibilidad de volver a intentarlo.
Condiciones de ejecución:	
Entradas/pasos de ejecución:	
Resultado esperado:	Debe ser verificado el directorio especificado y la existencia del archivo a traducir.

Tabla 36. Prueba de aceptación 2

Caso de prueba de aceptación 2	
Historia de usuario:	Cargar y validar archivo XML
Nombre:	Validar estructura XML.
Responsable:	Alex Fernández Castrillo
Descripción:	Esta prueba se llevará a cabo para identificar posibles errores en el código del archivo XML.
Condiciones de ejecución:	Tiene que estar cargado el documento.
Entradas/pasos de ejecución:	
Resultado esperado:	Debe realizarse la evaluación de la

	conformación XML, verificando que todas las etiquetas fueron construidas correctamente.

3.1.2 Segunda iteración

Planificación de la iteración

El objetivo de esta iteración es la implementación de las tareas de las historias de usuario que tienen relación con las funcionalidades de mayor peso dentro de la aplicación. Que son las relacionadas con la implementación de las dos fases principales que componen un traductor de lenguajes, análisis léxico y sintáctico. Se prevé como posible resultado una segunda versión de la aplicación con las principales funcionalidades de un traductor implementadas.

Tabla 37. Planificación de la segunda iteración

Historias de Usuario	**Tareas de ingeniería**	**Puntos estimados**
Realizar análisis léxico al código del lenguaje Diagrama de Escalera en el archivo XML. Realizar análisis sintáctico al código del lenguaje Diagrama de Escalera.	Estudio de la clase QXmlStreamReader. (0.5) Diseño e implementación de la clase AnalisisLexico. (1) Diseño e implementación de la clase Token. (0.5) Diseño e implementación de la clase TablaSimbolos. (0.5) Diseño e implementación de la clase Conexion. (0.5) Diseño e implementación de la clase AnalizadorSintactico. (1) Diseño e implementación de la clase POU. (1)	5

Versión de la aplicación obtenida

Se muestra en el Anexo 2 Sección B.

Pruebas de aceptación

Historia de usuario 2

Tabla 38. Prueba de aceptación 3

Caso de prueba de aceptación 3	
Historia de usuario:	Realizar análisis léxico al código del lenguaje Diagrama de Escalera en el archivo XML.
Nombre:	Implementación del analizador léxico.
Responsable:	Alex Fernández Castrillo
Descripción:	Esta prueba se realizará con el objetivo de identificar los elementos que contiene el archivo referente al código en lenguaje Diagrama de Escalera. Así como comprobar que la herramienta puede gestionar posibles errores léxicos, como etiquetas sin contenido, entre otros.
Condiciones de ejecución:	Tiene que estar cargado el documento. Tiene que haber sido verificado desde el punto de vista del lenguaje XML.
Entradas/pasos de ejecución:	
Resultado esperado:	Lista de tokens con su correspondiente información. En caso de la existencia de errores interrumpir el proceso de traducción e informar del error.

Historia de usuario 3

Tabla 39. Prueba de aceptación 4

Caso de prueba de aceptación 4	
Historia de usuario:	Realizar análisis sintáctico al código del

	lenguaje Diagrama de Escalera.
Nombre:	Implementación del analizador sintáctico.
Responsable:	Alex Fernández Castrillo
Descripción:	Esta prueba se realizará con el objetivo de reconocer las estructuras relacionadas al lenguaje Diagrama de Escalera, contenidas en el documento XML.
Condiciones de ejecución:	Tiene que estar cargado el documento. El archivo debe ser analizado lexicalmente primero.
Entradas/pasos de ejecución:	
Resultado esperado:	Generar las estructuras correspondientes para cada sección de código reconocida.

3.1.3 Tercera iteración

Planificación de la iteración

Durante el transcurso de esta iteración se prevé la implementación de la tarea de historia de usuario que permite la generación del código y el análisis semántico. Integrando de esta forma las anteriores iteraciones, para obtener como resultado la versión final del producto. Como resultado de la misma la aplicación es puesta a un período de tiempo de funcionamiento para valorar su desempeño.

Tabla 40. Planificación de la tercera iteración

Historias de Usuario	Tareas de ingeniería	Puntos estimados
Generación de código Análisis Semántico	Diseño e implementación de la clase GenerarCodigo. (1) Diseño e implementación del código para el análisis semántico. (1)	2

Versión de la aplicación obtenida

Se muestra en el Anexo 3 Sección C.

Prueba de aceptación

Historia de usuario 4

Tabla 41. Prueba de aceptación 5

Caso de prueba de aceptación 5	
Historia de usuario:	Generación de código.
Nombre:	Generación de código intermedio.
Responsable:	Alex Fernández Castrillo
Descripción:	Esta prueba se realizará con el objetivo de comprobar que la aplicación es capaz de generar código intermedio a partir del código cargado en el archivo XML.
Condiciones de ejecución:	Debe haber sido realizado de forma satisfactoria el análisis léxico y sintáctico.
Entradas/pasos de ejecución:	
Resultado esperado:	Generar el código correspondiente a la rutina contenida en el XML.

Tabla 42. Prueba de aceptación 6

Caso de prueba de aceptación 6	
Historia de usuario:	Generación de código.
Nombre:	Salvar código intermedio.
Responsable:	Alex Fernández Castrillo
Descripción:	Esta prueba se realizará con el objetivo de comprobar que la aplicación es capaz de salvar el código C/C++ generado.

Condiciones de ejecución:	Debe haberse generado el código intermedio en lenguaje C/C++.
Entradas/pasos de ejecución:	
Resultado esperado:	Archivo con el código correspondiente en C/C++ a la rutina contenida en el XML.

