



UNIVERSIDAD NACIONAL DE LOJA

ÁREA DE LA ENERGÍA, LAS INDUSTRIAS Y LOS RECURSOS NATURALES NO RENOVABLES

Carrera de Ingeniería en Sistemas

TÍTULO:

“DESARROLLO DE UNA HERRAMIENTA DE SOFTWARE QUE PERMITA GRAFICAR UN DIAGRAMA DE FLUJO GENERANDO SU PSEUDOCÓDIGO Y DE UN PSEUDOCÓDIGO OBTENER SU DIAGRAMA DE FLUJO, PARA LOS ALUMNOS DE LA CARRERA DE INGENIERÍA EN SISTEMAS DE LA UNIVERSIDAD NACIONAL DE LOJA.”

“TESIS PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN SISTEMAS”

AUTORA:

Andrea Natasha Salinas Ochoa

DIRECTOR:

Ing. Edison Leonardo Coronel Romero

Loja – Ecuador

2012



CERTIFICACIÓN DEL DIRECTOR

Loja, Junio de 2012

Ing. Edison Leonardo Coronel Romero

DIRECTOR DE TESIS

CERTIFICA:

Que la Srta. Egresada Andrea Natasha Salinas Ochoa, realizó el trabajo de investigación titulado “**DESARROLLO DE UNA HERRAMIENTA DE SOFTWARE QUE PERMITA GRAFICAR UN DIAGRAMA DE FLUJO GENERANDO SU PSEUDOCÓDIGO Y DE UN PSEUDOCÓDIGO OBTENER SU DIAGRAMA DE FLUJO, PARA LOS ALUMNOS DE LA CARRERA DE INGENIERÍA EN SISTEMAS DE LA UNIVERSIDAD NACIONAL DE LOJA**”, ha cumplido con todos los requerimientos y requisitos que contempla el reglamento general de la Universidad Nacional de Loja, además todo el proceso de desarrollo fue coordinado y revisado por mi persona, autorizo su presentación, sustentación y defensa ante el tribunal respectivo.

Es todo cuanto puedo certificar en honor a la verdad.

.....

Ing. Edison Leonardo Coronel Romero

DIRECTOR DE TESIS



AUTORÍA

Las ideas, opiniones, comentarios, conclusiones y recomendaciones que se encuentran en el presente proyecto son absoluta responsabilidad de la autora.

Andrea Natasha Salinas Ochoa

AGRADECIMIENTO

“Agradece a la llama su luz, pero no olvides el pie del candil que, constante y paciente, la sostiene en la sombra”

Antes que a todos quiero agradecer A Dios, a la Virgen del Cisne y al Divino niño Jesús por darme las fuerzas necesarias en los momentos en que más las necesité, gracias a ellos, pude guiar mi vida cada día, y culminar esta tesis.

Mis eternos agradecimientos a mi mami Naya Ochoa Álvarez y a mi papi Fabián Salinas Pacheco por darme la vida, su amor, confianza y apoyo incondicional; ya que gracias a ellos soy quien soy hoy en día, desde el día en que nací son los que han velado por mi salud, mis estudios, mi educación, alimentación; son a ellos a quien les debo todo, horas de consejos, de regaños, de reprimendas; compartiendo mis tristezas y alegrías, triunfos y fracasos; estoy segura que todo lo que mis padres hicieron por mí es para formarme como un ser integral. Mami y Papi me siento extremadamente orgullosa de ser su hija y gracias a Dios los tengo junto a mí para celebrar esta meta propuesta.

A mis queridos hermanos Stefany, Astrid, Fabián y Andrés por brindarme cada día sus consejos, confianza y apoyo incondicional.

A mi abuelita Amalia Álvarez Castillo, por su apoyo, cariño y amor. A pesar de la distancia siempre mi querida abuelita te has hecho presente conmigo, agradezco sus llamadas telefónicas para darme ánimo.

A mi novio, Danny Ontaneda, simplemente por ser como es. Con todos sus defectos y virtudes. Gracias por apoyarme, animarme, caminar a mi lado durante todo este tiempo y mostrarme que todo con esfuerzo se logra.

A mi director de tesis, Ing. Edison Coronel por su esfuerzo y dedicación, quien con sus conocimientos, experiencia, paciencia y motivación ha logrado en mí que pueda terminar mis estudios con éxito. También me gustaría agradecer a mis profesores de la carrera por todos sus conocimientos compartidos. Al Ing. Manuel Cordova e Ing. Michael Yanangomez por los consejos, el apoyo y el ánimo que me brindaron.

Un agradecimiento especial a los esposos Lcda. Luz Peralta y Lic. Edwin Ontaneda, por brindarme cada uno todo su apoyo, colaboración, cariño y amistad sin ningún interés desde el día en que me conocieron.

A todos MUCHAS GRACIAS.

Andrea Salinas Ochoa.

DEDICATORIA

Quiero dedicar esta tesis a Dios, a la Virgen del Cisne y al Divino Niño Jesús por darme paciencia y llenar mi alma de fortaleza en los momentos más difíciles de mi existencia y así poder hacer realidad este gran sueño.

A mi mami Naya Ochoa Álvarez y mi papi Fabián Salinas Pacheco, quienes han sido el pilar fundamental en mi vida, con cariño y sacrificio supieron motivarme para salir adelante enseñándome que el éxito se logra mediante la constancia.

A mis hermanos Stefany, Astrid, Fabián y Andrés, a mi abuelita Amalia Álvarez Castillo, que siempre me han acompañado, de una u otra forma me apoyaron incondicionalmente en mi formación personal y académica.

A mi novio Danny Ontaneda, por acompañarme, escucharme, apoyarme y ayudarme en cada momento.

A toda mi querida familia.

Andrea Salinas Ochoa.



CESIÓN DE DERECHOS

Andrea Natasha Salinas Ochoa, autora del presente Trabajo de Tesis certifica la propiedad intelectual a la Universidad Nacional de Loja, y autoriza a la misma para hacer uso del presente documento como considere conveniente.

Andrea Natasha Salinas Ochoa



A. TÍTULO

“DESARROLLO DE UNA HERRAMIENTA DE SOFTWARE QUE PERMITA GRAFICAR UN DIAGRAMA DE FLUJO GENERANDO SU PSEUDOCÓDIGO Y DE UN PSEUDOCÓDIGO OBTENER SU DIAGRAMA DE FLUJO, PARA LOS ALUMNOS DE LA CARRERA DE INGENIERÍA EN SISTEMAS DE LA UNIVERSIDAD NACIONAL DE LOJA.”

B. RESUMEN

En la actualidad la programación es una herramienta necesaria, que permite desarrollar actividades específicas de manera más rápida y eficiente; ahorrando tiempo, dinero y espacio. El proyecto de investigación que se detalla a continuación, está enfocado como ayuda a los estudiantes de la unidad de Metodología de Programación en la Carrera de Ingeniería en Sistemas perteneciente a la Universidad Nacional de Loja.

PsGram es una herramienta de software que permite graficar un Diagrama de Flujo generando su Pseudo-Código y de un Pseudo-Código obtener su respectivo diagrama de flujo, a su vez éste Pseudo-Código lo traduce a un programa equivalente en lenguaje JAVA (el lenguaje objeto). Otra función que puede desempeñar PsGram es informar al usuario de la presencia de errores (léxicos, sintéticos y semánticos).

En el desarrollo de éste software se utilizó los frameworks: JFlex, java-cup y JGraphX. Los dos primeros se utilizaron para el desarrollo del Pseudo-Código, éstos ayudaron a visualizar si el lenguaje fuente ingresado tiene correcta su escritura, sintaxis y semántica; mientras que JGraphX es una librería de java que facilita trabajar con gráficos permitiendo crear el módulo de Diagrama de Flujo. PsGram está desarrollado con la metodología ICONIX que permite un desarrollo ágil, con muy buena documentación, es bastante flexible y se adapta mejor a la Programación Orientada a Objetos; su filosofía se basa en que es interactivo e incremental.

La finalidad del proyecto es llegar a ser una herramienta de apoyo que facilite el aprendizaje en la unidad de Metodología de Programación, ayudando a implantar buenas bases para un futuro programador.



SUMMARY

Currently the programming is a necessary tool, which allows specific activities more quickly and efficiently, saving time, money and space. The research of the project that is described below is aimed to help students of the unit methodology of the Systems Engineering Career belonging to the National University of Loja.

PsGram is a software tool for graphing a flowchart generating it's pseudo-code and a pseudo-code to obtain their respective flowchart, at the same time this Pseudo-code translates it into an equivalent program in Java (the object language). Another role is to inform the user of PsGram the presence of errors (lexical, semantic and synthetic).

In developing this software were used frameworks: JFlex, java-cup and JGraphX. The first two were used for the development of pseudo-code; they helped to visualize if the source language has entered correct their writing, syntax and semantics, while JGraphX is a java library that facilitates working with graphics module allowing to create the flowchart. PsGram is developed with the ICONIX methodology that permits an agile development, with very good documentation; it's quite flexible and better suited to the Object-Oriented Programming, his philosophy it is based in that is interactive and incremental.

The project's goal is to become a support tool to facilitate learning in Programming Methodology unit, helping to establish good foundations for a future programmer.

ÍNDICE

ÍNDICE GENERAL

PORTADA.....	II
CERTIFICACIÓN DEL DIRECTOR.....	II
AUTORÍA.....	III
AGRADECIMIENTO.....	IV
DEDICATORIA.....	V
CESIÓN DE DERECHOS.....	VI
A. TÍTULO.....	1
B. RESUMEN.....	2
SUMMARY.....	3
ÍNDICE.....	4
ÍNDICE GENERAL.....	4
ÍNDICE DE FIGURAS.....	7
ÍNDICE DE TABLAS.....	8
ÍNDICE DE DIAGRAMAS.....	9
C. INTRODUCCIÓN.....	10
D. REVISIÓN DE LITERATURA O MARCO TEÓRICO.....	12
CAPÍTULO I.....	12
1. COMPILADORES.....	12
1.1. ANALIZADOR LÉXICO.....	13
1.2. ANALIZADOR SINTÁCTICO.....	15
1.3. ANALIZADOR SEMÁNTICO.....	16

1.4. GENERACIÓN DE CÓDIGO INTERMEDIO	17
1.5. OPTIMIZACIÓN DE CÓDIGO INTERMEDIO	17
1.6. GENERACIÓN DE CÓDIGO OBJETO	17
CAPÍTULO II	18
2. JAVA-CUP, JFLEX, JGRAPH.....	18
2.1. INSTALACIÓN DE JAVA-CUP Y JFLEX	18
2.1. JFLEX	21
2.3. JAVA-CUP	23
2.3. JGRAPHX	25
E. METODOLOGÍA	28
MÉTODOS	28
TÉCNICAS	28
METODOLOGÍA.....	29
F. RESULTADOS.....	30
1. DESARROLLO DE LA PROPUESTA ALTERNATIVA	30
1.1. RECOLECCIÓN DE INFORMACIÓN	30
1.2. ANALIZADOR LÉXICO SINTÁCTICO Y SEMÁNTICO	30
1.2.1. ANALIZADOR LÉXICO	31
1.2.2. ANALIZADOR SINTÁCTICO	39
1.2.3. ANALIZADOR SEMÁNTICO	51
1.3. MÓDULO DE DIAGRAMA DE FLUJO.....	55
1.4. DESARROLLO DE PsGram.....	56
1.5. PRUEBAS	68
1.5.1. PRUEBA DE VALIDACIÓN.....	68
2. VALORACIÓN TÉCNICA ECONÓMICA AMBIENTAL	76
G. DISCUSIÓN.....	78
1. METODOLOGÍA ICONIX	78
1.1. REQUERIMIENTOS.....	78
1.1.1. Panorama General.....	78
1.1.2. Metas.....	78
1.1.3. Funciones del Sistema.....	78
1.1.4. Atributos del Sistema	80
1.2. MODELO DEL DOMINIO	81
1.2.1. GLOSARIO DE TÉRMINOS	81
1.2.2. MODELO CONCEPTUAL DEL DOMINIO.....	82

1.3. MODELO DE CASOS DE USO	83
1.3.1. DETERMINACIÓN DE CASOS DE USO	83
1.3.2. DIAGRAMA DE CASOS DE USO	84
1.4. PROTOTIPOS DE PANTALLA, DESCRIPCION DE CASOS DE USO, DIAGRAMA DE SECUENCIA	85
1.6. ACTUALIZACIÓN DEL MODELO DEL DOMINIO	128
1.8. DIAGRAMA DE CLASES FINAL	129
1.9. DIAGRAMA DE PAQUETES	139
1.10. MODELO DE ARQUITECTURA	139
1.11. DIAGRAMA DE COMPONENTES	140
H. CONCLUSIONES	141
I. RECOMENDACIONES	142
J. BIBLIOGRAFÍA	143
K. ANEXOS	145
ANEXO A. Análisis de Resultados	145
ANEXO B. Anteproyecto	152
ANEXO C. Certificado de Aprobación	207
ANEXO D. DIAGRAMA DE ROBUZTEZ	208

ÍNDICE DE FIGURAS

ILUSTRACIÓN 1. Funcionamiento o estructura general de un compilador.	12
ILUSTRACIÓN 2. Interacción de un analizador léxico con el analizador sintáctico. ...	13
ILUSTRACIÓN 3. Esquema por etapas definitivo del traductor.....	14
ILUSTRACIÓN 4. Posición del analizador sintáctico en el modelo del compilador.....	15
ILUSTRACIÓN 5. Pasos para encontrar el archivo Jflex.jar.....	18
ILUSTRACIÓN 6. Pasos para integrar los archivos en una carpeta del proyecto.	18
ILUSTRACIÓN 7. Consola de netbeans con el archivo lexer.java.	19
ILUSTRACIÓN 8. Consola de netbeans con los archivo analizador.java y sym.java. .	20
ILUSTRACIÓN 9. Obtención de un programa portable en java a partir de una especificación JFLEX.	21
ILUSTRACIÓN 10. Interfaz de una aplicación desarrollada con JGRAPHX.....	27
11. ILUSTRACIÓN: PsGram amigable	70
12. ILUSTRACIÓN: Conveniente el diseño de PsGram.....	70
13. ILUSTRACIÓN: Informacion de PsGram es suficiente.....	70
14. ILUSTRACIÓN: Almacenamiento de informacion en PsGram es rápido y seguro.	71
15. ILUSTRACIÓN: Inconveientes al utilizar PsGram.	71
16. ILUSTRACIÓN: Razonable el tiempo de ejecución.....	71
17. ILUSTRACIÓN: Agil el proceso de corrida de Pseudo-Código y detección de errores.	72
18. ILUSTRACIÓN: Agil el proceso de corrida dedigramas de flujo.	72
19. ILUSTRACIÓN: Util para practicar y mejorar conociemitos adquiridos.....	72
20. ILUSTRACIÓN: Recomendaciones para PsGram.....	73
21. ILUSTRACIÓN: Diagrama de flujo favorece comprensión.	147
22. ILUSTRACIÓN: Diagrama de flujo excelente herramienta para capacitar.	148
23. ILUSTRACIÓN: Software con la opción modificar.....	148
24. ILUSTRACIÓN: Software en español.....	149
25. ILUSTRACIÓN: Opción guardar en el software.	150
26 ILUSTRACIÓN: Interfaz del software amigable.....	150
27. ILUSTRACIÓN: Software detecte errores antes de realizar una conversión.	151
28. ILUSTRACIÓN: Software que tenga opción deshacer errores.	151

ÍNDICE DE TABLAS.

