**Universidad de las Ciencias Informáticas**

**Facultad 9**

**Título: Herramienta de Compilación para Métodos Numéricos.**

**Trabajo de Diploma para optar por el título de**
**Ingeniero en Ciencias Informáticas.**


**Autores:**

Abdiel Cruz Robaina.

Yisnier Guerra García.

**Tutor:**

Lic. Yusnier Valle Martínez.

**Co-tutor:**

Lic. José Ángel Lago Graverán.


Julio, 2008

Año del 59 aniversario de la Revolución.

"Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad."

Albert Einstein.

**AGRADECIMIENTOS**

*Agradezco a mi madre por guiarme a ser la persona que soy, a ti te debo este y todos mis futuros logros, gracias MAMÁ.*
*A la Revolución Cubana.*
*A mis tíos y tías Carmen y Martha, Orlando, Ceferino y hermanas Yadira y Zuleika por su cariño.*
*A Arianna por su incomparable colaboración y por todo el amor que me brinda.*
*A mi padre Tatón quien me ha dado apoyo incondicional siempre.*
*A quien en la distancia aun sigue siendo mi tutor tanto personal como profesional, Carlos.*
*A Frey Ernesto por su amistad.*
*A toda mi familia que de una forma u otra ha dado su granito de arena para mi formación.*
*A Asmel y Nelson por ser los hermanos que nunca olvidaré.*
*A mi tutor, cotutor y a todos los profesores que hicieron posible este trabajo.*
*A mi más recientes familiares Urbano, Matilde, Alida, quienes también han contribuido a la realización de este trabajo.*
*A Yisnier por brindarme el privilegio de compartir esta experiencia.*
*Abdiel*

*A todos los que de una forma u otra han contribuido a mi formación profesional a lo largo de estos años quienes nos han acompañado en este largo camino, de la vida y la educación, dándonos fuerzas para salir adelante y realizar nuestros sueños.*
*A todos aquellos que hicieron posible la realización de este trabajo, especialmente a mi tutor Yusnier Valle Martínez por sus orientaciones certeras y a mi compañero de tesis y hermano Abdiel Cruz Robaina, por toda la ayuda que pudo darme y por estar presente en los momentos en los que de verdad necesité.*
*A mis compañeros de clases por estar cerca y darme la mano cuando tuve necesidad.*
*A mis amigos: Alexis, Omar, Yorjandis, Irenaldo, Pascual, Yenlys, Jenny, Yosber, Jose, Aliuska, Asmel, Nelsito, Orestes.*
*Mencionarlos a todos seria una lista interminable, pero al final eso queda en nuestros corazones.*
*Yisnier*

**DEDICATORIA**

*A mi madre Matilde, por haber confiado en mí.*
*A mis abuelos que les hubiera gustado ver en este momento.*
*A mis hermanas Yadira y Zuleika.*
*A Arianna por ser la persona que alegra mi vida día a día.*
*A mis tíos por su cariño.*
*A toda mi familia.*
*A mis amigos y compañeros de grupo.*
*Abdiel*

*A mi madre, por guiarme y apoyarme siempre que lo necesite, a ella,*
*que nunca dejó de tener esa esperanza en mí y a quien hoy le debo todo lo que soy.*
*A mi padre, quien es mi ejemplo a seguir y quien lo ha dado*
*todo porque este sueño se haya hecho realidad.*
*A mi hermana, quien es mi tesoro más preciado,*
*el mejor regalo que me han dado mis padres.*
*A mis segundos padres, Ana y Matos, quienes lo dieron todo,*
*hasta su corazón por mi, acogiéndome durante estos 5 largos años.*
*A mis abuelos, tíos y primas, por su presencia y colaboración.*
*Yisnier*

**DECLARACIÓN DE AUTORÍA**

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

| | |
|---|---|
| _____ | _____ |
| **Firma del Autor** | **Firma del Autor** |
| **Abdiel Cruz Robaina** | **Yisnier Guerra García** |
| _____ | _____ |
| **Firma del Tutor** | **Firma del Co-Tutor** |
| **Lic. Yusnier Valle Martínez** | **Lic. José Ángel Lago Graverán** |

**OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA**

**Título:** Herramienta de Compilación para Métodos Numéricos.

**Autores:** Abdiel Cruz Robaina.

Yisnier Guerra García.

El tutor del presente Trabajo de Diploma considera que durante su ejecución el estudiante mostró las cualidades que a continuación se detallan:

- Alto sentido de responsabilidad ante el trabajo.
- Elevada capacidad y habilidades para el autoaprendizaje.
- Alto sentido de la integridad, identificación y consagración ante las tareas asignadas.

Por todo lo anteriormente expresado considero que el estudiante está apto para ejercer como Ingeniero Informático; y propongo que se le otorgue al Trabajo de Diploma la calificación de Excelente (5 puntos).

Firma

Fecha

**OPINIÓN DEL USUARIO DEL TRABAJO DE DIPLOMA**

El Trabajo de Diploma, titulado: **"Herramienta de Compilación para Métodos Numéricos"**, fue realizado en la Universidad de las Ciencias Informáticas. Este centro considera que, en correspondencia con los objetivos trazados, el trabajo realizado le satisface:

Totalmente

Parcialmente en un 100%

Los resultados de este Trabajo de Diploma le reportan a la UCI los beneficios siguientes:

Se dispone de una Herramienta de Compilación en software libre, en principio concebida para ser embebida en software que necesite obtener en tiempo de real resultados de la ejecución de métodos numéricos aplicados al tratamiento de problemas específicos, pero que por su flexibilidad puede ser personalizada para satisfacer necesidades similares.

Como resultado de la implantación de este trabajo se reportará un efecto económico que asciende a ____. Y para que así conste, se firma la presente a los ____días del mes de _____del año _____

_____ _____

Representante de la entidad Cargo

_____ _____

Firma Cuño

**DATOS DE CONTACTO**

**Tutor:**
Nombre y Apellidos: Yusnier Valle Martínez
Teléfono: +53 (7) 837 – 2557.
email: yvm@uci.cu; valleml@gmail.com

Graduado de Nivel Superior (Licenciatura) en la especialidad de Ciencia de la Computación el 13 de Julio de 2004 en la Universidad de la Habana (UH). Actualmente tiene la categoría docente de Profesor Instructor. Se desarrolla como Arquitecto de Software del Grupo Sistemas de Información Geográfica (GSIG), en la Facultad 9, en la Universidad de Ciencias Informáticas.
Se desarrolla como arquitecto en el proyecto "Conceptualización de Soluciones Informáticas en Refinerías" [Información de Negocio de Refinación de Petróleo (áreas de Mantenimiento, Operaciones y Ambiente e Higiene Ocupacional), Ingeniería de Software (estudio de negocio)], con la empresa Petróleos de Venezuela S.A. (PDVSA).
Es líder del proyecto MCP (modelación de negocio, asimilación de herramientas de desarrollo en software libre), con la empresa Petróleos de Venezuela S.A. (PDVSA).
Labora también como Profesor Instructor de las asignaturas: Sistemas Operativos y Seguridad Informática a la carrera de Ingeniería en Informática en su Curso Regular Diurno (CRD) en la Universidad de las Ciencias Informáticas (UCI).

**DATOS DE CONTACTO**

**Co-tutor:**
Nombre y Apellidos: Lic. José Ángel Lago Graverán.
email: joseangel@uci.cu

Graduado con Título de Oro de Licenciado en Ciencia de la Computación, Universidad de la Habana año 2005. Durante el período de universidad integró el grupo UHSIS, en el cual participó como desarrollador de cuatro productos. Fue integrante del grupo de Geometría Computacional de la facultad de Matemática-Computación. Ha participado en varios eventos nacionales e internacionales, en algunos de ellos ha publicado trabajos. Actualmente trabaja como profesor en la UCI, en la cual ha impartido las asignaturas de Matemática Numérica y Gráfico por Computadoras, además de estar vinculado a la producción.

**RESUMEN**

El trabajo que se presenta a continuación se basa el desarrollo de una herramienta de compilación (*Numerical Compiler*) para métodos numéricos, con el fin de ser incluida como parte de un software de simulación de procesos químicos, así como otras aplicaciones que requieran del uso de estos métodos. El compilador está desarrollado en el lenguaje Python, auxiliándose en las bibliotecas de Ply, las cuales son una traducción de las tradicionales herramientas de construcción de compiladores LEX y YACC para el lenguaje antes citado.

El desarrollo de *Numerical Compiler* está sustentado por la metodología descrita por el Proceso Unificado de Desarrollo de Software (RUP), utilizando las técnicas de modelación establecidas por el Lenguaje Unificado de Modelado (UML).

Los resultados más relevantes del trabajo lo constituyen el estudio de los procesos de compilación realizados, y dentro de los mismos la búsqueda y utilización de nuevas representaciones para la generación de código intermedio; además del desarrollo de una aplicación bajo licencia GNU/GPL, con la capacidad de implementar métodos numéricos.

**PALABRAS CLAVES**

Métodos Numéricos, Compilador, Numerical Compiler, Fases de Compilación, Lexer, Parser, Análisis Lexicográfico, Análisis Sintáctico, Análisis Semántico, Formas Intermedias, Árbol de Sintaxis Abstracta, Notación Polaca, Grafo Dirigido. Grafo Acíclico Dirigido.

## ÍNDICE DE CONTENIDO

**ÍNDICE DE TABLAS**

**ÍNDICE DE FIGURAS**

## INTRODUCCIÓN

Como ciencia estructurada y rigurosa, la Matemática Numérica es relativamente joven pues data solamente de los siglos XIX y XX, pero sin descartar que se tienen registros de utilización de métodos de aproximación y cálculo de raíces desde el siglo III a. n. e, donde sobresalen los trabajos de Arquímedes en la cuadratura del círculo, utilizando polígonos inscritos y circunscritos donde obtuvo la aproximación de 3+10/71 <π <3+1/7. (Manuel Álvarez)

Otro ejemplo de la aplicación de estos métodos es la publicación en 1614 de las tablas de logaritmos de las funciones trigonométricas de los ángulos de 0 a 90 grados, con 8 cifras exactas, hechas por el holandés Neper. Como continuación al trabajo de Neper en 1628 el escoses Briggs y el holandés Vlacq publicaron las tablas de logaritmos decimales de los números del 1 al 100 000, calculadas con 10 cifras decimales exactas. (Manuel Álvarez)

A principios del siglo XVIII aparece otro gran aporte a lo que se denomina hoy Matemática Numérica, y fue la teoría del cálculo de diferencias finitas fundado por los ingleses Taylor y Stirling la cual constituye la base teórica de algunos métodos numéricos.

Con el surgimiento de la computación y el desarrollo alcanzado en dicha rama hoy en día se ha impulsado grandemente el desarrollo de la Matemática Numérica puesto que estas poderosas herramientas han ayudado a utilizar en la práctica innumerables métodos numéricos que hasta ese momento su alcance no pasaba más allá del ámbito teórico, así como el desarrollo de nuevos métodos.

Dentro de este desarrollo de las tecnologías informáticas surgió uno de los primero compiladores para el cálculo numérico FORTRAN (Formula Translator).

En los primeros tiempos de la informática cualquier cálculo que implicara la evaluación de fórmulas matemáticas había de hacerse mediante complicados programas, que traducían esas fórmulas al lenguaje del ordenador, muy primitivo. FORTRAN es un lenguaje de alto nivel, orientado a facilitar el trabajo del usuario con cálculos exhaustivos, y que permite escribir el código del programa de manera casi idéntica a como se escriben las fórmulas en un papel.

Ejemplo.

```
program ejemplo
 real a, b, c
 a=1
 b=a+1
print *, a,b
c=a+b+1
print *,'c es igual a=',c
end
```

A diferencia del lenguaje BASIC, que es un lenguaje interpretado, FORTRAN es un lenguaje compilado, esto significa que, una vez escrito el programa, éste ha de ser traducido en bloque al

lenguaje máquina, o sea, el lenguaje que entiende el procesador del ordenador, mediante un proceso de compilación.

Un proceso de compilación no es más que la acción realizada por un software, el cual toma como entrada un código fuente y le realiza los análisis lexicográficos, sintácticos, semánticos a demás de una traducción a un nuevo código que será el resultado de dicho proceso.

En la actualidad existen diferentes tipos de compiladores, que dan solución y soporte a diversos problemas planteados, estas soluciones van desde la compilación de diversos lenguajes de programación a un determinado lenguajes para su interoperabilidad, a la generación de código máquina para una arquitectura de hardware determinada, así como para la creación de programas de más alto nivel dentro una plataforma de desarrollo.

Tanto ha sido el alcance que han tomado las aplicaciones de compilación que hoy se pueden ver como parte de aplicaciones de mayores propósitos como son aplicaciones para la simulación de procesos químicos.

Con la simulación de procesos químicos se persigue modelar la dinámica de estos procesos, de forma tal que puedan ser evaluados y así obtener información de gran utilidad para tomar decisiones sobre los procesos, sin tener que experimentarlo en la práctica. El uso de la simulación trae consigo grandes beneficios e incrementa la seguridad en el ámbito donde se enmarcan los procesos (Refinerías, Industrias Petroquímicas, Industria azucarera, etc.).

Según empresas pioneras desarrolladoras de simuladores de procesos químicos a nivel mundial, una de las herramientas que debe tener un simulador de este tipo es un módulo que brinde al usuario la posibilidad de implementar sus propios métodos para el cálculo numérico, así como experimentar con nuevos modelos termodinámicos. Software como Hysys y ProII destinados a propósitos generales u otros como Hextran, Inplant , VisualFlow, PipePhase, destinados a tareas más especificas como la simulación de sistemas de transferencia de calor, o utilizados para la emulación y simulación de tuberías (redes de flujo); son ejemplos de aplicaciones que afirman el planteamiento expuesto anteriormente.

Esta problemática surge como parte de una de las investigaciones que se realizan dentro del proyecto de Conceptualización de Soluciones para Refinerías desarrollado en la facultad 9 de la Universidad de las Ciencias Informáticas.

En la Universidad de las Ciencias Informáticas, específicamente en la Facultad 9 se encuentra un Polo Productivo llamado Simulación de Procesos, en el cuál se desarrolla un software de simulación de procesos químicos dirigido a la industria azucarera. Se visualiza que en este polo se desarrolle un simulador que pueda responder a las necesidades de cualquier industria química. Por todo lo antes

expuesto se hace necesario el desarrollo de un compilador, como una herramienta que forme parte de un simulador de procesos, que sea robusto, confiable y eficiente, de forma tal que brinde una amplia gama de facilidades para el diseño e implementación de métodos numéricos.

La situación problemática anteriormente expuesta permite plantear el siguiente problema científico: ¿Cómo diseñar un compilador en software libre para métodos numéricos?

El anterior problema tiene como objeto de estudio los procesos de compilación en software libre y el análisis de los métodos numéricos utilizados en la modelación de procesos químicos. El cual da paso al siguiente campo de acción:

- Métodos numéricos.
- Herramientas de programación para el diseño de un compilador en software libre.

El presente trabajo tiene como Objetivo general: Diseñar un compilador en software libre para métodos numéricos.

De lo anteriormente expuesto se derivan los siguientes objetivos específicos:

- Explicar los fundamentos sobre los métodos numéricos y herramientas de programación numéricas para un compilador en software libre.
- Explicar los fundamentos sobre las tecnologías actuales que facilitan el diseño de compiladores en software libres.
- Describir las propuestas de compiladores para métodos numéricos.
- Diseñar un compilador en software libre para métodos numéricos.
- Desarrollar un compilador en software libre para métodos numéricos.
- Valorar la propuesta mediante un estudio de factibilidad.

Para darle cumplimiento se realizan las tareas que se mencionan:

- Explicación de los fundamentos sobre los métodos numéricos y herramientas de programación numéricas para un compilador en software libre.
- Explicación de los fundamentos sobre las tecnologías actuales que facilitan el diseño de compiladores en software libres.
- Descripción de las propuestas de compiladores para métodos numéricos.
- Desarrollo de un compilador en software libre para métodos numéricos.
- Diseño de un compilador en software libre para métodos numéricos.
- Valoración de la propuesta mediante un estudio de factibilidad.

Para la realización de este trabajo se utilizaron diferentes métodos para el desarrollo de la investigación

Los métodos utilizados fueron los siguientes:

- Métodos teóricos:
    - Dialéctico-Materialista: cuya esencia está determinada por las fuentes teóricas y científicas y por las categorías fundamentales del movimiento, del espacio y del tiempo.
    - Analítico-Sintético: su esencia es la investigación de los objetos separados del todo, para someterlas a estudio para captar las particularidades, en la génesis y desarrollo del objeto. Para reconstruir en el pensamiento toda la variedad de las mutuas vinculaciones del objeto como un todo.
    - Inductivo-Deductivo: se define como la utilización de la lógica para hacer definiciones, que pueden ir de lo general a lo específico o viceversa.
    - Histórico-Lógico: evalúa un objeto o suceso a través de la cronología desde su origen, desarrollo y decadencia.
- Los métodos empíricos:
    - Revisión de documentación: consiste en la consulta de documentación auxiliar, sobre uno o varios temas.
    - Investigación Científica: está basado en la realización de estudios bajo un marco científico acerca de un tema.

El trabajo de diploma se ha estructurado del siguiente modo: Introducción, Desarrollo, Conclusiones, Recomendaciones, Bibliografía y Anexos.

El Desarrollo se organiza en capítulos:

El Capítulo 1 explica los fundamentos sobre los métodos numéricos y procesos de la compilación. Se abordan los principales métodos numéricos utilizados en la modelación de procesos químicos, exponiendo algunas ventajas y desventajas sobre la utilización de los mismos. Además se expone un análisis sobre el proceso de compilación, su estructura y fases.

El Capitulo 2 expone una fundamentación de las tecnologías actuales que facilitan el diseño de compiladores en software libre. En él se justifican la utilización de todo el instrumental tecnológico utilizado en el desarrollo de *Numerical Compiler,* así como las metodologías y lenguajes utilizados.

El Capitulo 3 describe la propuesta de solución. Donde se expone el modelo del Negocio. Además de enumerar los requisitos funcionales y no funcionales contenidos en la solución propuesta.

El Capitulo 4 se detalla el análisis y diseño del sistema propuesto además de cómo se implementa el mismo. Se detalla la relación estructural entre los objetos que interactúan, dígase diagramas de clases, diagramas de secuencia y colaboración del análisis y del diseño.

El Capitulo 5 valora la propuesta mediante un estudio de factibilidad, donde de realiza un cálculo del tiempo de desarrollo del software mediante el ESTIMAC, software que realiza los cálculos basados en el método de estimación por puntos de casos de usos. Además se calcula el costo que tendría el desarrollo de la aplicación.

Las Conclusiones muestran de forma breve y precisa el cumplimiento de los objetivos del trabajo y se abordan los principales resultados obtenidos.

Las Recomendaciones proponen las acciones a seguir para mejorar la herramienta propuesta.

Las Bibliografía organiza las diferentes referencias bibliográficas utilizadas para la elaboración del trabajo.

Los Anexos registran los diferentes documentos complementarios al trabajo que servirán para una mayor comprensión del mismo.

La presente tesis de investigación y desarrollo de una herramienta de compilación contribuye a ampliar aún más el cúmulo de aplicaciones dentro del paradigma del software libre, para la eliminación de la brecha digital.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## INTRODUCCIÓN

El presente capítulo tiene como objetivo abordar elementos teóricos básicos en lo concerniente al uso de herramientas de programación para la creación de un compilador de métodos numéricos en software libre. Se destacan los principales métodos numéricos para la solución de ecuaciones diferenciales ordinarias (EDO), integración numérica y cálculo de raíces, utilizados en la simulación de procesos químicos, así como los principios de funcionamiento de un compilador, mostrando las diferentes fases contenidas dentro del proceso de compilación.

## 1.1 MÉTODOS NUMÉRICOS

La Matemática Numérica, es una rama de la matemática, en la cual el objetivo no es el estudio de un ente matemático en particular; la Matemática Numérica tiene como propósito el desarrollo de métodos para la solución de los diversos problemas mediante una cantidad finita de operaciones numéricas. Es decir lo que le da unidad a esta rama de la matemática no es el tipo de problema que se ha de resolver sino el método que se aplicará: operaciones numéricas en cantidad finita. (Manuel Álvarez)

Los métodos numéricos son técnicas mediante las cuales es posible formular problemas matemáticos de tal forma que puedan resolverse usando operaciones aritméticas. Existe gran cantidad de tipos de métodos numéricos, y comparten una característica común: invariablemente se deben realizar un buen número de veces determinados cálculos aritméticos específicos de cada método. Son herramientas muy poderosas para la solución de problemas matemáticos llevados a los ordenadores.

Estos métodos permiten solucionar problemas de:

- Interpolación.
- Integración Numérica.
- Ajustes de Curvas.
- Raíces de Ecuaciones.
- Ecuaciones Diferenciales Ordinaria de Orden N.

Una solución para estos problemas es la utilización de los software disponibles comercialmente que se basen en la creación y ejecución de métodos numéricos. Sin embargo, realizando estudios sobre

los métodos numéricos se pueden diseñar programas propios que no requieren de gastos o costos elevados.

### 1.1.1 MÉTODOS NUMÉRICOS PARA LA SOLUCIÓN DE RAÍCES.

Esta clase de métodos, como refleja su nombre, se basa en la búsqueda de Raíces reales de Ecuaciones y Sistemas de Ecuaciones dentro de un rango de valores. Cada método tiene sus propias características e incluso existen precondiciones para la utilización de algunos.

Los métodos más sencillos y usados son:

- El Método de Regular Falsi.
- El Método de Bisección.
- El Método de Newton-Raphson.
- El Método Mejorado de Newton-Raphson.
- El Método de Las Secantes.

### 1.1.2 MÉTODOS NUMÉRICOS PARA INTEGRACIÓN NUMÉRICA.

Los métodos de integración numérica pueden ser descritos generalmente como combinación de evaluaciones del integrando para obtener una aproximación a la integral. Una parte importante del análisis de cualquier método de integración numérica requiere un estudio de cómo se comporta el error de aproximación en función del número de evaluaciones del integrando.

Los métodos que producen un pequeño error para un número reducido de evaluaciones son normalmente considerados superiores. Reduciendo el número de evaluaciones del integrando se reduce el número de operaciones aritméticas involucradas, y por consiguiente se disminuye el error de redondeo total. También, es necesario tener en cuenta que cada evaluación posee un costo de tiempo, dependiendo del integrando que puede ser arbitrariamente complicado.

Algunos métodos utilizados para la integración numérica son:

- El Método de los Trapecios.
- El Método de Simpson.
- El Método de Gauss.
- Análisis de Fourier de una función periódica.

### 1.1.3 MÉTODOS NUMÉRICOS PARA HALLAR SOLUCIONES A EDO.

El comportamiento de muchos procesos Químicos-Físicos, que varían dependiendo del tiempo, se llaman procesos transitorios, y se modelan a través de un sistema de Ecuaciones Diferenciales Ordinarias (EDO). Por lo que es necesario adquirir el conocimiento sobre los métodos de solución de este tipo de ecuaciones. (Scenna, 1999)

Existen diversas ecuaciones diferenciales físicamente significativas que no pueden ser resueltas a través de métodos analíticos, lo que conduce a recurrir a métodos numéricos para su solución.

Para la solución de EDO existen dos tipos de métodos:

Los métodos de paso simple como:

- El método de Euler.
- El método de Runghe – Kuttha.
- El método de Taylor.

Y los métodos de paso múltiples como:

- El método de Adams-Bashforth.
- El método de Adams-Moulton.
- El método predictor-corrector.

### 1.1.4 VENTAJAS Y DESVENTAJAS DE LOS MÉTODOS NUMÉRICOS.

El uso de métodos numéricos en la resolución de problemas presenta ventajas significativas:

- Eventualmente son más eficientes que los métodos directos para sistemas de un elevado orden.
- Generalmente son más simples de programar que los métodos directos.
- Puede aprovecharse una aproximación a la solución, si tal aproximación existe.
- Utilizan menos recursos que los métodos directos a la hora de realizar los cálculos en un ordenador.

Por otro lado, es necesario tener en cuenta determinadas desventajas que presenta su uso:

- Aún cuando la convergencia esté asegurada, puede ser lenta y, por lo tanto, los cálculos requeridos para obtener una solución particular no son predecibles.

## 1.2 PROCESO DE COMPILACIÓN.

Entre el lenguaje natural de los seres humanos y las computadoras existen muchísimas diferencias, debido a que los seres humanos utilizan lenguajes basados en un conjunto de símbolos muy diversos y complejos en el cual intervienen disímiles componentes de la lengua (emisor-receptor-mensaje-

codificador-descodificador-ruidos-canales, y otros que influyen en la comunicación: su cantidad y su calidad). La computadora aunque utiliza la mayoría de estos componentes del lenguaje, solamente en esencia es un medio tecnológico que utiliza el lenguaje binario: ceros y unos, dejando fuera componentes de carácter subjetivo que tienen que ver con el hombre: sus intereses, motivaciones, necesidades, sentimientos que influyen en la comunicación y en la gama de su interpretación de los objetos y fenómenos.

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

Cada acción que realiza el hombre sobre una computadora se traduce a una secuencia de comandos, que son ejecutados por programas para dar la respuesta solicitada por el usuario. A los programas que realizan la traducción del lenguaje humano al lenguaje binario se les llama traductores, Figura 1.1, y más específicamente compiladores.

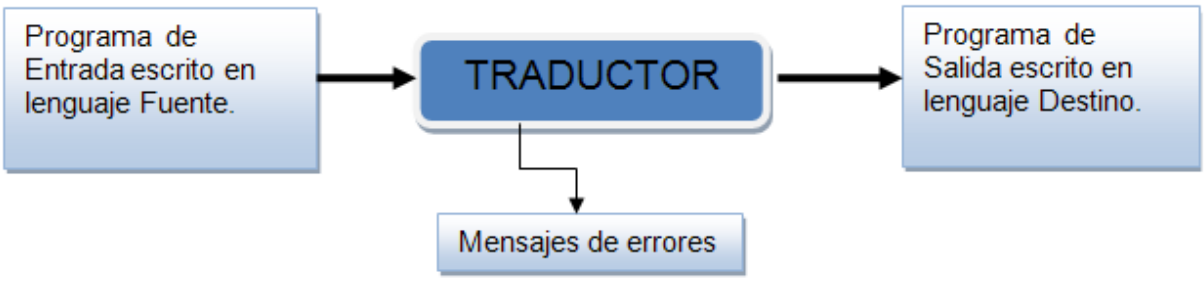



**Figura 1. 1: Esquema de un Traductor.**

El término de traductor engloba tanto a los compiladores como a los intérpretes, Figura 1.2.

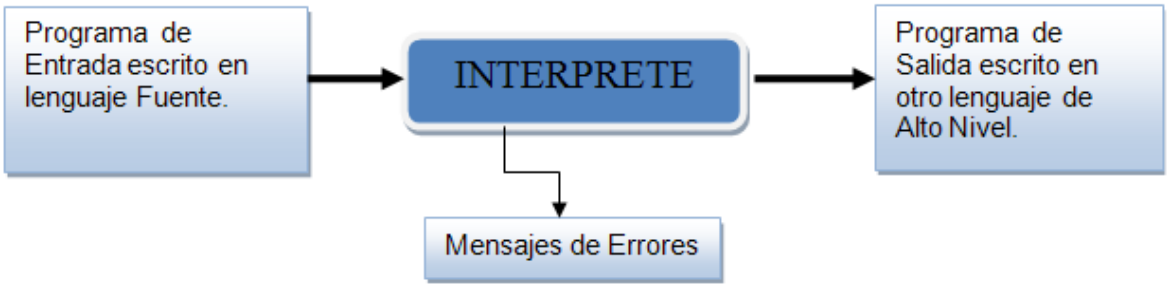



**Figura 1. 2: Esquema de un Intérprete.**

En la década de los 50, se consideraba a los traductores como programas muy difíciles de escribir. Un ejemplo de la anterior afirmación se refleja en la creación del primer compilador, Figura 1.3, de Fortran (Formula Translator), que necesitó para su implementación el equivalente a 18 años de trabajo individual lo cual realmente no se tardó tanto puesto que el trabajo se desarrolló en equipo. Objetivamente la aplicación de la teoría de autómatas y lenguajes formales son significativos avances

al desarrollo de traductores, erradicando problemas existentes antes de su aplicación. Sin embargo, hoy día un compilador básico puede ser el proyecto fin de carrera de cualquier estudiante universitario de Informática.

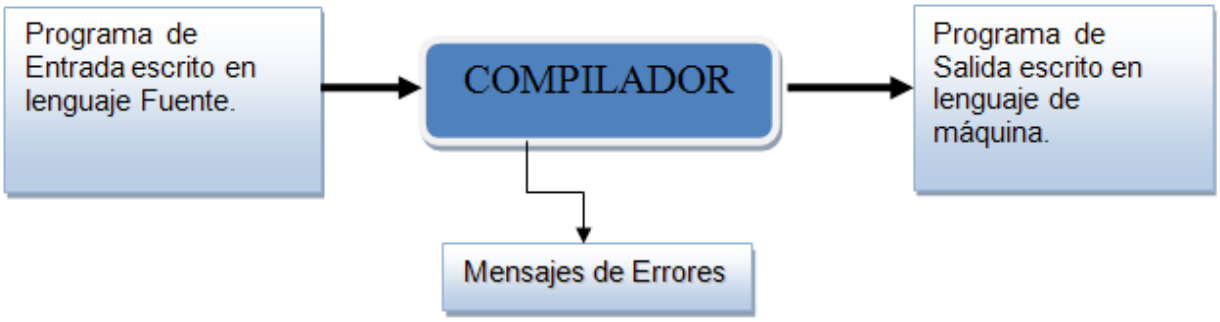



**Figura 1. 3: Esquema de un Compilador.**

### 1.2.1 CONCEPTOS BÁSICOS

DEFINICIÓN 1.2.1 Se denomina alfabeto (∑) a un conjunto arbitrario, finito y no vacío, de símbolos. Entre los alfabetos más comunes podemos mencionar el alfabeto binario {0, 1}, el alfabeto latino {a, b, c, d, e, f, g, . . . }, etc.