Historia de usuario 5

Tabla 43. Prueba de aceptación 7

Caso de prueba de aceptación 7	
Historia de usuario:	Análisis Semántico
Nombre:	Diseño e implementación del código para el análisis semántico.
Responsable:	Alex Fernández Castrillo
Descripción:	Esta prueba se realizará con el objetivo de comprobar que la aplicación es capaz de analizar semánticamente las instrucciones generadas en C/C++.
Condiciones de ejecución:	Debe haberse generado el código intermedio en lenguaje C/C++.
Entradas/pasos de ejecución:	
Resultado esperado:	Realización del análisis semántico y la capacidad de reconocer los errores que se encuentren.

3.2 Resumen del resultado de las iteraciones

En cada iteración que se ejecutó se concluyó con un conjunto de pruebas de aceptación para verificar el cumplimiento de los requisitos. Dichas pruebas permitieron comprobar el grado de cumplimiento de cada requisito definido acorde con el resultado final. Las no conformidades identificadas fueron mitigadas antes de pasar a la siguiente iteración. A continuación se muestra un gráfico que resume los resultados arrojados por cada iteración. (Ver Ilustración 14)

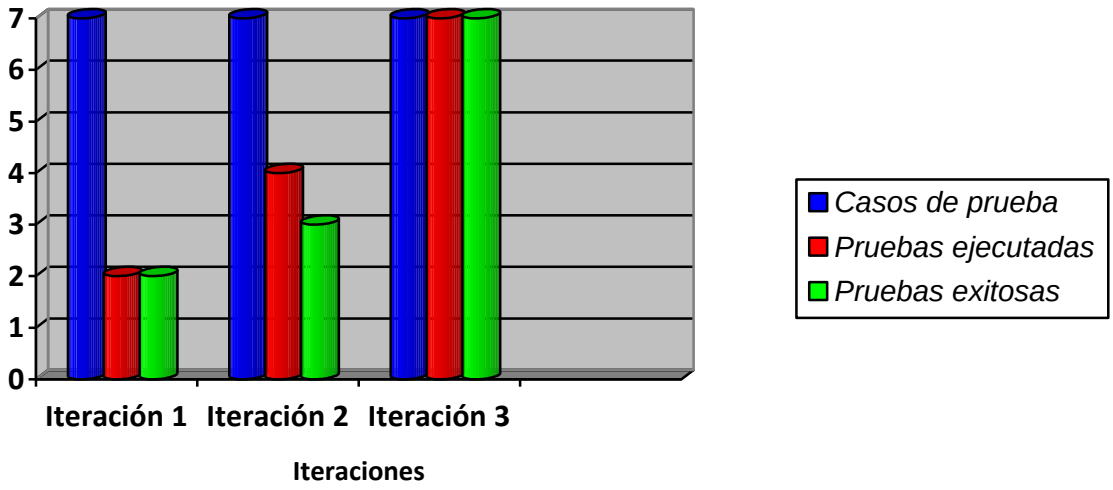

Ilustración 14. Gráfico de pruebas por iteración

Conclusiones parciales

Dentro de las actividades que propone la metodología seleccionada están el análisis de cada iteración y las pruebas de aceptación que se van a desarrollar en cada una de las mismas, para comprobar el correcto funcionamiento del sistema. Atendiendo a esto fueron definidas 7 pruebas de aceptación. Las mismas permitieron una evaluación del cumplimiento de los requisitos funcionales que debía cumplir el sistema, el cual respondió de forma satisfactoria a la mayoría de las pruebas hechas, presentando algunas no conformidades en la segunda iteración. Para el paso de una iteración a otra fueron solucionadas cada una de las deficiencias detectadas.

Conclusiones Generales

En el trabajo se ha logrado el diseño y desarrollo de una herramienta capaz de realizar la traducción y compilación de funciones implementadas en el lenguaje Diagrama de Escalera, respetando las definiciones establecidas en la Norma Internacional IEC 61131-3. Mediante el traductor, “TradArex”, se puede obtener archivos de extensión C/C++, que integrado al *toolchain* de C de la tarjeta CID 300/9 permite obtener directamente el código objeto para la arquitectura del PLC Arex.

A partir del análisis de un conjunto de herramientas, que realizan funciones similares, se establecieron una serie de elementos y definiciones en cuánto al diseño del sistema; como también el flujo para realizar el proceso de compilación.

La ejecución de las pruebas de aceptación a la aplicación desarrollada, contribuyó a concluir que el traductor “TradArex” cumple satisfactoriamente con los requerimientos especificados por el cliente.

Recomendaciones

1. Ampliar la capacidad de traducción mediante el procesamiento de variables de configuración y recursos en la programación de las rutinas.
2. Incorporación a la traducción todos los tipos de variables soportados por el lenguaje así como la llamada de funciones implementadas por los usuarios.
3. Integrar el compilador con la herramienta PLC HMI Arex, específicamente con la interfaz para implementar en el lenguaje Diagrama de Escalera.

Referencias Bibliográficas

1. **Internacional, Comisión Electrónica.** Norma Internacional IEC 61131-3. Tercera Edición *IEC 61131-3.* Madrid, España : s.n., 2003.

2. **Calvo, Luis.** [En línea] 22 de Abril de 2003. [Citado el: 15 de febrero de 2014.] en línea. http://www.scribd.com/doc/Lenguaje-de-Programacion.

3. **SENA.** Controladores Lógicos Programables. *Controladores Lógicos Programables.* 2005. Vol. Sena Virtual Distrito.