TABLA 1 .Descripción de token, numero token, lexemas, expresiones regulares y separadores.....	31
TABLA 2 . Archivo lex.txt, donde se indica los token de PsGram.....	35
TABLA 3 . Código de lex1.txt del programa PsGram	38
TABLA 4 . Parte de código del archivo jcup.txt, perteneciente a PsGram	50
TABLA 5 . Clase semantico.java.....	54
TABLA 6 .Clase styles.java.....	56
TABLA 7. Clase dibujaraction de PsGram	59
TABLA 8 . Verificaraction.java de PsGram.....	67
TABLA 9 . Modelo de encuesta para pruebas de PsGram.....	69
TABLA 10 . Informe de resultados de pruebas de validacion.....	75
TABLA 11 . Valoración técnica económica ambiental.	77
TABLA 12 . Resumen valoración técnica económica ambiental.....	77
TABLA 13 . Requerimientos Funcionales.....	80
TABLA 14 . Requerimientos no Funcionales.....	80
TABLA 15 . Determinación de Casos de Uso	83
TABLA 16 . Prototipo de pantalla del Caso de Uso: INGRESAR PSEUDO-CÓDIGO.....	88
TABLA 17 . Descripción del Caso de Uso: INGRESAR PSEUDO-CÓDIGO.....	93
TABLA 18 . Prototipo de pantalla del Caso de Uso: ANALIZAR LEXICAMENTE.....	95
TABLA 19 . Descripción del Caso de Uso: ANALIZAR LÉXICAMENTE.....	97
TABLA 20 . Prototipo de pantalla del Caso de Uso: ANALIZADOR SINTACTICO.....	98
TABLA 21 . Descripción del Caso de Uso: ANALIZADOR SINTACTICO.....	99
TABLA 22 . Prototipo de pantalla del Caso de Uso: ANALIZADOR SEMANTICO. ...	102
TABLA 23 . Descripción del Caso de Uso: ANALIZADOR SEMANTICO.	104
TABLA 24 . Prototipo de pantalla del Caso de Uso: GESTIONAR ERRORES.....	106
TABLA 25 . Descripción del Caso de Uso: GESTIONAR ERRORES.....	108
TABLA 26 . Prototipo de pantalla del Caso De Uso: CREAR DIAGRAMA.	110
TABLA 27 . Descripción del Caso de Uso: CREAR DIAGRAMA.....	111
TABLA 28 . Prototipo de pantalla del Caso de Uso: DIBUJAR DIAGRAMA.	115
TABLA 29 . Descripción del Caso de Uso: DIBUJAR DIAGRAMA.....	120
TABLA 30 . Prototipo de pantalla del Caso de Uso: REVISAR ERRORES.....	122
TABLA 31. Descripción del Caso de Uso: REVISAR ERRORES.....	124
TABLA 32 . Prototipo de pantalla del Caso de Uso: GENERAR PSEUDO-CODIGO.....	125
TABLA 33 . Descripción del Caso de Uso: GENERAR PSEUDO-CODIGO.....	126
TABLA 34 . Modelo de encuesta para obtener requerimientos.	147
TABLA 35 . Diagrama de flujo favorece comprensión.....	147
TABLA 36 . Diagrama de flujo excelente herramienta para capacitar.	148
TABLA 37 . Software con la opción modificar.	148
TABLA 38 . Software en español.	149
TABLA 39 . Opción guardar en el software.....	149
TABLA 40 . Interfaz del software amigable.	150
TABLA 41 . Software detecte errores antes de realizar una conversión.....	150
TABLA 42 . Software que tenga opción deshacer errores.....	151

ÍNDICE DE DIAGRAMAS.

DIAGRAMA 1. Modelo Inicial del Dominio	82
DIAGRAMA 2. Diagrama de Casos de Uso	84
DIAGRAMA 3. Diagrama de secuencia del Caso de Uso: Ingresar Pseudo-Código ...	94
DIAGRAMA 4. Diagrama de secuencia del Caso de Uso: Analizar Lexicamente.....	97
DIAGRAMA 5. Diagrama de secuencia del Caso de Uso: Analizador Sintactico.....	100
DIAGRAMA 6. Diagrama de secuencia del Caso de Uso: Analizador Semantico	105
DIAGRAMA 7 . Diagrama de secuencia del Caso de Uso: Gestionar Errores.....	109
DIAGRAMA 8 . Diagrama de secuencia del Caso de Uso: Crear Diagrama	112
DIAGRAMA 9. Diagrama de secuencia del Caso de Uso: Dibujar Diagrama	121
DIAGRAMA 10. Diagrama de secuencia del Caso de Uso: Revisar Errores	124
DIAGRAMA 11. Diagrama de secuencia del Caso de Uso: Generar Pseudo-Código	127
DIAGRAMA 12. Actualización del Modelo del Dominio	128
DIAGRAMA 13. Digrama de clases del paquete edu.psg.editor.dominio	129
DIAGRAMA 14. Digrama de clases del paquete edu.psg.editor.negocio	131
DIAGRAMA 15. Digrama de clases del paquete edu.psg.editor.vista	132
DIAGRAMA 16. Digrama de clases del paquete edu.psg.graficador.dominio	134
DIAGRAMA 17. Digrama de clases del paquete edu.psg.graficador.negocio	136
DIAGRAMA 18. Digrama de clases del paquete edu.psg.graficador.vista	138
DIAGRAMA 19. Diagrama de Paquetes.	139
DIAGRAMA 20. Modelo de Arquitectura.	139
DIAGRAMA 21. Diagrama de Componentes	140
DIAGRAMA 22. Diagrama de robustez del Caso de Uso: Ingresar Pseudo-Código .	209
DIAGRAMA 23. Diagrama de robustez del Caso de Uso: Analizar Lexicamente	210
DIAGRAMA 24. Diagrama de robustez del Caso de Uso: Analizador Sintactico	211
DIAGRAMA 25. Diagrama de robustez del Caso de Uso: Analizador Semantico	212
DIAGRAMA 26. Diagrama de robustez del Caso de Uso: Gestionar Errores	213
DIAGRAMA 27. Diagrama de robustez del Caso de Uso: Crear Diagrama.....	214
DIAGRAMA 28. Diagrama de robustez del Caso de Uso: Dibujar Diagrama	215
DIAGRAMA 29. Diagrama de robustez del Caso de Uso: Revisar Errores	216
DIAGRAMA 30. Diagrama de robustez del Caso de Uso: Generar Pseudo-Código..	217

C. INTRODUCCIÓN

La Universidad Nacional de Loja, como un ente de educación superior y en pos del desarrollo de la sociedad, cuyo objetivo es formar profesionales íntegros, en capacidad de utilizar lo aprendido en el proceso de formación académica y mediante la investigación, puedan generar nuevos conocimientos científicos, de esta manera aportan al desarrollo social, económico y técnico.

El Área de la Energía, las Industrias y los Recursos Naturales no Renovables, se enfoca desde el punto de vista técnico para implementar la investigación de campo e introducir a la comunidad profesionales que resuelvan dificultades con métodos adecuados y acordes a las necesidades de la sociedad actual.

La Carrera de Ingeniería en Sistemas, origina soluciones a los problemas de la vida diaria, enfocándose en algo diferente y ayudándose con los diversos avances tecnológicos que contamos en estos días.

En la actualidad los compiladores son una herramienta indispensable para el desarrollo de software; permitiendo facilitar las tareas de programación, ayudando a detectar de forma rápida los errores que se presentan. Cabe destacar que los lenguajes de programación son difíciles de comprender por lo que se evidencia la necesidad de contar con un lenguaje natural fácil de manejar.

Las herramientas graficas al elaborar diagramas de flujo ayudan a las personas a evitar procesos manuales muy largos y si se comete un error volverlos a empezar desde cero; permitiendo ser guardados y así poder editarlos al existir un error.

Con lo anteriormente mencionado, se planteó como objetivo principal **“Desarrollar una herramienta que permita graficar Diagramas de Flujo generando su respectivo Pseudocódigo y que a partir de un Pseudocódigo genere su Diagrama de Flujo, para los alumnos de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja”**, su cumplimiento conducirá a demostrar que el uso de esta herramienta mejorará y agilizará el proceso de enseñanza-aprendizaje de los estudiantes.

La Fundamentación Teórica recopila toda la información relacionada con respecto al proyecto y las herramientas utilizadas para la construcción del mismo.

En la obtención de los resultados esperados es conveniente el uso del método deductivo e inductivo y la técnica de entrevista estructurada, que permiten facilitar el desarrollo del trabajo, ayudando a detectar los problemas y sus causas, sirviendo como base para desarrollar el proyecto y proveer las respectivas soluciones, así mismo para obtener un software de calidad es importante utilizar la metodología ICONIX que es la más adaptable a las necesidades del proyecto, debido a que es iterativo e incremental, esto significa que durante este proceso se encontrara nuevas entidades y relaciones, que permita tomar en cuenta al iniciar la investigación actualizando cada vez el modelo del dominio o espacio del problema hasta que este quede completo.

En los resultados se explica el desarrollo de la propuesta alternativa, la forma en que se llevó a cabo el cumplimiento de los objetivos planteados.

En la valoración técnico-económica-ambiental se presenta los recursos (humanos, materiales, técnicos y tecnológicos) utilizados para la consecución del presente trabajo.

La discusión detalla las actividades realizadas en cada una de las etapas de la metodología ICONIX.

Las conclusiones son la parte final de los procesos que se relacionan entre sí, cumpliendo los objetivos trazados y los resultados alcanzados.

Las recomendaciones dan sugerencias a un futuro perfeccionamiento de la aplicación.

Es evidente que esto desafía a los profesionales a adquirir nuevos conocimientos y proceder con nuevas estrategias que logren un cambio significativo y a la vez importante, contribuyendo con el desarrollo de nuevas herramientas que permitan a los estudiantes su iniciación en este campo.

D. REVISIÓN DE LITERATURA O MARCO TEÓRICO

CAPÍTULO I

1. COMPILADORES

“Un compilador es un programa que lee un programa escrito en un lenguaje, el lenguaje fuente, y lo traduce a un programa equivalente en otro lenguaje, el lenguaje objeto. Como parte importante de éste proceso de traducción, el compilador informa a su usuario de la presencia de errores en el programa fuente”.¹

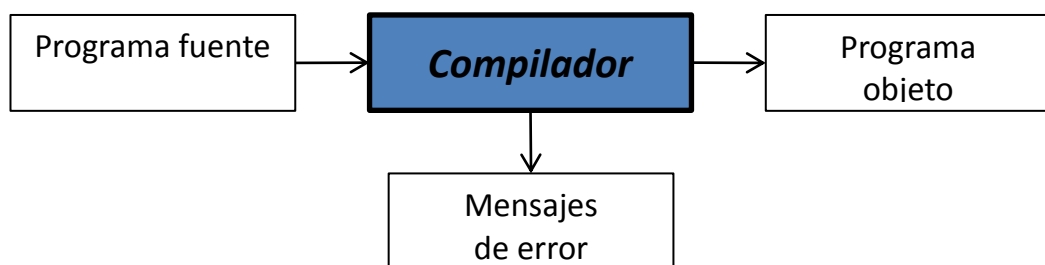


Ilustración 1. Funcionamiento o estructura general de un compilador.

Un compilador se estructura de la siguiente manera:

1. Análisis léxico.
2. Análisis sintáctico.
3. Análisis semántico.
4. Generación de código intermedio.
5. Optimización de código intermedio.
6. Generación de código objeto.

Con cada una de estas fases interactúa la Tabla de símbolos y la Gestión de errores.

¹BONILLA, Oscar. [en línea] Compiladores, Universidad Galileo, [http://74.125.45.104/search?q=cache:A9YLY0RcmuUJ:oscarbonilla.com/courses/compilers/materials/06_Analisis_Sintactico.ppt+analizador+sint%C3%A1ctico&hl=es&ct=clnk&cd=11].

1.1. ANALIZADOR LÉXICO

El analizador léxico o scanner, es la primera fase de un compilador, “lee un texto fuente y lo transforma en una secuencia ordenada de elementos léxicamente válidos. Un carácter o conjunto de estos que constituya un componente léxico se llama lexema (token). Como componentes léxicos consideramos: palabras reservadas, separadores, operadores, identificadores, constantes y signos de puntuación”².