DEFINICIÓN 1.2.2 Una cadena vacía no es más que una cadena sin elementos, y se denota ε.

DEFINICIÓN 1.2.3 Una cadena sobre un determinado alfabeto ∑ se define como sigue:

1. ε (cadena vacía) es una cadena sobre ∑.
2. Si x es una cadena sobre ∑, y a Є∑ entonces xa es una cadena sobre ∑, para
3. todo a Є∑.
4. Solo pueden obtenerse cadenas sobre ∑, aplicando 1 y 2.

DEFINICIÓN 1.2.4 Si x y y son cadenas sobre ∑, entonces xy es una cadena sobre ∑, que se denomina concatenación de x y y.

DEFINICIÓN 1.2.5 El reverso de una cadena x, se denota $x^R$, es la cadena que se obtiene escribiendo los símbolos de x en sentido inverso.

DEFINICIÓN 1.2.6 Un lenguaje sobre ∑ es un conjunto de cadenas sobre dicho alfabeto que obedece una cierta regla de formación.

DEFINICIÓN 1.2.7 El conjunto ∑* denota todas las cadenas sobre ∑, incluyendo la cadena vacía, y se denomina Clausura de ∑.

DEFINICIÓN 1.2.8 Definición 0.1.8 El conjunto ∑+ se denomina Clausura Positiva de ∑, y se define como sigue:

$$\Sigma^{+} = \Sigma^{*} - \mathcal{E}$$

DEFINICIÓN 1.2.9 Una gramática es un cuádruplo G = {N, ∑, P, S} donde:

N: Conjunto finito no vacío de símbolos no terminales.

∑: Conjunto finito no vacío, disjunto de N, de símbolos terminales.

P: Conjunto de Reglas de Producción de la forma:

$$(N \cup T)^{*} N (N \cup T)^{*} \longrightarrow (N \cup T)^{*}$$

S: Axioma o símbolo distinguido $(S \in N)$

DEFINICIÓN 1.2.10 Sea G = {N, ∑, P, S} una gramática, la relación ⇒(deriva directamente) se define como sigue:

Si $\alpha\beta\lambda\epsilon(N \cup \Sigma)^{+}\, y\ \beta \longrightarrow \delta\ \epsilon\ P$ entonces $\alpha\beta\lambda \Longrightarrow \alpha\delta\lambda$.

La derivación de longitud k que denotamos $\overset{k}{\Rightarrow}$ significa que si $\alpha \overset{k}{\Rightarrow} \beta$, entonces existe una secuencia de cadenas $\alpha = \alpha_0, \alpha_1, \ldots, \alpha_k = \beta$ $\alpha =$ $\Rightarrow$ $\Rightarrow \ldots \Rightarrow$ $=$ . (Partiendo de alfa se obtiene beta después de k derivaciones).

La relación $\overset{+}{\Rightarrow}$ se define como la clausura positiva de $\Rightarrow$, equivalente a $\overset{k}{\Rightarrow}$ con $k \geq 1$. (Derivando una o más veces).

DEFINICIÓN 1.2.11 Lenguaje generado por una Gramática se define como sigue:

$$L(G) = \{ w \mid w \in \Sigma^{*} \text{ y } S \overset{*}{\Rightarrow} w \}$$

DEFINICIÓN 1.2.12 Se denomina Léxico (vocabulario) a un conjunto de palabras que forman parte de un lenguaje específico.

DEFINICIÓN 1.2.13 Se denomina Sintaxis a un conjunto de de reglas necesarias para construir frases correctas en un lenguaje.

DEFINICIÓN 1.2.14 Se denomina Semántica al significado de frases generadas por la sintaxis y el léxico.

DEFINICIÓN 1.2.15 Se denomina Token o lexema a la unidad básica de un compilador, que puede representar un símbolo o conjunto de símbolos con un significado semántico.

## 1.3 FASES DEL PROCESO DE COMPILACIÓN.

El proceso de compilación, Figura 1.4, se divide en seis fases fundamentales.

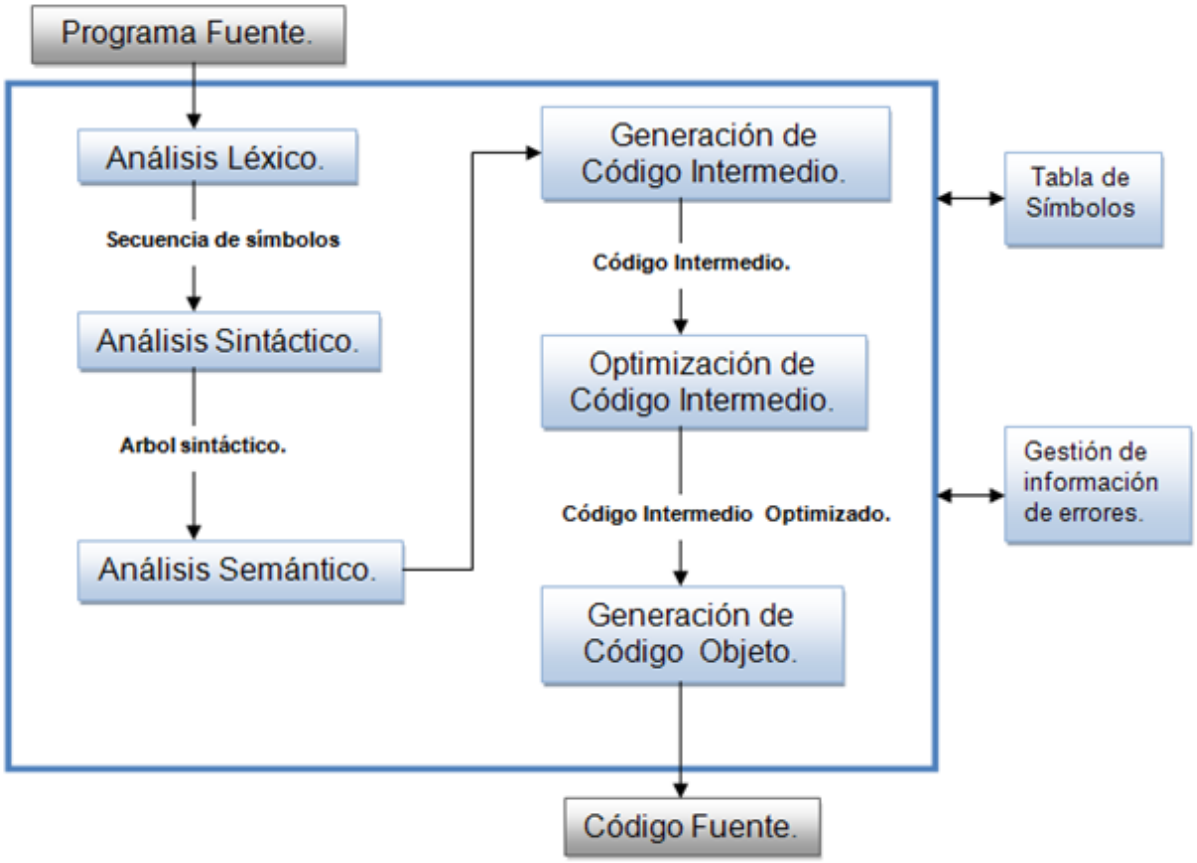



**Figura 1. 4: Fases del proceso de compilación.**

### 1.3.1 ANÁLISIS LEXICOLÓGICO

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

sintácticas simples o elementales denominadas tokens o lexemas. (Conferencia 1 Introducción al Proceso de Compilación., 2005-2006)

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

- palabras reservadas
- identificadores
- constantes numéricas y literales
- operadores

Se considera una buena práctica a cada token asignarle una estructura lexicológica, consistente en un par de la forma <tipo del token, info>, donde "tipo del token" almacena la categoría lexicológica del token, e "info" el valor del token en particular (ejemplo: el valor de la constante, nombre del identificador, etc.).

Por tanto, un scanner no es más que un traductor cuya entrada es una cadena de símbolos escritas en lenguaje natural (programa fuente) y cuya salida es una secuencia de estructuras lexicológicas o tokens.

Ejemplo: dada la siguiente cadena obtenemos una salida donde se divide por tokens cada segmento de la cadena.

```
    #No retorna valor. Token descartado
     MODULO PROCESO_GPI
     {
      float var1
     float Proceso1(int parm1,
float parm2)
       {
        return = parm1 + 10 / parm2 * 5
       }
     }
   Salida que obtenemos:
   LexToken(ID,'MODULO',3,44)
   LexToken(ID,'PROCESO_GPI',3,51)
   LexToken(LKEY,'{',4,67)
   LexToken(TYPE_FLOAT,'float',5,74)
   LexToken(ID,'var1',5,80)
   LexToken(TYPE_FLOAT,'float',6,90)
   LexToken(ID,'Proceso1',6,96)
   LexToken(LPAREN,'(',6,104)
   LexToken(TYPE_INT,'int',6,106)
   LexToken(ID,'parm1',6,110)
   LexToken(SEPARADOR,',',6,116)
   LexToken(TYPE_FLOAT,'float',6,118)
   LexToken(ID,'parm2',6,124)
   LexToken(RPAREN,')',6,130)
   LexToken(LKEY,'{',7,139)
   LexToken(RETURN,'return',8,150)
   LexToken(EQUAL,'=',8,157)
   LexToken(ID,'parm1',8,159)
   LexToken(SUMA,'+',8,165)
   LexToken(NUMERO,10,8,167)
   LexToken(DIV,'/',8,170)
   LexToken(ID,'parm2',8,172)
   LexToken(MULT,'*',8,178)
   LexToken(NUMERO,5,8,180)
   LexToken(RKEY,'}',9,189)
   LexToken(RKEY,'}',10,196)
```

### 1.3.2 ANÁLISIS SINTÁCTICO

El análisis sintáctico es un proceso en el cual se examina la secuencia de tokens para determinar si el orden de la secuencia es correcto de acuerdo a ciertas convenciones estructurales (reglas) de la definición sintáctica del lenguaje.

La entrada del analizador sintáctico o parser es la secuencia de tokens generada por el scanner. El parser analiza solamente la primera componente de cada tokens; la segunda componente se utiliza en otros pasos. (Conferencia 1 Introducción al Proceso de Compilación., 2005-2006)

Ejemplo: dada la siguiente entrada a nuestro analizador Sintáctico:

a) F= x * 2 /8.

b) G= g */ 45.

Y dadas las siguientes reglas gramaticales:

```
función_simple: id  '='  expresión
expresión: expresión  '+'  término
         | expresión  '-'  término
         | término;
término: término  '*'  factor
        | término  '/'  factor
        | factor
       ;
factor: id
       | '(' expresión ')'
       | número
       ;
```

Realizando las dos primeras fases del proceso se obtienen los siguientes resultados:

a) F = x * 2/8

El analizador lexicológico daría como resultado.

```
<id,1> ;<op_as,=>;<id,2>;<op_ar,*>;<const,2>;<op_ar,/>;<const,8>
```

La secuencia de derivaciones del analizador sintáctico sería como sigue:

$$expresión \Rightarrow término$$
$$término \Rightarrow termino/factor$$
$$factor \Rightarrow 8$$
$$término \Rightarrow término * factor$$
$$factor \Rightarrow 2$$
$$término \Rightarrow factor$$
$$factor \Rightarrow x$$

Si se logra reconocer la cadena completa según las reglas gramaticales se dice que la entrada esta sintácticamente correcta, por lo que solo quedaría revisar si semánticamente está correctamente escrita.

En caso contrario como se observa a continuación no se obtendrían los mismos resultados:

b) G= g */ 45.

El analizador lexicológico daría como resultado:

```
<id,1>; <op_as,=>;<id,2>;<op_ar,*>;<op_ar,/>;<const,45>
```

La secuencia de derivaciones del analizador sintáctico sería como sigue:

$$G \Rightarrow expresión$$
$$expresión \Rightarrow término$$
$$término \Rightarrow término/factor$$
$$factor \Rightarrow 45$$
$$término \Rightarrow término * factor$$

Al tratar de entrar en la reducción de esta regla se encontraría que g * no coincide con ninguna de las reglas gramaticales correspondientes a un término y se produciría un error sintáctico.

Al tratar de entrar en la reducción de esta regla se encontraría que g * no coincide con ninguna de las reglas gramaticales correspondientes a un termino y lanzaría un error sintáctico.

### 1.3.3 ANÁLISIS SEMÁNTICO

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

El analizador semántico revisa el programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. En ella se utiliza la estructura jerárquica determinada por la fase de análisis sintáctico para identificador los operadores y operandos de expresiones y proposiciones.

Un componente importante del análisis semántico es la verificación de tipos. Aquí, el compilador verifica si cada operador tiene operandos permitidos por la especificación del lenguaje fuente. Por ejemplo, las definiciones de muchos lenguajes de programación requieren que el compilador indique un error cada vez que se use un número real como índice de una matriz. Sin embargo, la especificación del lenguaje puede imponer restricciones a los operandos, por ejemplo, cuando un operador aritmético binario se aplica a un número entero y una cadena de caracteres.

En el siguiente ejemplo se muestra un código escrito en C++, sintácticamente correcto pero semánticamente erróneo:

```
class Error_Semántico
{
  private int a;
  private string b;
  private int c;

  public int  GetSuma()
  {
    return   c= a + b;
  }
}
```

Como se muestra anteriormente, sintácticamente el código que se presenta esta escrito correctamente pues no viola ninguna de las expresiones gramaticales establecidas dentro del lenguaje de Java, pero al compilador realizar el análisis semántico detecta la incompatibilidad de tipos de las variable presentes en la suma ( c = a + b ), observa que se intentó sumar una cadena de caracteres (string) , con un literal entero (int), lo cual es considerado un error semántico por el compilador y se produce un error semántico.

### 1.3.4 GENERACIÓN DE CÓDIGO INTERMEDIO

Después del análisis sintáctico un compilador puede generar una o varias representaciones explícitas intermedias del código fuente. Dicha representación puede servir para la realización de un análisis semántico del código fuente o para la optimización del código.

Ejemplo de estos formas intermedias son los códigos: MSIL, Java bytecode, Notación de Cuádruplos, Notación Polaca.

Las formas intermedias deben tener dos características muy importantes: debe ser fácil de producir y fácil de traducir al programa objeto.

Las mismas se clasifican en:

- *Formas Intermedias de Alto Nivel*: son aquellas que se suelen emplear en las primeras fases de análisis.
- *Formas Intermedias de Nivel Medio*: son válidas para representar un conjunto amplio de lenguajes fuente, no siendo dependientes de uno en concreto. Válidas para representar un conjunto extenso de arquitecturas de hardware.
- *Formas Intermedias de Bajo Nivel*: permiten traducir a distintos micros de una misma arquitectura, creando una dependencia a ésta.
- *Formas Intermedias Multinivel*: Aquéllas que conjugan varias de las características anteriores.

**Formas intermedias de Alto Nivel.**

Se empiezan a utilizar en la fase de análisis sintáctico. El analizador semántico recibe un árbol de sintaxis (abstracta) del analizador sintáctico. En ocasiones estas estructuras sirven de código intermedio. Otro uso más común es traducir estas formas a un código intermedio de medio o bajo nivel.

Este tipo de representaciones son:

- Dependientes de un lenguaje fuente.

- Independientes de una arquitectura destino.

Su representación interna facilita su procesamiento para tareas de:

- Inferencia y comprobación de tipos.
- Generación de código.
- Optimización de código independiente de la plataforma.

Los elementos de una representación de alto nivel, modelan las abstracciones de los lenguajes que procesan.

- Elementos condicionales.
- Representaciones de iteraciones.
- Sentencias de asignación e invocación.
- Expresiones (desde los operadores aritméticos y condicionales hasta accesos a arreglos, atributos, campos y métodos).
- Declaraciones de variables, funciones y métodos.

Las formas intermedias de medio y bajo nivel ofrecen abstracciones más cercanas a los microprocesadores.

Algunas de las representaciones intermedias de alto nivel más empleadas son Árboles de Sintaxis Abstracta (ASTs), Grafos Dirigidos y Grafos de Dependencia.

**Árbol de Sintaxis Abstracta.**

Son una simplificación de los árboles sintácticos. Representan el código mediante estructuras de árboles. Cada nodo representa una construcción del lenguaje (clase, método sentencia, expresión, etc.). Permiten una manipulación sencilla de los programas. En ocasiones se utilizan como código intermedio, de entrada al back-end. Tiene la bondad de poderse generar código intermedio (medio o bajo nivel) a partir de el.

En la Figura 1.5 se muestra un ejemplo de Árbol de Sintaxis Abstracta (AST) para la expresión:

a= 1 /(a+b)

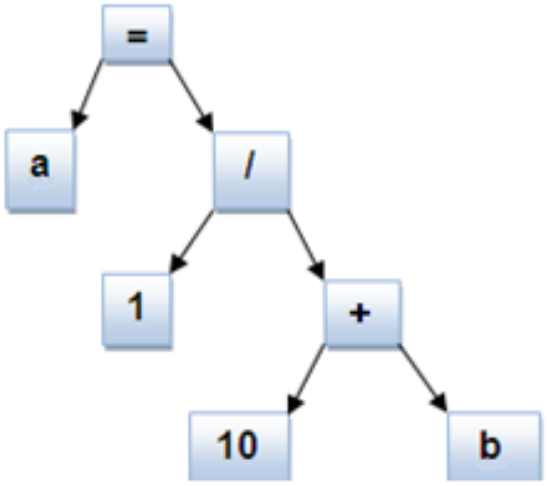



**Figura 1. 5: Representación de un AST.**

**Grafos Acíclicos Dirigidos.**

La representación de código mediante Grafos Acíclicos Dirigidos (GADs) supone una optimización de la representación de ASTs donde se reutilizan las expresiones comunes, por lo que se genera código optimizado respecto a los ASTs. Suele ser un soporte directo para la optimización de "subexpresiones comunes" independiente de la plataforma.

En la Figura 1.6 se muestra un ejemplo de un GAD para la expresión:

c=a*b/32+a*b

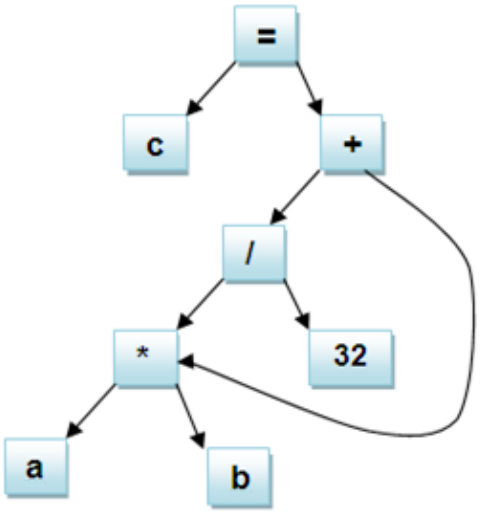



**Figura 1. 6: Representación de un Grafo Acíclico Dirigido.**

**Grafos de Dependencia**

Los grafos de dependencia se emplean en tareas de optimización de código. Coexisten con las representaciones anteriores y pueden ser de dos tipos:

Grafos de dependencia de flujo:

- Grafos dirigidos que representan dependencias entre instrucciones.
  - Nodos que representan secuencias de instrucciones contiguas.
  - Artistas (etiquetadas con identificadores) que representan saltos condicionales o incondicionales.
- Grafos de dependencia de datos: Grafos dirigidos que representan dependencias entre datos.
  - Nodos que representan instrucciones.
  - Artistas (etiquetadas con identificadores) que representan dependencia entre datos.

**Formas intermedias de nivel medio**

Las formas intermedias de nivel medio representan un gran número de lenguajes de programación sin sufrir dependencia de un lenguaje específico, y generan código a un elevado número de

plataformas; además realizan la mayoría de las optimizaciones de código independientes de la plataforma y del lenguaje. Es el formato de código intermedio más utilizado en la mayoría de los compiladores. Nexo entre el back-end y front-end (Anexo 1).

Estas formas proporcionan las siguientes abstracciones:

- Variables del código fuente.
- Variables temporales.
- El flujo de ejecución se realiza con saltos, llamadas a funciones y retornos de éstas.
- Asignación de variables.
- Obtención de direcciones (&) e indireccionamientos (*).

Las representaciones más empleadas son:

- Máquinas de pila.
- Código de tres direcciones.
- Notación polaca inversa.

**Máquinas de Pila**

El código que evalúa una expresión de una máquina de pila esta estrechamente relacionado a la notación postfija de esa expresión. (Alfred V. Aho, 1998)

Las operaciones trabajan con operandos y resultados sobre una estructura de pila. La Tabla 1.1 muestra ejemplos de esta clase de códigos para la expresión: c=a*b/32+a*b

| Código JVM: | Código MSIL (.net): |
|---|---|
| iload_1 // a | ldloc.0 // a |
| iload_2 // b | ldloc.1 // b |
| imul | mul |
| bipush 32 | ldc.i4.s 32 |
| idiv | div |
| iload_1 // a | ldloc.0 // a |
| iload_2 // b | ldloc.1 // b |
| imul | mul |
| iadd | add |
| dup | stloc.2 //c |
| istore_3 // c | |

**Tabla 1. 1: Código de Máquinas de Pilas.**

**Ventajas.**

- Generar este tipo de código es sencillo.
- Su interpretación o ejecución resulta directa empleando una estructura de pila.

**Inconvenientes.**

- Las optimizaciones de código son algo más complejas, puesto que los parámetros están implícitos en la pila.
- Aunque existen microprocesadores basados en máquinas de pila (HP3000 y Burroughs B5500), en la actualidad la mayoría están basados en registros.

**Código de tres direcciones.**

El Código de Tres Direcciones es una representación simbólica de una máquina basada en registros. El mismo consiste en una secuencia de instrucciones, cada una de las cuales tiene como máximo tres operandos, la estructura general de sus instrucciones posee la siguiente sintaxis, Figura 1.7. (Louden, 2004)


**Figura 1. 7: Sintaxis del Código de Tres Direcciones.**

Donde x, y, z son identificadores, constantes o variables temporales generadas por el compilador y op representa cualquier tipo de operador.

La generación de código de tres dimensiones requiere que el compilador genere identificadores temporales para cada componente del código de tres dimensiones.

Ventajas:

- Es más orientado a la optimización de código y traducción sencilla a micros hardware.

Inconvenientes:

- Su generación e interpretación es más compleja.
-

En la Tabla 1.2 se muestra un ejemplo de código de tres direcciones para la expresión c=a*b/32+a*b.

| Código de tres direcciones (no optimizado): | Código de tres direcciones (optimizado): |
|---|---|
| t1 ← a * b | t1 ← a * b |
| t2 ← t1 / 32 | t2 ← t1 / 32 |
| t3 ← a * b | c ← t2+ t1 |
| t4 ← t2 + t3 | |
| c ← t4 | |

**Tabla 1. 2: Código de tres direcciones.**

Como se ha visto, el Código de Tres Direcciones suele ser un formato interno del compilador mientras que los lenguajes intermedios de Máquinas de Pila son más comunes como formato externo.

**Notación Polaca Inversa**

La Notación Polaca Inversa o Postfija es una representación intermedia de nivel medio y es el resultado de un recorrido en postorden de un AST, los que pueden obtenerse directamente del AST o viceversa. Suele utilizarse como entrada a las herramientas de generación de código.

Ventaja:

- Fáciles de generar y ejecutar.

Inconvenientes:

- Difícil optimización del código y traducción a un microprocesador basado en registros.

La Tabla 1.3 muestra la Notación Postfija correspondiente a la expresión: c=a*b/32+a*b

| c | a | b | * | 32 | / | a | b | * | + | = |
|---|---|---|---|---|---|---|---|---|---|---|

**Tabla 1. 3: Notación Postfija.**

**Formas Intermedias de Bajo Nivel.**

Las formas intermedias de bajo nivel simbolizan un conjunto de lenguajes para una familia de microprocesadores de una misma arquitectura que permiten realizar la máxima optimización de código para un microprocesador concreto. Es la última (y opcional) representación de código intermedio de un compilador optimizador. La semántica de sus instrucciones es cercana a la traducción de una instrucción destino por cada instrucción origen y se especializa para un microprocesador específico, de modo que sea la más rápida posible, a través de la optimización a nivel de registro.

Todas se basan en lenguajes con registros simbólicos, muy cercanos a la arquitectura destino.

**Formas Intermedias con Múltiples niveles de Abstracción.**

Las formas intermedias con múltiples niveles de abstracción son representaciones intermedias que poseen abstracciones de distintos niveles de abstracción. Un ejemplo lo constituye el lenguaje de nivel intermedio de la máquina virtual de java (JVM) que posee representaciones de alto nivel, tales como la manipulación y el acceso indexado de arreglos multidimensionales.

Este criterio se suele emplear para facilitar la tarea de optimización del código, sin embargo, subir el nivel de abstracción de la forma intermedia, representa una dependencia del lenguaje de alto nivel en este caso, de Java; y bajar el nivel conlleva a una dependencia de una arquitectura específica.

Para no sufrir dependencias, la mayoría de los compiladores optimizadores emplean varias representaciones intermedias.

Los compiladores de HP para su arquitectura PA-RISC realizan el siguiente proceso:

1. Traducen el programa fuente (C, C++, Fortran y Pascal) a una representación de alto nivel (AST).
2. El AST es optimizado mediante transformaciones por el HLO (High Level Optimizer) obteniendo un GAD.
3. La salida del HLO es traducida a UCode (lenguaje de pila), siendo esta la salida del front-end.
4. De los distintos back-end existentes para HP, Figura 1.8, el de la arquitectura RISC traduce Ucode a SLLIC (lenguaje de bajo nivel de PA-RISC). En este último nivel se realizan la mayoría de las optimizaciones, generando el código objeto.

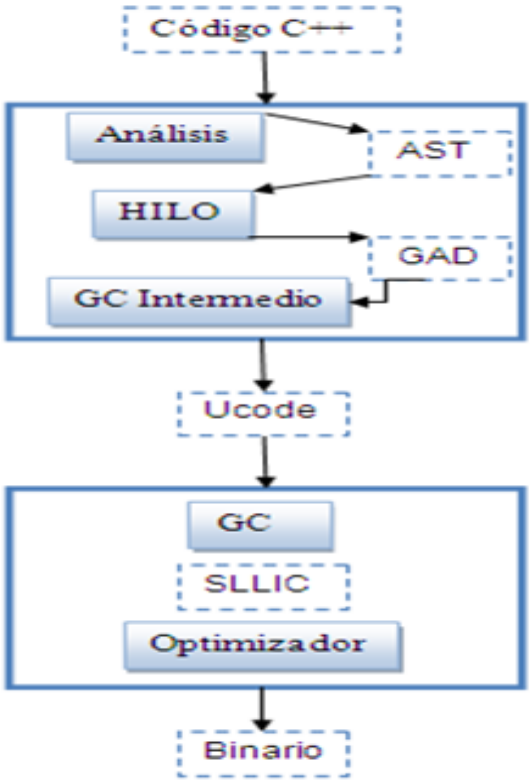



**Figura 1. 8: Representación de un compilador de HP para la arquitectura PA-RISC.**

### 1.3.5 OPTIMIZACIÓN DEL CÓDIGO INTERMEDIO

La fase de optimización se encarga de transformar el código intermedio en un nuevo código de función equivalente pero de menor tamaño o de menor tiempo de ejecución.

Algunas de las transformaciones que puede llevar a cabo la fase de optimización son:

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

### 1.3.6 GENERACIÓN DEL CÓDIGO OBJETO

La fase de generación de código objeto se encarga de generar el programa nativo usando el juego de instrucciones específico de la máquina o CPU objeto, y el formato para archivos ejecutables del

sistema operativo. Entre otras cosas, también se le asignan direcciones definitivas a las rutinas y variables que componen el programa.

### 1.3.7 GESTIÓN DE INFORMACIÓN DE ERRORES

Si los compiladores tuvieran que procesar solamente programas correctos, su diseño e implementación se simplificaría en buena medida. Pero los programadores escriben programas incorrectos frecuentemente, y un buen compilador debe ayudar al programador a localizar e identificar los errores. (Conferencia 1 Introducción al Proceso de Compilación., 2005-2006)

Los errores en un programa pueden clasificarse en 4 grandes grupos:

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

**Errores Lexicológicos**.

Los errores lexicológicos son aquellos que ocurren cuando un símbolo es analizado por el scanner y este no encuentra ningún token válido, por lo que informa que ha ocurrido un error lexicológico.

Ejemplos de ello puede ser cuando se escribe mal un identificador: ID@1

Si el lenguaje dentro del que será definido este identificador solo establece que un identificador está constituido por letras y números, ocurriría un error al no poder el analizador léxico encontrar un token válido para el símbolo '@'.

**Errores Sintácticos**.

Los errores sintácticos son detectados durante el análisis sintáctico y se producen cuando los tokens no cumplen con las reglas gramaticales del lenguaje.

Ejemplo de ello puede ser cuando se escribe una expresión aritmética con paréntesis no balanceados:

F= (x+ (6 – 7).

**Errores Semánticos.**

Estos errores se producen durante el proceso de análisis semántico, detectándose al comprobar que cada operando involucrado en las operaciones sea de un tipo permitido por la misma.

Ejemplo de ello puede ser si se trata de sumar dos valores: un valor numérico y una cadena de caracteres:

1+ QWERTY = ?

Como se observa, sintácticamente está bien escrita la instrucción anterior, pero a la hora de realizar la acción, el analizador semántico de la operación suma al comprobar los tipos involucrados detecta que la suma entre un número y una cadena de caracteres no está permitida, por lo que ocurriría un error semántico.

**Errores lógicos.**

Los errores lógicos están relacionados con el cumplimiento de condiciones en un ambiente dado, o con redundancia en el código.

Ejemplo de ellos pudieran ser que se hace referencia a una variable no declarada aún, o se escribe una sentencia repetitiva donde la condición de parada nunca se cumpla, lo cual conlleva a los llamados ciclos infinitos.