4. **Technical, PLCopen.** XML Formats for IEC 61131-3. *XML Formats for IEC 61131-3.* [Standard]. 2009. Vol. 2.01, pág. 80.

5. **Paniagua, Angel Barbero.** www.dat.etsit.upm.es. *www.dat.etsit.upm.es.* [En línea] 2012. [Citado el: 10 de noviembre de 2013.] http://www.dat.etsit.upm.es.

6. **Díaz, Emiliano Llano.** *Análisis y diseño de compiladores.* 1era Edición. Mexico D.F. : Exa Ingeniería SA., 2002.

7. **Bolton, W.** *Programmable Logic Controllers.* Fourth Edition. s.l. : Elsevier's Science & Technology Rights, 2006. Vol. 4.

8. **Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman.** www.pearsoned.com. *www.pearsoned.com.* [En línea] 2007. http://www.pearsoned.com.

9. **Mogensen, Torben Ægidius.** *Basics of Compiler Design.* 3ra . Copenhagen : s.n., 2009. pág. 500. Vol. 1.

10. **Dr. Liesner Acevedo Martínez, Msc. Karel Osorio Ramírez.** Técnicas de Compilación. *Manual Práctico para Estudiantes de Infórmatica.* La Habana, Cuba : s.n., 2011. Manual práctico docente.

11. **Daniel Vargas Vela**, **Fernando Martinez Santa**. *Diseño e implentación del compilador STL para el PLC-UD. Diseño e implentación del compilador STL para el PLC-UD.*

12. **Andres Felipe Vallejo, Erick Damian Jimenez.** *Diseño e implementación de un laboratorio de emulación remota de PLC.* Santiago de Cali, Colombia, 2011.

13. **Corbilla, Jordi.** Blog dedicated to software development in Delphi and others, Agile Development, eXtreme programming, Pragmatic thinking and learning, technology, entrepreneurship, photography and science. [En línea] 2006. http://Randomthoughtsoncoding&technology/Beremiz/SistemaOpenSourceparalaautomatización.com.

14. **Maikel Ramírez, Eduardo Millán y Valery Moreno.** *Herramienta para programar un controlador lógico programable basado en hardware reconfigurable.* La Habana, Cuba : s.n., 2011. Herramienta para programar un controlador lógico programable basado en hardware reconfigurable, Vol. XXXII. 1815-5928.

15. **Sousa, Mario de.** bitbucket.org. *bitbucket.org.* [En línea] 2003-2012. The following compiler

has been based on the FINAL DRAFT IEC 61131-3, 2nd Ed. https://bitbucket.org/mjsousa/matiec.

16. **Corp, Nokia.** qt.nokia.com. *qt.nokia.com.* [En línea] 2012. [Citado el: 20 de noviembre de 2013.] http://qt.nokia.com.

17. **Amaro Calderón, S.D.** *Metodologías Ágiles.* Facultad de Ciencias Físicas y Matemáticas, Universidad Nacional de Trujillo. Trujillo : s.n., 2007. Trabajo investigativo.

18. **STROUSTRUP., B.** *The Design and Evolution of C++.* s.l. : Addison-Wesley, 1994. 978-0-201-54330-8.

19. **Booch, Grady, Rumbaugh, James y Jacobson, Ivar.** *El Lenguaje Unificado de Modelado.* Tercera. 2001.

20. **Vargas, Arianna Gómez.** *Herramienta de Programación y Configuración para el PLC HMI Arex.* Centro de Informática Industrial, Universidad de las Ciencias Informáticas, UCI. La Habana : s.n., 2013. Tesis. Investigación y Desarrollo.

21. **León, E.** es.scribd.com. *es.scribd.com.* [En línea] 2000. [Citado el: 11 de febrero de 2013.] http://es.scribd.com/doc/36636137/Tutorial-Visual-Paradigm.

22. **Saburit, MCs. Arelys Borrell.** ¿Le resulta difícil hacer la bibliografía_ Los gestores de referencias bibliográficas pueden ayudarlo — Centro de Ayuda de la Biblioteca Virtual en Salud. *¿Le resulta difícil hacer la bibliografía_ Los gestores de referencias bibliográficas pueden ayudarlo — Centro de Ayuda de la Biblioteca Virtual en Salud.* [En línea] 206. [Citado el: 24 de enero de 2014.] http://gestores.com/¿Le resulta difícil hacer la bibliografía/ Los gestores de referencias bibliográficas pueden ayudarlo/Centro de Ayuda de la Biblioteca Virtual en Salud.

23. **Otero, Aranzazú Sanchis.** *Gestor de Referencias Bibliografícas.* La Habana : Bibliographies Made Easy, 2011. Manual Básico.

Bibliografía

Internacional, Comisión Electrónica. Norma Internacional IEC 61131-3. Tercera Edición *IEC 61131-3.* Madrid, España : s.n., 2003.

Calvo, Luis. [En línea] 22 de Abril de 2003. [Citado el: 15 de febrero de 2014.] en línea. http://www.scribd.com/doc/Lenguaje-de-Programacion.

SENA. Controladores Lógicos Programables. *Controladores Lógicos Programables.* 2005. Vol. Sena Virtual Distrito.

Technical, PLCopen. XML Formats for IEC 61131-3. *XML Formats for IEC 61131-3.* [Standard]. 2009. Vol. 2.01, pág. 80.

Paniagua, Angel Barbero. www.dat.etsit.upm.es. *www.dat.etsit.upm.es.* [En línea] 2012. [Citado el: 10 de noviembre de 2013.] http://www.dat.etsit.upm.es.

Díaz, Emiliano Llano. *Análisis y diseño de compiladores.* 1era Edición. Mexico D.F. : Exa Ingeniería SA., 2002.