- **Token:** Nombre que se da a cada componente léxico.
- **Lexema:** o valor léxico es la secuencia de caracteres de la entrada que corresponden a un token.
- **Patrón:** Forma compacta de describir conjuntos de lexemas.

FUNCIONES

Su función principal es leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis. Esta interacción suele aplicarse convirtiendo al analizador léxico en una subrutina del analizador sintáctico. El analizador sintáctico da la orden “Dame el siguiente componente léxico” al analizador léxico, y éste lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico, el cual devuelve al sintáctico según el formato convenido.

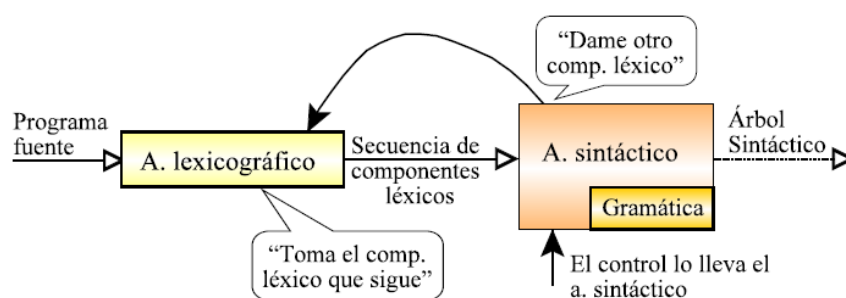


Ilustración 2. Interacción de un analizador léxico con el analizador sintáctico. ³

² TREJO AVILA, Mary Carmen. 2 de septiembre de 2004, RELIPMOC: Construcción de un Compilador Básico haciendo uso de las herramientas JLex y CUP, Edición Electrónica

³ AHO, Alfred y ULLMAN, Jeffrey. 2005, Compiladores Principios, Técnicas y Herramientas, Edición Electrónica

Otras funciones de un analizador léxico son:

- Manejar el archivo fuente (abrirlo, leerlo, cerrarlo).
- Generar y entregar tokens bajo la petición del analizador sintáctico.
- Rechazar un carácter o conjunto de estos que no concuerden con patrones especificados.
- Entendamos como patrón una expresión regular que se define en el lenguaje.
- Ignorar comentarios, espacios en blanco y tabuladores.
- Reconocer las palabras reservadas del lenguaje.
- Gestionar errores, contando los saltos de línea y asociando los mensajes de error con el número de la línea del archivo fuente donde se producen.
- Guardar tokens junto con su atributo en una tabla de símbolos. Este atributo es información adicional relevante, habitualmente con relación a los identificadores.

TABLA DE SÍMBOLOS

Es una estructura de datos que posee información sobre los identificadores definidos por el usuario, que pueden ser: constantes, variables, tipos y otros. Dado que puede contener información de diversa índole, no se debe guardar la misma información sobre una variable del programa que sobre un tipo definido por el usuario. Hace funciones de diccionario de datos y su estructura puede ser: una tabla hash, un árbol binario de búsqueda, para que las operaciones de acceso sean lo bastante eficientes.

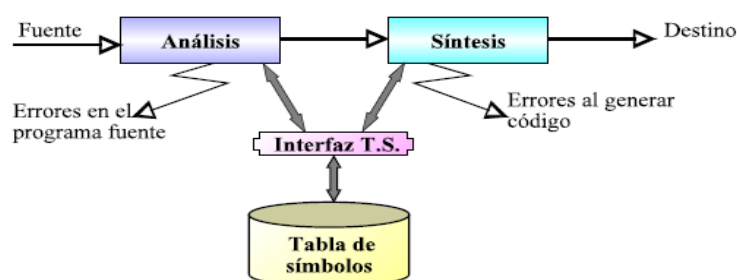


Ilustración 3. Esquema por etapas definitivo del traductor.

Tanto la etapa de análisis como la de síntesis acceden a esta estructura, por lo que se halla muy acoplada al resto de fases del compilador. Por ello conviene dotar a la tabla de símbolos de una interfaz lo suficientemente genérica como para permitir el cambio de las estructuras internas de almacenamiento sin que estas fases deban ser retocadas. El esquema general definitivo de un traductor se detalla en la Figura 3.

1.2. ANALIZADOR SINTÁCTICO

“Es la fase del analizador que se encarga de chequear la secuencia de tokens que representa al texto de entrada, en base a una gramática dada. En caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce en base a una representación computacional.”⁴.

Las gramáticas formales ofrecen ventajas significativas a los diseñadores de lenguajes y a los desarrolladores de compiladores:

- Las gramáticas son especificaciones sintácticas y precisas de lenguajes de programación.
- A partir de una gramática se puede generar automáticamente un analizador sintáctico.
- El proceso de generación automática anterior puede llevar a descubrir ambigüedades.
- Una gramática proporciona una estructura a un lenguaje de programación, siendo más fácil generar código y detectar errores.
- Es más fácil ampliar/modificar el lenguaje si está descrito con una gramática.

EL PAPEL DE ANALIZADOR SINTÁCTICO

El analizador obtiene una cadena de componentes léxicos del analizador léxico, como se presenta en la figura 4, y comprueba si la cadena puede ser generada por la gramática del lenguaje fuente. El analizador sintáctico informara de cualquier error de sintaxis de manera inteligente. También debería recuperarse de los errores que ocurren frecuentemente para poder continuar procesando el resto de su entrada.

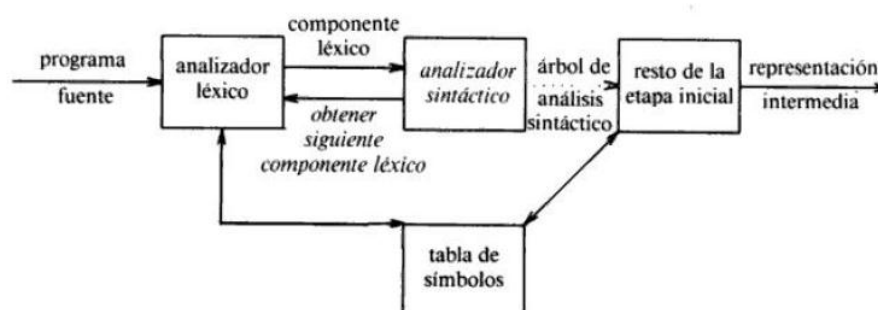


Ilustración 4. Posición del Analizador sintáctico en el modelo del compilador.

⁴GÁLVEZ ROJAS, Sergio Y MORA MATA, Miguel Ángel. .2005, Java a Tope: Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC, Edición Electrónica

MANEJO DE ERRORES SINTÁCTICOS

Los errores en la programación pueden ser de los siguientes tipos:

- Léxicos, producidos al escribir mal un identificador, una palabra clave o un operador.
- Sintácticos, por una expresión aritmética o paréntesis no equilibrados.
- Semánticos, como un operador aplicado a un operando incompatible.
- Lógicos, puede ser una llamada infinitamente recursiva.
- De corrección, cuando el programa no hace lo que el programador realmente deseaba.

GRAMÁTICAS INDEPENDIENTES DEL CONTEXTO

Consta de terminales, no terminales, un símbolo inicial y producciones.

- Los terminales son los símbolos básicos con que se forman las cadenas.
- Los no terminales:
 - Son variables sintácticas que denotan conjuntos de cadenas.
 - Definen conjuntos de cadenas que ayudan a definir el lenguaje generado por la gramática.
- En una gramática, un no terminal es considerado como el símbolo inicial, y el conjunto de cadenas que representan es el lenguaje definido por la gramática.
- Las producciones de una gramática especifican cómo se pueden combinar los terminales y los no terminales para formar cadenas. Cada producción consta de un terminal, seguido por algún símbolo y seguida por una cadena de no terminales y terminales.

1.3. ANALIZADOR SEMÁNTICO

El análisis semántico dota de un significado coherente a lo que se hace en el análisis sintáctico. El chequeo semántico se encarga de que los tipos que intervienen en las expresiones sean compatibles o que los parámetros reales de una función sean coherentes con los parámetros formales: por ejemplo: no suele tener mucho sentido el multiplicar una cadena de caracteres por un entero. Además de controlar que un programa cumple con las reglas de la gramática del lenguaje, hay que comprobar que lo que se quiere hacer tiene sentido.

Esta fase revisa el árbol sintáctico junto con los atributos y la tabla de símbolos para tratar de encontrar errores semánticos. Para todo esto se analizan los operadores y operandos de expresiones y proposiciones. Finalmente reúne la información necesaria sobre los tipos de datos para la fase posterior de generación de código. El componente más importante del análisis semántico es la verificación de tipos. Aquí, el compilador verifica si los operandos de cada operador son compatibles según la especificación del lenguaje fuente.

1.4. GENERACIÓN DE CÓDIGO INTERMEDIO

El generador de código intermedio transforma la salida del análisis semántico, en una representación cercana a un lenguaje intermedio cercano al código objeto. Esta representación intermedia debe tener dos propiedades importantes; debe ser fácil de producir y fácil de traducir al programa objeto.

1.5. OPTIMIZACIÓN DE CÓDIGO INTERMEDIO

En esta fase se realiza medicaciones sobre el código intermedio, de modo que en la siguiente fase resulte un código de máquina más rápido de ejecutar.

1.6. GENERACIÓN DE CÓDIGO OBJETO

La generación de código objeto es la fase final de un compilador, que consiste en transformar el código intermedio optimizado en código objeto de bajo nivel, que puede ser ensamblador o código máquina.

Cada una de las variables usadas por el programa se traduce a una dirección de memoria (esto también se ha podido hacer en la fase de generación de código intermedio). Después, cada una de las instrucciones intermedias se traduce a una secuencia de instrucciones de máquina que ejecuta la misma tarea. Un aspecto decisivo es la asignación de variables a registros.

CAPÍTULO II

2. JAVA-CUP, JFLEX, JGRAPH

2.1. INSTALACIÓN DE JAVA-CUP Y JFLEX

A continuación se explicará, para estudiantes que quieren trabajar por primera vez, con las herramientas JFlex y Cup en NetBeans, cómo realizar su instalación.

En primer lugar debemos tener ya en nuestro ordenador instalado Java, NetBeans y descargado los archivos de JFlex y Java-Cup; seguidamente se procede a descomprimir el archivo de jflex-1.4.3.zip.

Entramos en la carpeta de jflex-1.4.3 y buscamos la carpeta lib dentro de la cual estará un archivo llamado JFlex.jar

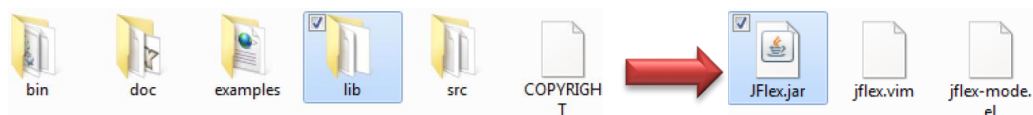


Ilustración 5. Pasos para encontrar el archivo JFlex.jar

Para la integración con el NetBeans solo ocupamos el archivo de JFlex.jar y el java-cup-11a.jar al cual le podemos cambiar el nombre a Cup.jar.

Para integrarlo en un proyecto, es recomendable copiar los archivos en una carpeta “X”, en este caso Tools dentro del proyecto.

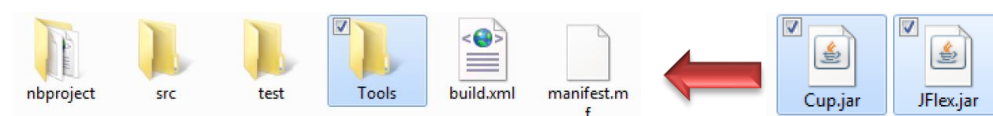


Ilustración 6. Pasos para integrar los archivos en una carpeta del proyecto.

Después nos pasamos a NetBeans y en el proyecto le damos click secundario y seleccionamos propiedades, luego seleccionamos donde dice librerías y cliqueamos en la opción de “Agregar jar/folder”. Buscamos la ubicación de la carpeta Tools y seleccionamos los dos archivos jar.

Con esto ya tenemos agregados las herramientas al NetBeans. Para poder usarlas solo ocupamos el siguiente código:

Codigo: MLexer

```
package Temporal;

import java.io.File; //Como usamos new File() ocupamos importar esta librería

public class MLexer{

public static void main(String[] args){

JFlex.Main.generate(new File("src\\Temporal\\archivo.flex"));

}} //Donde archvio.flex es su archivo .flex
```

Lo ejecutan y les generara un archivo llamado Lexer.java

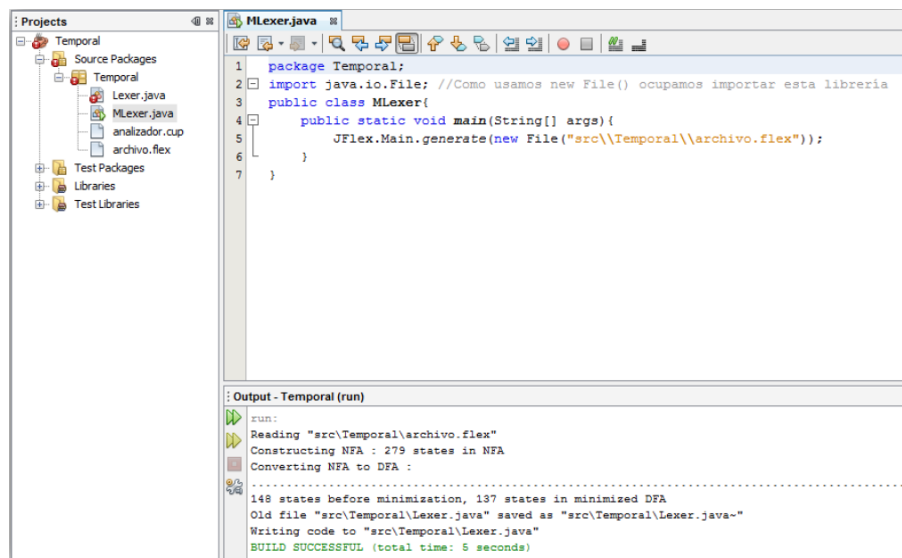


Ilustración 7. Consola de NetBeans con el archivo Lexer.java.