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

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

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

**CONCLUSIONES.**

En este capítulo se han abordado elementos teóricos básicos relativos a métodos numéricos, los cuales han permitido explicar las características fundamentales que describen el comportamiento de estos para el cálculo de raíces, solución de ecuaciones diferenciales ordinarias e integración numérica, además de las ventajas su utilización como herramientas de aproximación para el cálculo. Se abordaron también las características fundamentales de las diferentes fases del proceso de compilación, así como los distintos tipos de formas o representaciones intermedias que se utilizan.

# CAPITULO 2: TENDENCIAS Y TECNOLOGÍAS ACTUALES

## INTRODUCCIÓN

El presente capítulo tiene como objetivo abordar elementos fundamentales sobre las tecnologías actuales que permiten el diseño de compiladores. El estudio de las tendencias actuales de la tecnología de la informática brinda los conceptos indispensables para la explicación de los diferentes programas que se trabajan y diseñan, así como el por qué de su utilización en la sociedad moderna.

## 2.1 LAS TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES (TIC)

Las Tecnologías de la Informática y las Comunicaciones (TIC) han jugado un papel importante a lo largo de la historia de la humanidad dado el avance de las tecnologías de la información. Incluso se dice que dado el desarrollo alcanzado por las TIC se pudiera decir que existe una tercera revolución industrial. (Conferencia de Rectores de las Universidades Españolas (CRUE), 2004)

Las TIC son incuestionables y están presentes en la mayoría de los procesos que ocurren dentro de la sociedad, forman parte de la cultura tecnológica que nos rodea y con la que debemos convivir. Son herramientas que amplían nuestras capacidades físicas y mentales así como las posibilidades de desarrollo social.

Hoy en día se define a las TIC como los instrumentos y procesos utilizados para recuperar, almacenar, organizar, manejar, producir, presentar e intercambiar información por medios electrónicos y automáticos. Incluimos en el concepto TIC no solamente la informática y sus tecnologías asociadas, telemática y multimedia, Figura 2.1, sino también los medios de comunicación de todo tipo: los medios de comunicación social y los medios de comunicación interpersonales tradicionales con soporte tecnológico como el teléfono, fax y otros.

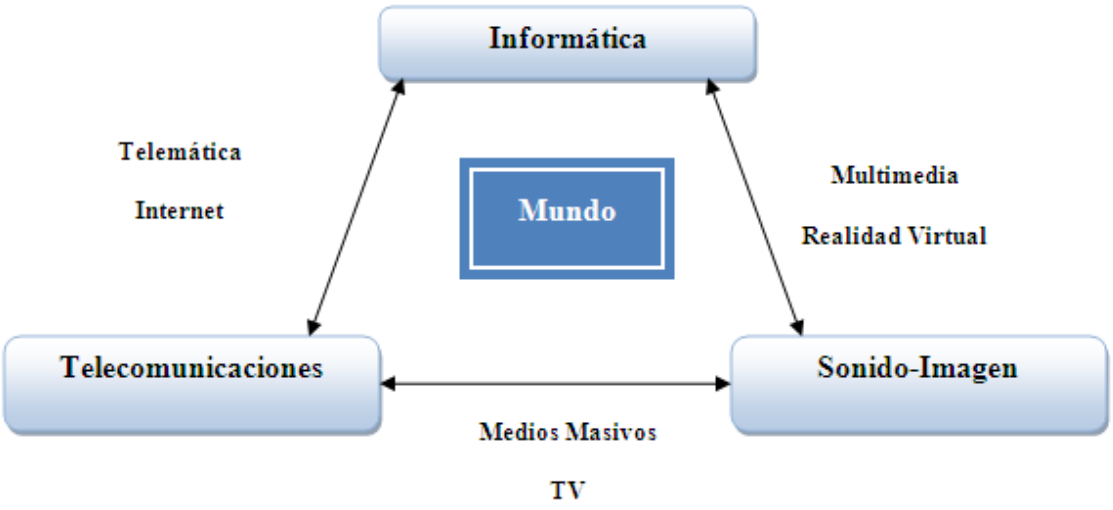



**Figura 2. 1: Representación de las TIC**

Hoy en día unas de las principales ramas donde las TIC han marcado una pauta y han creado una nueva era es en el proceso de enseñanza-aprendizaje del hombre, no se considera ya un proceso de aprendizaje donde no se haga uso del las tecnologías de la información.

## 2.2 EL LENGUAJE UNIFICADO DE MODELADO (UML) COMO SOPORTE A LA PROGRAMACIÓN ORIENTADA A OBJETOS

"El UML se ha vuelto el estándar de facto (impuesto por la industria y los usuarios) para el modelado de aplicaciones de software. En los últimos años, su popularidad trascendió al desarrollo de software y, en la actualidad, el UML es utilizado para modelar muchos otros dominios, como por ejemplo el modelado de procesos de negocios". (Dan Pilone, 2005)

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

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

**Unificado**: Unifica varias técnicas de modelado en una única.

El UML proviene de técnicas orientadas a objetos, se crea con la fuerte intención de que este permita un correcto modelado orientado a objetos. Es un lenguaje que no describe métodos o procesos, sino que los especifica. (Dan Pilone, 2005)

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

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

La ventaja principal que presenta UML es la combina de diferentes notaciones, como se muestra en la Figura 2.3.

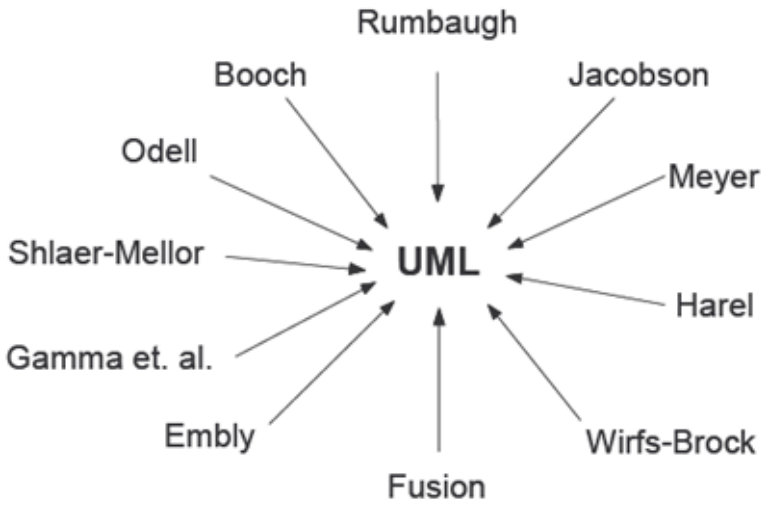



**Figura 2. 2: Integración de Notación en UML.**

Aunque también presenta inconvenientes como:

- Falta de integración con respecto a otras técnicas tales como diseño interfaces de usuario, documentación etc.
- Excesivamente complejo. Y no esta del todo libre de ambigüedades.
  El 80% de los problemas puede modelarse con el 20% de UML. (Grady Booch)

Como aspecto fundamental podemos destacar que será el lenguaje de modelado orientado a objeto estándar predominante los próximos años.

Razones:

- Participación de metodólogos influyentes.
- Participación de importantes empresas.
- Aceptación del OMG como notación estándar.

UML combina notaciones provenientes desde:

- Modelado Orientado a Objetos.
- Modelado de Datos.
- Modelado de Componentes.
- Modelado de Flujos de Trabajo.

### 2.3.1 VISUAL PARADIGM COMO HERRAMIENTA CASE PARA EL MODELADO CON UML

Visual Paradigm es una herramienta con un diseño centrado en casos de uso y enfocado al negocio que genera un software de calidad, tiene la particularidad de ser un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación. Posee capacidades de ingeniería directa (versión profesional) e inversa, modelo y código que permanece sincronizado en todo el ciclo de desarrollo, disponibilidad de múltiples versiones, disponibilidad en múltiples plataformas.

Existen varias versiones de Visual Paradigm para UML. La gratuita no permite realizar ingeniería inversa, pero permite crear diagramas y generar código a partir de ellos. Las versiones comerciales difieren entre ellas por su funcionalidad y su capacidad de integración con otras herramientas; también hay disponible un programa para acceder a las herramientas a un precio bajo si se desea emplear con fines académicos.

### ¿POR QUÉ VISUAL PARADIGM PARA UML?

Visual Paradigm para UML ha sido actualizado con rapidez en la sincronización con el nuevo desarrollo de su versión 2.1 para proporcionar un entorno de modelado visual que reúne el software y la tecnología según las necesidades de comunicación. Es un valioso producto que facilita a las organizaciones el diseño visual y el diagrama, integra y despliega sus aplicaciones empresariales de misión crítica y sus bases de datos subyacentes.

La herramienta de desarrollo de software ayuda a su equipo a sobresalir todo el modelo de construcción-despliegue software de proceso de desarrollo, y aumentar al máximo la aceleración de ambos equipos y de los individuos.

Requisitos del sistema:

- Compatible Procesador Intel Pentium III a 1,0 GHz o superior.
- Mínimo 256 MB RAM, pero se recomienda 1,0 GB.
- Mínimo 400 MB de espacio en disco.
- Microsoft Windows (98, 2000, XP o Vista), Linux, Mac OS X, Solaris o todos los demás con Java de plataformas.

Integración con los IDE

- Eclipse 3 o superior.
- IntelliJ IDEA 4 o superior (6,0 listo).
- JBuilder 9 o superior.
- JDeveloper 10.
- NetBeans 4,0 o superior.
- Sun Studio Enterprise WebLogic Workshop 8,1 o superior.

## 2.3 EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE (RUP) COMO BASE EN EL DESARROLLO DE LA SOLUCIÓN

El Proceso Unificado de Desarrollo de Software (RUP) es un Proceso de Ingeniería de Software que proporciona un enfoque disciplinado a la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es garantizar la producción de alta calidad de software que satisface las necesidades de sus usuarios finales, en un previsible calendario y el presupuesto.

RUP constituye una guía de cómo utilizar de forma eficaz el Lenguaje Unificado de Modelado (UML). Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.

RUP se agrupa en actividades de grupos lógicos definiéndose 9 flujos de trabajo principales, Figura 2.6. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo.

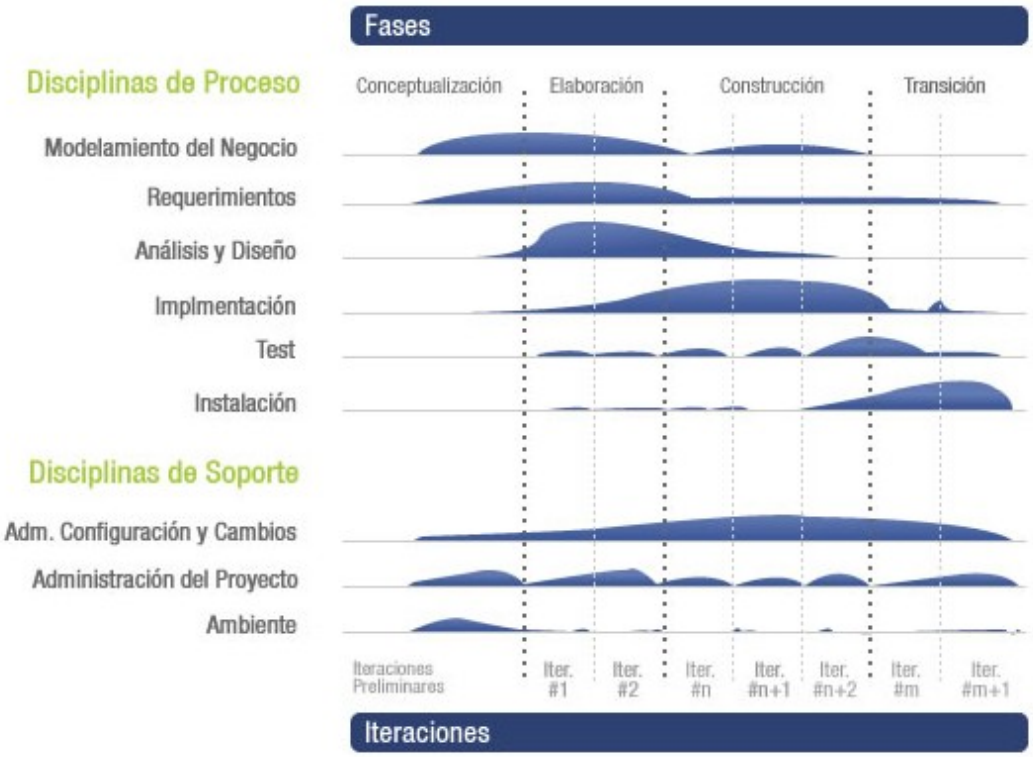



**Figura 2. 3: Fases y Flujos de Trabajo de RUP.**

## 2.4 ¿POR QUÉ PYTHON COMO LENGUAJE DE PROGRAMACIÓN?

Python es un lenguaje de programación de alto nivel y de propósito general, legible, elegante, simple y minimalista, o sea, todo aquello innecesario no hay que escribirlo (por ejemplo ;,{, }, '\n').

Python soporta estructuras de datos de alto nivel: cadenas de caracteres (strings), listas, diccionarios, etc. Presenta múltiples niveles de organización del código: funciones, clases, módulos, y paquetes, y contiene un gran número de clases de utilidades, además de que si se necesita eficiencia se pueden crear plugins en C o C++, siguiendo la API para extender o empotrar Python en una aplicación, o a través de herramientas como SWIG, Sip o Pyrex.

Python no está diseñado para la programación de bajo nivel: como la programación de sistemas operativos, drivers o kernels. Además por ser un lenguaje de demasiado alto nivel, no hay control directo sobre memoria y otras tareas de bajo nivel. Por el contrario resulta ideal como lenguaje "pegamento" para combinar varios componentes juntos, o para llevar a cabo prototipos de sistema, o la elaboración de aplicaciones cliente y para desarrollo Web, además es eficiente en sistemas distribuidos y para el desarrollo de tareas científicas, en los que hay que simular y prototipar rápidamente.

Es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación estructurada, programación funcional y programación orientada a aspectos. Otros paradigmas están soportados mediante el uso de extensiones. Python usa tipado dinámico de datos y conteo de referencia (reference counting) para el manejo de memoria. Python provee herramientas adicionales que permiten alterar la manera en que se establece la correspondencia entre la forma en que se definen los parámetros de una función y el modo en que se invoca.

Como lenguaje de muy alto nivel, presenta muchas características a su favor:

- Posee tipos de datos incorporados como arreglos flexibles y diccionarios los cuales costarían días implementar eficientemente. Esta característica hace que los códigos sean bastante escuetos, sencillos y fáciles de trasladar de una plataforma a otra.
- Una característica importante del Python es la resolución dinámica de nombres, que permite enlazar un método y un nombre de variable durante la ejecución del programa.
- Ofrece una mejor estructuración y soporte para programas extensos que el shell y a su vez mejor chequeo de sintaxis que el C.
- Permite dividir los programas en módulos que pueden ser usados por otros programas Python.
- Soporta números complejos.
- El agrupamiento de sentencias se realiza mediante el uso de sangría (indentación) en lugar de begin/end o llaves.
- No es necesario declarar los argumentos ni las variables.

## 2.5 ¿POR QUÉ ECLIPSE COMO IDE PARA LA PROGRAMACIÓN DE COMPILADORES?

Eclipse es una comunidad de código abierto cuyos proyectos están centrados en la construcción de una plataforma de desarrollo extensible. Es un IDE (Integrated Development Environment) muy aceptado por la comunidad desarrolladora en software libre muy flexible y robusto, para el que se han desarrollado varias extensiones (plugins) que permiten desarrollar en varios lenguajes.

Eclipse es bien conocido como IDE para desarrollo en Java, sin embargo soporta la mayoría de los lenguajes de programación más populares. Dentro de los plugins que se han desarrollado se encuentra Pydev, muy usado para programar en Python. (Burnette, 2005)

## 2.6 PLY COMO ANALIZADOR LÉXICO - SINTÁCTICO

Ply es una implementación en Python de lex y yacc, dos herramientas para apoyar el desarrollo de compiladores (generan lexer y parser dependiendo de ciertos parámetros: expresiones regulares para lex, y gramática para yacc).

PLY consta de dos módulos separados; lex.py y yacc.py. El objetivo principal de PLY es permanecer fiel a la forma tradicional de las herramientas de trabajo lex / yacc. Esto incluye el apoyo a LALR, así como analizar el suministro de una amplia entrada de validación, reporte de errores, y diagnósticos.

El módulo lex.py se utiliza para convertir la entrada de texto en una colección de tokens especificados por un conjunto de reglas de expresiones regulares. Yacc.py se utiliza para reconocer la sintaxis de lenguaje que se ha especificado en la forma de una Gramática Libre del Contexto. Las dos herramientas están diseñadas para trabajar en conjunto.

Algunas características:

- Es totalmente implementado en Python.
- Utiliza técnicas de análisis LR, es razonablemente eficiente y muy adecuado para el concepto más amplio de gramáticas.
- Sencillo de utilizar y proporciona muy extensa comprobación de errores.
- Proporciona la mayor parte de la norma lex / yacc incluyendo características que presentan el apoyo a las producciones vacías, reglas de prioridad, recuperación de errores y el apoyo a las gramáticas ambiguas.

## 2.7 ESTIMAC

ESTIMAC, es una herramienta que permite automatizar los cálculos necesarios para realizar la estimación de un proyecto informático basándose en el análisis de puntos de casos de uso.

Fue realizado en la Universidad de las Ciencias Informáticas por el ingeniero Henry Raúl González Brito en el curso 2005-2006 como apoyo a la actividad docente de la asignatura Ingeniería de Software.

## CONCLUSIONES

En la actualidad existe gran variedad de tecnologías para el desarrollo de aplicaciones. Utilizar una u otra debe ser una decisión que se tome después de hacer un estudio de los requerimientos de la aplicación a desarrollar en correspondencia con las potencialidades que ofrece cada tecnología. En este capítulo se resumió el funcionamiento de cada herramienta a utilizar, así como algunas de las razones del por qué fueron utilizadas.

# CAPITULO 3: DESCRIPCIÓN DE LA PROPUESTA DE SOLUCIÓN.

## INTRODUCCIÓN

El presente capítulo tiene como objetivo describir la propuesta del sistema a desarrollar, en el se describen los procesos del negocio que intervienen en la realización de las actividades vinculadas al objeto de estudio. Dada la compleja y abstracta organización de estos procesos, se decide representar los diferentes conceptos que intervienen en un Modelo de Dominio. También se exponen los requisitos funcionales y no funcionales que se tienen en cuenta para el diseño e implementación de *Numerical Compiler.*

## 3.1 DESCRIPCIÓN DE LOS PROCESOS DEL NEGOCIO PROPUESTOS

Para la descripción de los procesos del negocio que se llevan a cabo en el campo de acción de este trabajo, se toma como objeto de estudio los procesos de compilación y la definición y ejecución de los métodos numéricos que se utilizan para la simulación de procesos químicos.

El primer paso dentro del modelado del negocio es la identificación de los diferentes procesos del negocio que intervienen dentro de nuestro campo de acción.

En la simulación de procesos químicos se llevan a cabo diferentes cálculos de variables, cuyos comportamientos están definidos por ecuaciones matemáticas que pueden ir desde la evaluación de una simple ecuación lineal hasta cálculo de integrales.

Como se ha visto anteriormente algunos software de simulación como Hysys, PipiPahse y otros incorporan una herramienta de compilación para la comprobación o experimentación de cómo se comportarían los diferentes modelos utilizando nuevas definiciones matemáticas, sin tener que afectar el proceso en cuestión.

Los usuarios de estos sistemas de simulación pudieran necesitar observar cómo se comportan algunos valores dentro del software definiendo una nueva manera de calcular uno o varios valores a través de la escritura de un nuevo método. Este proceso se podría llevara a cabo utilizando hojas de cálculo y otras herramientas, pero esto traería como consecuencias tener que escribir los datos para estas hojas para posteriormente analizar los resultados, debido a la no interoperabilidad entre la hoja e cálculo y el simulador. A través de los métodos numéricos estos cálculos en un ordenador se pueden realizar en segundos, siempre y cuando el usuario esté claro de que siempre existirá un margen de error con el cual debe manejar su resultado.

## 3.2 MODELO DEL DOMINIO

Dada las descripciones de los procesos en el epígrafe anterior, se llega a la conclusión que el negocio que se está estudiando, contiene un nivel de abstracción bastante elevado, con un nivel muy bajo en la estructuración de sus procesos, lo cual conlleva a varias soluciones.

Debido a lo anterior se utilizará un modelo de dominio para la visualización de los principales conceptos presentes en el desarrollo del sistema. Con el objetivo que los usuarios, clientes y desarrolladores puedan tener un entendimiento común de los procesos del negocio y además se utilice un vocabulario común para comprender mejor el contexto en que se desenvolverá el sistema.

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

DEFINICIÓN 3.2.1 Se considera un Token o Lexema a la unidad básica de un compilador, que puede representar un símbolo o conjunto de símbolos con un significado semántico.

DEFINICIÓN 3.2.2 Se considera un Lexer, al componente del sistema que lee el código escrito por el usuario y produce como salida una secuencia de tokens que utiliza el parser para sus funciones, también realiza otras funciones, como la eliminación de los comentarios, caracteres tabs, espacios en blanco, etc.

DEFINICIÓN 3.2.3 Se considera un Parser al componente del sistema que toma como entrada una lista de tokens y que valida el orden en que estos aparecen dándole su significado semántico cada vez que se reduce una regla del mismo.

DEFINICIÓN 3.2.4 Se le considera Programa a un conjunto de instrucciones que entra el usuario y que tienen un orden lógico bien definido y que puede ser ejecutado.

DEFINICIÓN 3.2.5 Se considera una Cadena a una secuencia de símbolos de cierto alfabeto colocados uno a continuación del otro.

DEFINICIÓN 3.2.6 Un Usuario es la persona que interactúa con el sistema.

DEFINICIÓN 3.2.7 Se considera un Árbol de Sintaxis Abstracta (AST por sus siglas en inglés.) a la presentación en forma de árbol de un código.

DEFINICIÓN 3.2.8 El Código Intermedio es una interpretación que genera un compilador de las cadenas entradas al que sirve para ser trasladada o interpretada por otros sistemas.

DEFINICIÓN 3.2.9 Una Lista de Variables es el conjunto de varias variables declaradas y en uso del código fuente escrito por el usuario.

DEFINICIÓN 3.2.10 Se considera una Variable a todo aquel elemento que sea capaz de guardar una información para ser utilizada o modificada en alguna instrucción.

DEFINICIÓN 3.2.11 Se considera una Instrucción a la agrupación lógica de un conjunto de tokens definidas dentro del parser.

DEFINICIÓN 3.2.12 Se considera un Intérprete al componente del sistema o sistema externo capaz de reconocer y ejecutar el código intermedio generado por un compilador.

DEFINICIÓN 3.2.13 Se considera como Resultado al conjunto de valores provenientes de la ejecución de un programa.

DEFINICIÓN 3.2.14 Se considera un Lenguaje al conjunto de cadenas sobre un alfabeto que obedecen reglar gramaticales.

El modelo del dominio que se describe en la Figura 3.1, utilizando el Lenguaje Unificado de Modelado (UML) y más específicamente un diagrama de clases. Cada clase representa un concepto significativo para el dominio del problema.

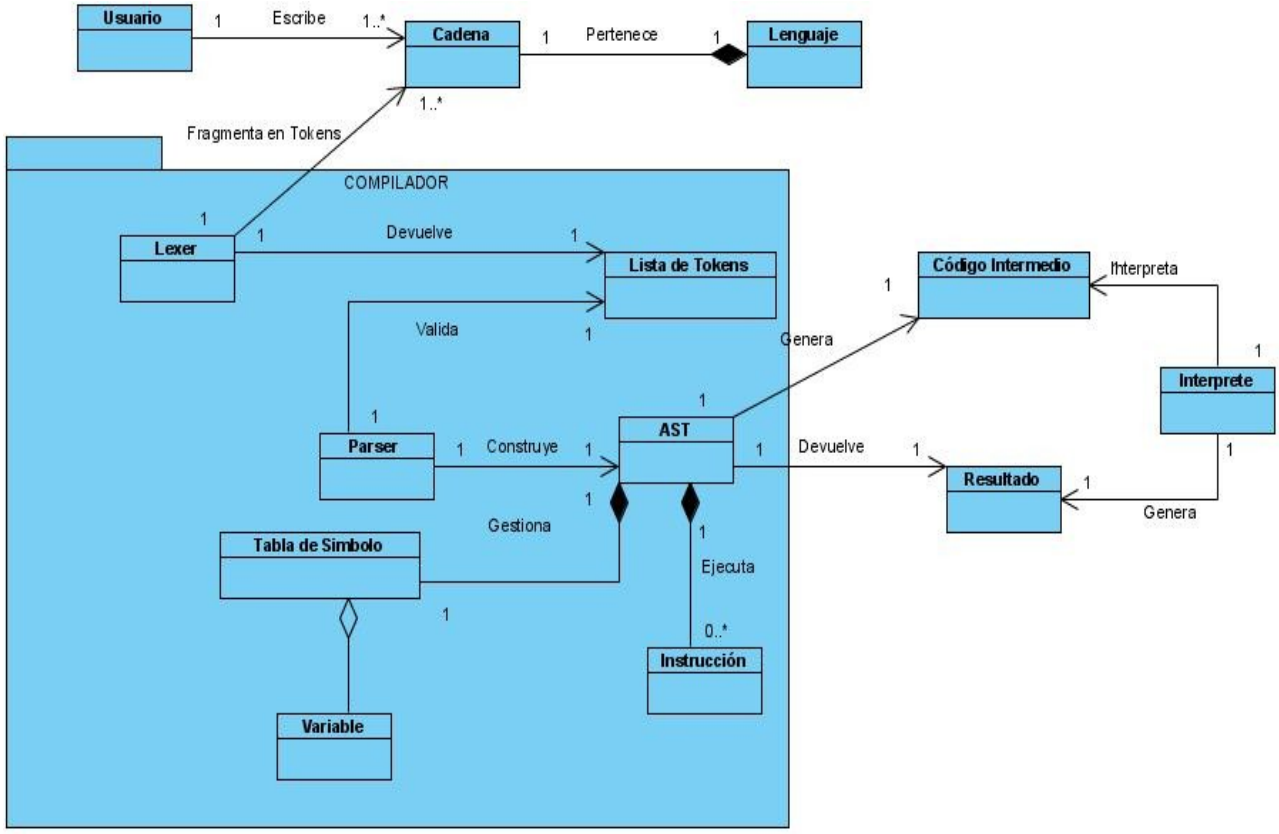



**Figura 3. 1: Diagrama del Modelo del Dominio.**

## 3.3 REQUERIMIENTOS DEL SISTEMA

"Todos querernos implementar muchos requisitos apasionantes, pero debernos ser cuidadosos. Podemos concretar requisitos inútiles, o conducir a requisitos innovadores que conducen a un producto demasiado futurista." (Lerman, 1999)

Una vez conocidos los conceptos que rodean al objeto de estudio, se debe analizar: ¿Qué debe hacer el sistema para que se cumplan los objetivos planteados al inicio de este trabajo?, para ello se enumeran, a través de requerimientos. Dentro de ellos se incluyen las acciones que podrán ser ejecutadas por el usuario, las acciones ocultas que debe realizar el sistema, y las condiciones extremas a determinar por el sistema.

### 3.3.1 REQUISITOS FUNCIONALES

R.1 El sistema debe ser capaz de evaluar las expresiones numéricas, soportando las operaciones aritméticas (Suma, Resta, Multiplicación, División, Potencia, Raíz, Resto, Expresiones con Paréntesis).

R.2 El sistema debe ser capaz de evaluar de expresiones lógicas, soportando las operaciones de comparación Menor, Mayor, Menor Igual, Mayor Igual, Igualdad, Diferencia, O lógico, Y lógico).

R.3 El sistema debe ser capaz de reconocer, analizar e interpretar las intrusiones creadas por el usuario. (IF-ELSE, FOR, WHILE, Definiciones de Métodos).

R.4 El sistema debe propiciar un mecanismo de almacenamiento para las variables que el usuario describa.

R.5 El sistema debe ser capaz de informar los diferentes errores que puedan presentarse en el compilador (Errores, Lexicográficos, Errores Semánticas, Errores Sintácticos, Errores Lógicos).

R.6 El sistema debe generar un lenguaje intermedio para después ser interpretado por los sistemas clientes. Por ejemplo código de tres direcciones, notación polaca o código XML.

### 3.3.2 REQUERIMIENTOS NO FUNCIONALES

**REQUERIMIENTOS DE USABILIDAD**

- El sistema deberá tener una interfaz amigable con una buena utilización de los elementos de diseño, con adecuada combinación de colores.

- El sistema deberá ser usado por usuarios capacitados para el uso de la herramienta y que hayan leído previamente el manual de ayuda.

**REQUERIMIENTOS DE SOPORTE**

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

**REQUERIMIENTOS DE PORTABILIDAD Y OPERATIVIDAD**

- El sistema debe ser compatible con los Sistemas Operativos: Windows 2000 NT, Windows XP y GNU/Linux.

**REQUERIMIENTOS AMBIENTALES**

**REQUERIMIENTOS DE HARDWARE DEL SISTEMA**

- Las computadoras que utilizarán el software a desarrollar deberán tener 64 MB de Memoria tipo RAM como mínimo.

**REQUERIMIENTOS DE SOFTWARE DEL SISTEMA**

- Las computadoras que utilizarán el software deben tener instalado:
- Windows 2000 NT, Windows XP Professional ó alguna distribución GNU/Linux.
- Python 2.5 o superior.

## 3.4 DESCRIPCIÓN DEL SISTEMA PROPUESTO

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

Como el sistema tiene un sólo módulo y además, un sólo rol, este será el encargado de utilizarlo donde el módulo tiene como objetivo resolver el código del usuario y brindarle los resultados que se obtendrían mediante una herramienta de compilación.

En resumen, este sistema brindara las facilidades de implementar algoritmos basados en el cálculo numérico, permitiendo la ejecución de los mismos y realizando un chequeo de errores con el fin de ayudar al usuario en su corrección.