Bolton, W. *Programmable Logic Controllers.* Fourth Edition. s.l. : Elsevier's Science & Technology Rights, 2006. Vol. 4.

Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. www.pearsoned.com. *www.pearsoned.com.* [En línea] 2007. http://www.pearsoned.com.

Mogensen, Torben Ægidius. *Basics of Compiler Design.* 3ra . Copenhagen : s.n., 2009. pág. 500. Vol. 1.

Dr. Liesner Acevedo Martínez, Msc. Karel Osorio Ramírez. Técnicas de Compilación. *Manual Práctico para Estudiantes de Infórmatica.* La Habana, Cuba : s.n., 2011. Manual práctico docente.

Stallman, Charles Donnelly y Richard. bison 1.27.org. *bison 1.27.org.* [En línea] 2005. [Citado el: 13 de diciembre de 2014.] www://bison 1.27.org.

Daniel Vargas Vela, **Fernando Martinez Santa**. Diseño e implentación del compilador STL para el PLC-UD. Diseño e implentación del compilador STL para el PLC-UD.

Corbilla, Jordi. Blog dedicated to software development in Delphi and others, Agile Development, eXtreme programming, Pragmatic thinking and learning, technology, entrepreneurship, photography and science. [En línea] 2006. http://Randomthoughtsoncoding&technology/Beremiz/SistemaOpenSourceparalaautomatización.com.

Maikel Ramírez, Eduardo Millán y Valery Moreno. *Herramienta para programar un controlador lógico programable basado en hardware reconfigurable.* La Habana, Cuba : s.n., 2011. Herramienta para programar un controlador lógico programable basado en hardware reconfigurable, Vol. XXXII. 1815-5928.

Sousa, Mario de. bitbucket.org. *bitbucket.org.* [En línea] 2003-2012. The following compiler has been based on the FINAL DRAFT - IEC 61131-3, 2nd Ed.. https://bitbucket.org/mjsousa/matiec.

Bibliografía

Corp, Nokia. qt.nokia.com. *qt.nokia.com.* [En línea] 2012. [Citado el: 20 de noviembre de 2013.] http://qt.nokia.com.

Amaro Calderón, S.D. *Metodologías Ágiles.* Facultad de Ciencias Físicas y Matemáticas, Universidad Nacional de Trujillo. Trujillo : s.n., 2007. Trabajo investigativo.

STROUSTRUP., B. *The Design and Evolution of C++.* s.l. : Addison-Wesley, 1994. 978-0-201-54330-8.

Booch, Grady, Rumbaugh, James y Jacobson, Ivar. *El Lenguaje Unificado de Modelado.* Tercera. 2001.

Vargas, Arianna Gómez. *Herramienta de Programación y Configuración para el PLC HMI Arex.* Centro de Informática Industrial, Universidad de las Ciencias Informáticas, UCI. La Habana : s.n., 2013. Tesis. Investigación y Desarrollo.

León, E. es.scribd.com. *es.scribd.com.* [En línea] 2000. [Citado el: 11 de febrero de 2013.] http://es.scribd.com/doc/36636137/Tutorial-Visual-Paradigm.

Saburit, MCs. Arelys Borrell. ¿Le resulta difícil hacer la bibliografía_ Los gestores de referencias bibliográficas pueden ayudarlo — Centro de Ayuda de la Biblioteca Virtual en Salud. *¿Le resulta difícil hacer la bibliografía_ Los gestores de referencias bibliográficas pueden ayudarlo — Centro de Ayuda de la Biblioteca Virtual en Salud.* [En línea] 206. [Citado el: 24 de enero de 2014.] http://gestores.com/¿Le resulta difícil hacer la bibliografía/ Los gestores de referencias bibliográficas pueden ayudarlo/Centro de Ayuda de la Biblioteca Virtual en Salud.

Otero, Aranzazú Sanchis. *Gestor de Referencias Bibliografícas.* La Habana : Bibliographies Made Easy, 2011. Manual Básico.

Dr. Cs. Carlos Álvarez de Zayas. *METODOLOGIA DE LA INVESTIGACION CIENTIFICA.* La Habana. Centro de Estudios de Educación Superior “Manuel F. Gran”, 2011.