El cup, para este ejemplo solo utilizaremos las opciones de destino y de nombre.

Codigo: MCup

```
package Temporal;

public class MCup{

public static void main(String[] args){

String opciones[] = new String[5];

opciones[0] = "-destdir"; //Seleccionamos la opción de dirección de destino
```

```

opciones[1] = "src\\Temporal\\";           //Le damos la dirección
//Seleccionamos la opción de nombre de archivo

opciones[2] = "-parser";

//Le damos el nombre que queremos que tenga

opciones[3] = "Analizador";

//Le decimos donde se encuentra el archivo .cup

opciones[4] = "src\\Temporal\\analizador.cup";

try {
java_cup.Main.main(opciones);
} catch (Exception e) {
System.out.print(e);
}}

```

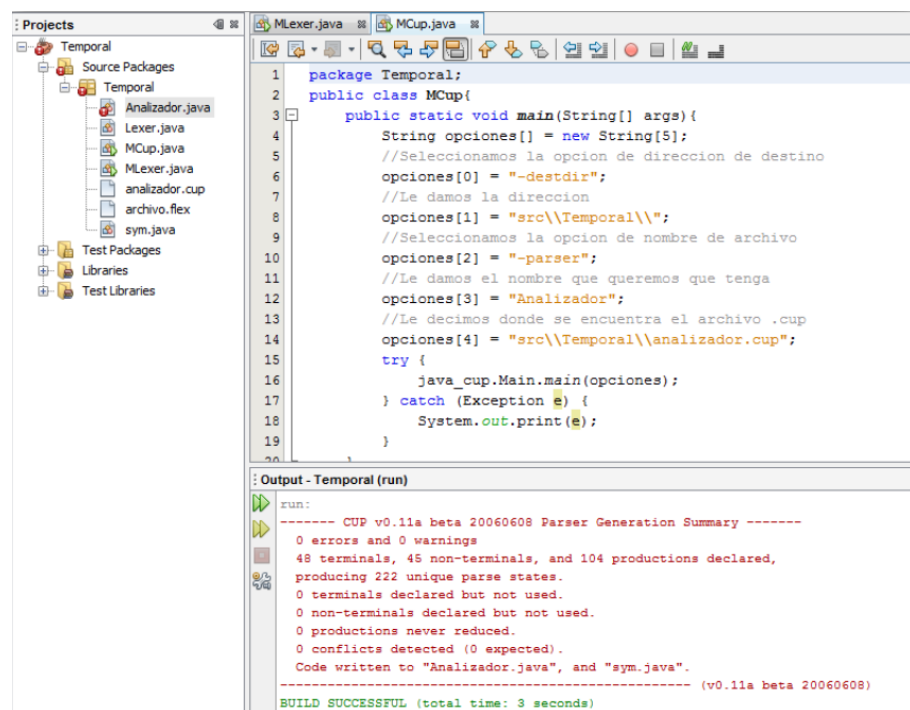


Ilustración 8. Consola de NetBeans con los archivos Analizador.java y sym.java.

Lo cual nos generará 2 archivos .java, uno es el Analizador.java o el nombre que hayan elegido y el otro es sym.java.

2.1. JFLEX

JFlex es desarrollado por Gerwin Klein y Régis Décamps, es un generador de analizadores lexicográficos, reimplementado de la herramienta JLex desarrollada en la Universidad de Princeton. JFlex está desarrollado en Java y genera código Java, tiene mayor compatibilidad con algunos parsers como CUP; además permite actuar sobre aquellas cadenas de un fichero de texto que encajan en una expresión regular.

La clase Main del paquete JFlex es la que se encarga de metacompilar (herramienta que a partir de la especificación de un lenguaje construye un programa o analizador que es capaz de reconocer sentencias de dicho lenguaje) nuestro programa .jflex de entrada; de esta manera, una invocación típica es de la forma: `java JFlex.Main fichero.jflex` lo que generará un fichero `Yylex.java` que implementa al analizador lexicográfico. JFlex genera los archivos Léxico y Scanner.



Ilustración 9. Obtención de un programa portable en java a partir de una especificación JFlex.

La especificación de JFLEX se divide por %% en tres partes y tiene el siguiente formato:

Código de usuario

%%

Opciones y Declaraciones (Directivas JFlex)

%%

Zonas de reglas Léxicas (Expresiones regulares)

🌱 Área de código de usuario, importaciones y paquete: Esta destinada a la importación de los paquetes que se vayan a utilizar en las acciones regulares situadas al lado de cada patrón en la zona de reglas. También sirve para escribir clases completas e interfaces.

☀ **Área de opciones y declaraciones:** “Esta área permite indicar a JFlex una serie de opciones para adaptar el fichero .java resultante de la meta-compilación y que será el que implemente nuestro analizador lexicográfico en Java. También permite asociar un identificador de usuario a los patrones más utilizados en el área de reglas.”⁵.

a) Opciones: Todas ellas empiezan por el carácter % y no pueden estar precedidas por nada en la línea en que aparecen. Las opciones más interesantes se pueden clasificar en opciones de:


- Opciones de clase.
- Opciones de la función de análisis: Estas opciones permiten modificar el método o función encargada de realizar el análisis lexicográfico en sí.
- Opciones de fin de fichero: Por defecto, cuando yylex() se encuentra el carácter fin de fichero, retorna un valor null. En caso de que se haya especificado %int en las opciones anteriores, se retornará el valor YYEOF definido como public static final en la clase Yylex.
- Opciones de juego de caracteres: Estas opciones permiten especificar el juego de caracteres en el que estará codificada la entrada al analizador lexicográfico generado por JFlex.
- Opciones de contadores: Estas opciones hacen que el analizador lexicográfico almacene en contadores el número de caracteres, líneas o columnas en la que comienza el lexema actual.

b) Declaraciones: Además de opciones, el programador puede indicar declaraciones de dos tipos en el área que nos ocupa, a saber, declaraciones de estados léxicos y declaraciones de reglas.

- Declaraciones de estados léxicos: Se declaran mediante la opción: %state estado1, estado2, etc. Si existen muchos patrones en el área de reglas que comparten el mismo estado léxico, es posible indicar éste una sola vez y agrupar a continuación todas estas reglas entre llaves. El estado inicial viene dado por la constante YYINITIAL.
- Declaraciones de reglas: En caso de que un patrón se utilice repetidas veces o cuando su complejidad es elevada, es posible asignarle un nombre y utilizarlo posteriormente en cualquier otra regla encerrándolo entre llaves, su sintaxis es:

<nombre macro>= <expresión regular valida>

⁵ GÁLVEZ ROJAS, Sergio Y MORA MATA, Miguel Ángel. .2005, Java a Tope: Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC, Edición Electrónica
 Universidad Nacional de Loja

 **Zonas de reglas Léxicas:** En esta sección se definen las expresiones regulares que representan los tokens de nuestro lenguaje y que debemos hacer una vez detectados dichos tokens. Las reglas tienen 3 partes:

$$[\text{estado(s)}] \text{ <expresión> } \{ \text{acción} \}$$

En el área de reglas es posible agrupar las reglas a aplicar en un mismo estado léxico. Como ejemplo, las reglas:

```
<YYINITIAL>"=" { System.out.println("Encontrado ="); }
```

```
<YYINITIAL>"!=" { System.out.println("Encontrado !="); }
```

también pueden escribirse como:


```
<YYINITIAL>{
  "=" { System.out.println("Encontrado ="); }
  "!=" { System.out.println("Encontrado !="); }
}
```


2.3. JAVA-CUP





El segundo paso para construir un compilador es desarrollar el analizador sintáctico de nuestro lenguaje. CUP es una herramienta desarrollada en Java en el Instituto de Tecnología de Georgia (EE.UU.) para crear analizadores sintácticos y semánticos.


En Java genera las clases `sym` (está constituida por los símbolos terminales declarados en la gramática, la cual es utilizada para hacer referencia a los mismos) y `parser` (`parser` contiene al analizador sintáctico), a partir de una especificación en la que se indica una gramática formal y también se asocian una serie de acciones a los símbolos aparecidos en cada regla gramatical.

En un archivo de entrada para CUP se pueden diferenciar cinco secciones:

 **Especificaciones de “package” e “imports”:** Indica que las clases Java generadas a partir de este archivo pertenecen a un determinado paquete e importa las clases de Java necesarias.



Componentes de código de usuario: Esta sección se divide en 4 partes, donde se puede incluir código Java que el usuario desee en el analizador sintáctico que se va a obtener con CUP.


- 
 actioncode{: código :}, permite incluir código Java que es utilizado por las acciones especificadas en la gramática.
- 
 parsercode{: código :}, permite incluir código Java en la clase parser. Aquí se pueden redefinir los métodos que se invocan como consecuencia de errores de sintaxis.
- 
 initwith{: código :}, permite incluir código Java el cual será ejecutado por el traductor antes de que este pregunte por el primer token.
- 
 scanwith{: código :}, permite indicar cómo el traductor preguntará por el siguiente token al analizador léxico.


Lista de símbolos de la gramática (terminales y no terminales): Se declaran los símbolos terminales y no terminales de la gramática que define el analizador sintáctico que deseamos producir. Tanto los símbolos no terminales como los símbolos terminales pueden, opcionalmente, tener asociado un objeto Java de una cierta clase. La sintaxis es:

terminal [<nombre clase>] nombre01, nombre02, ..., nombreN y

non terminal [<nombre clase>] nombre01, nombre02, ..., nombreN


Declaraciones de precedencia: Es posible definir niveles de precedencia y la asociatividad de símbolos terminales. Las declaraciones de precedencia de un archivo CUP consisten en una secuencia de construcciones que comienzan con la palabra clave precedence. A continuación, viene la declaración de asociatividad, que puede tomar los valores left (el terminal se asocia por la izquierda), y right (el terminal se asocia por la derecha). Finalmente, la construcción termina con una lista de símbolos terminales separados por comas, seguido del símbolo ;.


Especificación de la gramática (definición del símbolo inicial de la gramática y las reglas de producción): startwith < no terminal > se utiliza para definir el símbolo inicial de la gramática. Las reglas de producción tienen esta sintaxis:

expresión ::= expresión <símbolo terminal>expresión {: código :};

Donde expresión son no terminales, y a la derecha está el código Java que se ejecutará al aplicarse esta producción.

2.3. JGRAPHX

JGraphX es la mejor biblioteca Java de fuente Abierta para la Visualización de Gráficos. Sigue los parámetros de diseño Swing para brindar un API familiar para los programadores Swing y una funcionalidad que brinda una gran cantidad de características. La visualización de gráficos es un requerimiento central para aplicaciones tales como editores de volúmenes de trabajo, monitoreo de redes de ordenadores y telecomunicaciones, diagramas de flujo, modelado de proceso de negocios, gráficos organizacionales, diagramas de relaciones de entidades, diagramas de causa-efecto y muchos más.

JGraphX está diseñado principalmente para su uso en un entorno de escritorio, aunque Java tiene características que permiten trabajar en web, por lo que es posible implementar JGraphX en entorno web.

Es bastante interesante JGraph ya que además se puede exportar a formatos como .SVG (Gráficos vectoriales), XML, y otros.

El término celda se usa para describir un elemento de un gráfico, ya sea bordes, vértices y aristas (las líneas de conexión entre los nodos).

INTERACCIÓN GRÁFICA

La interacción es la forma en que una aplicación que utiliza JGraphX puede alterar el modelo de gráfico a través de la interfaz gráfica de usuario de aplicaciones Web. JGraphX soporta arrastrar y clonación de las celdas, configuración del tamaño, conexión de las celdas y desconexión, arrastrar y soltar, editar las etiquetas de las celdas y mucho más. Una de las principales ventajas de JGraphX es la flexibilidad de cómo la interacción se puede programar.

INSTALACIÓN JGRAPHX

Tanto la evaluación como las versiones completas de JGraphX se entregan como archivos zip. Descomprimir el paquete a su lugar preferido, una carpeta con el nombre JGraphX se creará allí, esta carpeta es la carpeta raíz de la instalación JGraphX.



ESTRUCTURA DEL PROYECTO Y LAS OPCIONES DE GENERACIÓN

Una vez descomprimido se le presentará con una serie de archivos y directorios en la raíz de la instalación.

/ doc	Raíz de la documentación, incluye este manual de usuario
/ src	Fuente de la biblioteca
/ lib	Contiene el jar de la biblioteca.
/examples	Ejemplos que muestran el uso de JGraphX
license.txt	Los términos de la licencia bajo la cual usted debe usar la biblioteca

INSERTAR CELDAS

Usted puede agregar vértices y aristas utilizando el método `add ()` en el modelo. Sin embargo, para los fines de uso general de esta biblioteca, saber que `mxGraph.insertVertex ()` y `mxGraph.insertEdge ()` son el núcleo API para la adición de las celdas. El método del modelo requiere que la celdas que se añade ya se ha creado, mientras que el `mxGraph.insertVertex ()` crea la celda para usted.

CORE API MÉTODOS:

`mxGraph.insertVertex (padre, id, valor, x, y, ancho, alto, de estilo)`: crea e inserta un nuevo vértice en el modelo, dentro de una llamada de actualización `begin / end`.

`mxGraph.insertEdge (padre, id, valor, origen, destino, estilo)`: crea e inserta una nueva ventaja en el modelo, en un comienzo / fin de llamada de actualización.