### 3.4.1 MODELO DE CASOS DE USO DEL SISTEMA

Utilizando las facilidades que brinda el UML, se representarán los requisitos funcionales del sistema mediante un diagrama de casos de uso. Para ello hay que definir de acuerdo a lo planteado en los epígrafes anteriores, cuáles serían los actores que van a interactuar con el sistema, y los casos de uso que van a representar las funcionalidades.

Un caso de uso es una secuencia de acciones que el sistema lleva a cabo para ofrecerle algún resultado de valor para un actor... (I.Jacobson, 2000).

Los casos de usos están diseñados para cumplir los deseos del usuario cuando utiliza el sistema. (I.Jacobson, 2000)

Un actor no es parte del sistema, sino un rol que se juega dentro del sistema, que puede intercambiar información o puede ser un recipiente pasivo de información y representa a un ser humano, a un software o a una máquina que interactúa con el sistema. En este caso interactúa un único actor que se define a continuación en la Tabla 3.1.

| ACTORES | JUSTIFICACIÓN |
|---|---|
| Usuario | Representa a una persona que tiene la capacidad de realizar la programación de un método numérico para su utilización en la modelación de procesos químicos. |

**Tabla 3. 1: Actores del sistema.**

A continuación se presentan los casos de uso determinados para satisfacer los requerimientos funcionales del sistema.

Casos de Usos del Sistema.

| | |
|---|---|
| CU-1 | Compilar Código. |
| Actor | Usuario |
| Descripción | El usuario introduce el código que desea compilar y se realizan los análisis Lexicográficos, Sintácticos y Semánticos. |
| Referencia | R3,R4,R5 |

**Tabla 3. 2: Caso de Uso Compilar Código.**

| | |
|---|---|
| CU-2 | Ejecutar Código. |
| Actor | Usuario |
| Descripción | El usuario desea observar los resultados de la ejecución del código escrito por él. |
| Referencia | R1,R2,R3,R4,R5 |

**Tabla 3. 3: Caso de Uso Ejecutar Código.**

| | |
|---|---|
| CU-3 | Generar Código Intermedio. |
| Actor | Usuario |
| Descripción | El usuario después de haber compilado su código primeramente puede solicitar una forma intermedia del algoritmo programado por él. |
| Referencia | R5,R6 |

**Tabla 3. 4: Caso de Uso Generar Código Intermedio.**

| | |
|---|---|
| CU-4 | Gestionar Errores. |
| Actor | Usuario |
| Descripción | Cuando el usuario realice la compilación del código se deben gestionar los errores que se encuentren. |
| Referencia | R5 |

**Tabla 3. 5: Caso de Uso Gestionar Errores.**

El diagrama donde se representa la relación existente entre los actores y los casos de uso se presenta a continuación en la Figura 3.2.

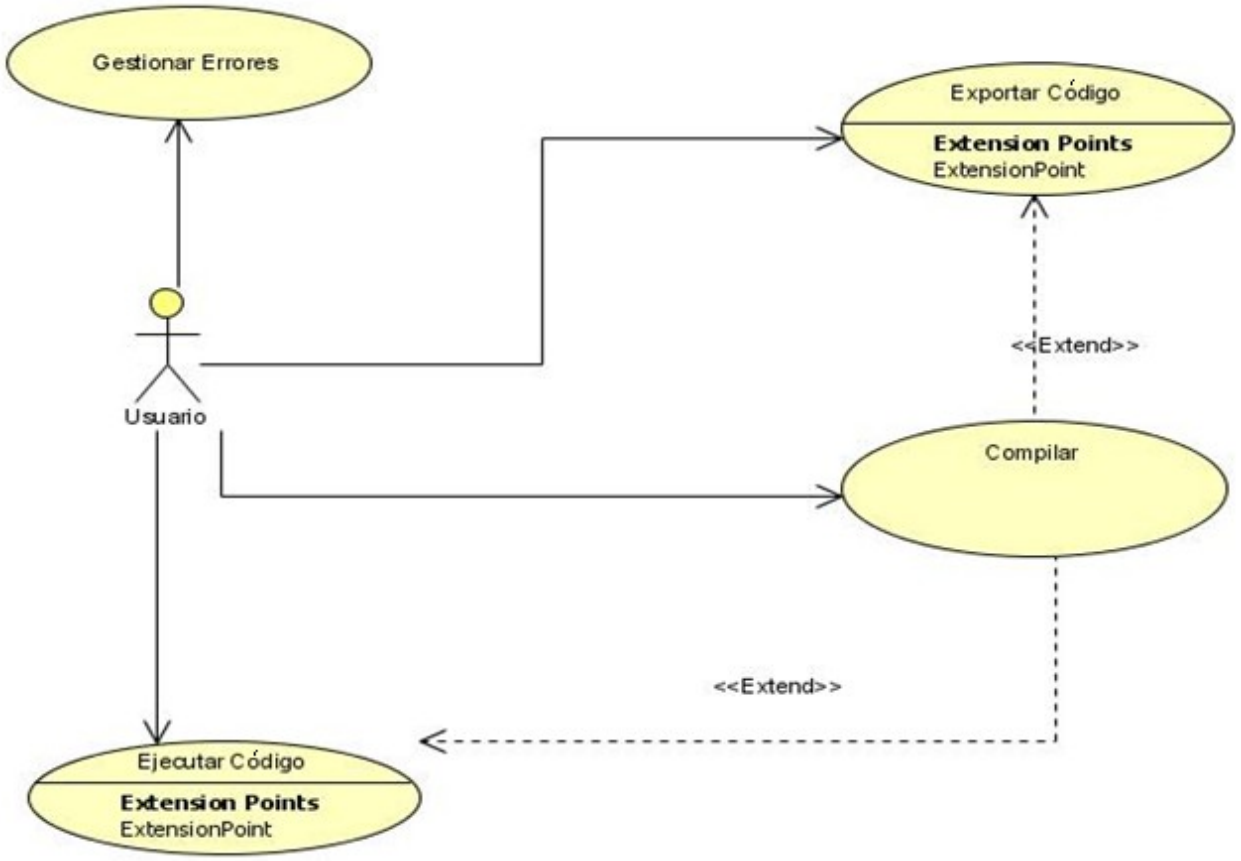
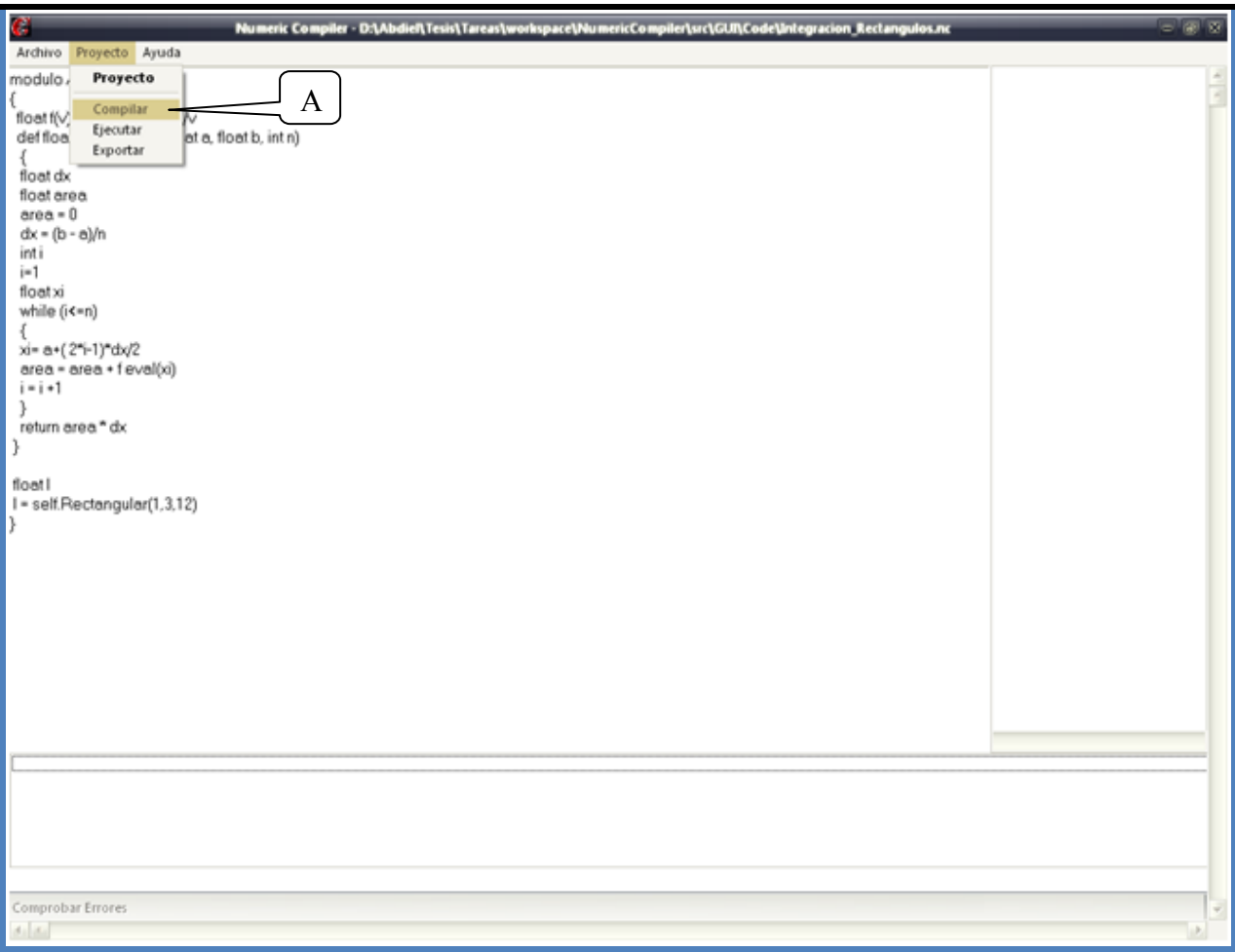



**Figura 3. 2: Modelo de Casos de Usos del Sistema.**

**Descripción Textual de los Casos de Uso**

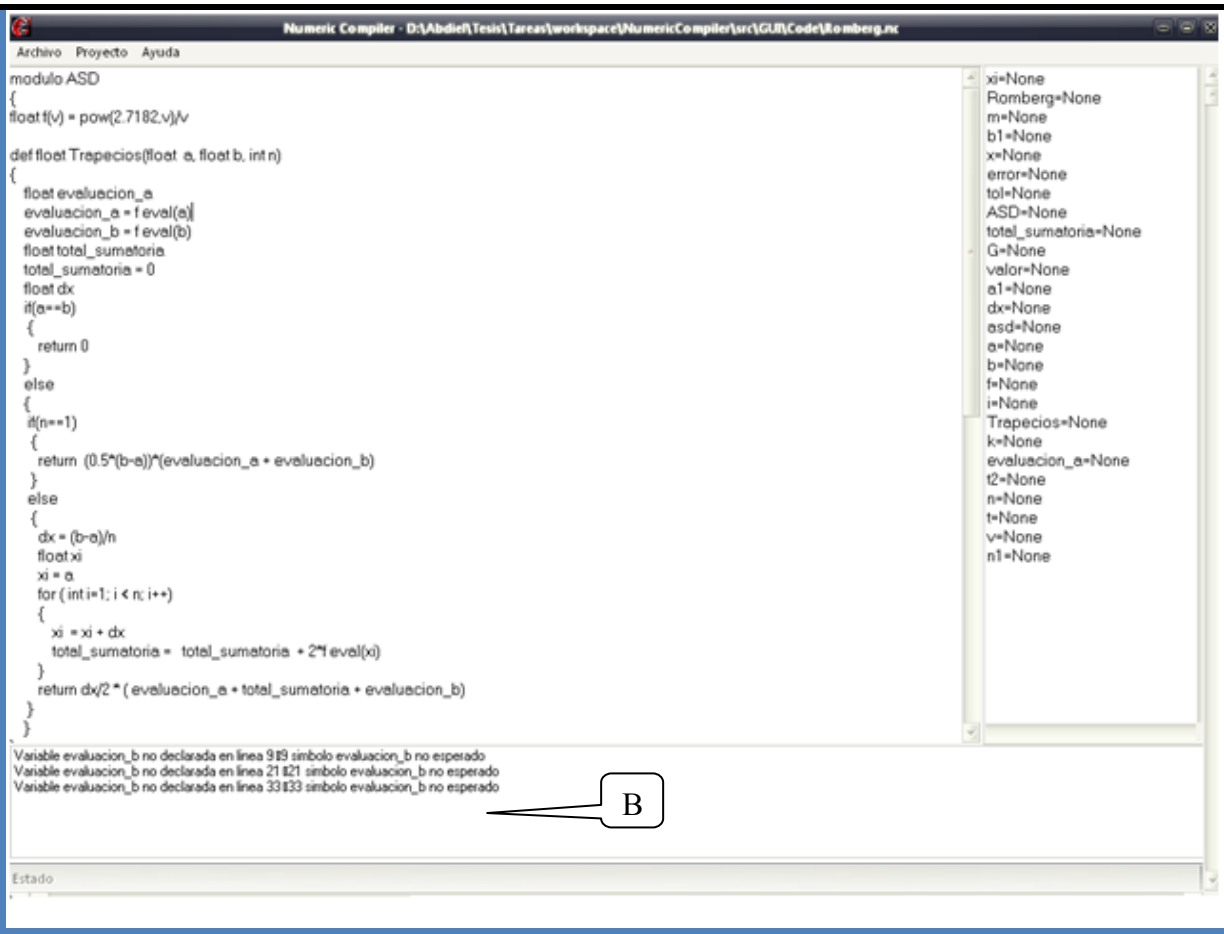
Mediante los casos de uso expandidos se describe paso a paso la secuencia de eventos que los actores utilizan para completar un proceso a través del sistema.

| | |
|---|---|
| Caso de Uso: | Compilar. |
| Actor(es): | Usuario |
| Propósito: | Reconocer y almacenar todas las instrucciones declaradas por el usuario mientras que sean soportadas por el lenguaje. |
| Resumen: | El caso se inicia con la acción del usuario |
| Referencias: | R3,R4,R5 |
| Precondiciones: | El Usuario debe de encontrarse en la ventana principal y debe haber introducido algún código en el editor de código (B). |
| | |



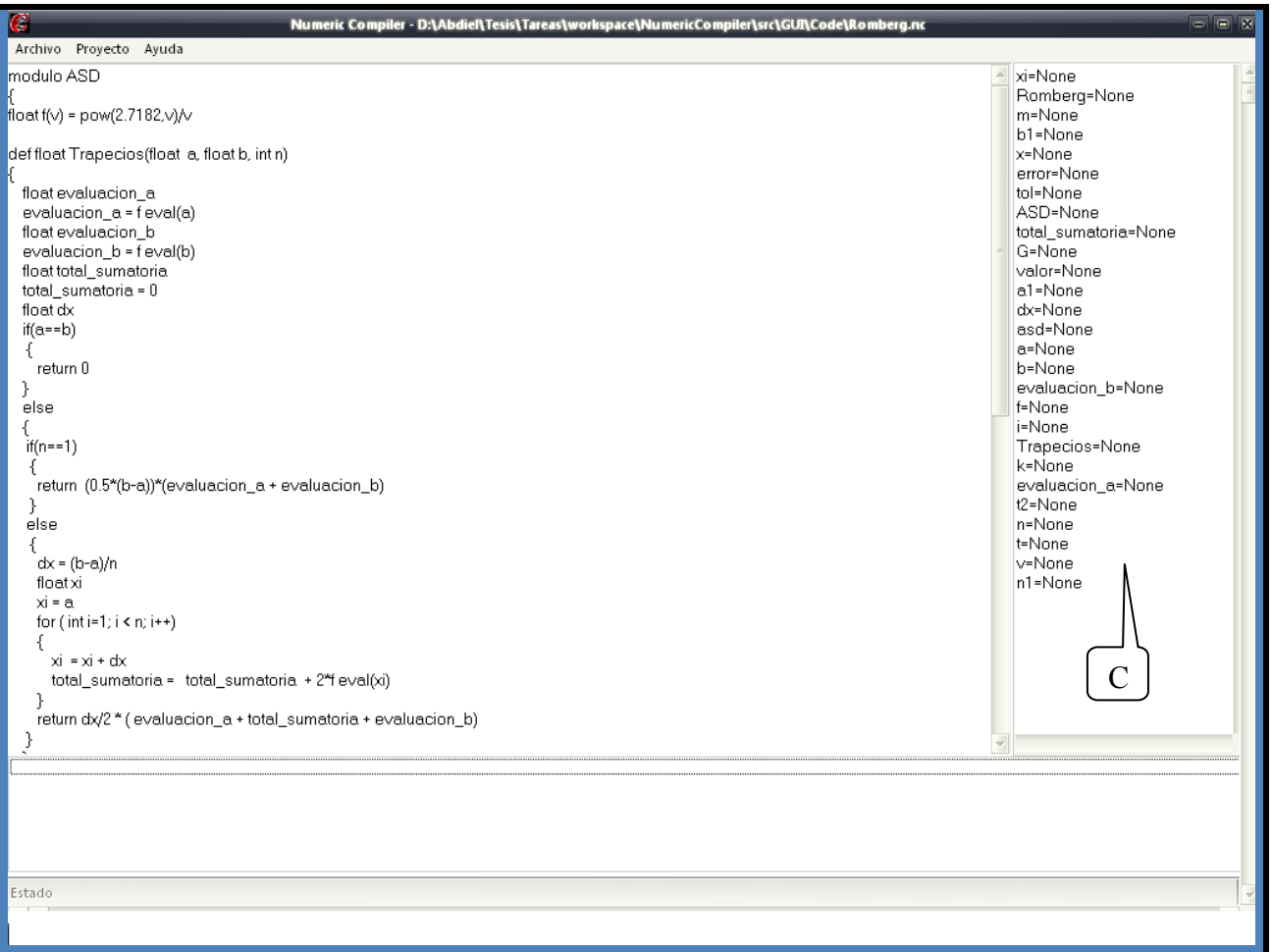

**Pantalla 1**

| Acción del Actor | Respuesta del Sistema |
|---|---|
| El usuario escoge la opción de compilar(A). | El sistema compila el código escrito por el usuario. |




**Pantalla 2**

| Sección: “Si se detectan errores.” | |
|---|---|
| | El sistema debe mostrar la información de los errores encontrados en el panel de errores (B). |




**Pantalla 3**

| Sección: "No se detectan errores." | |
|---|---|
| | El sistema mostrara las variables definidas por el sistema en la ventana de Variables (C). |

**Tabla 3. 6: Caso de Uso Expandido Compilar Código.**

Nota: Para ver los restantes casos de usos expandidos ver Anexo 2.

## CONCLUSIONES

En este capítulo se desarrolló la propuesta de solución, y se obtuvo a partir del análisis de los procesos del negocio, un listado con las funciones que debe tener el sistema representadas mediante un Diagrama de Casos de Uso. Finalmente se describieron paso a paso todas las acciones de los actores del sistema con los casos de uso que interactúa. Llegado a este punto se puede diseñar el sistema, teniendo en cuenta que se cumplan todos los requerimientos que han sido considerados necesarios en este capítulo.

# CAPÍTULO 4: CONSTRUCCIÓN DE LA PROPUESTA DE SOLUCIÓN.

## INTRODUCCIÓN

En este capítulo se recoge en término general, utilizando las extensiones del UML, el análisis y diseño del sistema propuesto además de cómo se implementa el mismo. Se detalla la relación estructural entre los objetos que interactúan, dígase diagramas de clases, diagramas de secuencia y colaboración del análisis y del diseño.

## 4.1 FASE DE ANÁLISIS DE LA PROPUESTA DE SOLUCIÓN

En la fase de análisis del desarrollo se da prioridad al conocimiento de los requerimientos, los conceptos y las operaciones relacionadas con el sistema. A menudo la investigación y el análisis se caracterizan por centrarse en cuestiones concernientes a los procesos, los conceptos, etcétera. (Lerman, 1999)

En esta fase se decide cuál será la arquitectura a seguir por los diseñadores y arquitectos del software, en este caso, se utilizará la arquitectura en capas.

El objetivo primordial de la programación por capas es la separación de la lógica de negocios de la lógica de diseño, la arquitectura más simple aplicando este patrón consiste en separar la capa de datos de la capa de presentación al usuario, actividad que usualmente suele hacerse estando inconscientemente aplicando esta pauta de diseño de software.

La mayor ventaja de este patrón es que como el desarrollo está distribuido en varios niveles o capas, en caso que sea necesario algún cambio, sólo se cambiaría lo necesario en la capa que se encontrará el componente, sin tener que revisar el código.

Además, permite distribuir el trabajo de desarrollo de un software por niveles; pudiendo trabajar varios grupos al mismo tiempo sin tener que preocuparse unos del otro, de forma que bastaría con conocer la API que existe entre niveles.

Por estas razones se decide desarrollar el software aplicando esta arquitectura.

A continuación se exponen los diagramas de clases y de colaboración del análisis con el objetivo de que comprender mejor los procesos que se llevan a cabo en el dominio del sistema, así como cuáles son los eventos y operaciones que realizará.

Los diagramas han sido organizados por casos de usos para una mayor comprensión.