Anexos

Anexo 1 Sección A

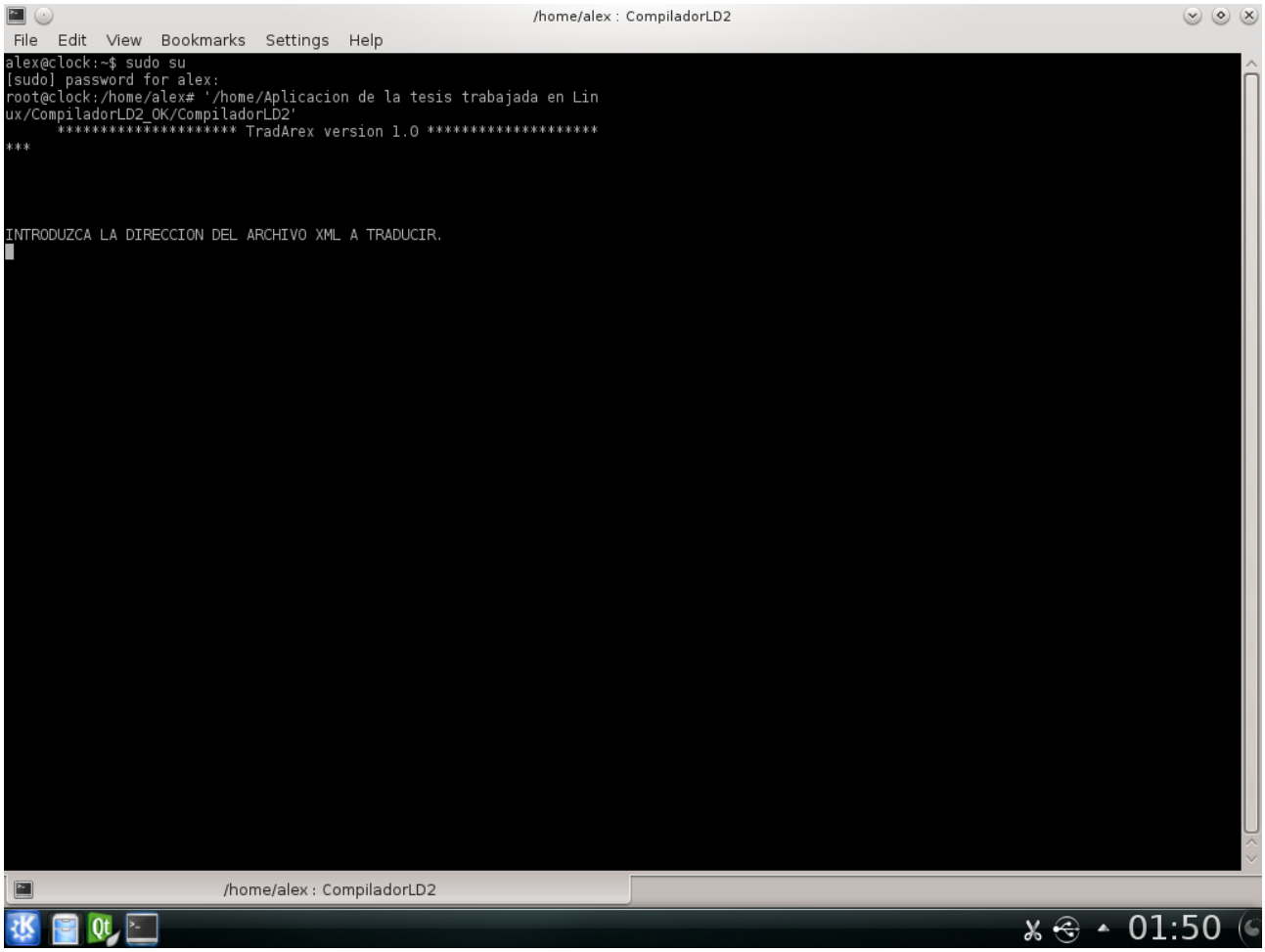

Ilustración 15. Interfaz inicial

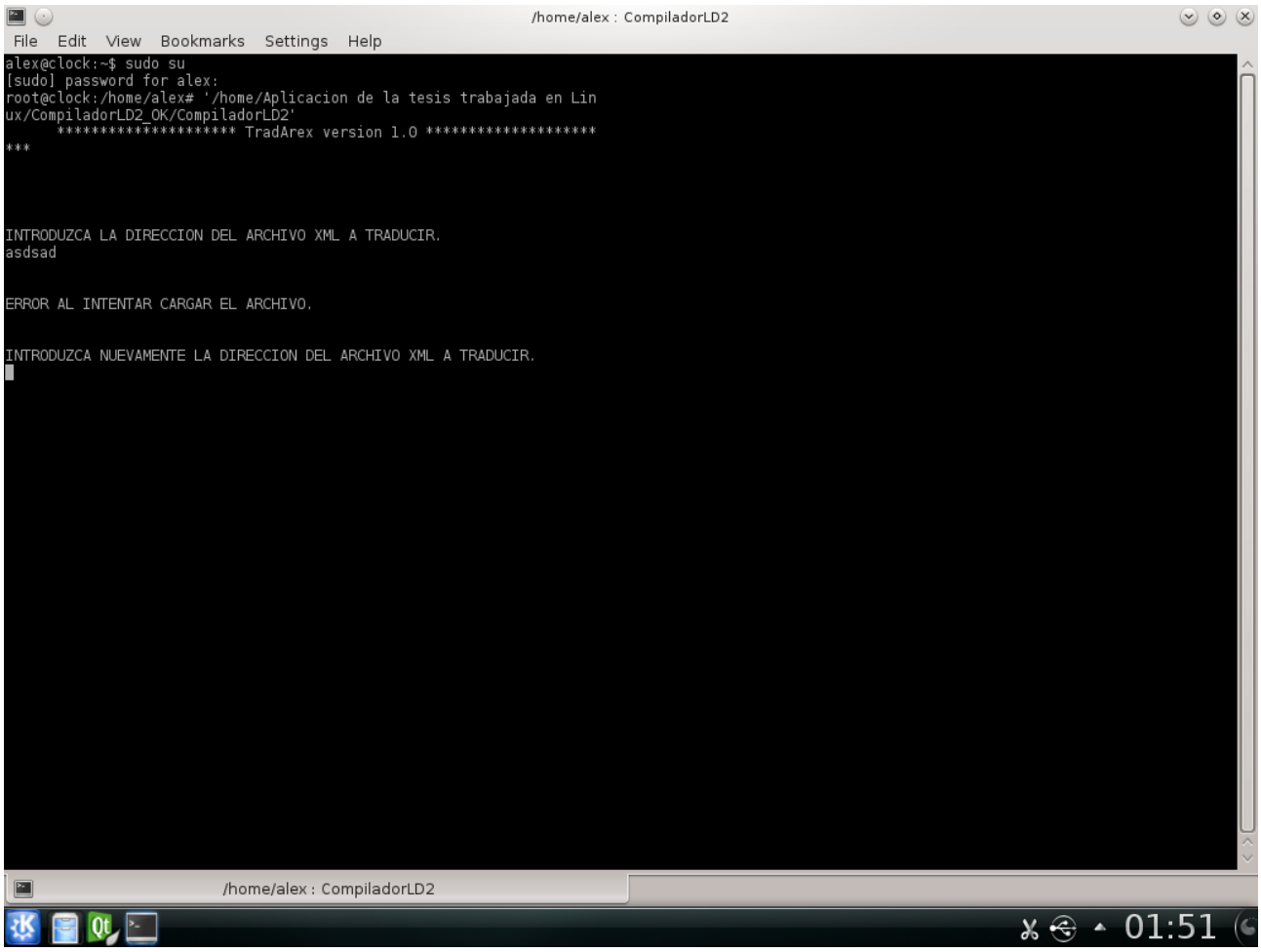