`mxGraph.insertVertex ()`: creará un objeto `mxCell` y devolverlo a partir del método utilizado. Los parámetros del método son:

- **Los padres:** la celda que es el padre inmediato de la nueva celda en la estructura del grupo. Utilizar `graph.getDefaultParent ()`, como su padre por defecto.
- **Identificación:** este es un identificador único global que describe la celda, siempre es una cadena. Esto se debe principalmente para hacer referencia a las celdas en la salida persistente externamente. Si usted no desea mantener los identificadores, pase el valor `null` en este parámetro y asegurarse de que

mxGraphModel.isCreatelds () devuelve true. De esta manera el modelo se encargará de los identificadores y asegurarse de que son únicos.

- **valor:** este es el objeto de usuario de la celdas. Objeto de usuario son simplemente eso, sólo objetos, sino que forman los objetos que le permiten asociar a la lógica de negocio de una aplicación con la representación visual de JGraphX. Si usted usa una cadena como el objeto de usuario, este se mostrará como la etiqueta en el vértice o borde.
- **x, y, ancho, alto:** como el nombre sugiere, estas son las posiciones x e y de la esquina superior izquierda de la celda, su anchura y altura.
- **Estilo:** la descripción del estilo que se aplica a este vértice. Este parámetro es una cadena que sigue un formato particular. En la cadena aparece cero o más nombres de estilos y un cierto número de pares de clave / valor que reemplazar el estilo global o establecer un nuevo estilo.

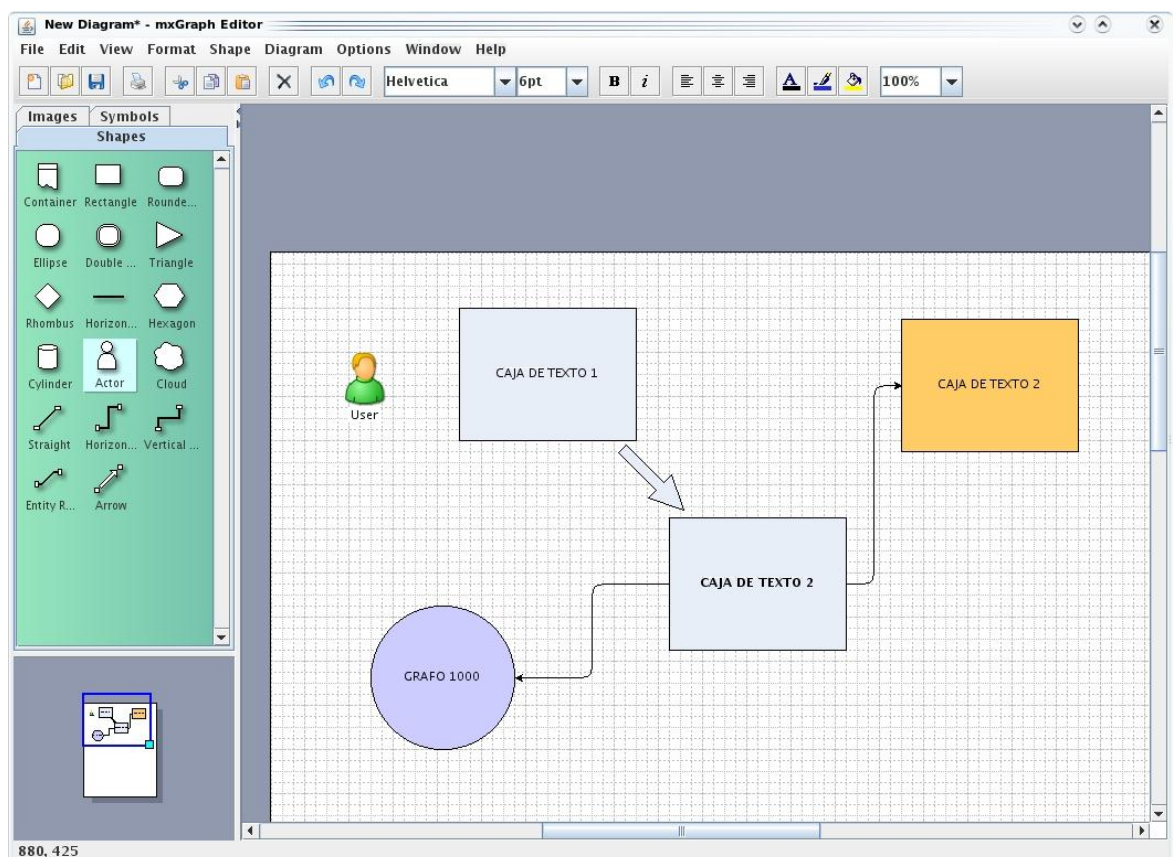


Ilustración 10. Interfaz de una aplicación desarrollada con JGraphX

E. METODOLOGÍA

Para la realización del presente proyecto se empleó diversos métodos y técnicas de investigación los cuales sirvieron dentro del desarrollo de la investigación. Entre los métodos utilizados tenemos los siguientes:

MÉTODOS

- **Método Inductivo.-** Se lo utilizó para poder identificar los inconvenientes que se presentan al no tener una herramienta que genere el Pseudocódigo a partir de un diagrama de Flujo y viceversa.
- **Método Deductivo.-** Sirvió para buscar alternativas de solución a las casusas determinadas a partir del problema.
- **Método Analítico.-** Se utilizó para realizar un minucioso análisis del objeto en estudio como son los problemas, causas y consecuencias que se están presentando en los alumnos que empiezan a recibir las bases para aprender a programar, sin una herramienta de apoyo que les ayude a su mejor entendimiento.
- **Método Sintético.-** Se lo consideró para realizar la construcción teórica de la investigación.

TÉCNICAS

- **La Encuesta.-** Esta técnica es muy importante para el análisis porque permite obtener la información en forma escrita, a través de preguntas a los alumnos que empiezan sus estudios de programación en la Carrera de Ingeniería en Sistemas, de ésta manera saber si es necesario que los estudiantes tengan conocimiento sobre éste tema y una herramienta de software para mayor comprensión y así en

el futuro poder programar. Esta técnica también es utilizada para las pruebas de validación del software.

✿ **Lectura Comprensiva.-** Consiste en obtener un conocimiento ordenado y sistemático de como e efectuará la implementación de la aplicación.

METODOLOGÍA

La metodología utilizada en el desarrollo de este proyecto es ICONIX, debido a que permite un desarrollo ágil, con muy buena documentación, es bastante flexible y se adapta mejor a la Programación Orientada a Objetos ya que emplea UML. ICONIX, permite inicializar la ejecución del presente proyecto, en la etapa de análisis con toda la información necesaria, la filosofía en que se basa este tipo de metodología es de ser iterativo e incremental, esto significa que durante este proceso se encontrará nuevas entidades y relaciones, que no se tomaron en cuenta al iniciar la investigación lo cual nos obliga a actualizar cada vez el modelo del dominio o espacio del problema hasta que este quede completo.

A continuación estas son las etapas de este método:

Análisis: en esta etapa se recolectó toda la información necesaria, analizándola y seleccionándola para la siguiente etapa.

Diseño: en esta etapa se desarrolló los prototipos de pantalla conforme a la información que se recolectó y a las necesidades de los alumnos.

Desarrollo: en esta etapa se puso como base los prototipos de pantalla que anteriormente se diseñó para poder codificar y darle la funcionalidad que se esperaba.

Pruebas: en esta etapa se terminó la aplicación y se realizó las pruebas suficientes para poder corregir errores. Al final se entregó una herramienta funcionando correctamente.

F. RESULTADOS

1. DESARROLLO DE LA PROPUESTA ALTERNATIVA

1.1. RECOLECCIÓN DE INFORMACIÓN

OBJETIVO 1: Analizar y recolectar información para el desarrollo de la herramienta.

Para cumplir con este objetivo se utilizó dos técnicas de estudio como son la encuesta y la lectura comprensiva. En la primera, que fue realizada en Marzo del 2010 a los estudiantes de quinto módulo y algunos docentes de la carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja (*ver ANEXO A*), se pudo saber si los estudiantes tienen la necesidad de una herramienta de software que les facilite la comprensión de los siguientes procedimientos: pasar un Pseudo-Código a Diagrama de Flujo y viceversa; del mismo modo si los docentes creían necesario la misma herramienta de software para facilitar la comprensión de sus estudiantes. A las encuestas se las analizó detalladamente dando como resultado la necesidad de un software que ayude a comprender los procedimientos antes mencionados y sus necesidades primordiales como posibles usuarios del sistema (*ver requerimientos del Sistema*).

A la técnica de lectura comprensiva, se la utilizó para sacar información (*ver Marco Teórico*) de: el procedimiento manual de la conversión de un Pseudo-Código a Diagrama de Flujo y la transformación de Diagrama de Flujo a Pseudo-Código; las herramientas y librerías más convenientes para el desarrollo del software PsGram. Como resultado se utilizó Java, Netbean y las librerías: JFlex, Java-Cup para el desarrollo del módulo de Pseudo-Código; y JGraphyX para el módulo de Diagrama de Flujo.

1.2. ANALIZADOR LÉXICO SINTÁCTICO Y SEMÁNTICO

OBJETIVO 2: Desarrollar un analizador léxico, sintáctico y semántico para el pseudocódigo.

1.2.1. ANALIZADOR LÉXICO

DESCRIPCIÓN DE TOKEN, NUMERO TOKEN, LEXEMAS, EXPRESIONES REGULARES Y SEPARADORES

A continuación se presenta una tabla en la que constan la descripción de un token, el índice o número único con el que se reconoce cada token, el lexema, la expresión regular (es decir la manera en cómo se debe escribir en el Editor de PsGram), los separadores que indica donde empieza y termina cada uno de los lexemas y expresiones regulares.

Tabla 1 .Descripción de Token, Numero Token, Lexemas, Expresiones Regulares y Separadores

DESCRIPCION TOKEN	N° TOKEN	LEXEMA	EXPRESION REGULAR	SEPARADORES
Tipo de dato entero	2	ENTERO	Entero	\n,\t,\r
Operador aritmético (Suma)	3	MAS	+	\n,\t,\r, ENTERO_S, DECIMAL_S
Operador de aritmético de suma (más más)	4	MASMAS	++	\n,\t,\r,)
Operador aritmético (menos unitario)	5	MENOS	-	\n,\t,\r, ENTERO_S, DECIMAL_S
Operador de aritmético de diferencia (menos menos)	6	MENOSMENOS	--	\n,\t,\r,)
Operador aritmético (producto)	7	POR	*	\n,\t,\r, ENTERO_S, DECIMAL_S
Operador de asignación (división)	8	DIVIDIR	/	\n,\t,\r, ENTERO_S, DECIMAL_S
Operador asignación	9	IGUAL	=	\n,\t,\r, ENTERO_S, DECIMAL_S, CADENA_S

Operador de igualdad	10	IGUALIGUAL	==	\n,\t,\r, ENTERO_S, DECIMAL_S, BOOL
Operador menor que	11	MENQ	<	\n,\t,\r, ENTERO_S, DECIMAL_S
Operador mayor que	12	MAYQ	>	\n,\t,\r, ENTERO_S, DECIMAL_S
Operador diferente	13	DIFERENTE	!=	\n,\t,\r, ENTERO_S, DECIMAL_S, BOOL
Operador Residuo	14	RES	%	\n,\t,\r, ENTERO_S, DECIMAL_S
Operador mayor igual que	15	MAYIGUAL	>=	\n,\t,\r, ENTERO_S, DECIMAL_S
Operador menor igual que	16	MENIGUAL	<=	\n,\t,\r, ENTERO_S, DECIMAL_S
Abrir Paréntesis	17	PARENTI	(\n,\t,\r, ENTERO_S, DECIMAL_S, BOOL , VARIABLE
Cerrar Paréntesis	18	PARENTF)	\n,\t,\r, ENTERO_S, DECIMAL_S, OPERADOR ARITMETICO, OPERADOR RELACIONAL
Punto y coma	19	PUNTOCOMA	;	\n,\t,\r
Condición Si	20	SI	Si	\n,\t,\r,(
Instrucción sino	21	SINO	Sino	\n,\t,\r
Fin Condición Si	22	FINSI	Finsi	\n,\t,\r
Bucle para	23	PARA	Para	\n,\t,\r,(
Fin bucle para	24	FINPARA	Finpara	\n,\t,\r
Bucle mientras	25	MIENTRAS	Mientras	\n,\t,\r,(
Fin bucle mientras	26	FINMIENTRAS	Finmientras	\n,\t,\r
Escribir en consola	27	ESCRIBIR	Escribir	\n,\t,\r
Tipo de dato	28	CADENA	Cadena	\n,\t,\r

cadena				
Tipo de dato booleano	29	AFIRMACION	Afirmación	\n,\t,\r
Inicio del Pseudo-Código	30	INICIO	Inicio	\n,\t,\r
Fin del Pseudo-Código	31	FIN	Fin	\n,\t,\r
Instrucción hacer	32	HACER	Hacer	\n,\t,\r
Instrucción entonces	33	ENTONCES	Entonces	\n,\t,\r
Tipo de dato real	34	DECIMAL	Decimal	\n,\t,\r
Operador ó lógico	35	OPO	Oo	\n,\t,\r, ENTERO_S, DECIMAL_S
Operador y lógico	36	OPY	Yy	\n,\t,\r, ENTERO_S, DECIMAL_S
Instrucción Swich	37	SWITCH	Paracaso	\n,\t,\r,(
Instrucción caso	38	CASO	Caso	\n,\t,\r
Instrucción parar	39	PARAR	Parar	\n,\t,\r
Instrucción por defecto	40	DEFAULT	Default	\n,\t,\r,:
Fin Instrucción Swich	41	FINSWITCH	Finparacaso	\n,\t,\r
Potencia	42	POTENCIA	Potencia	\n,\t,\r
Leer	43	LEER	leer	\n,\t,\r
Dos puntos	44	DOSPUNTOS	:	\n,\t,\r
Números enteros	45	ENTERO_S	numero [0-9]+	\n,\t,\r, VARIABLE, OPERADOR ARITMETICO, OPERADOR RELACIONAL
Booleano	46	BOOL	V F	\n,\t,\r,)
Cadenas	47	CADENA_S	“A-Z- a-z[[0-9]][_]*”	\n,\t,\r
Números reales	48	DECIMAL_S	[[0-9]+(.) [0-9]+]	\n,\t,\r, OPERADOR ARITMETICO, OPERADOR RELACIONAL
Variable	49	VARIABLE	A-Z- a-z[[0-9]][_]*	\n,\t,\r, IGUAL

DEFINICIÓN DE LOS TOKENS EN EL ARCHIVO lex.txt

Para desarrollar el analizador léxico en el caso de PsGram después de agregar la librería JFlex se procedió a crear lex.txt, en este archivo se encuentran los tokens de PsGram; e inmediatamente se empezó a crear el archivo GenLexer.java, el que hace la llamada al archivo lex.txt para generar la clase Scanner.java.