### 4.1.1 CASO DE USO COMPILAR

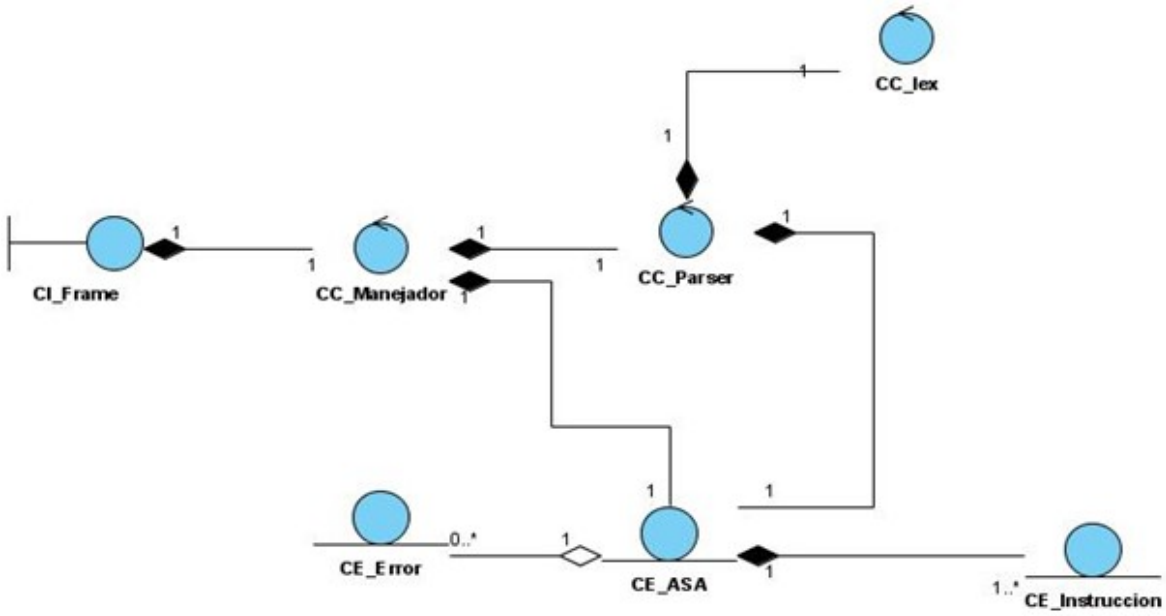



**Figura 4. 1: Diagrama de Clases del Análisis CU Compilar.**

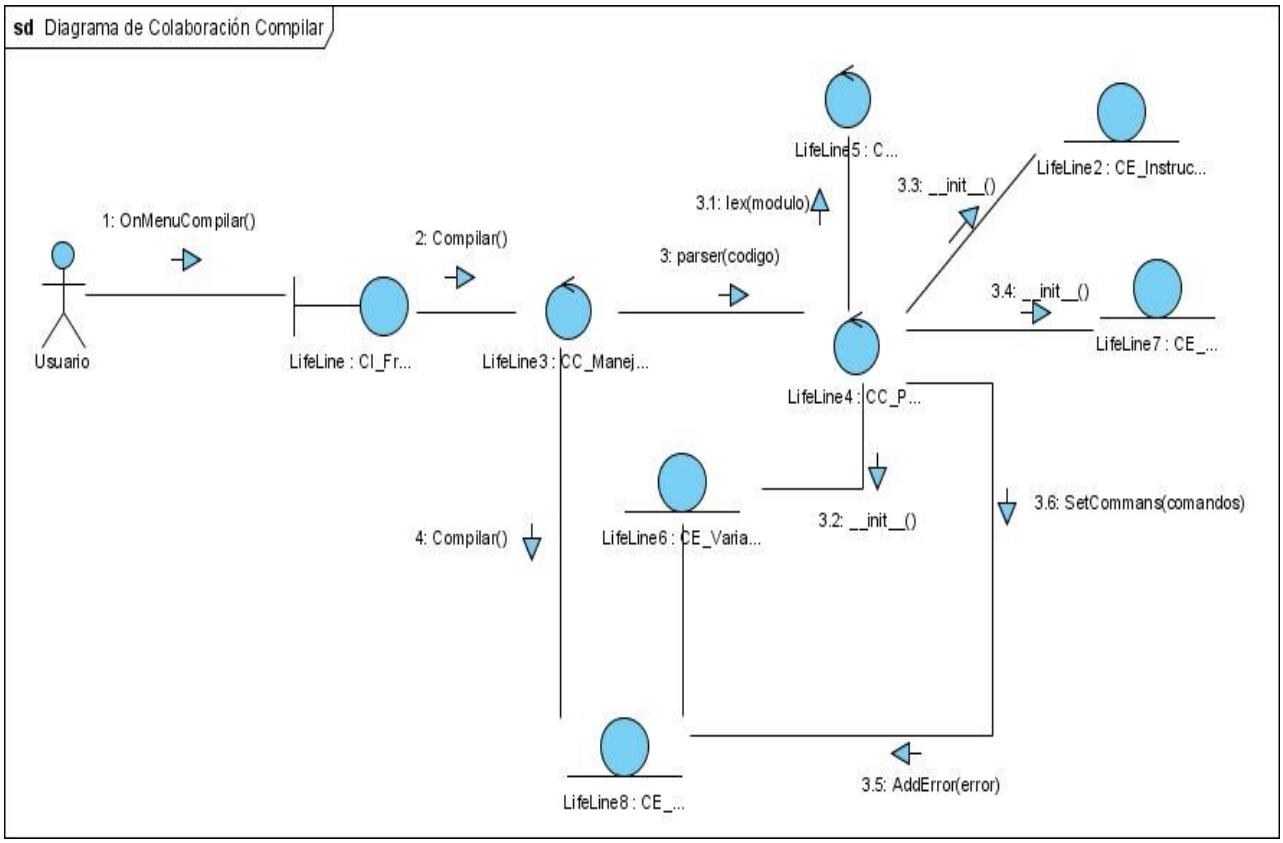



**Figura 4. 2: Diagrama de Colaboración del Análisis CU Compilar.**

### 4.1.2 CASO DE USO EJECUTAR CÓDIGO

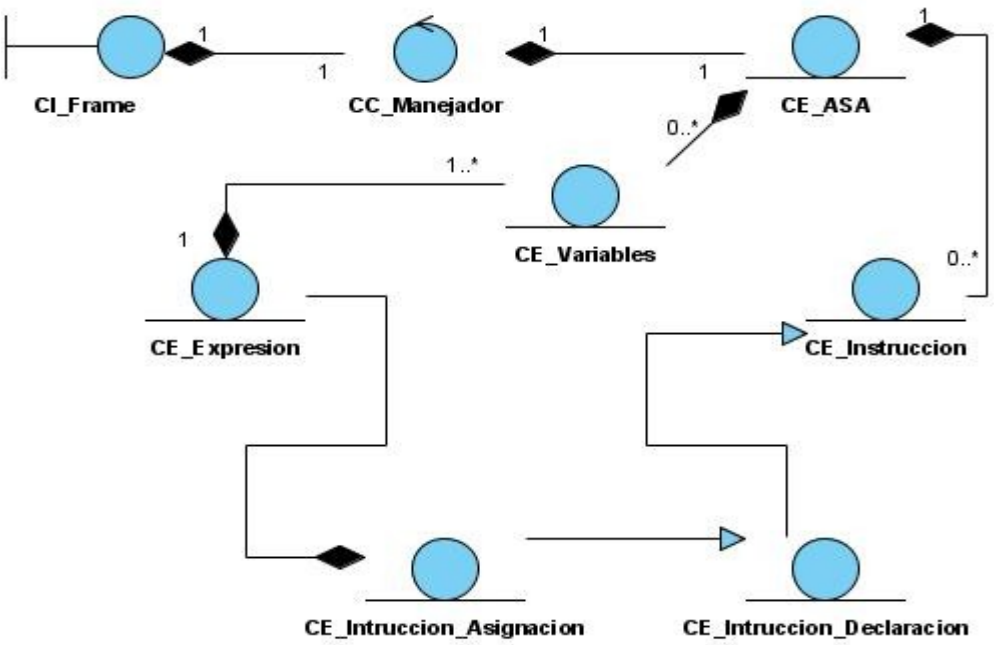



**Figura 4. 3: Diagrama de Clases del Análisis del CU Ejecutar Código.**

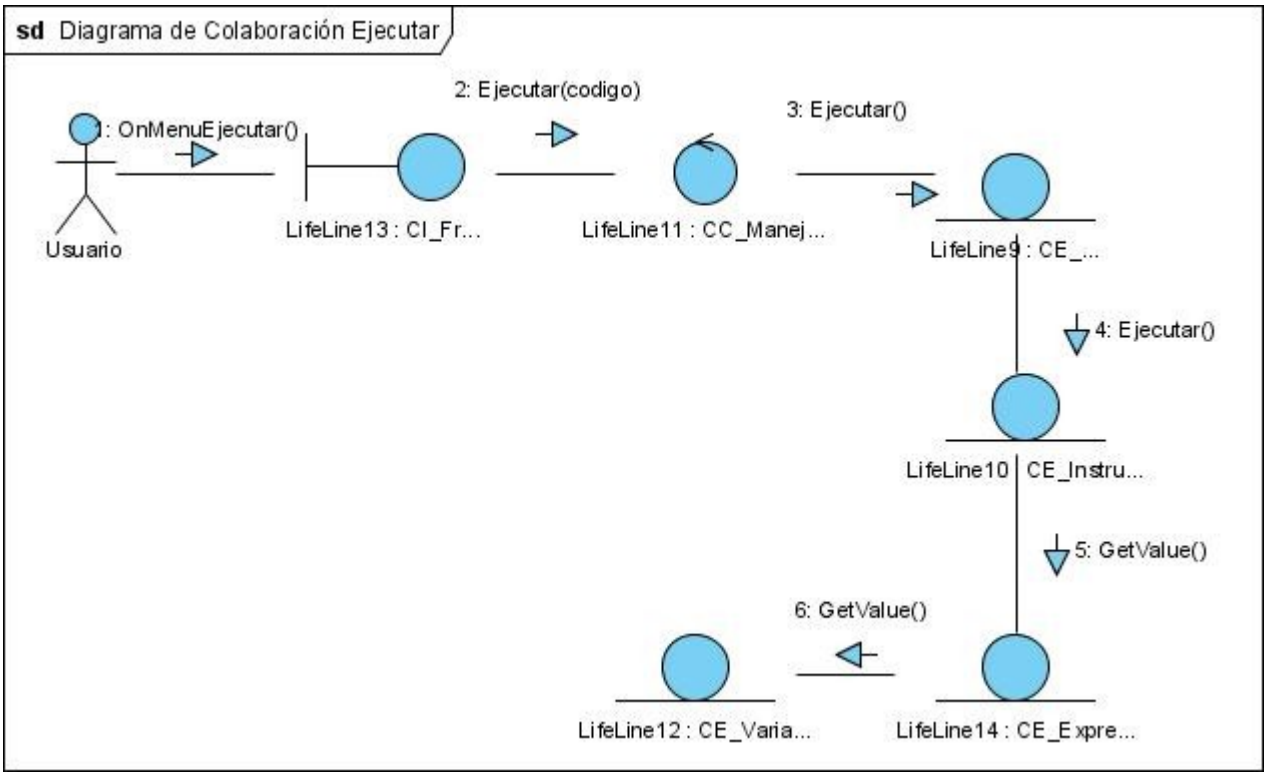



**Figura 4. 4: Diagrama de Colaboración del Análisis CU Ejecutar Código.**

### 4.1.3 CASO DE USO EXPORTAR CÓDIGO

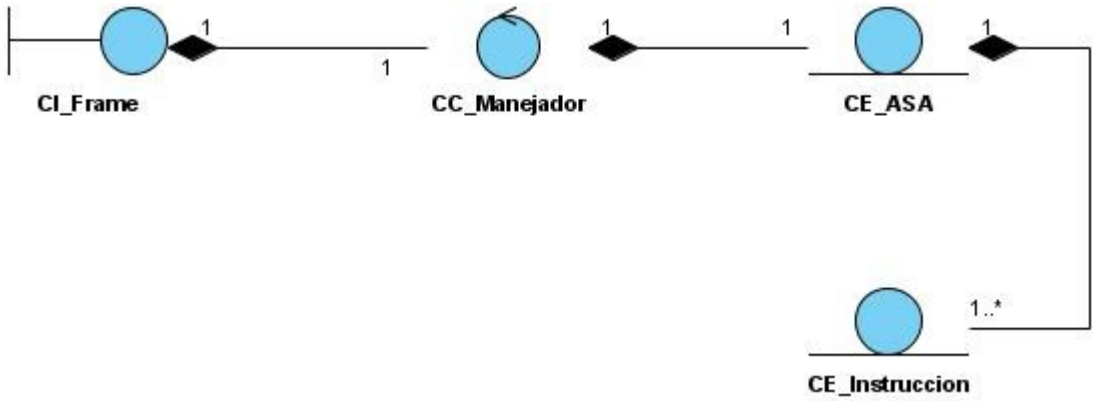



**Figura 4. 5: Diagrama de Clases del Análisis CU Exportar Código.**

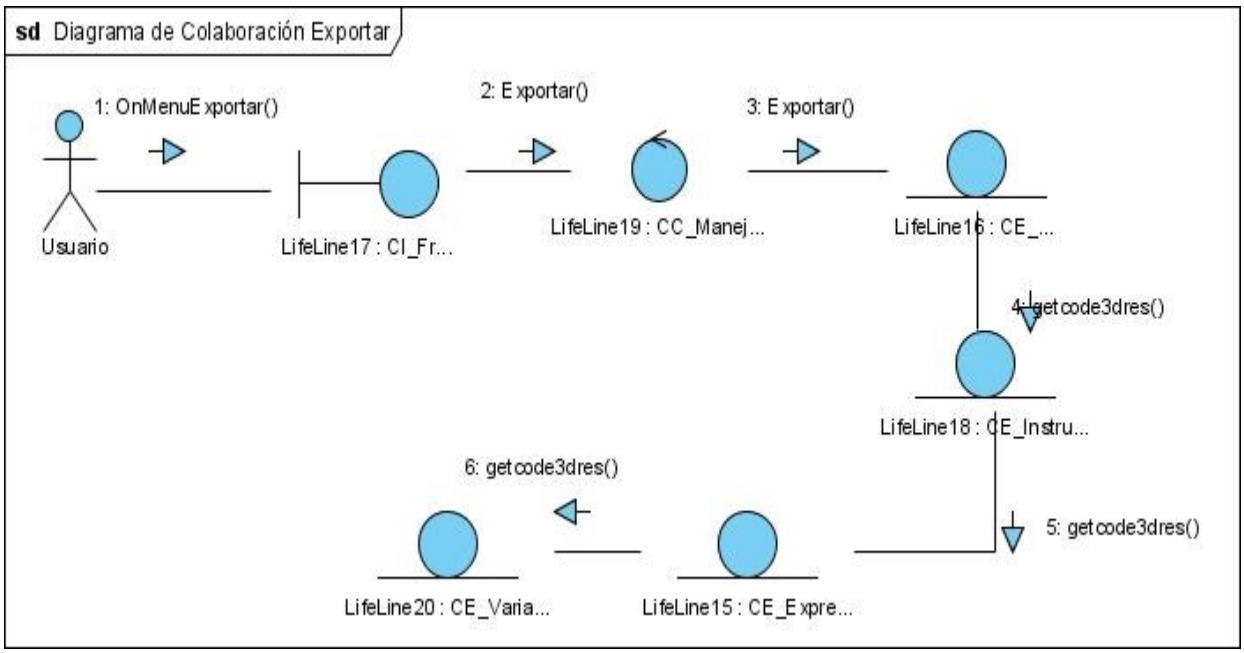



**Figura 4. 6: Diagrama de Colaboración del Análisis CU Exportar Código.**

### 4.1.4 CASO DE USO GESTIONAR ERRORES

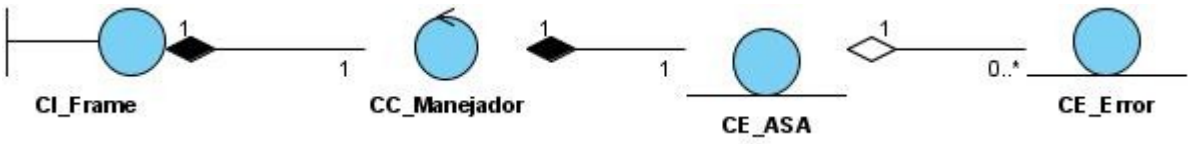



**Figura 4. 7: Diagrama de Clases del Análisis CU Gestionar Errores.**

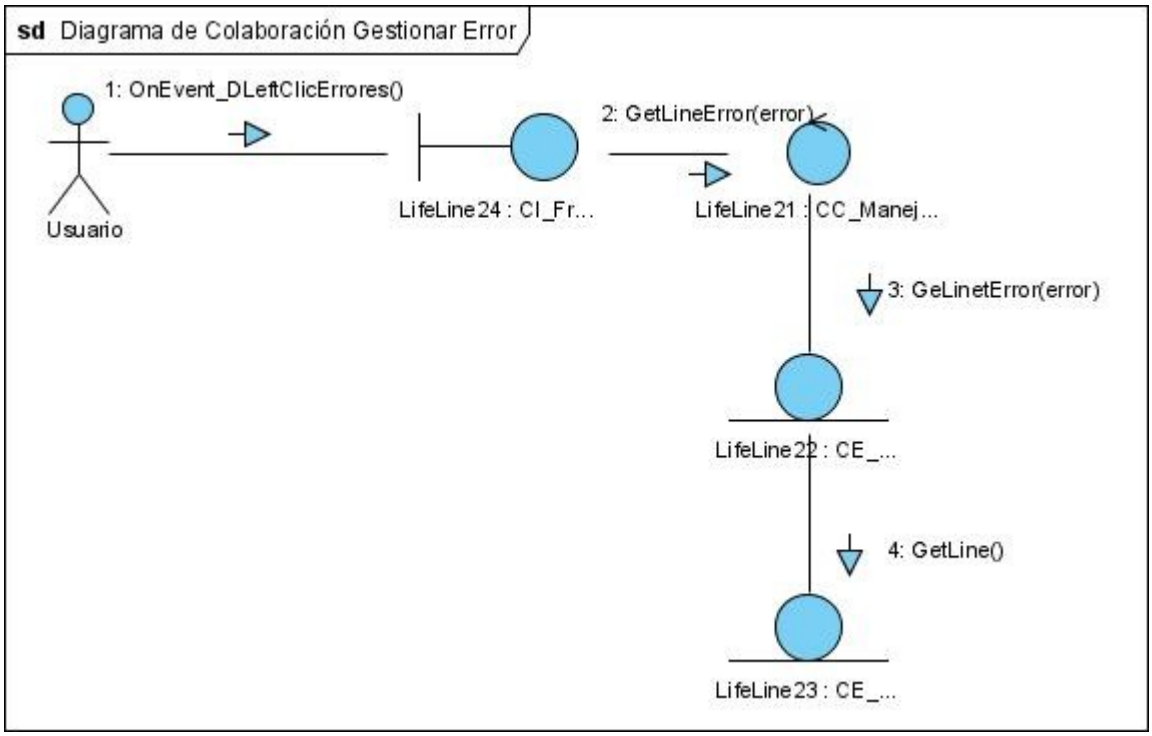



**Figura 4. 8: Diagrama de Colaboración del Análisis CU Gestionar Errores.**

Después de haberse expuesto los principales eventos y operaciones que se desarrollan dentro de cada caso de uso en el análisis, se tiene una primera aproximación de lo que será el sistema.

## 4.2 FASE DE DISEÑO DE LA PROPUESTA DE SOLUCIÓN

Los diagramas de clases del diseño están compuestos por clases y sus atributos o sus relaciones con las demás clases. Esto posibilita que se tenga una representación de la estructura del sistema, son los más utilizados en el modelado de sistemas orientados a objetos. Se utilizan para modelar las colaboraciones o modelar esquemas.

“Los diagramas de interacción son algunos de los artefactos más importantes que se preparan en el análisis y diseño orientados a objetos.” (Grady Booch, 1999)

‘Es muy importante asignar acertadamente las responsabilidades al momento de elaborar los diagramas de interacción.” (Grady Booch, 1999)

“El tiempo y el esfuerzo que se dedican a su elaboración, así como un examen riguroso de la asignación de responsabilidades, deberían absorber parte considerable de la fase de diseño de un proyecto.” (Grady Booch, 1999)

“Los patrones, principios y expresiones especializadas codificados sirven para mejorar la calidad del diseño.” (Grady Booch, 1999)

Como se analizó, los diagramas de clases y los de interacción, juegan en el flujo de trabajo de diseño un rol importante.

Se presenta a continuación como se muestra en la Figura 4.9 un diagrama de clases donde están presentes los paquetes del análisis, los cuales contienen las clases del diseño del software.

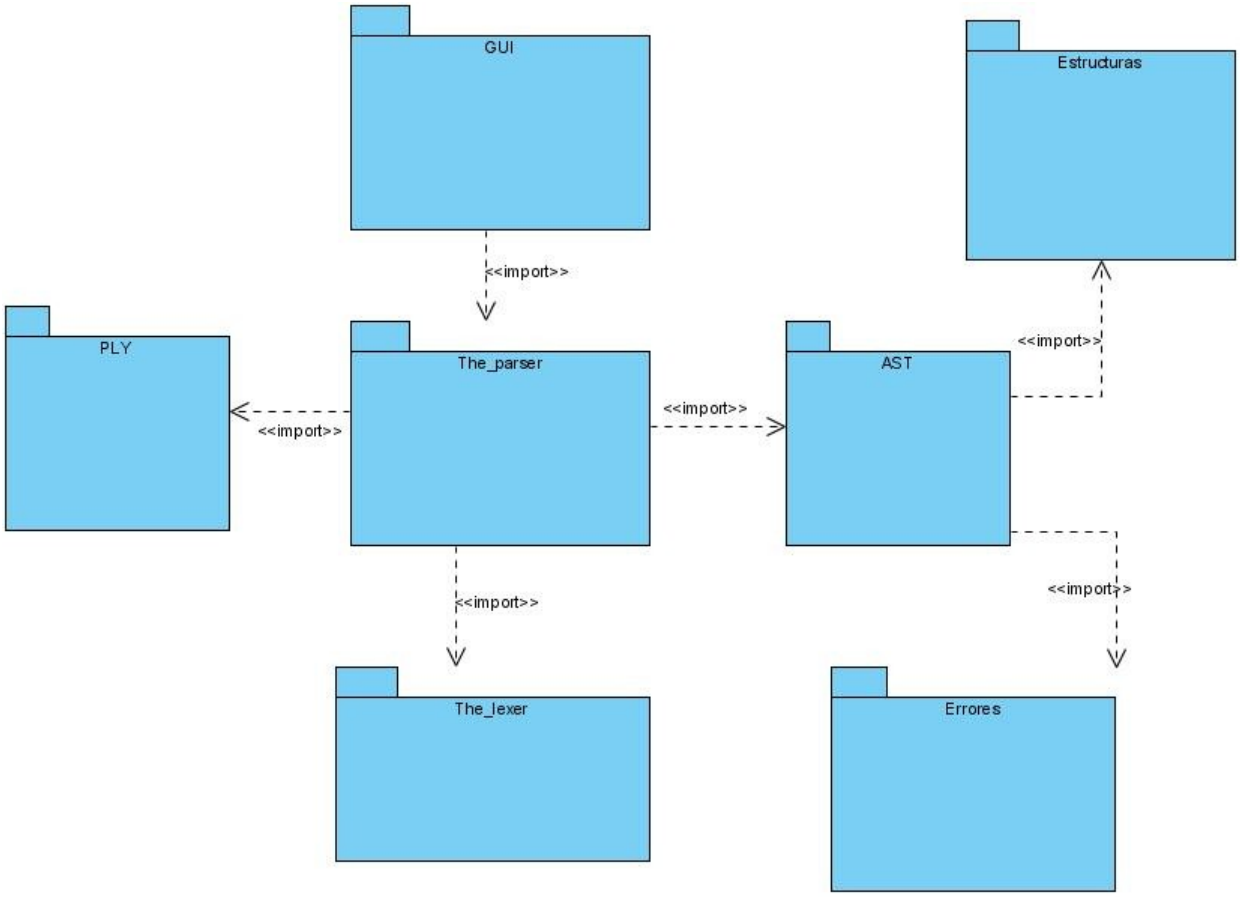



**Figura 4. 9: Diagrama de Paquetes del Diseño.**

Nota: El diagrama está compuesto por varios paquetes y sus dependencias, para una mayor comprensión del diagrama ver Anexo 3.

De acuerdo a la forma en que se ha organizado el contenido del trabajo, se deben presentar los modelos organizados por los casos de uso, de forma que pueda entenderse mejor la lógica del negocio.

En el sistema se tiene una sola interfaz que se comunica a través de una clase controladora que maneja todos los datos desde la capa de la interfaz de usuario con la lógica del negocio, y a su vez

existe otra que relaciona esta última con la capa de datos, aplicando así el diseño de arquitectura de 3 capas.

Para una mejor comprensión de los modelos se explicará el propósito de algunas clases.

| CLASE | PROPÓSITO |
|---|---|
| CE_Programa | Clase para el manejo del conjunto de instrucciones declaradas por el usuario. Desde ella se puede compilar, ejecutar, exportar u obtener los errores que están presentes en el código, es decir contiene todos los datos necesarios del sistema. |
| CE_Instrucción | Representa la súper clase en la jerarquía de instrucciones que están presentes dentro de nuestro compilador. |
| CC_Controladora | Es la clase que comunica la capa de la interfaz de usuario con la capa de la lógica del negocio. |
| CI_Frame | Define el diseño de la interfaz de usuario además de tener toda la lógica de la misma. |
| CE_Compiler_Error | Súper clase que representa la jerarquía de errores que se manejan dentro de la lógica del negocio. |
| CC_Parser | Es la clase que maneja la lógica del negocio, en ella se definen todas la reglas semánticas y sintácticas del lenguaje utilizado en el compilador. Tiene relación con la clase Lexer. |
| CC_Lexer | Es la clase que define las palabras, símbolos y números que son aceptados por el sistema. |
| CE_Tabla_Simbolo | Clase que guarda el conjunto de variables que se manejan dentro de cada programa definido por el usuario. |

**Tabla 4. 1: Propósito de algunas clases.**

A continuación se presentan los diagramas de los casos de usos que representan los procesos más importantes, se ocultan los atributos y las operaciones de las clases de la capa intermedia para facilitar la comprensión de los mismos.

### 4.2.1 CASO DE USO COMPILAR

Este caso de uso describe todo el proceso de compilación, y actualiza la vista de la ventana de errores y de variables.

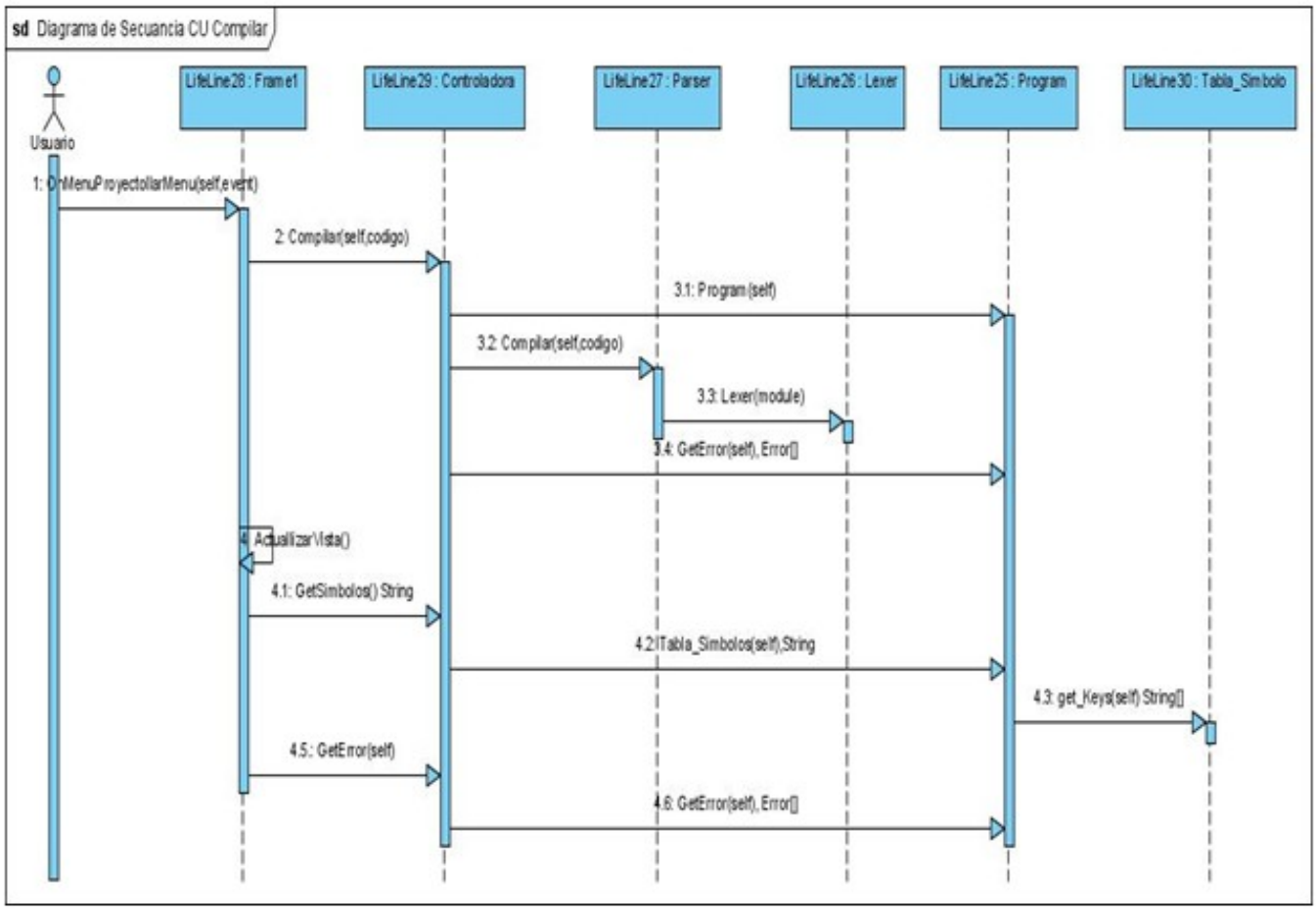



**Figura 4. 10: Diagrama de Secuencia del Caso de Uso Compilar.**

### 4.2.2 CASO DE USO EJECUTAR CÓDIGO

Este caso de uso ejecuta cada instrucción que esté presente dentro del Árbol de Sintaxis Abstracta obteniéndose como resultado los valores finales que tomarían todas las variables de la tabla de símbolo. Es válido resaltar que este sólo ocurrirá en el caso que no exista ningún error dentro del Árbol de Sintaxis Abstracta.

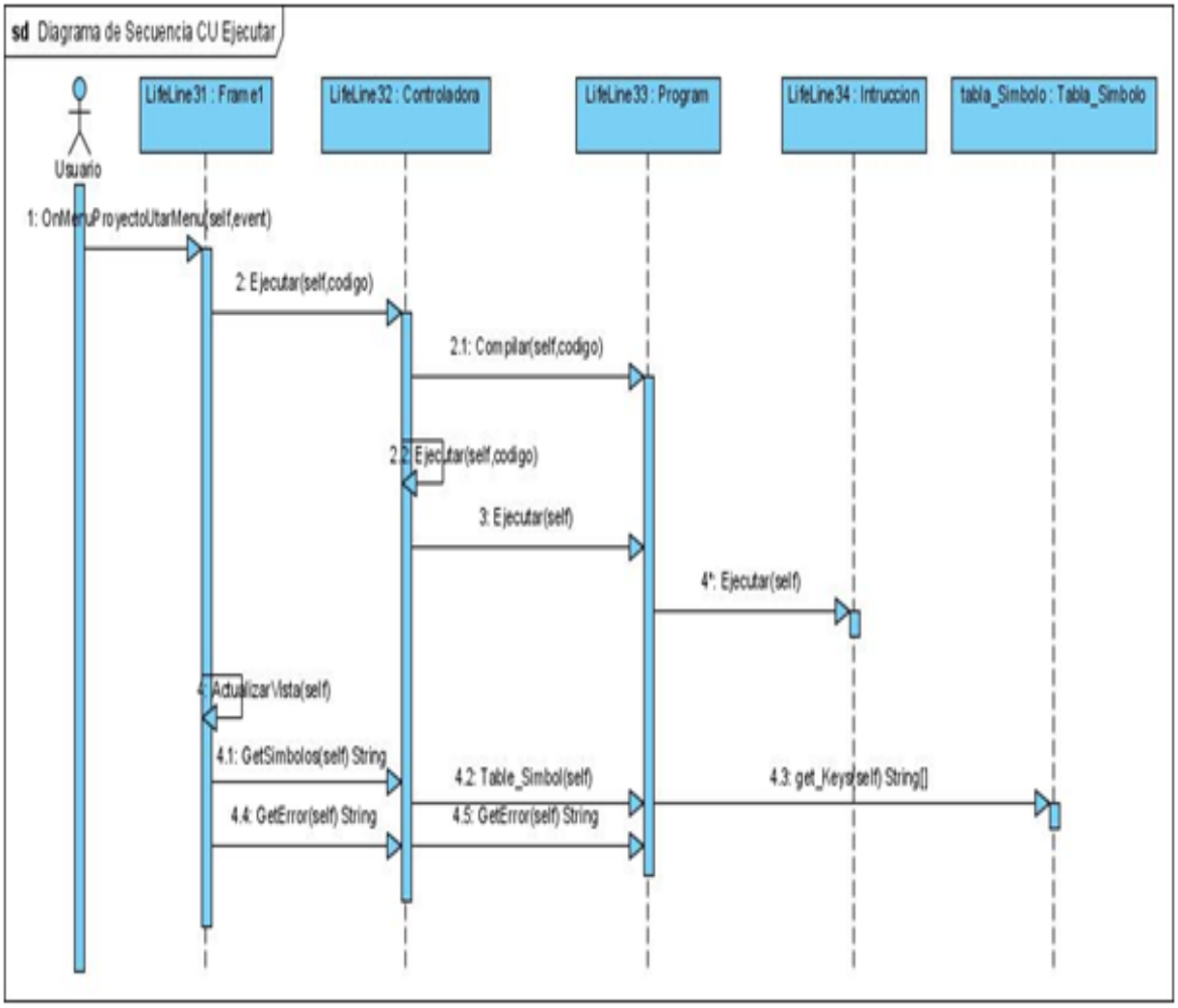



**Figura 4. 11: Diagrama de Secuencia CU Ejecutar Código.**

### 4.2.3 CASO DE USO EXPORTAR CÓDIGO

En este caso de uso se obtiene un código de tres direcciones, para ser utilizado como notación o un lenguaje intermedio entre el compilador y otras aplicaciones como pudiera ser un intérprete para dicho lenguaje.

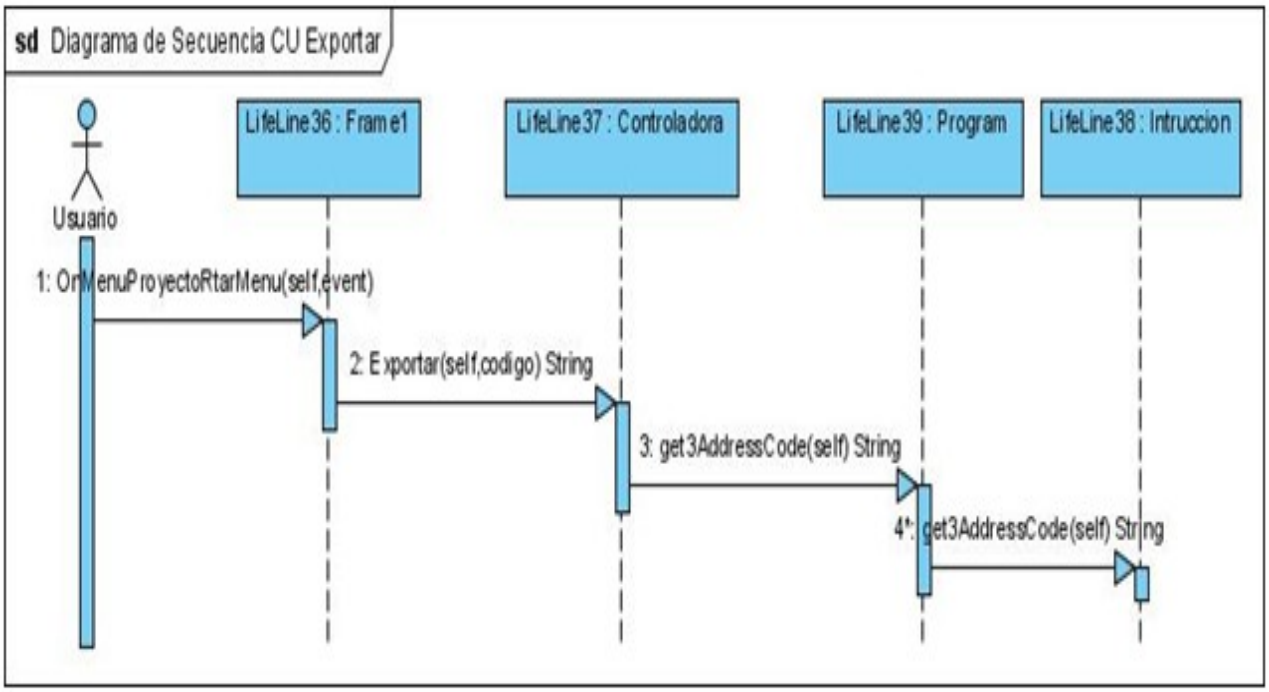



**Figura 4. 12: Diagrama se Secuencia del Caso de Uso Exportar Código.**

### 4.2.4 CASOS DE USO GESTIONAR ERROR

En este caso de uso se describe el proceso de gestión de errores en la interfaz de usuario con el objetivo de ayudar al usuario a encontrar sus errores.