Ilustración 16. Gestionando la dirección del XML

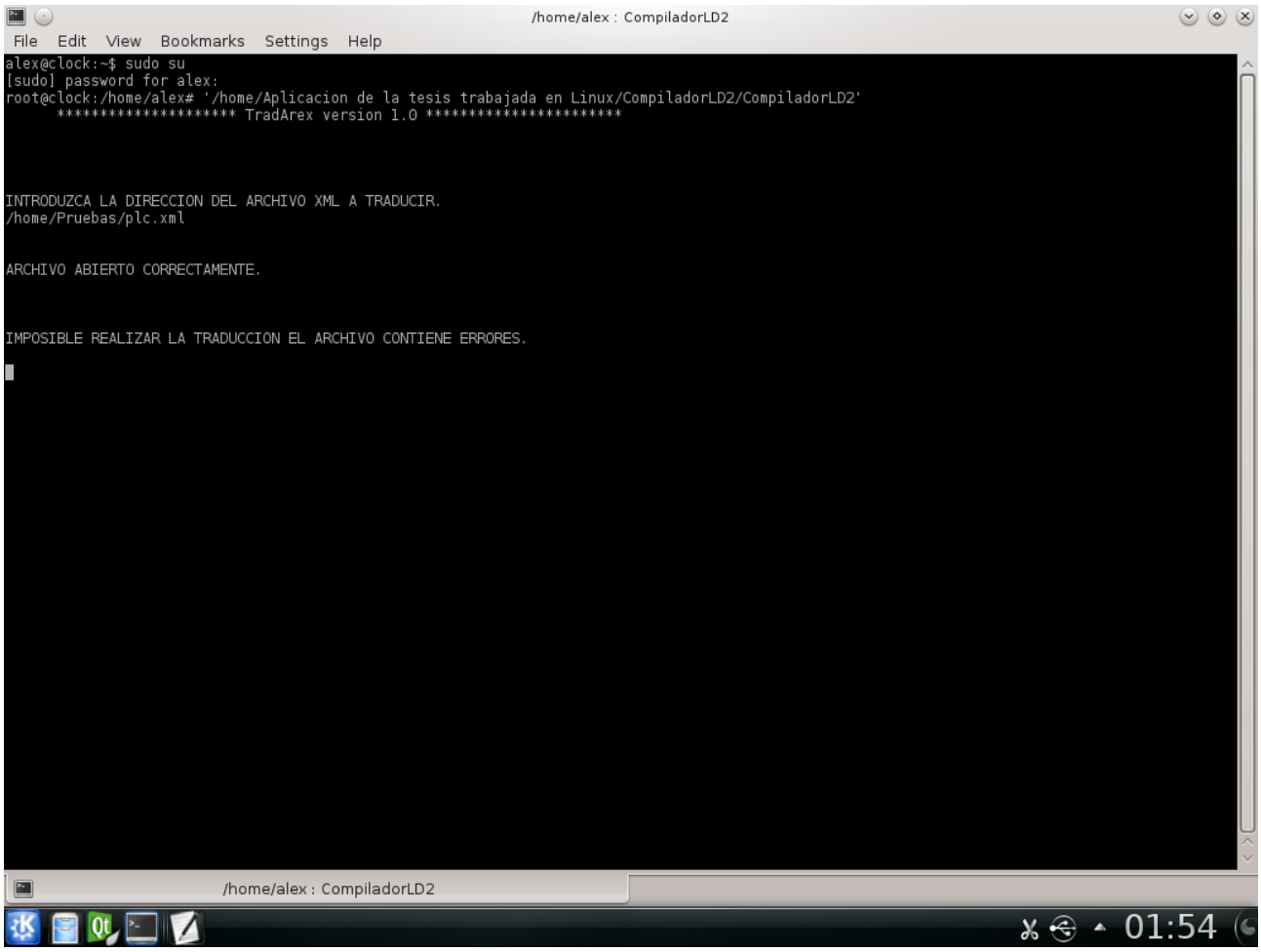

Ilustración 17. Gestionando errores XML