A continuación, se presenta una parte del código del archivo lex.txt, donde se indica los token de PsGram:

```

<YYINITIAL> {
/* keywords */
"si"           { return symbol(sym.SI); }
"fin"si"      { return symbol(sym.FINSI); }
"mientras"    { return symbol(sym.MIENTRAS); }
"finmientras" { return symbol(sym.FINMIENTRAS); }
"para"        { return symbol(sym.PARA); }
"finpara"     { return symbol(sym.FINPARA); }
"entero"      { return symbol(sym.ENTERO); }
"afirmacion"  { return symbol(sym.AFIRMACION); }
"sino"        { return symbol(sym.SINO); }
"cadena"      { return symbol(sym.CADENA); }
"escribir"    { return symbol(sym.ESCRIBIR); }
"inicio"      { return symbol(sym.INICIO); }
"fin"         { return symbol(sym.FIN); }
"hacer"       { return symbol(sym.HACER); }
"entonces"    { return symbol(sym.ENTONCES); }
"decimal"     { return symbol(sym.DECIMAL); }
"paracaso"    { return symbol(sym.SWITCH); }
"caso"        { return symbol(sym.CASO); }
"parar"       { return symbol(sym.PARAR); }
"default"     { return symbol(sym.DEFAULT); }
"finparacaso" { return symbol(sym.FINSWITCH); }
"pot"         { return symbol(sym.POTENCIA); }
"leer"        { return symbol(sym.LEER); }

/* boolean literals */
"v"           { return symbol(sym.BOOL, new Boolean(true)); }
"f"           { return symbol(sym.BOOL, new Boolean(false)); }

/* separators */
"("           { return symbol(sym.PARENTI); }
")"           { return symbol(sym.PARENTF); }
";"           { return symbol(sym.PUNTOCOMA); }
"."           { return symbol(sym.DOSPUNTOS); }

/* operators */
"="           { return symbol(sym.IGUAL); }
"=="         { return symbol(sym.IGUALIGUAL); }
">"          { return symbol(sym.MAYQ); }
"<"          { return symbol(sym.MENQ); }

```



```

"+"          { return symbol(sym.MAS); }
"++"        { return symbol(sym.MASMAS); }
"-"         { return symbol(sym.MENOS); }
"--"        { return symbol(sym.MENOSMENOS); }
"*"         { return symbol(sym.POR); }
"/"         { return symbol(sym.DIVIDIR); }
"!="        { return symbol(sym.DIFERENTE); }
"oo"        { return symbol(sym.OPO); }
"yy"        { return symbol(sym.OPY); }
"%"         { return symbol(sym.RES); }
">="        { return symbol(sym.MAYIGUAL); }
"<="        { return symbol(sym.MENIGUAL); }

/* string literal */
\"          { yybegin(sym.CADENA_S); string.setLength(0); }

/* numeric literals */
{DeclIntegerLiteral} { return symbol(sym.ENTERO_S, new Integer(yytext()));}
{DecimalLiteral}     { return symbol(sym.DECIMAL_S, new Double(yytext()));}

/* comments */
{Comment}           { /* ignore */ }

/* whitespace */
{WhiteSpace}        { /* ignore */ }

/* identifiers */
{Identifier}         { return symbol(sym.VARIABLE, yytext()); }
\"{STRING_TEXT}\"    { return symbol(sym.CADENA_S, yytext()); }
\"{STRING_TEXT}\"    { throw new RuntimeException("Cadena mal definida"); }
}
<STRING> {
  \"                { yybegin(YYINITIAL); return symbol(sym.CADENA_S,
string.toString()); }
  {StringCharacter}+ { string.append( yytext() ); }

  /* error cases */
  \\.                { throw new RuntimeException("Illegal escape sequence
\""+yytext()+"\""); }
  {LineTerminator}   { throw new RuntimeException("Unterminated string at end of
line"); }
}

/* error fallback */
.|\n                { throw new RuntimeException("Illegal character \""+yytext()+
\"\" at line "+yyline+", column "+yycolumn"); }

```

Tabla 2 . Archivo lex.txt, donde se indica los token de PsGram.

Después de generar la clase Scanner, se procede a crear el archivo lex1.txt, en donde se indica el índice donde empieza y termina cada token, su ancho y en qué línea se encuentra ubicado dentro del código ingresado en la interfaz editor de Psgram.


```

";" { return (new
    Ytoken(sym.PUNTOCOMA,yytext(),yyline,yychar,yychar+1,"OPERADOR")); }
":" { return (new
    Ytoken(sym.DOSPUNTOS,yytext(),yyline,yychar,yychar+1,"OPERADOR")); }
"si" { return (new Ytoken(sym.SI,yytext(),yyline,yychar,yychar+2,"RESERVADA")); }
"sino" { return (new
    Ytoken(sym.SINO,yytext(),yyline,yychar,yychar+4,"RESERVADA")); }
"finsi" { return (new
    Ytoken(sym.FINSI,yytext(),yyline,yychar,yychar+5,"RESERVADA")); }
"para" { return (new
    Ytoken(sym.PARA,yytext(),yyline,yychar,yychar+4,"RESERVADA")); }
"finpara" { return (new
    Ytoken(sym.FINPARA,yytext(),yyline,yychar,yychar+7,"RESERVADA")); }
"mientras" { return (new
    Ytoken(sym.MIENTRAS,yytext(),yyline,yychar,yychar+8,"RESERVADA")); }
"finmientras" { return (new
    Ytoken(sym.FINMIENTRAS,yytext(),yyline,yychar,yychar+11,"RESERVADA")); }
"escribir" { return (new
    Ytoken(sym.ESCRIBIR,yytext(),yyline,yychar,yychar+8,"RESERVADA")); }
"afirmacion" { return (new
    Ytoken(sym.AFIRMACION,yytext(),yyline,yychar,yychar+10,"RESERVADA")); }
"cadena" { return (new
    Ytoken(sym.CADENA,yytext(),yyline,yychar,yychar+6,"RESERVADA")); }
"entero" { return (new
    Ytoken(sym.ENTERO,yytext(),yyline,yychar,yychar+6,"RESERVADA")); }
"inicio" { return (new
    Ytoken(sym.INICIO,yytext(),yyline,yychar,yychar+6,"RESERVADA")); }
"fin" { return (new Ytoken(sym.FIN,yytext(),yyline,yychar,yychar+3,"RESERVADA")); }
"v" { return (new Ytoken(sym.BOOL,yytext(),yyline,yychar,yychar+1,"RESERVADA")); }
"f" { return (new Ytoken(sym.BOOL,yytext(),yyline,yychar,yychar+1,"RESERVADA")); }
"hacer" { return (new
    Ytoken(sym.HACER,yytext(),yyline,yychar,yychar+5,"RESERVADA")); }
"entonces" { return (new
    Ytoken(sym.ENTONCES,yytext(),yyline,yychar,yychar+8,"RESERVADA")); }
"decimal" { return (new
    Ytoken(sym.DECIMAL,yytext(),yyline,yychar,yychar+8,"RESERVADA")); }
"paracaso" { return (new
    Ytoken(sym.SWITCH,yytext(),yyline,yychar,yychar+8,"RESERVADA")); }
"caso" { return (new
    Ytoken(sym.CASO,yytext(),yyline,yychar,yychar+4,"RESERVADA")); }
"parar" { return (new
    Ytoken(sym.PARAR,yytext(),yyline,yychar,yychar+5,"RESERVADA")); }
"default" { return (new
    Ytoken(sym.DEFAULT,yytext(),yyline,yychar,yychar+7,"RESERVADA")); }
"finparacaso" { return (new
    Ytoken(sym.DEFAULT,yytext(),yyline,yychar,yychar+11,"RESERVADA")); }
"pot" { return (new
    Ytoken(sym.POTENCIA,yytext(),yyline,yychar,yychar+3,"RESERVADA")); }
"leer" { return (new
    Ytoken(sym.LEER,yytext(),yyline,yychar,yychar+4,"RESERVADA")); }
"!=" { return (new
    Ytoken(sym.DIFERENTE,yytext(),yyline,yychar,yychar+2,"OPERADOR")); }
"||" { return (new Ytoken(sym.OPO,yytext(),yyline,yychar,yychar+2,"OPERADOR")); }
"&&" { return (new Ytoken(sym.OPY,yytext(),yyline,yychar,yychar+2,"OPERADOR")); }
"%" { return (new Ytoken(sym.RES,yytext(),yyline,yychar,yychar+1,"OPERADOR")); }
">=" { return (new
    Ytoken(sym.MAYIGUAL,yytext(),yyline,yychar,yychar+2,"OPERADOR")); }

```

```

"<=" { return (new
    Ytoken(sym.MENIGUAL,yytext(),yyline,yychar,yychar+2,"OPERADOR")); }

{NONNEWLINE_WHITE_SPACE_CHAR}+ { }

"/*" { yybegin(COMMENT); comment_count++; }

\"{STRING_TEXT}\" {
    String str = yytext().substring(1,yylength()-1);
    return (new Ytoken(sym.CADENA_S,str,yyline,yychar,yychar+yylength(),"CADENA"));
}
\"{STRING_TEXT} {
    String str = yytext().substring(1,yytext().length());
    //javax.swing.JOptionPane.showMessageDialog(null,"Error Lexico");
    return (new Ytoken(sym.error,str,yyline,yychar,yychar + str.length(),"CADENA"));
}

{DIGITO}+ { return (new
Ytoken(sym.ENTERO_S,yytext(),yyline,yychar,yychar+yylength(),"NUMERO")); }
{DECIM}+ { return (new
Ytoken(sym.DECIMAL_S,yytext(),yyline,yychar,yychar+yylength(),"NUMERO")); }

{Ident} { return (new
Ytoken(sym.VARIABLE,yytext(),yyline,yychar,yychar+yylength(),"VARIABLE")); }

}
<COMMENT> {
    "/" { comment_count++; }
    "*/" { if (--comment_count == 0) yybegin(YYINITIAL); }
    {COMMENT_TEXT} { }
}

{NEW_LINE} { }

. {
    javax.swing.JOptionPane.showMessageDialog(null,"ERROR AL COMPILAR:\nError léxico
    encontrado\nCaracter no válido");
}
  
```

Tabla 3. Código de lex1.txt del programa PsGram

1.2.2. ANALIZADOR SINTÁCTICO

Para desarrollar en analizador sintáctico se utilizó la librería Java-Cup la cuál genera los archivos Cup.java y sym.java