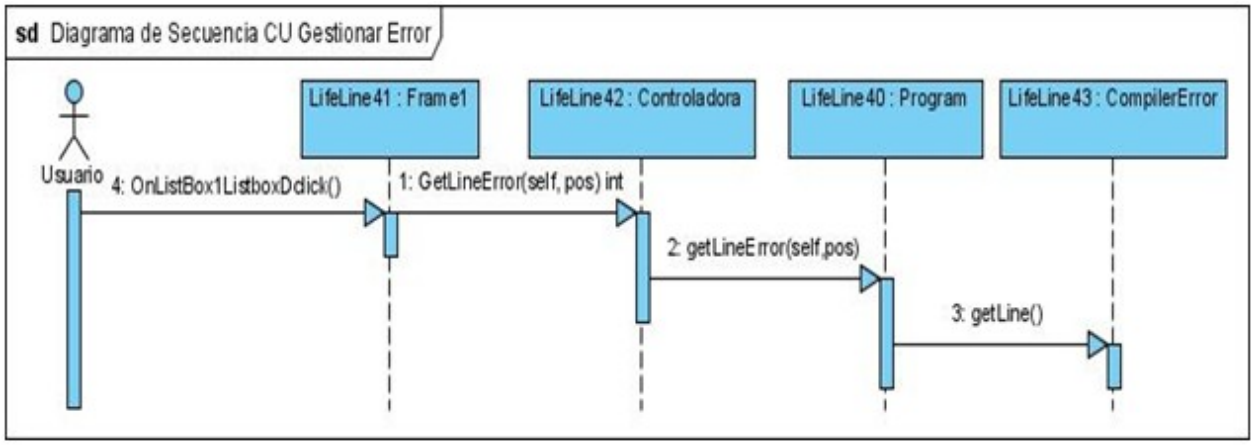



**Figura 4. 13: Diagrama de Secuencia del Caso de Uso Gestionar Errores.**

## 4.3 PRINCIPIOS DE DISEÑO

El diseño de un sistema informático sea cual sea su finalidad, siempre tiene que favorecer al usuario. Por lo cual se utilizan ciertos principios y patrones que en general ayudarán a garantizar la usabilidad de cualquier sistema.

Después de haberse visto los diagramas, se explican los principios y patrones que se tienen en cuenta a lo hora de diseñar los diagramas de clases y de colaboración.

Patrón Experto: la clase que cuenta con la información necesaria es la ideal para cumplir la responsabilidad.

"Es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen". (Lerman, 1999)

- **Creador**: guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos.

  "El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento." (Lerman, 1999)

  Nota: Para observar los casos en que se pudiera utilizar este patrón ver Anexo 4.

- **Alta Cohesión**: asignar una responsabilidad de modo que la cohesión siga siendo alta. Es un principio que se debe tener presente en todas las decisiones de diseño; es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño. (Lerman, 1999)

- **Bajo Acoplamiento**: Asignar una responsabilidad para mantener bajo acoplamiento. Es un principio que debemos recordar durante las decisiones de diseño; es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. (Lerman, 1999)

- **Controlador**: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase.

  Nota: Para observar los casos en que se pudiera utilizar el patrón Ver Anexo 5.

- **Principio de Sustitución de Liskov**: "Si para cada objeto O1 de tipo S hay un objeto O2 de tipo T tal que para todos los programas P definidos en términos de T, el comportamiento [interno] de P no cambia cuando O1 es sustituido por O2, entonces S es un subtipo de T." (Barbara Liskov, 1986)

### 4.3.1 ESTÁNDARES EN LA INTERFAZ DE LA APLICACIÓN

Los saltos cualitativos en eficacia se encuentran en la arquitectura del sistema, no en su superficie, en el diseño visual de la interfaz. Pero siempre hay que tener en cuenta que una aplicación con una interfaz de usuario amigable da una buena imagen del software.

- **Eficacia del usuario**: se busca la productividad del usuario, no solo del ordenador. Cuando se trate la eficacia de un sistema, va más allá de la simple eficacia de la máquina.
- **Mantén ocupado al usuario**: El gasto más alto en un negocio es el trabajo humano. Cada vez que el usuario tiene que esperar la respuesta del sistema.
- **Maximizar la eficacia**: de un negocio u organización debes maximizar la eficacia de todos y no sólo de un grupo.
- **Ayuda al Usuario**: Escribe mensajes de ayuda concisos y que ayuden a resolver el problema: un buen texto ayuda mucho en comprensión y eficacia.
- **Diseño de Menús y Etiquetas de Botones**: estos deben comenzar con la palabra más importante para no sobrecargar la interfaz con demasiadas letras y para ayudar al usuario a una rápida navegación.
- **Legibilidad**: Se utiliza texto con alto contraste. Se utiliza negro sobre blanco. Además se utilizan tamaños de letra adecuados para que puedan ser bien leídos.
- **Guardar el Estado**: La información puede guardarse en cualquier momento y después volverse a cargar si se desea continuar trabajando.

Esta es la razón por la que es tan importante que todo el mundo involucrado en un proyecto de software aprecie la importancia de hacer de la productividad del usuario el objetivo principal y entender la diferencia entre diseñar un sistema eficaz o potenciar la productividad del usuario. Esto implica que es fundamental la colaboración, comunicación y complicidad entre ingenieros y diseñadores de interacción si se quiere conseguir este objetivo.

### 4.3.2 TRATAMIENTO DE ERRORES

Los errores son tratados en un componente de la ventana inicial, la cual brinda la facilidad de identificar la línea de error en el editor de texto para aumentar la efectividad del usuario a la hora de corregir el error. En algunos casos incluye la forma de solucionar el error.

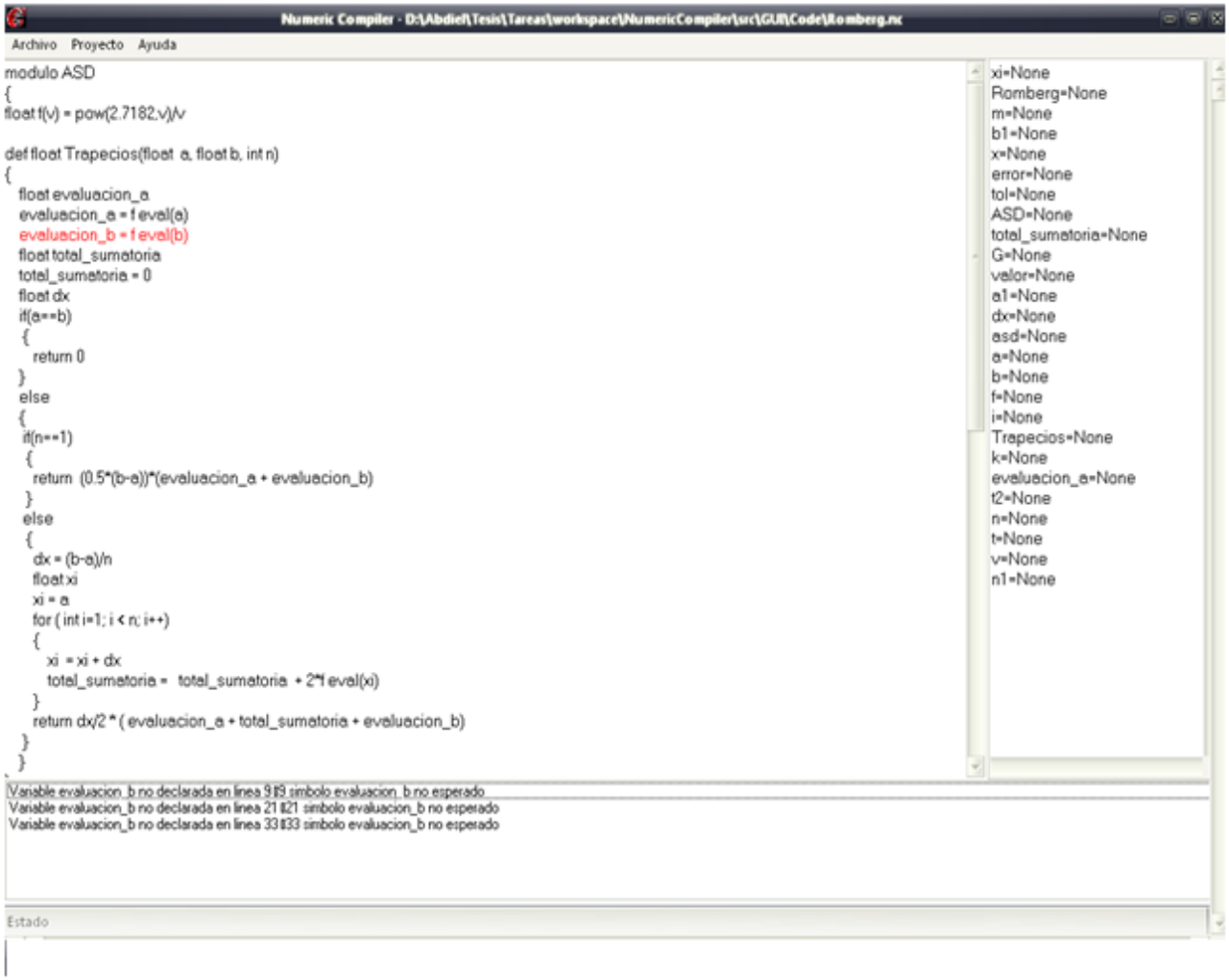



**Figura 4. 14: Tratamiento de errores.**

## 4.4 ESTÁNDARES DE CODIFICACIÓN

Un estándar de codificación comprende los aspectos de la generación de código y repercute directamente en la legibilidad y la extensibilidad de cualquier proyecto de Software, haciendo que nuevos desarrolladores se acoplen rápidamente al proceso de desarrollo.

Utilizar estándares de codificación ayuda a reducir errores a la hora de reutilizar un código a demás de garantizar la obtención de un código claro y comprensible para la comunicación entre los programadores del equipo por lo que facilita el mantenimiento del software.

```
   def p_expresion_inic(p):
         'BEGIN : HEAD PROG_ID BODY '

   def p_expresion_inic_error2(p):
         'BEGIN  :  HEAD  PROG_ID  error
BODY '
```

```
   def p_expresion_inic_error1(p):
         'BEGIN : error '

   def p_expresion_id(p):
         'PROG_ID : ID '
         prog.setName(p[1])
```

## 4.5 MODELO DE DESPLIEGUE

En este caso no tiene sentido realizar un modelo de despliegue dado que se desarrolla una aplicación centralizada donde todos los componentes estarán estáticamente en un único ordenador.

## 4.6 MODELO DE IMPLEMENTACIÓN

Durante el flujo de trabajo en la fase de implementación unos de los artefactos que se elaboran es el diagrama de componentes que muestra las organizaciones y dependencias lógicas entre componentes del software, sean estos de código fuente, binarios o ejecutables.

Los elementos de modelado dentro del diagrama de componentes serán componentes y paquetes.

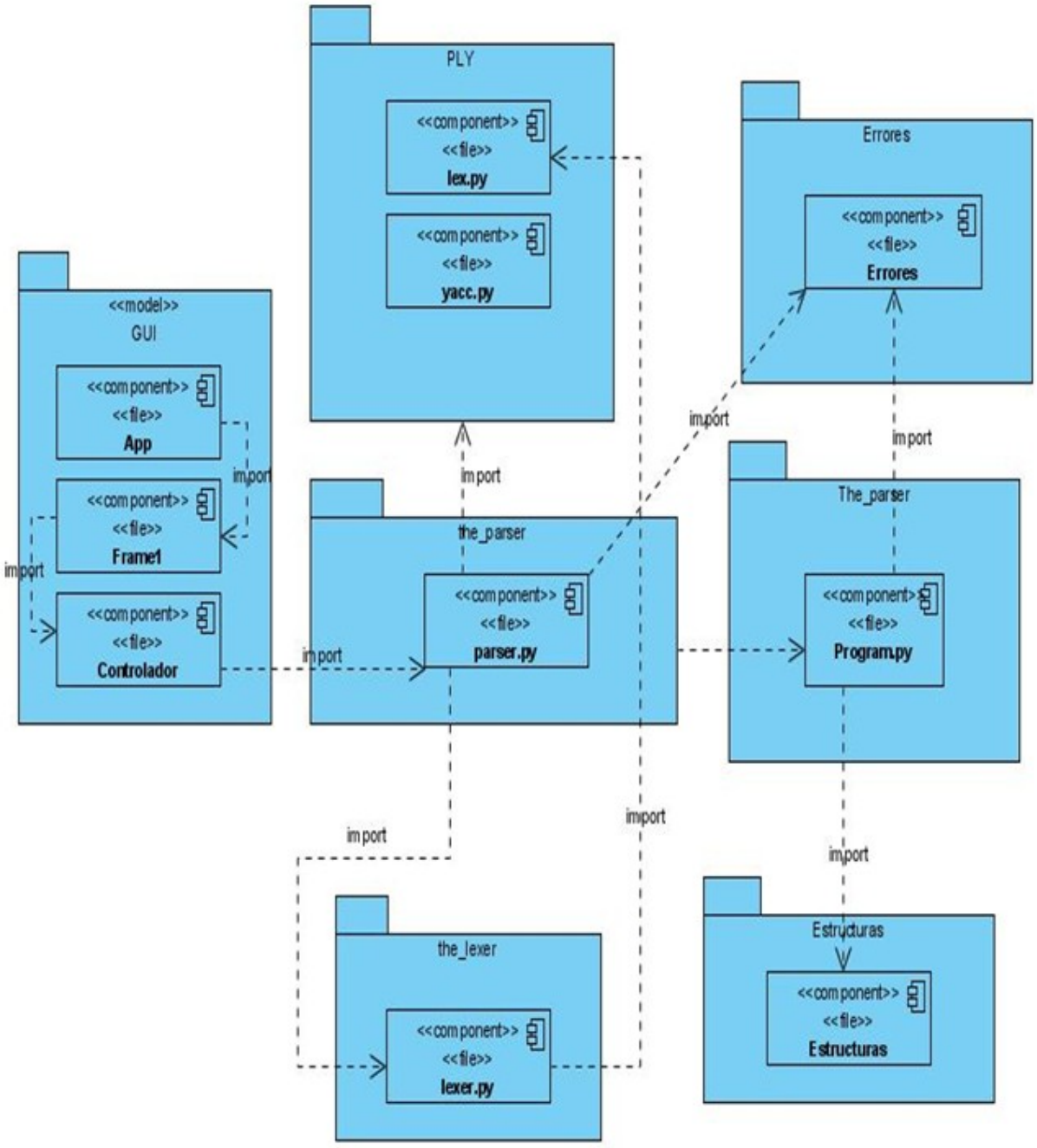



**Figura 4. 15: Diagrama de Componentes.**

## CONCLUSIONES

Este capítulo se ha encargado de ofrecer todos los aspectos necesarios para el correcto entendimiento del modelado del problema. Se mostraron las principales clases del dominio para un mayor entendimiento del negocio, se describió la propuesta de solución para el desarrollo del software, a través de la muestra, de varias vistas para llevar a cabo el proceso de análisis, diseño e implementación del sistema. Se exponen los distintos patrones de diseños utilizados.

# CAPITULO 5: ESTUDIO DE FACTIBILIDAD

## INTRODUCCIÓN

Para llevar a cabo un buen proyecto de desarrollo de software, debemos comprender el ámbito del trabajo a realizar, los recursos requeridos, las tareas a ejecutar, las referencias a tener en cuenta y la agenda a seguir. El estudio de la factibilidad es un paso importante, pues es imprescindible a la hora de acometer una tarea. Brinda al equipo de trabajo información inicial relacionada con el costo del producto, permite conocer si la realización del mismo es factible o no. Ello ayuda a precisar con más exactitud sus características, de modo que permita obtener un sistema con mayor calidad desde el punto de vista de la utilidad.

Se utilizó la Estimación basada en Puntos de Casos de Uso, procedimiento efectivo para la estimación del costo de un producto informático. En este capítulo se abordarán aspectos relacionados con la estimación de esfuerzos (costos) de desarrollo del sistema y los beneficios.

## 5.1 ESTIMACIÓN DEL TIEMPO DE DESARROLLO

La estimación mediante el análisis de Puntos de Casos de Uso es un método propuesto originalmente por Gustav Karner de Objetory AB, y posteriormente refinado por muchos otros autores. Se trata de un método de estimación del tiempo de desarrollo de un proyecto mediante la asignación de "pesos" a un cierto número de factores que lo afectan, para finalmente, contabilizar el tiempo total estimado para el proyecto a partir de esos factores. (Conferencia 5 Planificación y Estimación de Proyectos, 2006-2007)

### 5.1.1 CALCULAR LOS PUNTOS DE CASOS DE USO SIN AJUSTAR

El proyecto tiene un actor complejo, el usuario y cuatro casos de uso simples.

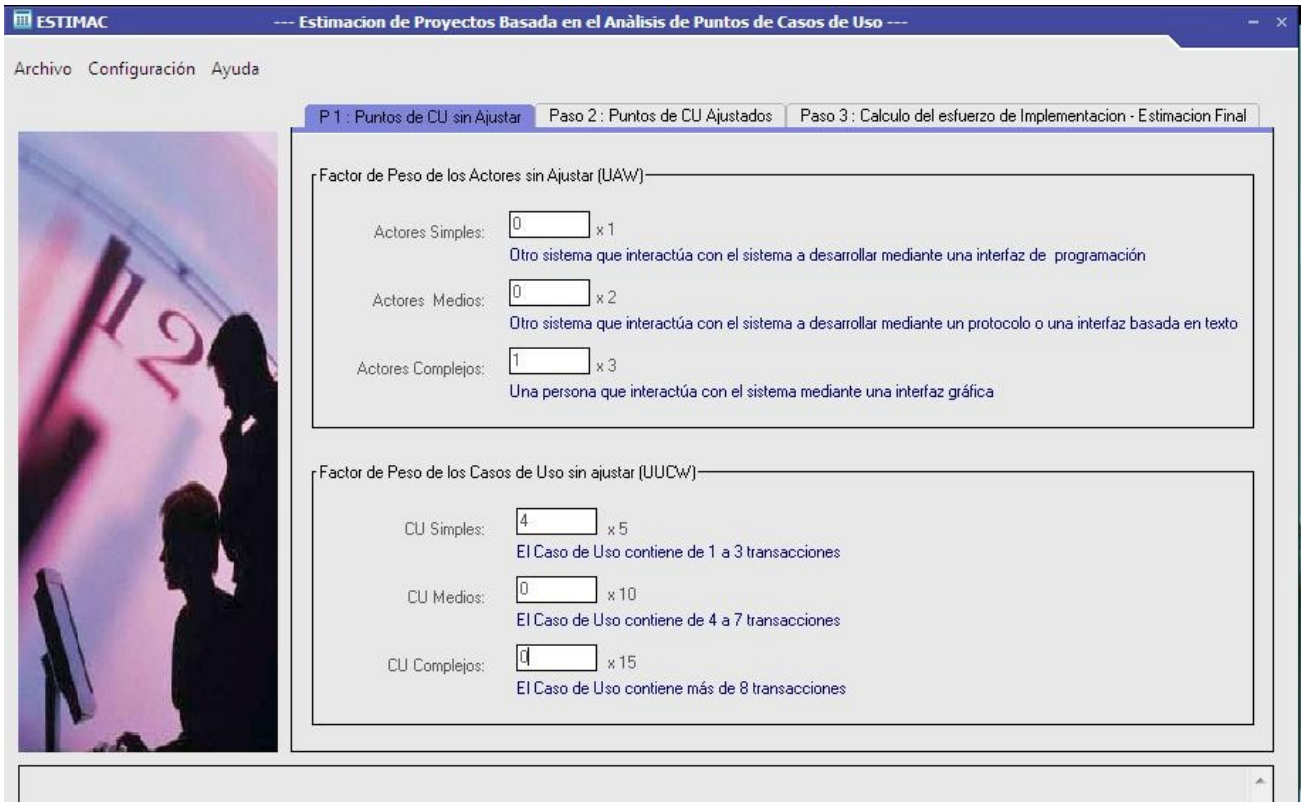



**Figura 5. 1: Puntos de Casos de Usos sin Ajustar.**

### 5.1.2 PUNTOS DE CASOS DE USOS AJUSTADOS

**Factor de Complejidad Técnica:**

Este coeficiente se calcula mediante la cuantificación de un conjunto de factores que determinan la complejidad técnica del sistema. Cada uno de los factores se cuantifica con un valor de 0 a 5, donde 0 significa un aporte irrelevante y 5 un aporte muy importante. En la siguiente tabla se muestra el significado y el peso de cada uno de estos factores:

| Factor | Descripción | Peso | Valor Asignado | Comentarios |
|---|---|---|---|---|
| T1 | Sistema distribuido | 2 | 0 | Sistema centralizado. |
| T2 | Objetivos del rendimiento o tiempo de respuesta | 1 | 4 | La velocidad debe ser lo más rápida posible. |
| T3 | Eficacia del usuario | 1 | 3 | Se necesita un usuario con conocimientos medios. |

| | | | | |
|---|---|---|---|---|
| T4 | Procesamiento interno complejo | 1 | 4 | Se realizan cálculos complejos. |
| T5 | El código debe ser reutilizable | 1 | 3 | El código debe ser reutilizable. |
| T6 | Facilidad de instalación | 0.5 | 1 | Escasos requerimientos de facilidad de instalación. |
| T7 | Facilidad de uso | 0.5 | 1 | Normal |
| T8 | Portabilidad | 2 | 3 | Se requiere portabilidad. |
| T9 | Facilidad de cambio | 1 | 3 | Se requiere facilidad de cambio. |
| T10 | Concurrencia | 1 | 0 | No hay. |
| T11 | Incluye objetivos especiales de seguridad | 1 | 1 | Seguridad mínima. |
| T12 | Provee acceso directo a terceras partes | 1 | 0 | No existe acceso a terceros. |
| T13 | Se requieren facilidades especiales de entrenamiento a los usuarios | 1 | 3 | Se requiere de previo adiestramiento. |

**Tabla 5. 1: Variables para el cálculo del Factor de Complejidad Técnica.**

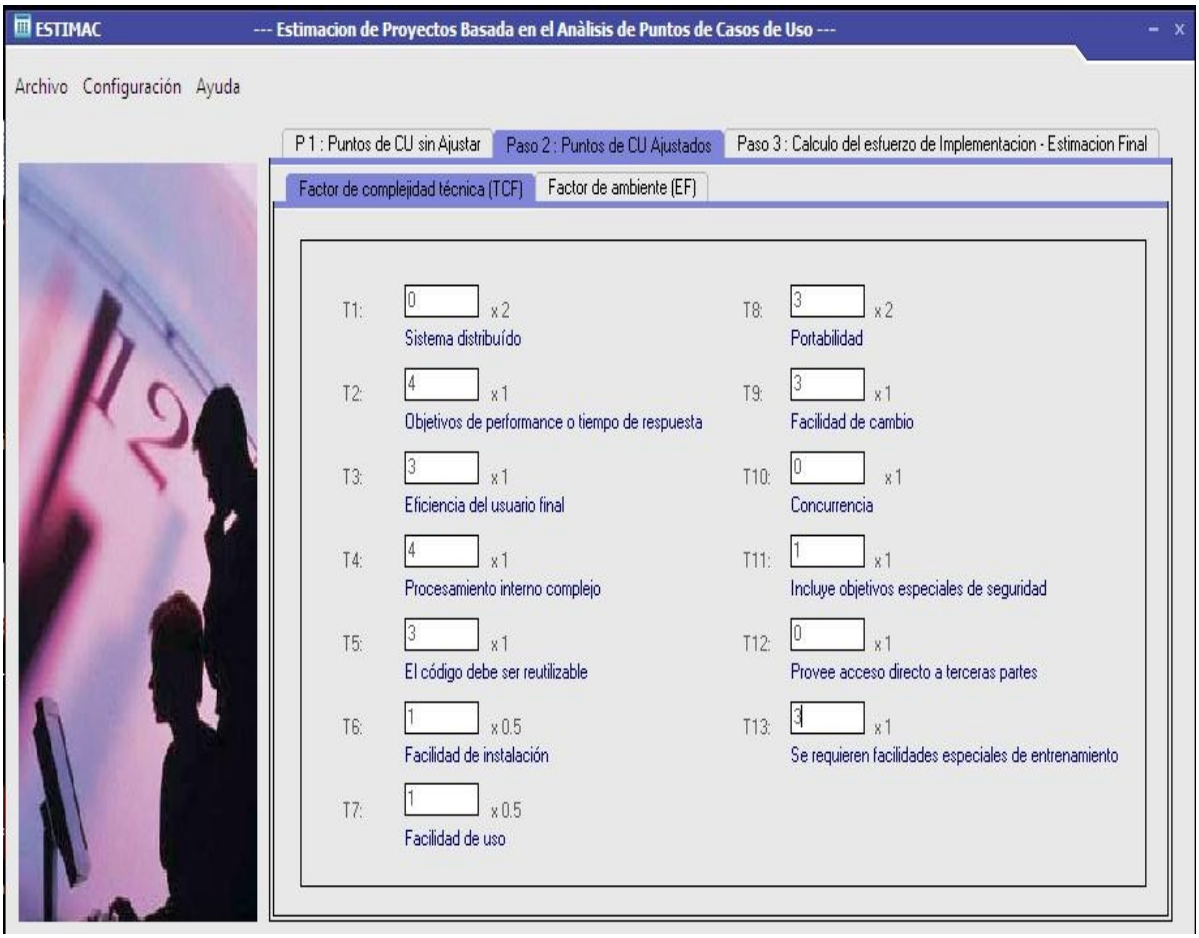



**Figura 5. 2: Variables para el cálculo del Factor de Complejidad Técnica.**

**Factor de Ambiente**

| Factor | Descripción | Peso | Valor Asignado | Comentarios |
|---|---|---|---|---|
| E1 | Familiaridad con el modelo de proyecto utilizado | 1.5 | 4 | El grupo está bastante familiarizado con el proyecto. |
| E2 | Experiencia en la aplicación | 0.5 | 4 | Se ha dedicado gran tiempo al estudio de este tipo de aplicación |
| E3 | Experiencia en orientación a objetos | 1 | 4 | El equipo ya tiene buen tiempo programando orientado a objeto. |
| E4 | Capacidad del analista líder | 0.5 | 5 | Se tiene a un profesional del tema. |
| E5 | Motivación | 1 | 5 | El grupo tiene gran motivación. |

| | | | | |
|---|---|---|---|---|
| E6 | Estabilidad de los requerimientos | 2 | 3 | Se pudieran presentar algunos cambios. |
| E7 | Personal part-time | -1 | 0 | No existen. |
| E8 | Dificultad del lenguaje de programación | -1 | 4 | Es un lenguaje con una complejidad media-alta. |

**Tabla 5. 2: Variables para el cálculo del Factor de ambiente.**

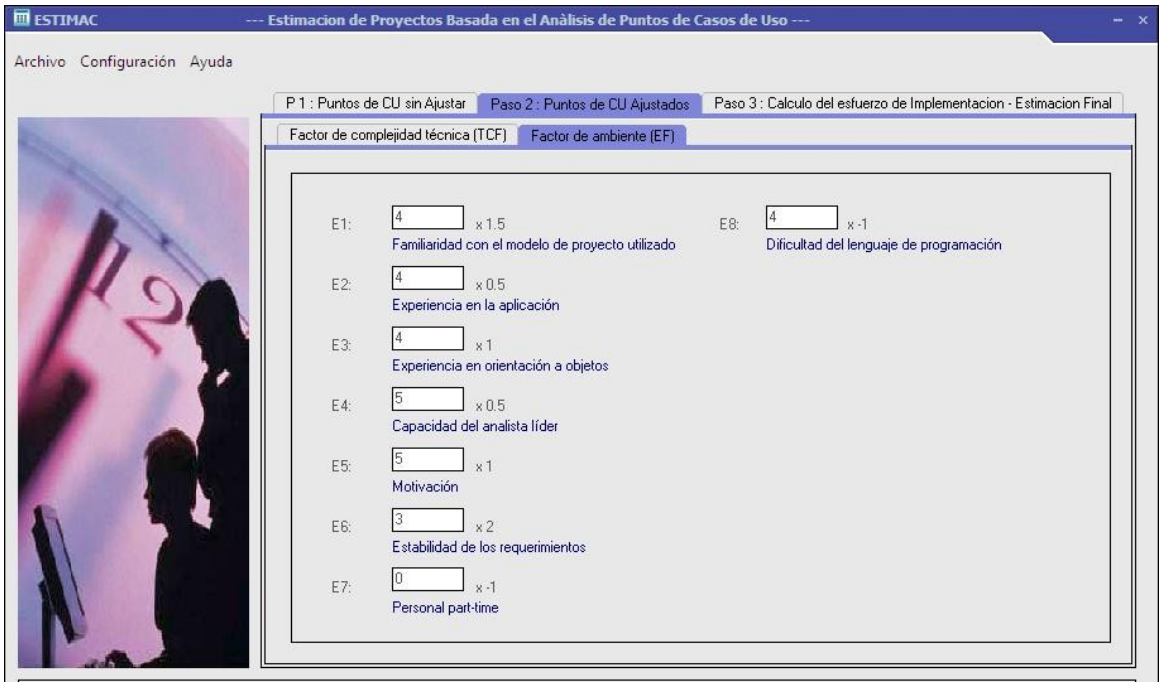



**Figura 5. 3: Variables para el cálculo del Factor de ambiente.**

**Estimación Final.**

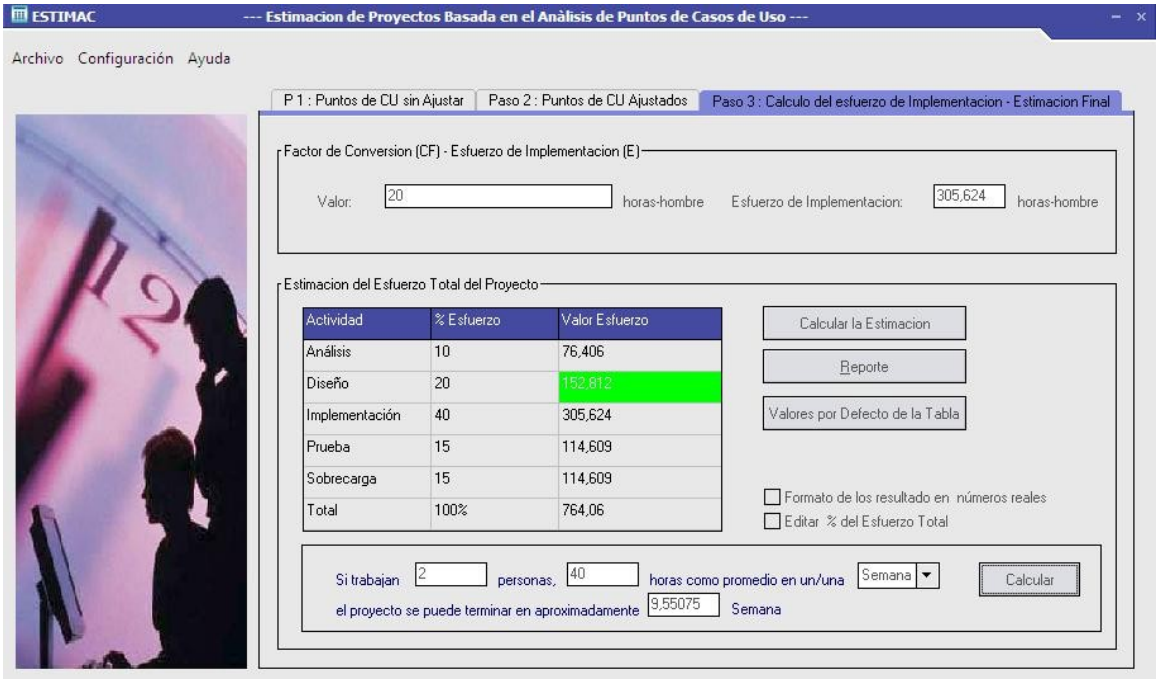



**Figura 5. 4: Cálculo del esfuerzo de implementación. Estimación final.**

Se concluyó que para realizar el proyecto, trabajando 8 horas diarias, 5 días a la semana, 2 personas, se necesitan aproximadamente 10 semanas.