Anexo 2 Sección B

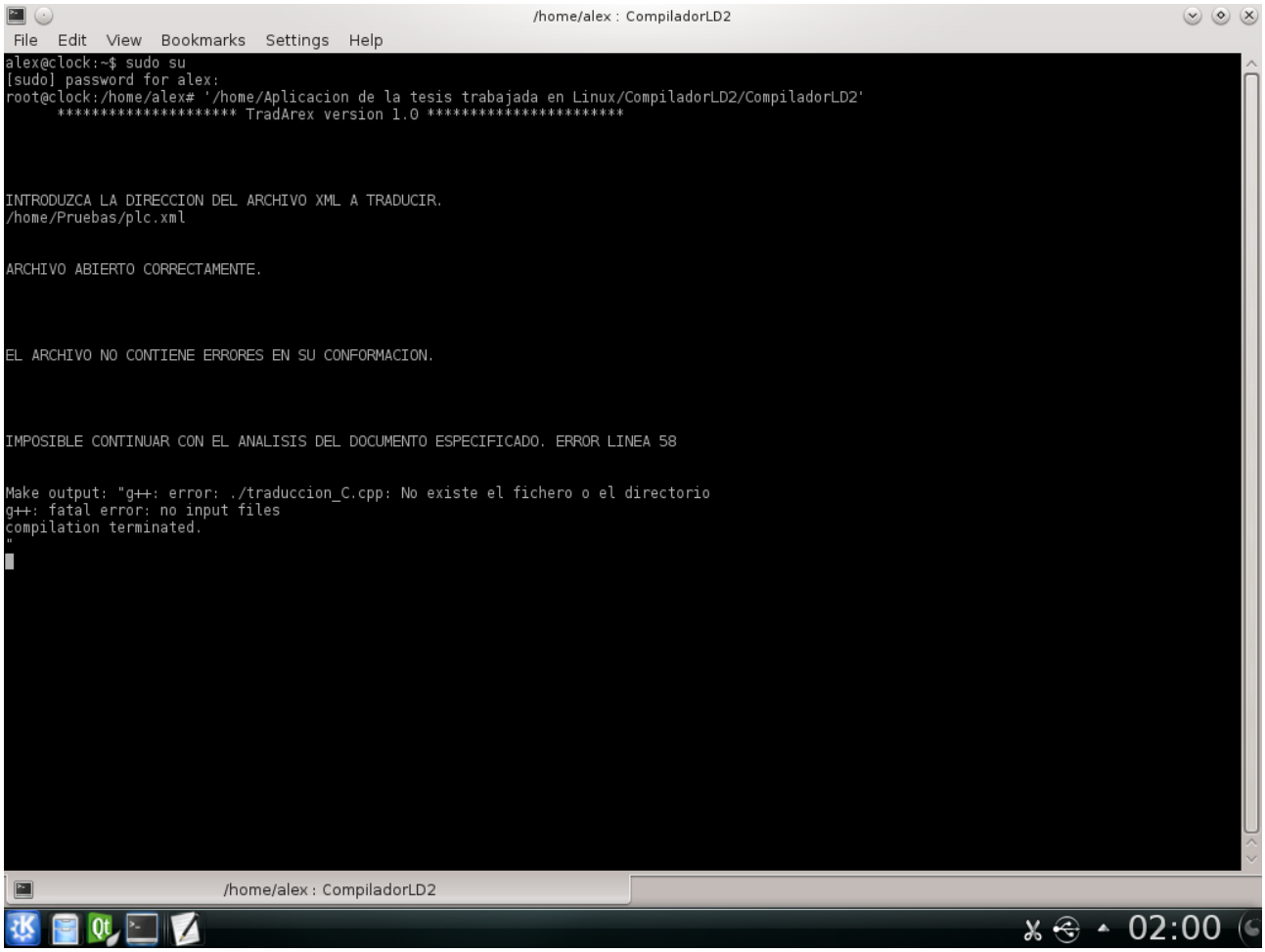

Ilustración 18. Error léxico

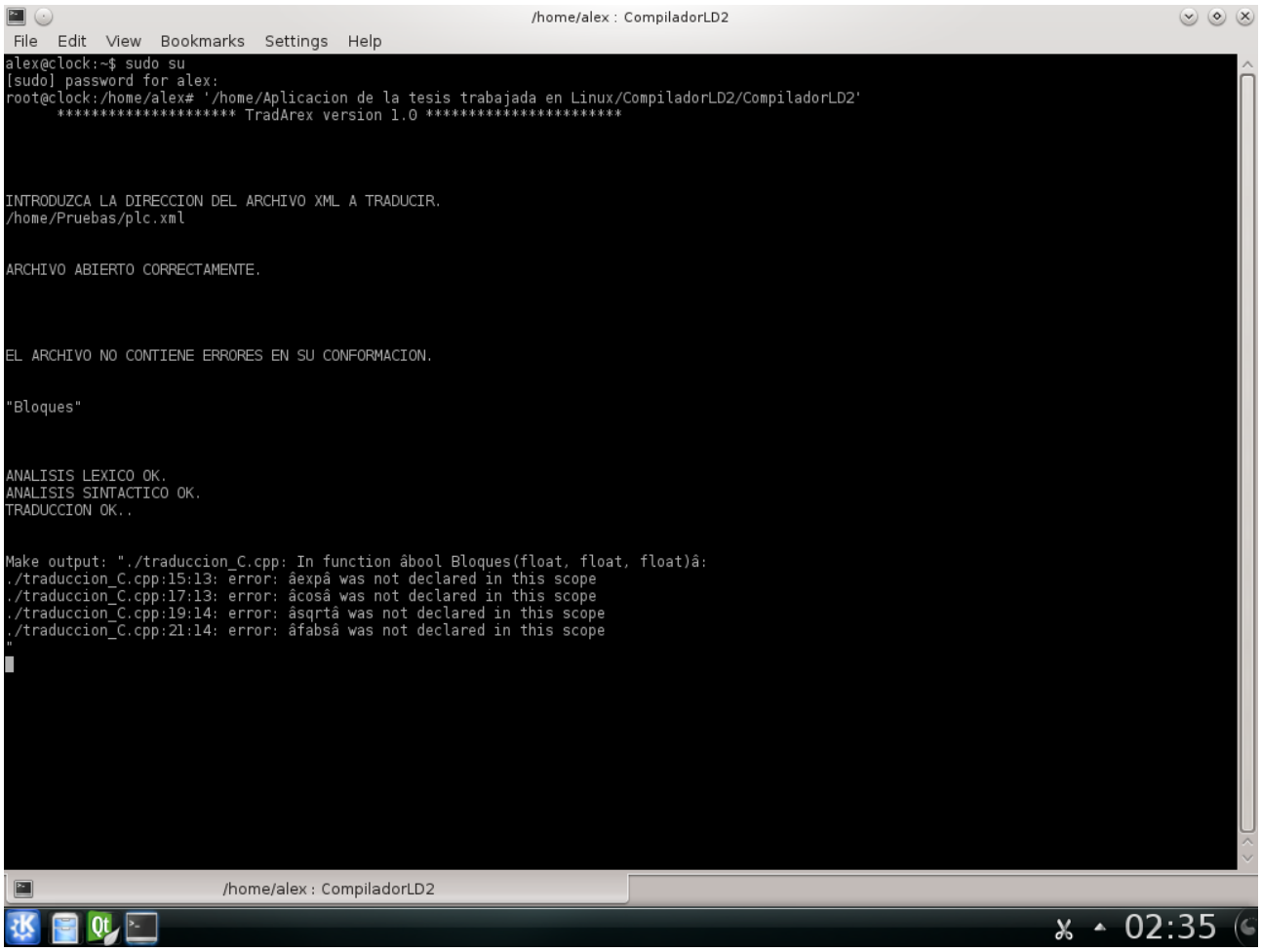