DEFINICIÓN DE LA GRAMÁTICA EN EL ARCHIVO jcup.txt

Para el desarrollo del analizador sintáctico se empezó creando jcup.txt (aquí se establece la gramática de PsGram y lee a java-cup), seguidamente se realizó GenCup.java el cual hace la llamada al primer archivo de texto creado y genera los archivos Cup.java y sym.java. Lo que se programó en jcup.txt es lo siguiente:

```
terminal
ENTERO,MAS,MASMAS,MENOS,MENOSMENOS,POR,DIVIDIR,IGUAL,IGUALIGUAL,MEN
Q,MAYQ,DIFERENTE,RES,MAYIGUAL,MENIGUAL;
terminal
PARENTI,PARENTF,PUNTOCOMA,SI,SINO,FINSI,PARA,FINPARA,MIENTRAS,FINMIENTR
AS,ESCRIBIR;
terminal
CADENA,AFIRMACION,INICIO,FIN,HACER,ENTONCES,DECIMAL,OPO,OPY,SWITCH,CA
SO,PARAR,DEFAULT,FINSWITCH,POTENCIA,LEER,DOSPUNTOS;

terminal java.lang.Number ENTERO_S;
terminal java.lang.Boolean BOOL;
terminal java.lang.String CADENA_S;
terminal java.lang.Number DECIMAL_S;
terminal java.lang.String VARIABLE;

non terminal programa, init_programa,cuerpo_programa,sentencias;
non terminal expresion,operadores_relacionales,residuo,operador_mod;
non terminal declaracion_for,expresion_for,incrementos;
non terminal expresiones_aritmeticas,factor,poten,termino;
non terminal inicializar_variable,concatenar_variable,concatenar,sentenciaO,sentenciaY;
non terminal declaracion_switch,casos_switch,todos_casos;

start with programa;
programa ::= init_programa: ip { RESULT=ip; };

//init_programa ::= INICIO cuerpo_programa:cp FIN{ RESULT="public void
cuerpoPrograma(){\n"+cp+"}\n\npublic static void main(String[]args){\nCodigoGenerado
codg = new CodigoGenerado();\ntry{\nFileWriter archivo = new FileWriter(new
File("Resultados.txt").getAbsolutePath(),false);\nPrintWriter escritor = new
PrintWriter(archivo);\nescritor.print(codg.get());\nescritor.close();\n}catch(IOException
ioe){\nJOptionPane.showMessageDialog(null,"Error al guardar los
resultados"+ioe,"PSGRAM",JOptionPane.ERROR_MESSAGE);\n}\n"; :} | INICIO
FIN { RESULT="public void cuerpoPrograma(){\n\n}\n\npublic static void main(Stri
CodigoGenerado();\ncodg.cuerpoPrograma()\ntry{\nFileWriter archivo = new
FileWriter(new File("Resultados.txt").getAbsolutePath(),false);\nPrintWriter escritor = new
ng[]args){\n\nCodigoGenerado codg = new
PrintWriter(archivo);\nescritor.print("\n");\nescritor.close();archivo.close();\n}catch(IOExcepi
onex){\nJOptionPane.showMessageDialog(null,"Error al guardar los
resultados"+ex,"PSGRAM",JOptionPane.ERROR_MESSAGE);\n}\n"; :};
```

```

init_programa ::= INICIO cuerpo_programa:cp FIN{: RESULT="public void
cuerpoPrograma()\n"+cp+"}\n\npublic static void main(String[]args)\nCodigoGenerado
codg = new CodigoGenerado();\ncodg.cuerpoPrograma();\n"; :} | INICIO FIN {:
RESULT="public void cuerpoPrograma()\n\n\npublic static void
main(String[]args)\n\nCodigoGenerado codg = new
CodigoGenerado();\ncodg.cuerpoPrograma()\n\n"; :};

cuerpo_programa ::= cuerpo_programa:ncp sentencias:dv {: RESULT=ncp+" "+dv; :} |
sentencias:dv1 {: RESULT=dv1; :};

sentencias ::= CADENA VARIABLE:v2 IGUAL CADENA_S:cs1 {: RESULT="String "+v2+" =
"+cs1+";\n";
Object[] o = Semantico.verificarVariablesRepetidas(getVector(),"cadena",v2);
setVector((Vector<Object[]> o[0]);
if( Boolean.parseBoolean(o[1].toString())==false){
setErrorEncontrado("Variable repetida\ncadena "+v2+" = "+cs1);
return null;
} :}
| AFIRMACION VARIABLE:a1 IGUAL BOOL:as1 {: RESULT = "boolean "+a1+" =
"+as1+";\n";
Object[] o=Semantico.verificarVariablesRepetidas(getVector(), "afirmacion",a1);
setVector((Vector<Object[]> o[0]);
if( Boolean.parseBoolean(o[1].toString()) == false){
if(as1== true){
setErrorEncontrado("Variable repetida\nafirmacion "+a1+" = v");
}else{
setErrorEncontrado("Variable repetida\nafirmacion "+a1+" = f");
}
return null;
} :}
| ESCRIBIR PARENTI concatenar_variable:covs PARENTF {:
RESULT="System.out.println("+covs+")\n"; :}

| SI PARENTI expresion:exp PARENTF ENTONCES cuerpo_programa:cp FINSI {:
RESULT="if ( "+exp+" ) {\n"+cp+"}\n"; :}
| SI PARENTI expresion:exp PARENTF ENTONCES FINSI {: RESULT="if ( "+exp+"
) {\n}\n"; :}
| SI PARENTI expresion:exp PARENTF ENTONCES cuerpo_programa:cp SINO
cuerpo_programa:cp2 FINSI {: RESULT="if ( "+exp+" ) {\n"+cp+"} else
{\n"+cp2+"}\n"; :}
| SI PARENTI expresion:exp PARENTF ENTONCES SINO cuerpo_programa:cp2
FINSI {: RESULT="if ( "+exp+" ) {\n} else {\n"+cp2+"}\n"; :}
| SI PARENTI expresion:exp PARENTF ENTONCES cuerpo_programa:cp SINO
FINSI {: RESULT="if ( "+exp+" ) {\n"+cp+"} else {\n}\n"; :}
| SI PARENTI expresion:exp PARENTF ENTONCES SINO FINSI {: RESULT="if (
"+exp+" ) {\n} else {\n}\n"; :}
| SI PARENTI residuo:res PARENTF ENTONCES cuerpo_programa:cp FINSI {:
RESULT="if ( "+res+" ) {\n"+cp+"}\n"; :}
| SI PARENTI residuo:res PARENTF ENTONCES FINSI {: RESULT="if ( "+res+" )
{\n}\n"; :}
| SI PARENTI residuo:res PARENTF ENTONCES cuerpo_programa:cp SINO
cuerpo_programa:cp2 FINSI {: RESULT="if ( "+res+" ) {\n"+cp+"} else
{\n"+cp2+"}\n"; :}
| SI PARENTI residuo:res PARENTF ENTONCES SINO cuerpo_programa:cp2
FINSI {: RESULT="if ( "+res+" ) {\n} else {\n"+cp2+"}\n"; :}
| SI PARENTI residuo:res PARENTF ENTONCES cuerpo_programa:cp SINO
FINSI {: RESULT="if ( "+res+" ) {\n"+cp+"} else {\n}\n"; :}

```

```

| SI PARENTI sentenciaO:sen PARENTF ENTONCES cuerpo_programa:cp FINSI
|   { RESULT="if ( "+sen+" ) {\n"+cp+"}\n"; ;}
| SI PARENTI sentenciaO:sen PARENTF ENTONCES FINSI { RESULT="if (
|   "+sen+" ) {\n}\n"; ;}
| SI PARENTI sentenciaO:sen PARENTF ENTONCES cuerpo_programa:cp SINO
|   cuerpo_programa:cp2 FINSI { RESULT="if ( "+sen+" ) {\n"+cp+"} else
|   {\n"+cp2+"}\n"; ;}
| SI PARENTI sentenciaO:sen PARENTF ENTONCES SINO cuerpo_programa:cp2
|   FINSI { RESULT="if ( "+sen+" ) {\n} else {\n"+cp2+"}\n"; ;}
| SI PARENTI sentenciaO:sen PARENTF ENTONCES cuerpo_programa:cp SINO
|   FINSI { RESULT="if ( "+sen+" ) {\n"+cp+"} else {\n}\n"; ;}
| SI PARENTI sentenciaO:sen PARENTF ENTONCES SINO FINSI { RESULT="if (
|   "+sen+" ) {\n} else {\n}\n"; ;}

| SI PARENTI sentenciaY:sen PARENTF ENTONCES cuerpo_programa:cp FINSI
|   { RESULT="if ( "+sen+" ) {\n"+cp+"}\n"; ;}
| SI PARENTI sentenciaY:sen PARENTF ENTONCES FINSI { RESULT="if (
|   "+sen+" ) {\n}\n"; ;}
| SI PARENTI sentenciaY:sen PARENTF ENTONCES cuerpo_programa:cp SINO
|   cuerpo_programa:cp2 FINSI { RESULT="if ( "+sen+" ) {\n"+cp+"} else
|   {\n"+cp2+"}\n"; ;}
| SI PARENTI sentenciaY:sen PARENTF ENTONCES SINO cuerpo_programa:cp2
|   FINSI { RESULT="if ( "+sen+" ) {\n} else {\n"+cp2+"}\n"; ;}
| SI PARENTI sentenciaY:sen PARENTF ENTONCES cuerpo_programa:cp SINO
|   FINSI { RESULT="if ( "+sen+" ) {\n"+cp+"} else {\n}\n"; ;}
| SI PARENTI sentenciaY:sen PARENTF ENTONCES SINO FINSI { RESULT="if (
|   "+sen+" ) {\n} else {\n}\n"; ;}

| MIENTRAS PARENTI expresion:exp PARENTF HACER cuerpo_programa:cp
|   FINMIENTRAS { RESULT="while ( "+exp+" ) {\n"+cp+"}\n"; ;}
| MIENTRAS PARENTI expresion:exp PARENTF HACER FINMIENTRAS {
|   RESULT="while ( "+exp+" ) {\n}\n"; ;}

| PARA PARENTI declaracion_for:df PUNTOCOMA expresion_for:expf
|   PUNTOCOMA incrementos:inc PARENTF HACER cuerpo_programa:cp
|   FINPARA { RESULT="for ( "+df+" ; "+expf+" ; "+inc+" ) {\n"+cp+"}\n"; ;}
| PARA PARENTI inicializar_variable:ivf PUNTOCOMA expresion_for:expf
|   PUNTOCOMA incrementos:inc PARENTF HACER cuerpo_programa:cp
|   FINPARA { RESULT="for ( "+ivf+" ; "+expf+" ; "+inc+" ) {\n"+cp+"}\n"; ;}
| PARA PARENTI inicializar_variable:ivf PUNTOCOMA expresion_for:expf
|   PUNTOCOMA incrementos:inc PARENTF HACER FINPARA { RESULT="for
|   (" +ivf+" ; "+expf+" ; "+inc+" ) {\n}\n"; ;}
| PARA PARENTI declaracion_for:df PUNTOCOMA expresion_for:expf
|   PUNTOCOMA incrementos:inc PARENTF HACER FINPARA { RESULT="for
|   (" +df+" ; "+expf+" ; "+inc+" ) {\n}\n"; ;}
| VARIABLE:var PARENTI concatenar:co PARENTF { RESULT = var+ " =
|   \""+co+"";
|   if(Semantico.verificarDeclaracion(getVector(), var) == false){
|       setErrorEncontrado("La variable "+var+" no está declarada");
|       return null;
|   }
|   if(Semantico.esCadena(getVector(), var) == false){
|       setErrorEncontrado("Incompatibilidad de tipos de datos, la variable
|       <"+var+"> no es de tipo cadena");
|       return null;
|   }
|   ;}

```

```

|   ENTERO VARIABLE:var IGUAL expresiones_aritmeticas:opar {: RESULT="int
|       "+var+" = "+opar+"\n";
|   Object[] o=Semantico.verificarVariablesRepetidas(getVector(), "entero", var);
|   setVector((Vector<Object[]> o[0]);
|   if(Boolean.parseBoolean(o[1].toString()) == false){
|       setErrorEncontrado("Variable repetida\nentero "+var+" = "+opar);
|       return null;
|   };}
|   DECIMAL VARIABLE:vr1 IGUAL expresiones_aritmeticas:opra {:
|       RESULT="double "+vr1+" = "+opra+"\n";
|   Object[] o=Semantico.verificarVariablesRepetidas(getVector(), "decimal", vr1);
|   setVector((Vector<Object[]> o[0]);
|   if(Boolean.parseBoolean(o[1].toString()) == false){
|       setErrorEncontrado("Variable repetida\ndecimal "+vr1+" = "+opra);
|       return null;
|   };}
|   SWITCH declaracion_switch:ds HACER todos_casos:td FINSWITCH {:
|       RESULT="switch "+ ds +"{\n"+ td+"\n"; ;}
|   inicializar_variable:iv {: RESULT=iv+"\n"; ;}
;
declaracion_switch ::=
    PARENTI ENTERO_S:es PARENTF {: RESULT=" (" + es +)"; ;}
|   PARENTI VARIABLE:va PARENTF {: RESULT=" (" + va +)"; ;}
;
;
todos_casos ::=
    casos_switch:cs DEFAULT DOSPUNTOS cuerpo_programa:cp PARAR {:
    RESULT=cs+"\ndefault:\n"+cp+"\nbreak;\n"; ;}
;
;
casos_switch ::=
    CASO ENTERO_S:es DOSPUNTOS cuerpo_programa:cp PARAR {:
    RESULT="case "+ es +":\n"+ cp +"break;\n"; ;}
|   CASO VARIABLE:va DOSPUNTOS cuerpo_programa:cp1 PARAR {:
    RESULT="case "+ va +":\n"+ cp1 +"break;\n"; ;}
|   SWITCH declaracion_switch:ds HACER todos_casos:td FINSWITCH {:
    RESULT="switch "+ ds +"{\n"+ td+"\n"; ;}
|   casos_switch:cs CASO ENTERO_S:es1 DOSPUNTOS cuerpo_programa:cp1
    PARAR{: RESULT=cs+"\ncase "+es1+":\n"+ cp1 +"break;\n"; ;}
;
;
inicializar_variable ::=
    VARIABLE:variab IGUAL expresiones_aritmeticas:exp {: RESULT=variab+" =
    "+exp;
    if(Semantico.verificarDeclaracion(getVector(), variab) == false){
        setErrorEncontrado("La variable "+variab+" no está declarada");
        return null;
    }
    /*if(Semantico.esEntero(getVector(), variab) == false){
        setErrorEncontrado("Incompatibilidad de tipos de datos, la variable
        <"+variab+"> no es de tipo entero");
        return null;
    }*/
    if(Semantico.esEnteroDecimal(getVector(), variab) == false){
        setErrorEncontrado("Incompatibilidad de tipos de datos, la variable
        <"+variab+"> no es de tipo entero o decimal");
        return null;
    }
    ;}
;
;

```



```

concatenar_variable ::=
  concatenar_variable:cv MAS VARIABLE:cv1 {
    if(Semantico.verificarDeclaracion(getVector(), cv1) == false){
      setErrorEncontrado("La variable "+cv1+" no está declarada");
      return null;
    }
    if(Semantico.esAfirmacion(getVector(), cv1) == true){
      if(Boolean.parseBoolean(cv1) == true){
        RESULT = cv+" + \"v\"";
      }else{
        RESULT = cv+" + \"f\"";
      }
    }else{
      RESULT =cv+" + "+cv1;
    }
  }
  :}
| concatenar_variable:cv MAS ENTERO_S:ev1 {:RESULT = cv+" + "+ev1;:}
| concatenar_variable:cv MAS DECIMAL_S:dv1 {:RESULT = cv+" + "+dv1;:}
| concatenar_variable:cv MAS CADENA_S:cav1 {:RESULT = cv+" + "+cav1;:}
| concatenar_variable:cv MAS BOOL:bv1 {
  if(bv1 == true){
    RESULT = cv+" + \"v\"";
  }else{
    RESULT = cv+" + \"f\"";
  }
  :}
| VARIABLE:v1 {:RESULT =v1;
  if(Semantico.verificarDeclaracion(getVector(), v1) == false){
    setErrorEncontrado("La variable "+v1+" no está declarada");
    return null;
  }
  :}
| ENTERO_S:e1 {:RESULT = e1;:}
| DECIMAL_S:d1 {:RESULT = d1;:}
| CADENA_S:c1 {:RESULT = c1;:}
| BOOL:a1 {
  if(a1 == true){
    RESULT = "\"v\"";
  }else{
    RESULT = "\"f\"";
  }
  :}
;
concatenar ::=
  concatenar:cv MAS VARIABLE:cv1 {
    if(Semantico.verificarDeclaracion(getVector(), cv1) == false){
      setErrorEncontrado("La variable "+cv1+" no está declarada");
      return null;
    }
    if(Semantico.esAfirmacion(getVector(), cv1) == true){
      if(Boolean.parseBoolean(cv1) == true){
        RESULT = cv+" + \"v\"";
      }else{
        RESULT = cv+" + \"f\"";
      }
    }
  }else{
    RESULT =cv+" + "+cv1;
  }