## 5.2 ANÁLISIS DE COSTOS Y BENEFICIOS

Un producto informático en su desarrollo siempre tiene un costo. Que puede estar justificado por los beneficios tanto tangibles como intangibles que origina el mismo.

Teniendo en cuenta que la tecnología utilizada para el desarrollo del sistema es totalmente libre, la implementación de la aplicación no tendrá un coste elevado. Se tienen presente las importantes contribuciones que trae consigo la elaboración del mismo con la utilización de software libre, por tanto no es necesario incurrir en gastos, en el pago de licencias de uso. Sólo se pudiera tomar en cuenta para el costo del software el salario que reciben los desarrolladores.

Costo_Producto = Cant_Personas * Salario_Basico * Duración_Meses

Costo_Producto = 2 * 225 * 10 = 4500

Dada la anterior ecuación de costo como resultado se obtuvo que el producto tenga un costo relativo de $ 4500 pesos cubanos.

## CONCLUSIONES

En este capítulo se utilizó la estimación basada en Puntos de Casos de Uso, procedimiento efectivo para la estimación del tiempo de desarrollo de un producto informático. Además se abordaron aspectos relacionados con la estimación de esfuerzos (costos) de desarrollo del sistema, así como los beneficios. Por último se realizó una valoración de sostenibilidad del producto atendiendo a la dimensión del mismo.

## CONCLUSIONES

En el presente trabajo se realizó una investigación del estado del arte de los diferentes métodos de cálculo numérico, utilizados para la modelación de procesos químicos. Además de realizarse un análisis del proceso de compilación por sus distintas fases, así como se abordaron los temas de generación de código intermedio a través de las representaciones intermedias.

Se utilizó Visual Paradigm como herramienta CASE para el modelado y diseño de Numerical Compiler, a través de las técnicas de UML y utilizando RUP como metodología de desarrollo. El software fue desarrollado en el lenguaje de programación Python en su versión 2.5, utilizando las librerías PLY como herramientas para la construcción de compiladores del lenguaje utilizado.

A la culminación de este trabajo se cumplió con el objetivo general propuesto: “Diseñar un compilador en software libre para métodos numéricos.”, el cual puede ser utilizado para la implementación y ejecución de métodos numéricos.

## RECOMENDACIONES

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

- Continuar el análisis de nuevas formas intermedias con el objetivo de optimizar las salidas del Árbol de Sintaxis Abstracta utilizado en la solución del problema.
- Implementar intérpretes para las formas intermedias, tanto para la que se propone en el documento como para las que se incluyan en las futuras versiones.
- Implementar una versión de la herramienta donde se genere la forma intermedia directamente a partir de la herramienta del parser, para posteriormente interpretarla y realizar comparaciones con la herramienta desarrollada en este trabajo.

## BIBLIOGRAFÍA

### REFERENCIAS BIBLIOGRÁFICAS

**Alfred V. Aho, Ravi Sethi, Jeffrey DUllman. 1998.** *Compiladores: principios, tecnicas y herramientas.* s.l. : Pearson Educación, 1998. 9684443331.

**Barbara Liskov, John V. Guttag. 1986.** *Abstraction and Specification in Program Development.* Estados Unidos : s.n., 1986. 978-0262121125.

*Conferencia 1 Introducción al Proceso de Compilación.* **Informaticas, Universidad de las Ciencia. 2005-2006.** La Habana, Cuba : s.n., 2005-2006.

*Conferencia 5 Planificación y Estimación de Proyectos.* **Informáticas., Universidad de las Ciencias. 2006-2007.** Ciudad de la Habana, Cuba : Departamento Central del Ciencias Humanas., 2006-2007.

**Conferencia de Rectores de las Universidades Españolas (CRUE). 2004.** *Las tecnologías de la información y las comunicaciones en el sistema universitario español.* España : s.n., 2004. 84-932783-3-5.

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

**Burnette, Ed. 2005.** *Eclipse IDE Pocket Guide.* s.l. : O'Reilly, 2005. 0596100655.

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

**I.Jacobson, G. Booch, J.Rumbaugh. 2000.** *El Proceso Unificado de Desarrollo de Software* . Madrid. España : Person Educación, 2000. 84-7829-036-2.

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

**Louden, Kenneth C. 2004.** *Construcción de Compiladores.* 2004. 9706862994.

**Manuel Álvarez, Alfredo Guerra, Rogelio Lau.** *Matemática Numérica.*

**Scenna, Nicolás J. 1999.** *Modelado, Simulación y Optimización de Procesos Químicos.* 1999. 950-42-0022-2.

## BIBLIOGRAFÍA CONSULTADA.

**Beazley, David M. 2006.** ***Python Essential Reference: Essential Reference.*** **2006. 0672328623.**

**Cormen, Thomas H. 2001.** ***Introduction to Algorithms.*** **s.l. : MIT Press, 2001. 0262032937.**

**Kuo, Shan S. 1972.** ***Computer Applications of Numerical Methods.*** **s.l. : Addison Wesley, 1972. 0201039567.**

**Levine, John R. 1992.** ***Lex & Yacc: UNIX Programming Tools.*** **s.l. : O'Reilly, 1992. 1565920007.**

***MISSION CRITICAL DEVELOPMENT WITH XP AGILE PROCESSES.*** **People's Computer Company, EbscoHost. 1976. Michigan : M & T Pub., 1976.**

**Muchnick., Steven. 1997.** ***Advanced Compiler Design and Implementation.*** **s.l. : Morgan Kaufman, 1997. 1558603204.**

**Ralston, Anthony. 1986.** ***Introducción al análisis numérico.*** **1986. 9681806743.**

**Scott, Michael Lee. 2000.** ***Programming Language Pragmatics.*** **s.l. : Morgan Kaufmann, 2000. 0126339511.**

**Sergio Gálvez Rojas, Miguel Ángel Mora Mata. 2005.** ***Java a Tope: Compiladores Traductores y Compiladores con LEX/YACC , JFLEX/CUP y JavaCC.*** **Málaga : Universidad de Málaga, 2005. 84-689-1031-6.**

## GLOSARIO DE TÉRMINOS

**API**: (Application Program Interface). Conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación. Cuando se intenta estandarizar una plataforma, se estipulan unos APIs comunes a los que deben ajustarse todos los desarrolladores de aplicaciones.

**UML**:( Unified Modeling Language) Lenguaje Unificado de Modelado, es un lenguaje que proporciona un vocabulario y reglas para permitir una comunicación.

**Lex**: es un programa que genera analizadores léxicos ("scanners" o "lexers"). Se utiliza comúnmente con el generador de análisis sintáctico yacc.

**Yacc**: (Yet Another Compiler-Compiler) se trata de un generador de analizadores sintácticos, común en los sistemas Unix.

**TIC**: (Tecnologías de la Informática y las Comunicaciones) son herramientas que amplían nuestras capacidades físicas y mentales así como las posibilidades de desarrollo social.

**GRASP**: (General Responsibility Assignment Software Patterns) en español patrones generales de software para asignar responsabilidades, el nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

**JVM**: (Java Virtual Machina o Máquina virtual Java) es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial, el cual es generado por el compilador del lenguaje Java.

**RAM** :(Random Access Memory Module) se compone de uno o más chips y se utiliza como memoria de trabajo para programas y datos. Es un tipo de memoria temporal que pierde sus datos cuando se queda sin energía (por ejemplo, al apagar la computadora), por lo cual es una memoria volátil.

**OMG**: (Object Management Group) es un grupo destinado originalmente a establecer normas para distribuir sistema orientado a objetos y ahora se centra en el modelado (programas, sistemas y procesos de negocio).

**LR**: Constituye una de las más poderosas familias de algoritmos de análisis sintáctico para gramáticas independientes del contexto.

**LALR**: Intenta construir un árbol de análisis sintáctico, empezando desde la raíz y descendiendo hacia las hojas. Lo que es lo mismo que intentar obtener una derivación por la izquierda para una cadena de entrada, comenzando desde la raíz y creando los nodos del árbol en orden previo.

**PDVSA** (Petróleos de Venezuela S.A.)Es la corporación estatal de la República Bolivariana de Venezuela que se encarga de la exploración, producción, manufactura, transporte y mercadeo de los hidrocarburos, de manera eficiente, rentable, segura, transparente y comprometida con la protección ambiental; con el fin último de motorizar el desarrollo armónico del país, afianzar el uso soberano de los recursos, potenciar el desarrollo endógeno y propiciar una existencia digna y provechosa para el pueblo venezolano, propietario de la riqueza del subsuelo nacional y único dueño de esta empresa operadora.

**FORTRAN**: Lenguaje de programación informática, de alto nivel y propósito general, ha sido ampliamente adoptado por la comunidad científica para escribir aplicaciones de cálculos intensivos.

**EDO**: (Ecuación Diferencial Ordinaria) Es un tipo de ecuación diferencial caracterizada porque la variable dependiente está en función de una variable independiente, lo que las distingue de las ecuaciones derivadas.

**MSIL**: (Microsoft Intermediate Language) es un lenguaje intermedio común a todos los sistemas operativos que soporten .net framework.

**RUP**: (Rational Unified Process o Proceso Unificado de Desarrollo de Software) Su objetivo es garantizar la producción de alta calidad de software que satisface las necesidades de sus usuarios finales, en un previsible calendario y el presupuesto.

**Brecha digital:** es una expresión que hace referencia a la diferencia socioeconómica entre las distantes partes del mundo donde existe una desigual utilización a las nuevas tecnologías de la información y la comunicación (teléfonos móviles, cajeros automáticos, bíper, etc.). Como tal, la brecha digital se basa en diferencias previas al acceso a las tecnologías. Este término también hace referencia a las diferencias que hay entre grupos según su capacidad para utilizar las TIC (Tecnologías de la Información y la Comunicación) de forma eficaz, debido a los distintos niveles de alfabetización y capacidad tecnológica. También se utiliza en ocasiones para señalar las diferencias entre aquellos grupos que tienen acceso a contenidos digitales de calidad y aquellos que no.

**Front-End**: es la parte del software que interactúa con el o los usuarios.

**Back-End**: es la parte que procesa la entrada desde el front-end.

**GNU/GPL**: Licencia Pública General de GNU o más conocida por su nombre en inglés GNU General Public License o simplemente su acrónimo del inglés GNU GPL, es una licencia creada por la Free Software Foundation a mediados de los 80, y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

**GNU:** es un acrónimo recursivo que significa GNU No es Unix (GNU is Not Unix) iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre.

# ANEXOS

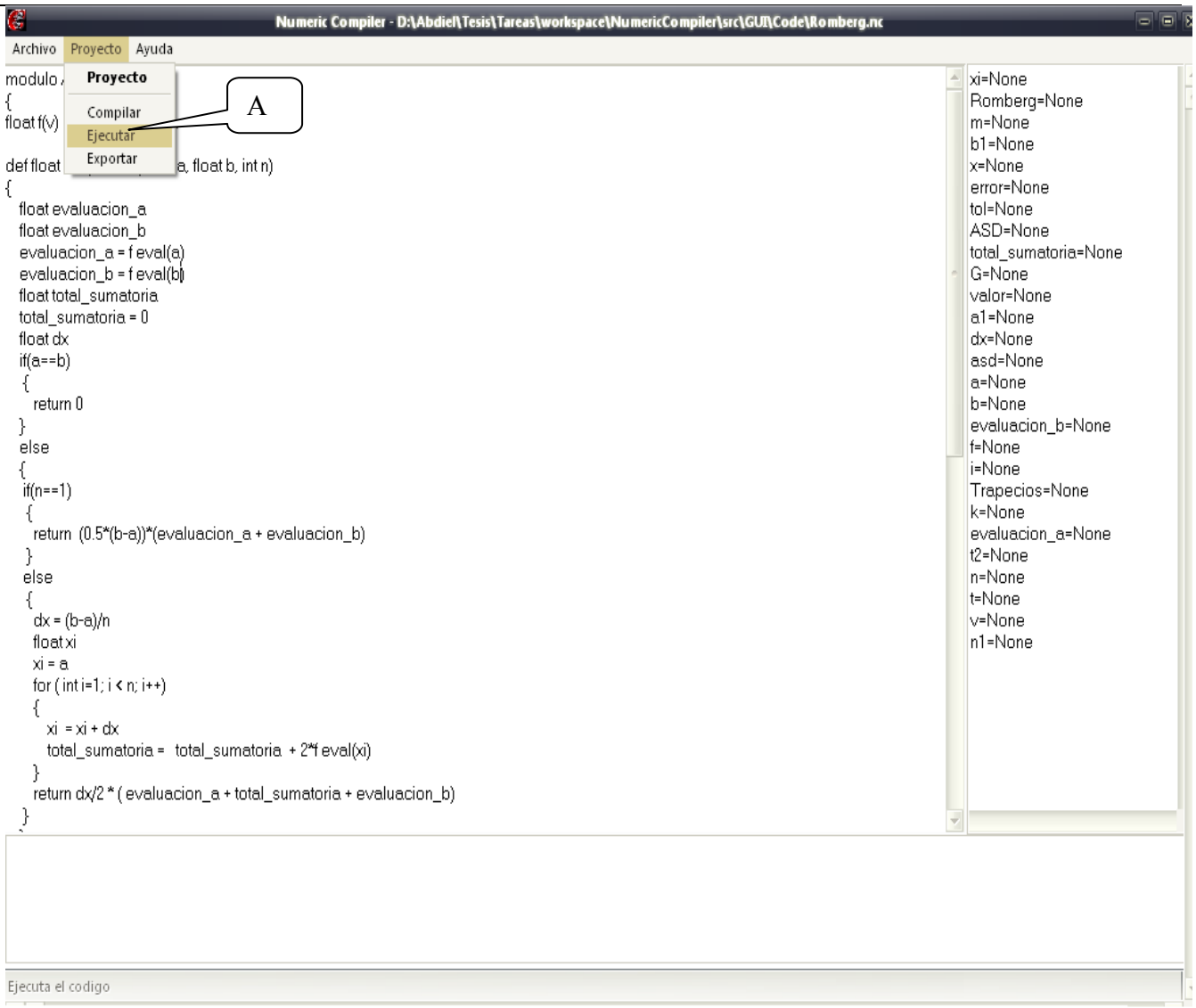
## Anexo 1

Nexo entre Front-End y Back-End: La separación del sistema en "front ends" y "back ends" es un tipo de abstracción que ayuda a mantener las diferentes partes del sistema separadas. La idea general es que el front-end sea el responsable de recolectar los datos de entrada del usuario, que pueden ser de muchas y variadas formas, y procesarlas de una manera conforme a la especificación que el back-end pueda usar. La conexión del front-end y el back-end es un tipo de interfaz.

## Anexo 2

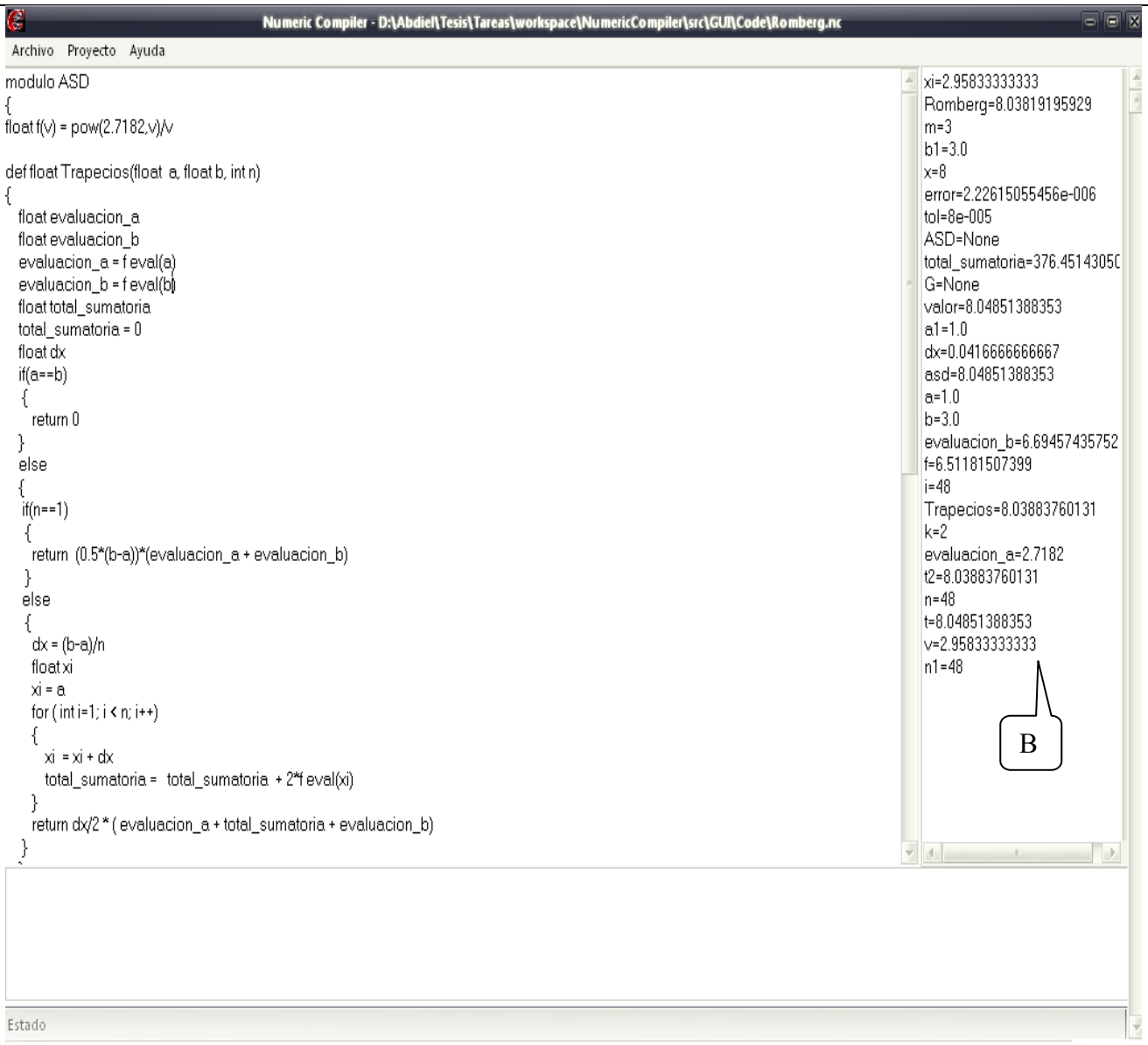
**Caso de Uso Expandido Ejecutar Código.**

| | |
|---|---|
| Caso de Uso: | Ejecutar Código. |
| Actor(es): | Usuario |
| Propósito: | Ejecutar el código escrito por el usuario para obtener los valores de las variables definidas en el código del usuario. |
| Resumen: | El caso se uso se inicia cuando el usuario desea obtener los resultados de los valores al interpretar cada paso del código de su programa. |
| Referencias: | R1, R2,R3 y R4 |
| Precondiciones: | El Usuario accede al sistema y se encuentra en la ventana principal.<br>No deben existir errores en el código. |



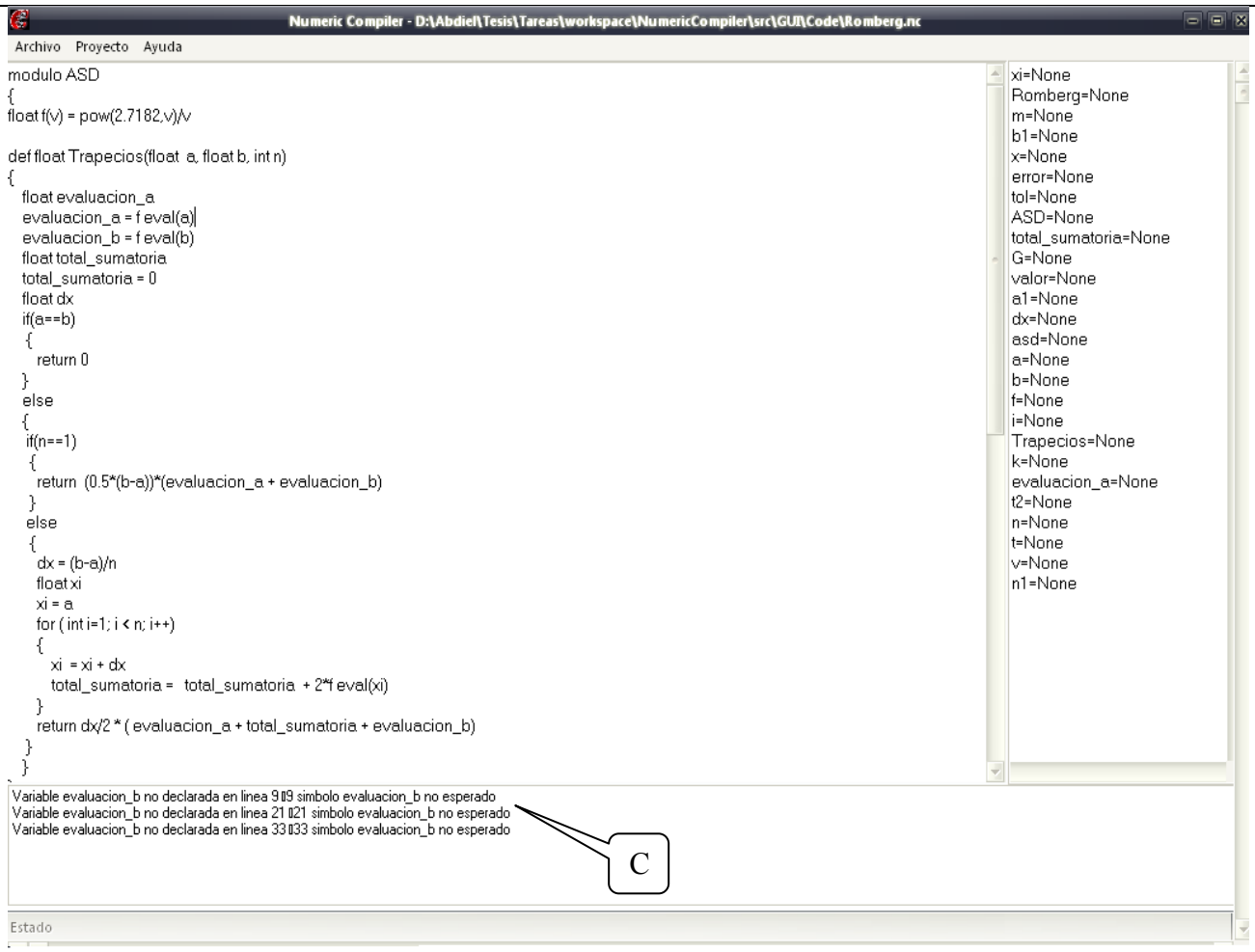

Pantalla 1

| Acción del Actor | Respuesta del Sistema |
|---|---|
| El usuario escoge la opción de ejecutar código (A). | El sistema compila si es necesario el código.<br>El sistema ejecuta el código. |




Pantalla 2

| | |
|---|---|
| Sección “No existen Errores”. | |
| | El sistema muestra todos los valores de las variables que gestiona (B). |




Pantalla 3

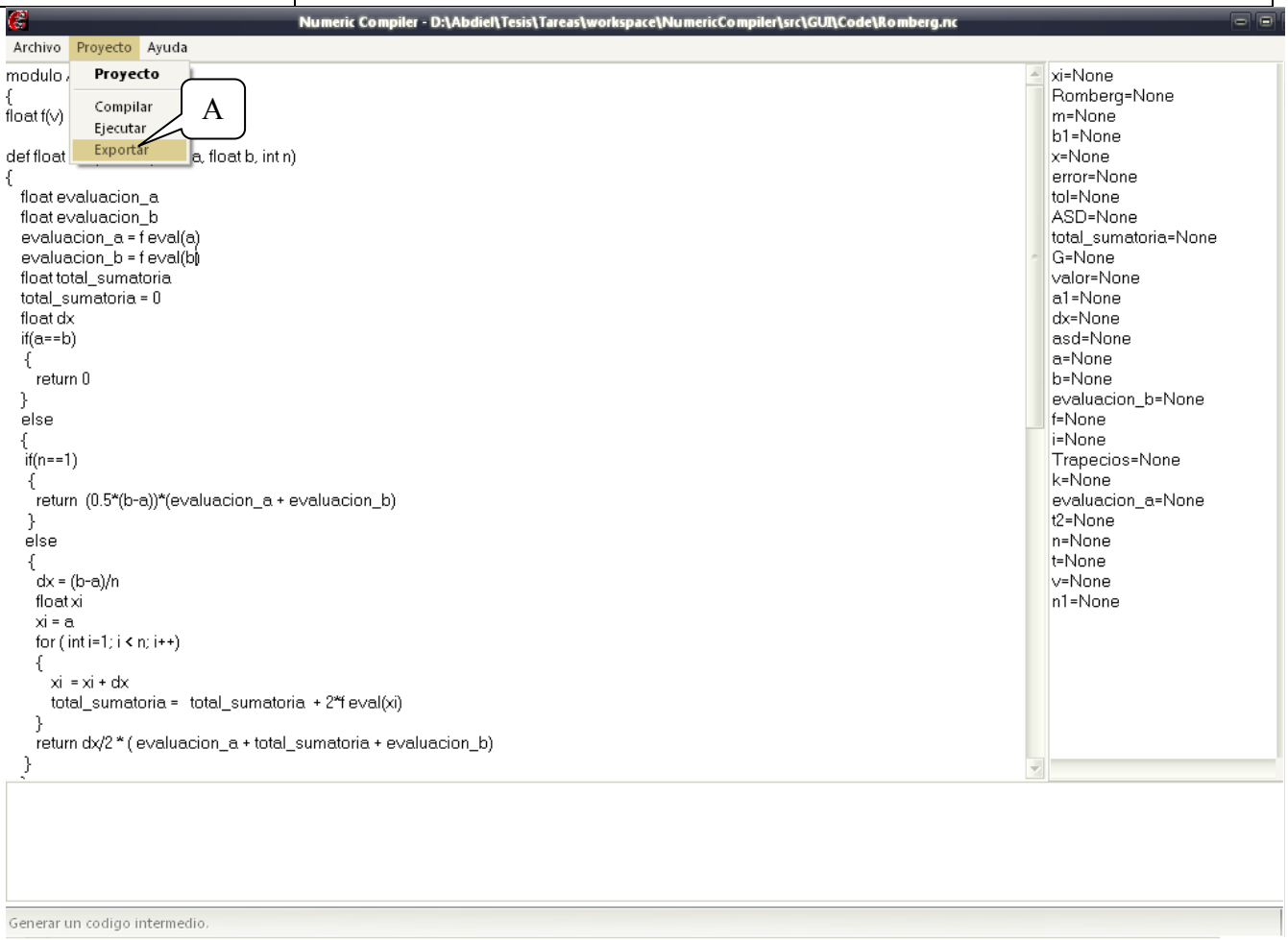
| | |
|---|---|
| Sección “Existen Errores”. | |
| | El sistema muestra en el panel de errores la información de los errores encontrados (C). |