Ilustración 19. Errores semánticos

Anexo 3 Sección C

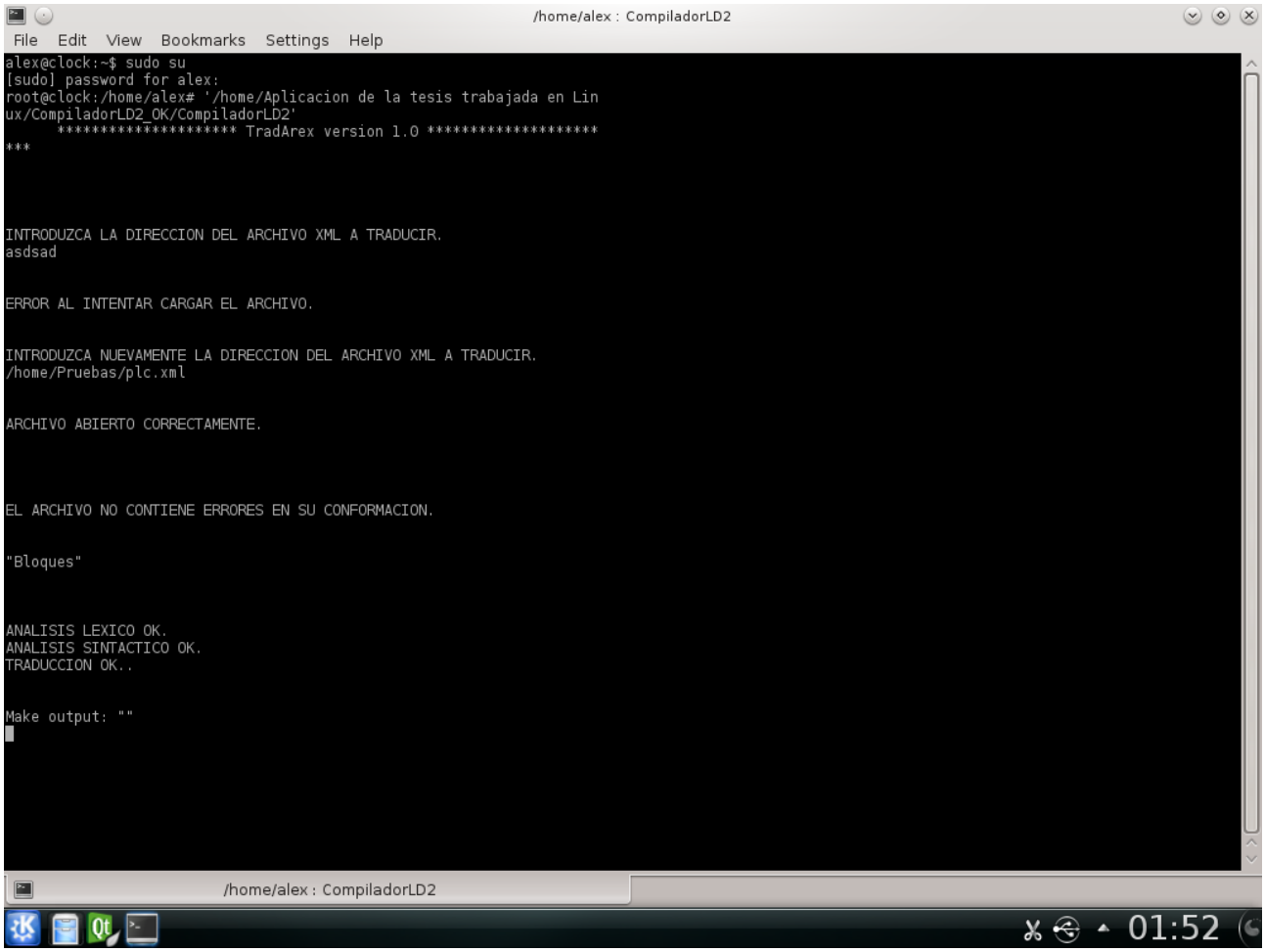

Ilustración 20. Salida final de la aplicación