**Tabla A. 1: Caso de Uso Expandido Ejecutar Código.**

**Caso de Uso Expandido Exportar Código.**

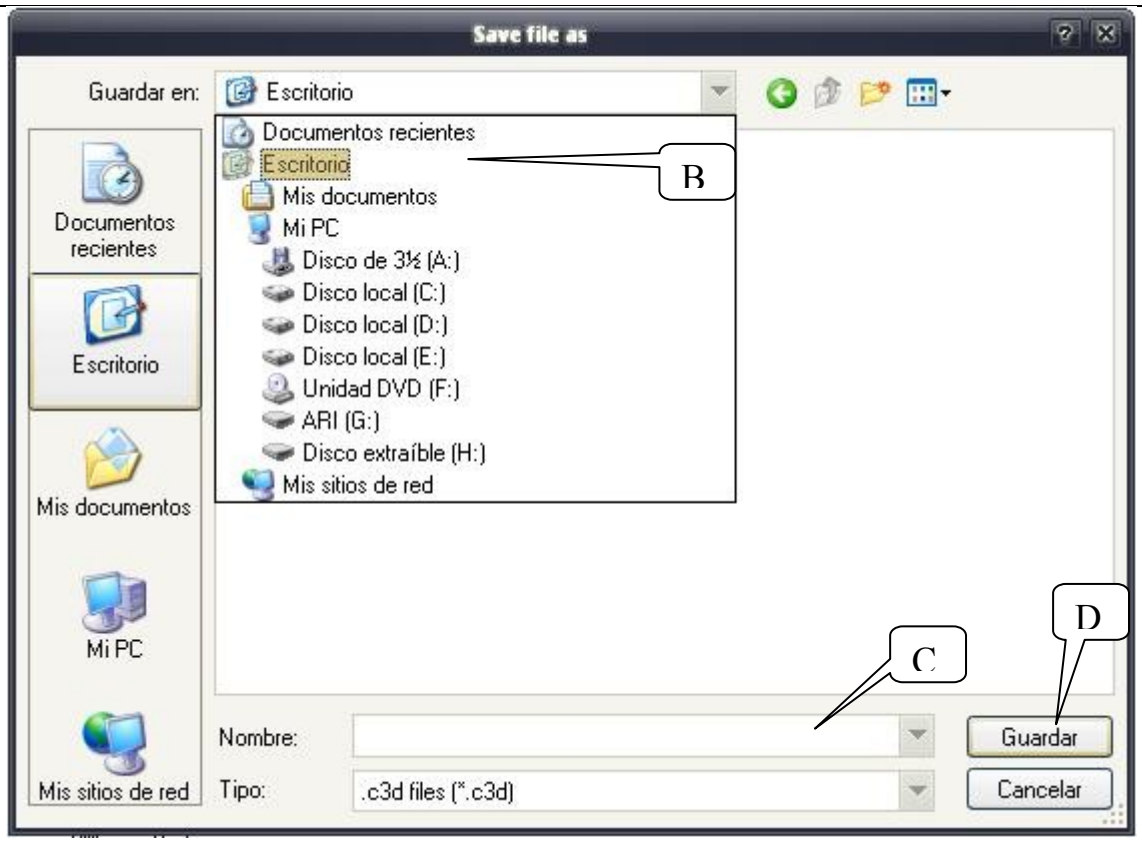
| | |
|---|---|
| Caso de Uso: | Exportar Código. |
| Actor(es): | Usuario |
| Propósito: | Generar el código intermedio del código escrito por el usuario. |
| Resumen: | El caso de uso se inicia cuando el usuario requiere exportar su código para ser utilizado por otro sistema. |

| Referencias: | |
|---|---|
| Precondiciones: | El Usuario accede al sistema y se encuentra en la ventana principal.<br>No deben existir errores en el código. |

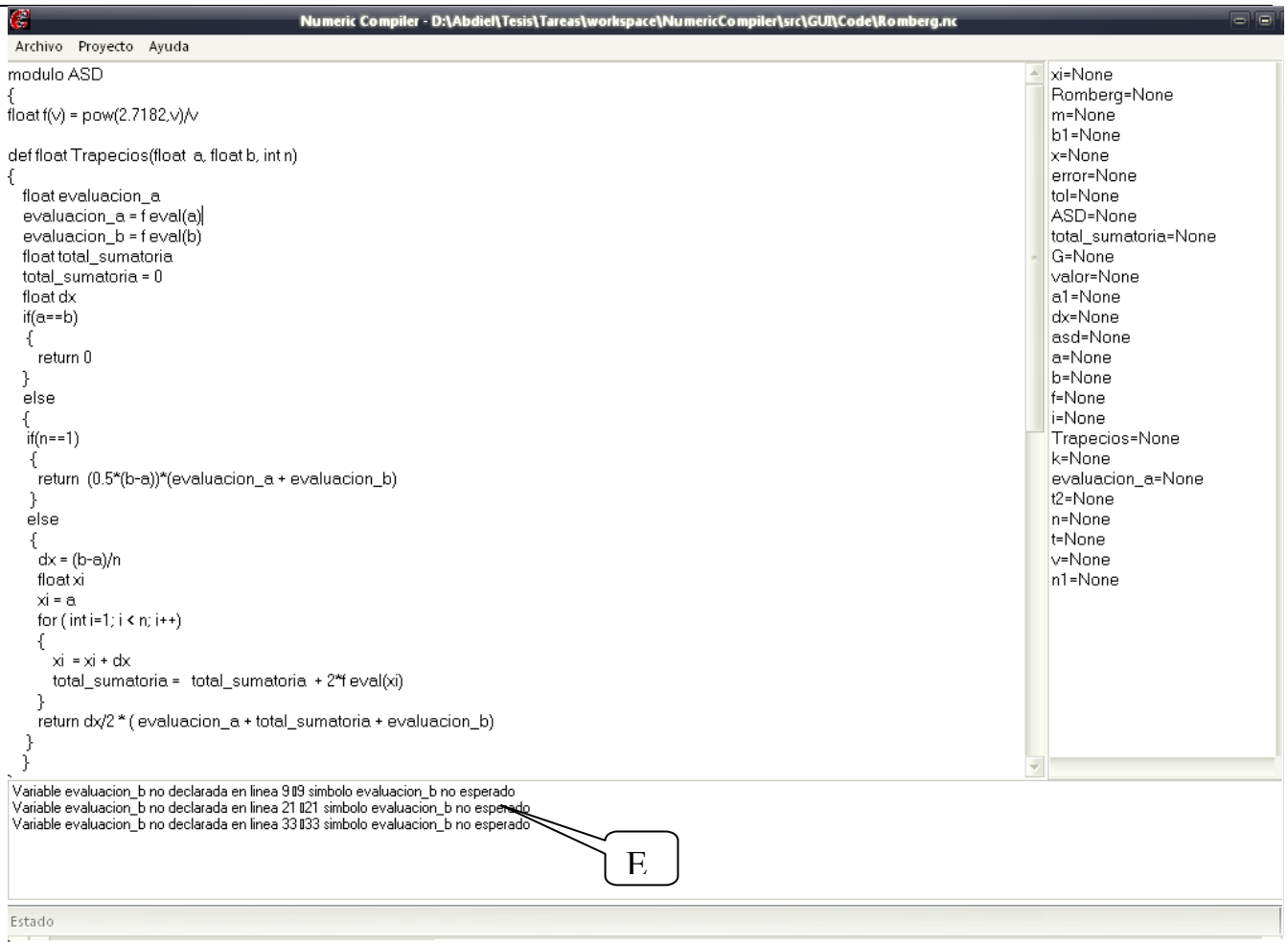



**Pantalla 1**

| Acción del Actor | Respuesta del Sistema |
|---|---|
| El usuario escoge la opción de exportar del menú Proyecto (A). | El sistema compila si es necesario el código. |

<table>
<tr><td colspan="2"> 

Pantalla 2</td></tr>
<tr><td colspan="2">Sección “No existen Errores”.</td></tr>
<tr><td></td><td>El sistema muestra una ventana de diálogo para seleccionar la carpeta donde se desea guardar el archivo (Pantalla 2).<br>.</td></tr>
<tr><td>El usuario selecciona la carpeta donde desea guardar el archivo.(B)<br>El usuario escribe el nombre del archivo. (C)<br>El usuario da clic en el botón Guardar. (D)</td><td>El sistema exporta el archivo con el nombre seleccionado por el usuario y con extensión c3d.</td></tr>
</table>




**Pantalla 3**

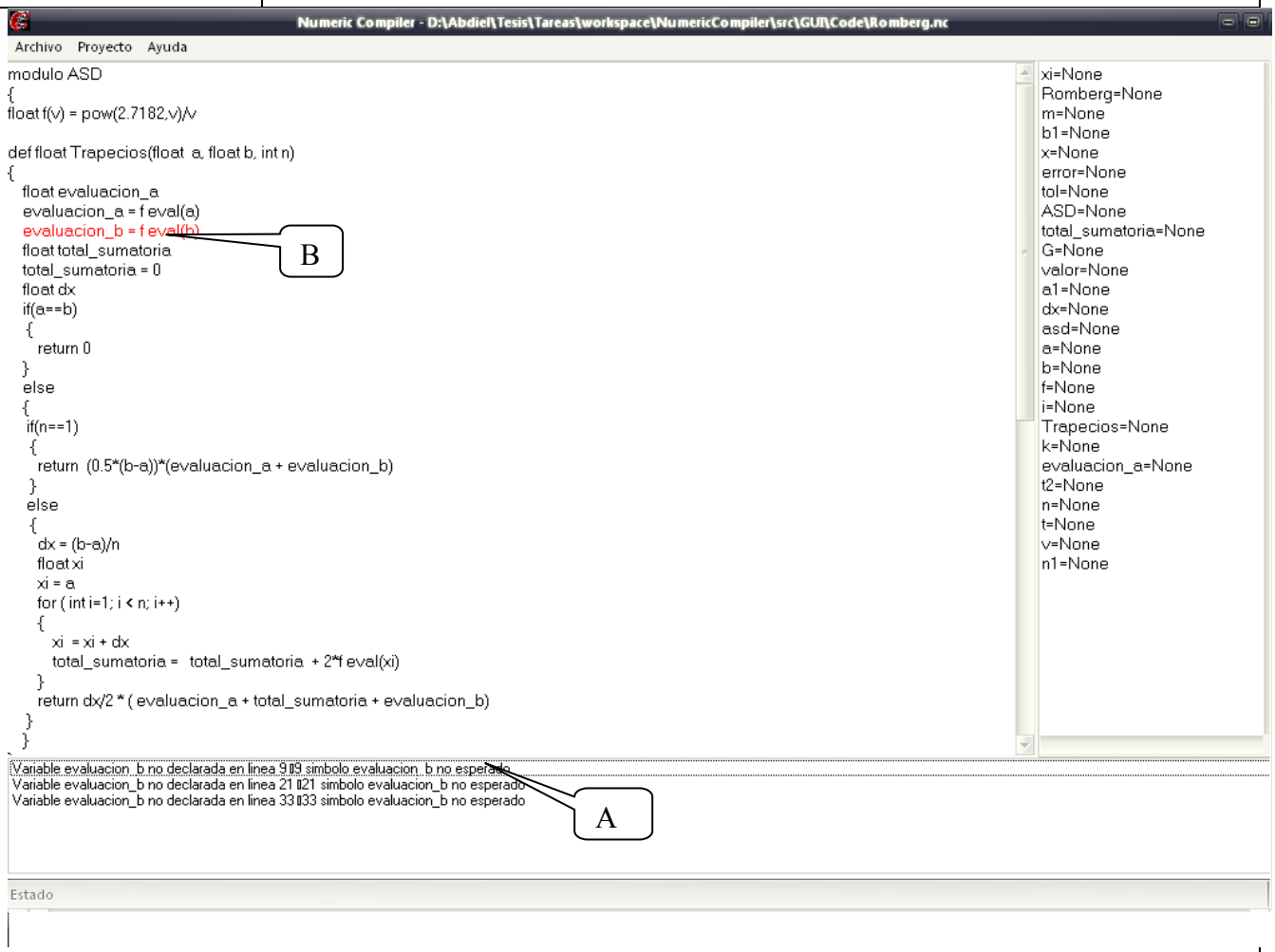
| | |
|---|---|
| Sección “Existen Errores”. | |
| | El sistema muestra en el panel de errores la información de los errores encontrados (E). |

**Tabla A. 2: Caso de Uso Expandido Exportar Código.**

**Caso de Uso Expandido Gestionar Errores.**

| | |
|---|---|
| Caso de Uso: | Gestionar Errores. |
| Actor(es): | Usuario |
| Propósito: | Ayudar al usuario a encontrar sus errores dentro del código escrito por él. |
| Resumen: | El caso de uso se inicia cuando el usuario da clic sobre algún error dentro |

| | |
|---|---|
| | de la ventana de errores, entonces se localiza el error dentro del código dentro de ventada de edición. |
| Referencias: | R5 |
| Precondiciones: | El Usuario accede al sistema y se encuentra en la ventana principal. Se debe haber compilado el código y deben existir errores en el mismo. |




Pantalla 1

| Acción del Actor | Respuesta del Sistema |
|---|---|
| El usuario da clic en el error que desea buscar.(A) | El sistema busca la línea que generó el error dentro del código del usuario.<br>La línea se resalta en color rojo (B). |

**Tabla A. 3: Caso de Uso Expandido Gestionar Errores.**

## Anexo 3

## Diagramas de Clases del Diseño por Paquetes.

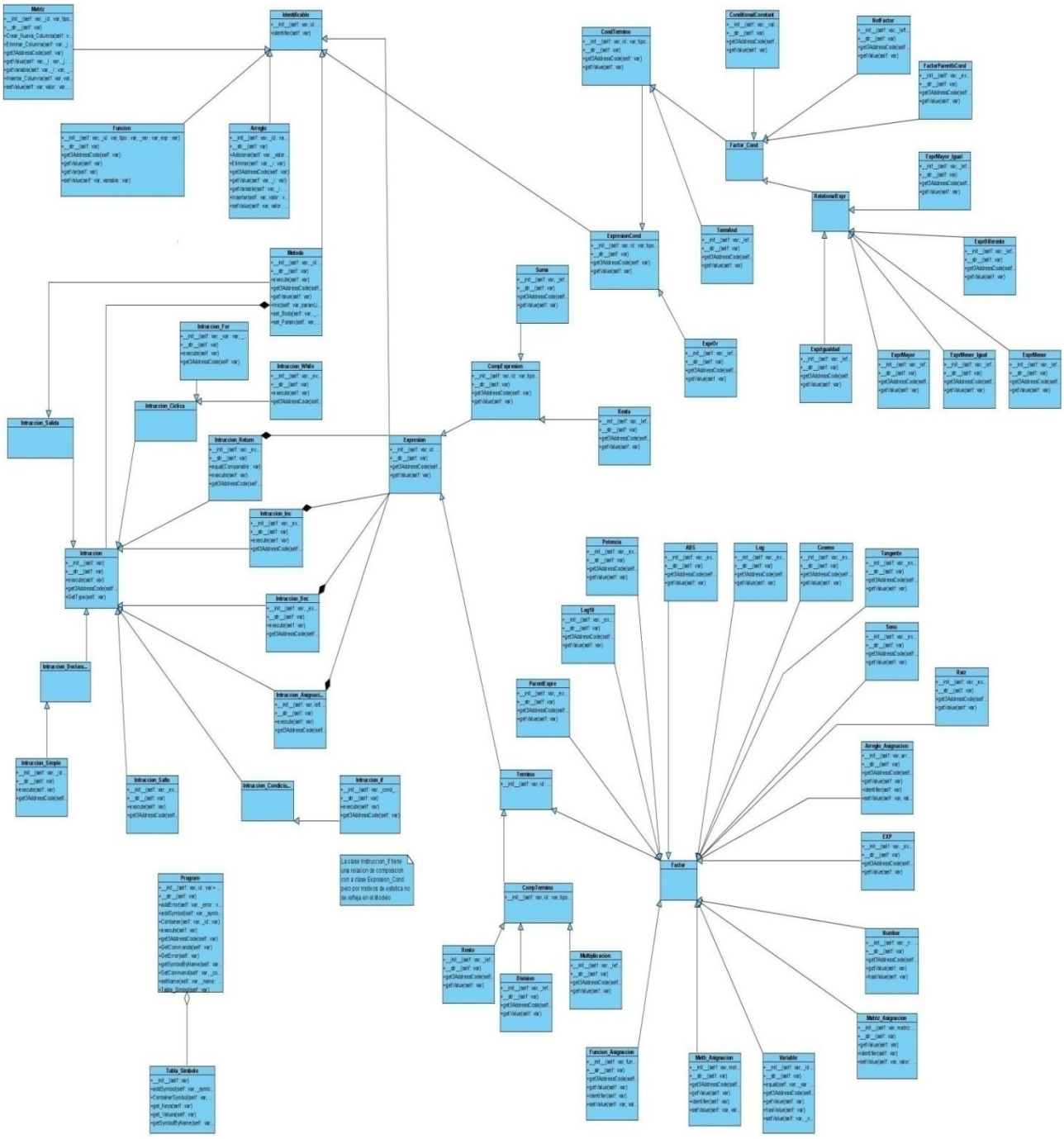


**Figura A. 1: Diagrama de clases del Diseño del Paquete AST.**

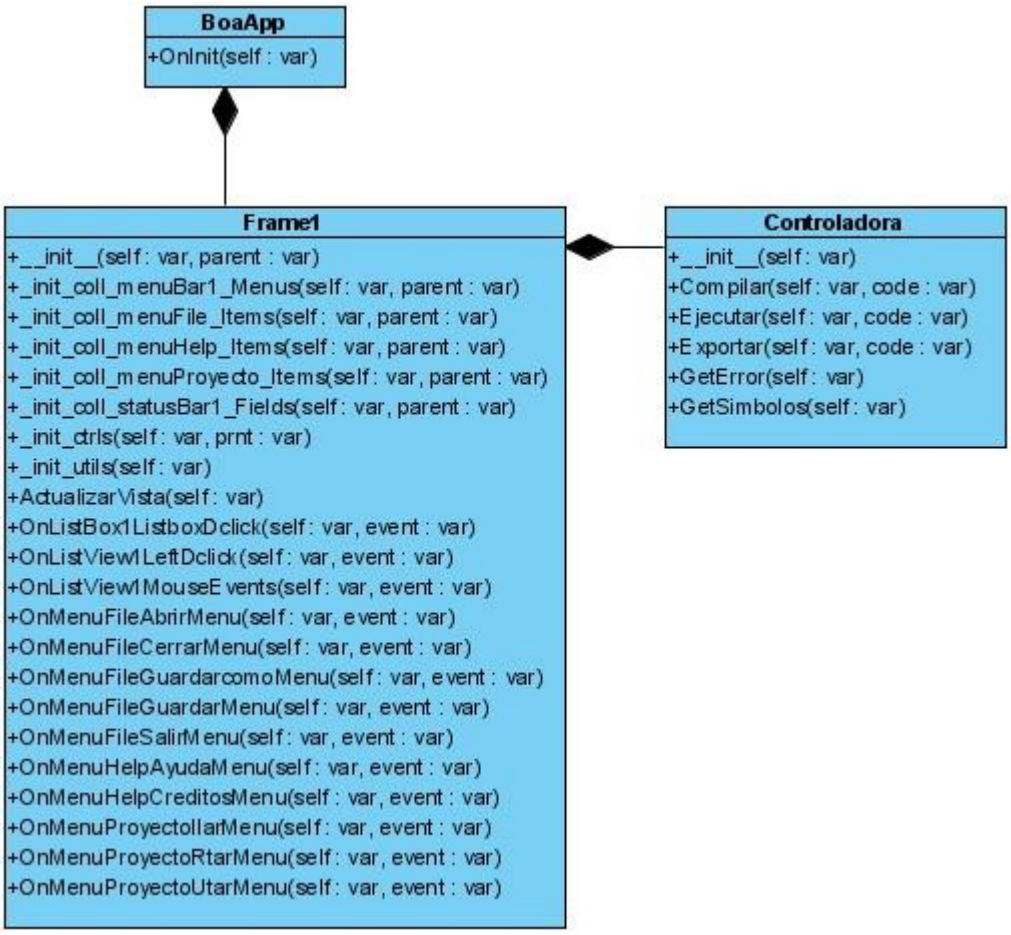



**Figura A. 2: Diagrama de Clases del Diseño del Paquete GUI**

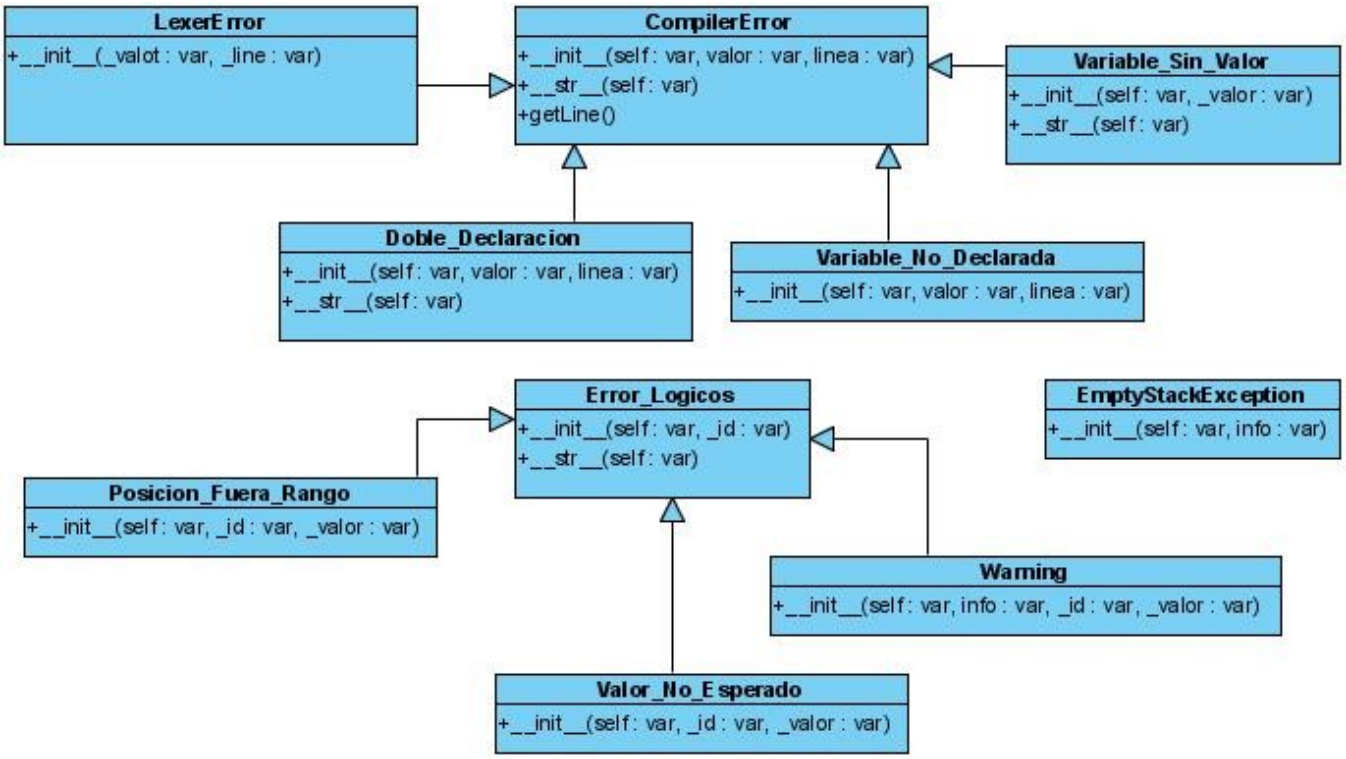



**Figura A. 3: Diagrama de Clases del Diseño del Paquete Error**

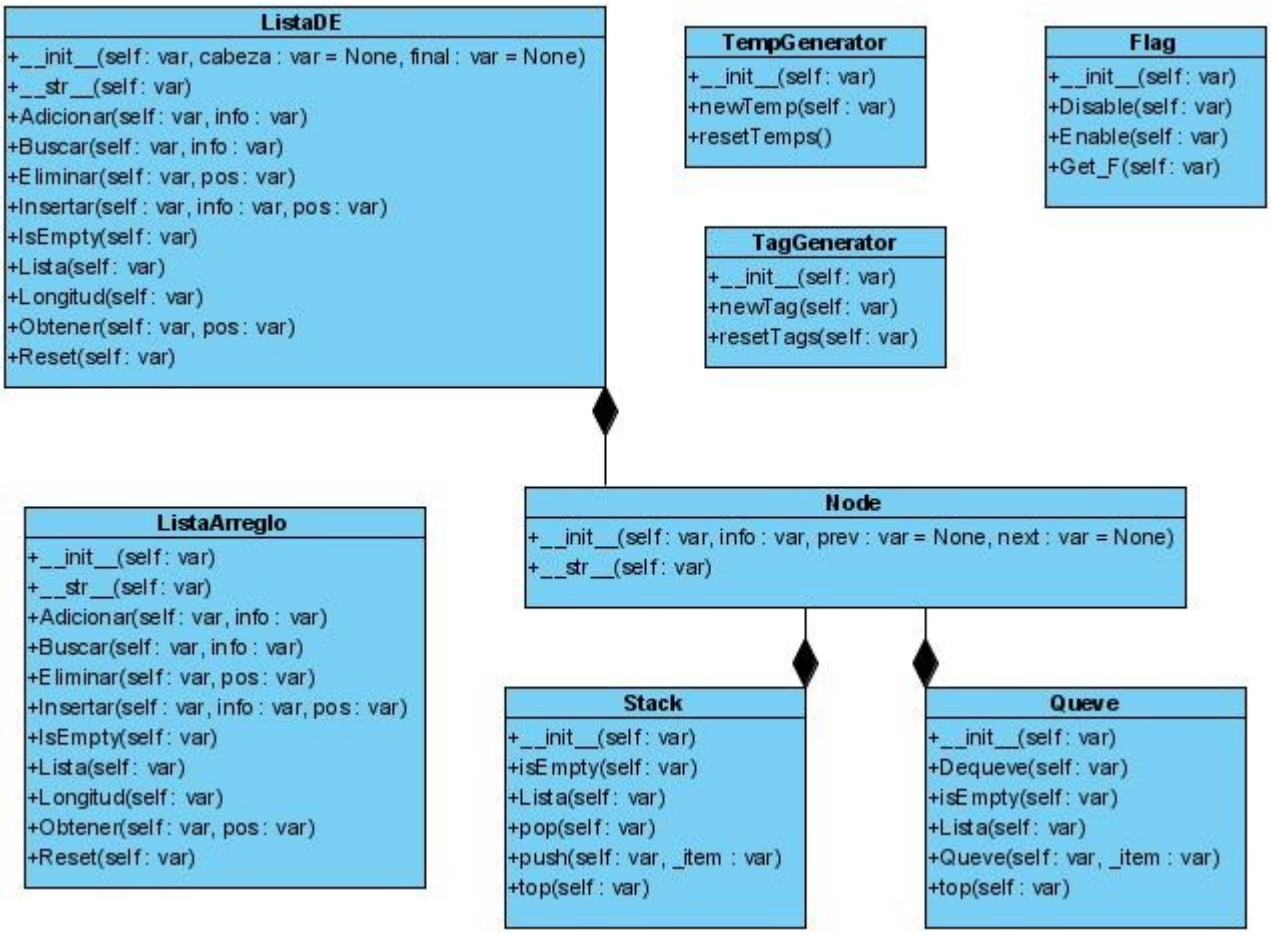



**Figura A. 4: Diagrama de Clases del Diseño del Paquete Estructuras**

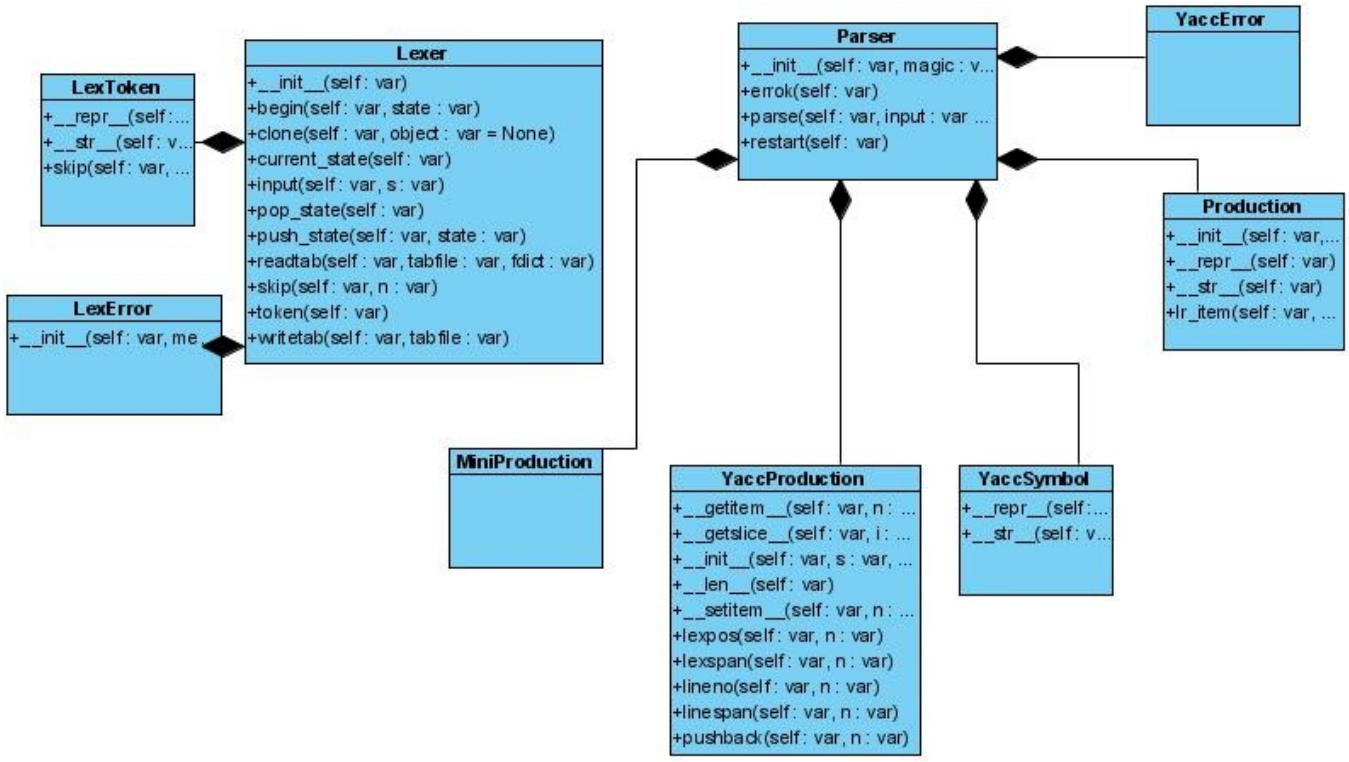



**Figura A. 5: Diagrama de Clases del Diseño del Paquete PLY**

**Anexo 4**

Patrón Creador: Asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos:

- B agrega 10s objetos A.
- B contiene 10s objetos A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serían transmitidos a A cuando este objeto sea creado (así que B es un experto respecto a la creación de A).
- B es un creador de los objetos A.
- Si existe más de una opción, prefiera la clase B que agregue o contenga la clase A. (Lerman, 1999)

**Anexo 5**

Patrón Controlador:

- El "sistema" global (controlador de fachada).
- La empresa u organización global (controlador de fachada).
- Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
- Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados "Manejador<NombreCasodeUso>"(controlador de casos de uso). (Lerman, 1999)