```

```

    }
    ;}
    concatenar:cv MAS ENTERO_S:ev1 {:RESULT = cv+ " +ev1;};
    concatenar:cv MAS DECIMAL_S:dv1 {:RESULT = cv+ " +dv1;};
    concatenar:cv MAS CADENA_S:cav1 {:RESULT = cv+ " +cav1;};
    concatenar:cv MAS BOOL:av1 {:
    if(av1 == true){
        RESULT = cv+ " +\"v\"";
    }else{
        RESULT = cv+ " +\"f\"";
    }
    ;}
    VARIABLE:v1 {:RESULT =v1;
    if(Semantico.verificarDeclaracion(getVector(), v1) == false){
        setErrorEncontrado("La variable "+v1+" no está declarada");
        return null;
    }
    ;}
    CADENA_S:c1 {:RESULT = c1;};
    ENTERO_S:e1 {:RESULT = e1;};
    DECIMAL_S:d1 {:RESULT = d1;};
    BOOL:a1 {:
    if(a1 == true){
        RESULT = "\"v\"";
    }else{
        RESULT = "\"f\"";
    }
    ;}
;

sentenciaO ::=
    sentenciaO:s1 OPO expresion:ex {: RESULT=s1+ " || "+ex; ;}
    | PARENTI sentenciaY:s1 PARENTF OPO expresion:ex {: RESULT="("+s1+") || "+ex; ;}
    ;}
    | expresion:ex OPO PARENTI sentenciaY:s1 PARENTF {: RESULT=ex + " || (" + s1 +
    ")"; ;}
    | sentenciaO:s1 OPO PARENTI expresion:ex PARENTF {: RESULT=s1+ " || (" + ex
    + ")"; ;}
    | expresion:ex1 OPO expresion:ex2 {: RESULT=ex1+ " || "+ex2; ;}
    | expresion:ex1 OPO PARENTI expresion:ex2 PARENTF {: RESULT= ex1+ " || (" + ex2
    + ")"; ;}
    | PARENTI expresion:ex1 PARENTF OPO expresion:ex2 {: RESULT= "(" + ex1 + ") ||
    "+ ex2; ;}
    | PARENTI expresion:ex1 PARENTF OPO PARENTI expresion:ex2 PARENTF {:
    RESULT= "(" + ex1 + ") || (" + ex2 + ")"; ;}
    | PARENTI sentenciaO:s1 PARENTF OPO PARENTI sentenciaY:s2 PARENTF {:
    RESULT="("+ s1 + ") || (" + s2 + ")"; ;}
;

sentenciaY ::=
    sentenciaY:s1 OPY expresion:ex {: RESULT=s1+ " && "+ex; ;}
    | PARENTI sentenciaO:s1 PARENTF OPY expresion:ex {: RESULT="("+s1+") &&
    "+ex; ;}
    | expresion:ex OPY PARENTI sentenciaO:s1 PARENTF {: RESULT=ex + " && (" + s1
    + ")"; ;}
    | sentenciaY:s1 OPY PARENTI expresion:ex PARENTF {: RESULT=s1+ " && (" + ex
    + ")"; ;}

```

```

| expresion:ex1 OPY expresion:ex2 {: RESULT=ex1+" && "+ex2; :}
| expresion:ex1 OPY PARENTI expresion:ex2 PARENTF {: RESULT= ex1+" && ("+
  ex2 +)"; :}
| PARENTI expresion:ex1 PARENTF OPY expresion:ex2 {: RESULT= "("+ ex1 +) &&
  "+ ex2; :}
| PARENTI expresion:ex1 PARENTF OPY PARENTI expresion:ex2 PARENTF {:
  RESULT= "("+ ex1 +) && ("+ ex2 +)"; :}
| PARENTI sentenciaY:s1 PARENTF OPY PARENTI sentenciaO:s2 PARENTF {:
  RESULT="("+ s1 +) && ("+ s2 +)"; :}
;
residuo ::=
  VARIABLE:v1 operador_mod:om VARIABLE:v2 operadores_relacionales:op
  ENTERO_S:es {: RESULT=v1+" "+GetSym.get(Integer.parseInt(om.toString()))+"
  "+v2+" "+GetSym.get(Integer.parseInt(op.toString()))+" "+es;
  if(Semantico.verificarDeclaracion(getVector(), v1) == false){
    setErrorEncontrado("La variable "+v1+" no está declarada");
    return null;
  }
  if(Semantico.verificarDeclaracion(getVector(), v2) == false){
    setErrorEncontrado("La variable "+v2+" no está declarada");
    return null;
  }
};}
| VARIABLE:v1 operador_mod:om VARIABLE:v2 operadores_relacionales:op
  DECIMAL_S:es {: RESULT=v1+"
  "+GetSym.get(Integer.parseInt(om.toString()))+" "+v2+"
  "+GetSym.get(Integer.parseInt(op.toString()))+" "+es;
  if(Semantico.verificarDeclaracion(getVector(), v1) == false){
    setErrorEncontrado("La variable "+v1+" no está declarada");
    return null;
  }
  if(Semantico.verificarDeclaracion(getVector(), v2) == false){
    setErrorEncontrado("La variable "+v2+" no está declarada");
    return null;
  }
};}
| VARIABLE:v3 operador_mod:omo VARIABLE:v4 operadores_relacionales:op1
  VARIABLE:v5 {: RESULT=v3+"
  "+GetSym.get(Integer.parseInt(omo.toString()))+" "+v4+"
  "+GetSym.get(Integer.parseInt(op1.toString()))+" "+v5;
  if(Semantico.verificarDeclaracion(getVector(), v3) == false){
    setErrorEncontrado("La variable "+v3+" no está declarada");
    return null;
  }
  if(Semantico.verificarDeclaracion(getVector(), v4) == false){
    setErrorEncontrado("La variable "+v4+" no está declarada");
    return null;
  }
  if(Semantico.verificarDeclaracion(getVector(), v5) == false){
    setErrorEncontrado("La variable "+v5+" no está declarada");
    return null;
  }
};}
| ENTERO_S:ens operador_mod:od VARIABLE:v6 operadores_relacionales:ore
  ENTERO_S:ent {: RESULT=ens+"
  "+GetSym.get(Integer.parseInt(od.toString()))+" "+v6+"
  "+GetSym.get(Integer.parseInt(ore.toString()))+" "+ent;
  if(Semantico.verificarDeclaracion(getVector(), v6) == false){
    setErrorEncontrado("La variable "+v6+" no está declarada");
    return null;
  }
};}

```

```

    };}
|   DECIMAL_S:ens operador_mod:od VARIABLE:v6 operadores_relacionales:ore
    DECIMAL_S:ent {: RESULT=ens+
    "+GetSym.get(Integer.parseInt(od.toString()))+" "+v6+"
    "+GetSym.get(Integer.parseInt(ore.toString()))+" "+ent;
    if(Semantico.verificarDeclaracion(getVector(), v6) == false){
        setErrorEncontrado("La variable "+v6+" no está declarada");
        return null;
    };}
|   ENTERO_S:en1 operador_mod:odm ENTERO_S:en2 operadores_relacionales:opd
    ENTERO_S:en3 {: RESULT=en1+
    "+GetSym.get(Integer.parseInt(odm.toString()))+" "+en2+"
    "+GetSym.get(Integer.parseInt(opd.toString()))+" "+en3;
    };}
|   DECIMAL_S:en1 operador_mod:odm DECIMAL_S:en2 operadores_relacionales:opd
    DECIMAL_S:en3 {: RESULT=en1+
    "+GetSym.get(Integer.parseInt(odm.toString()))+" "+en2+"
    "+GetSym.get(Integer.parseInt(opd.toString()))+" "+en3;
    };}
|   VARIABLE:v7 operador_mod:opm ENTERO_S:ents operadores_relacionales:opr
    ENTERO_S:ens {: RESULT=v7+
    "+GetSym.get(Integer.parseInt(opm.toString()))+" "+ents+
    "+GetSym.get(Integer.parseInt(opr.toString()))+" "+ens;
    if(Semantico.verificarDeclaracion(getVector(), v7) == false){
        setErrorEncontrado("La variable "+v7+" no está declarada");
        return null;
    };}
|   VARIABLE:v7 operador_mod:opm DECIMAL_S:ents operadores_relacionales:opr
    DECIMAL_S:ens {: RESULT=v7+
    "+GetSym.get(Integer.parseInt(opm.toString()))+" "+ents+
    "+GetSym.get(Integer.parseInt(opr.toString()))+" "+ens;
    if(Semantico.verificarDeclaracion(getVector(), v7) == false){
        setErrorEncontrado("La variable "+v7+" no está declarada");
        return null;
    };}
;
expresion ::=

    VARIABLE:v1 operadores_relacionales:op1 VARIABLE:v2 {: RESULT=v1+
    "+GetSym.get(Integer.parseInt(op1.toString()))+" "+v2;
    if(Semantico.verificarCondiciones(getVector(), v1, v2,
        GetSym.get(Integer.parseInt(op1.toString())) == false){
        return null;
    };}
|   VARIABLE:v3 operadores_relacionales:op2 ENTERO_S:es1 {: RESULT=v3+
    "+GetSym.get(Integer.parseInt(op2.toString()))+" "+es1;
    if(Semantico.verificarDeclaracion(getVector(), v3) == false){
        setErrorEncontrado("La variable "+v3+" no está declarada");
        return null;
    }
    if(Semantico.esEntero(getVector(), v3) == false){
        setErrorEncontrado("Incompatibilidad de tipos de datos, la variable <"+v3+>
        no es de tipo entero");
        return null;
    };}
|   VARIABLE:va1 IGUALIGUAL BOOL:as1 {: RESULT=va1+ == "+as1;
    if(Semantico.verificarDeclaracion(getVector(), va1) == false){

```

```

        setErrorEncontrado("La variable "+va1+" no está declarada");
        return null;
    }
    if(Semantico.esAfirmacion(getVector(), va1) == false){
        setErrorEncontrado("Incompatibilidad de tipos de datos, la variable <"+va1+">
        no es de tipo afirmacion");
        return null;
    }
}
VARIABLE:vai1 IGUAL BOOL:asi1 {: RESULT=vai1+" = "+asi1;
if(Semantico.verificarDeclaracion(getVector(), vai1) == false){
    setErrorEncontrado("La variable "+vai1+" no está declarada");
    return null;
}
if(Semantico.esAfirmacion(getVector(), vai1) == false){
    setErrorEncontrado("Incompatibilidad de tipos de datos, la variable
    <"+vai1+"> no es de tipo afirmacion");
    return null;
}
}
| ENTERO_S:es2 operadores_relacionales:op3 VARIABLE:v4 {: RESULT=es2+"
    "+GetSym.get(Integer.parseInt(op3.toString()))+" "+v4;
f(Semantico.verificarDeclaracion(getVector(), v4) == false){
    setErrorEncontrado("La variable "+v4+" no está declarada");
    return null;
}
if(Semantico.esEntero(getVector(), v4) == false){
    setErrorEncontrado("Incompatibilidad de tipos de datos, la variable <"+v4+">
    no es de tipo entero");
    return null;
}
}
| BOOL:as2 IGUALIGUAL VARIABLE:va4 {: RESULT=as2+" == "+va4;
if(Semantico.verificarDeclaracion(getVector(), va4) == false){
    setErrorEncontrado("La variable "+va4+" no está declarada");
    return null;
}
if(Semantico.esAfirmacion(getVector(), va4) == false){
    setErrorEncontrado("Incompatibilidad de tipos de datos, la variable <"+va4+">
    no es de tipo afirmacion");
    return null;
}
}
| ENTERO_S:es3 operadores_relacionales:op4 ENTERO_S:es4 {: RESULT=es3+"
    "+GetSym.get(Integer.parseInt(op4.toString()))+" "+es4; ;}
| BOOL:af1 {: RESULT=af1; ;}
| VARIABLE:v5 {: RESULT=v5;
if(Semantico.verificarDeclaracion(getVector(), v5) == false){
    setErrorEncontrado("La variable "+v5+" no está declarada");
    return null;
}
if(Semantico.esAfirmacion(getVector(), v5) == false){
    setErrorEncontrado("Condición no válida, la variable <"+v5+"> no es de tipo
    afirmación");
    return null;
}
}
| VARIABLE:v3 operadores_relacionales:op2 DECIMAL_S:ds1 {: RESULT=v3+"
    "+GetSym.get(Integer.parseInt(op2.toString()))+" "+ds1;
if(Semantico.verificarDeclaracion(getVector(), v3) == false){
    setErrorEncontrado("La variable "+v3+" no está declarada");
    return null;
}

```

```

    }
    if(Semantico.esDecimal(getVector(), v3) == false){
        setErrorEncontrado("Incompatibilidad de tipos de datos, la variable
        <"+v3+"> no es de tipo decimal");
        return null;
    }
}
| DECIMAL_S:ds2 operadores_relacionales:op3 VARIABLE:v4 {: RESULT=ds2+"
  "+GetSym.get(Integer.parseInt(op3.toString()))+" "+v4;
  if(Semantico.verificarDeclaracion(getVector(), v4) == false){
      setErrorEncontrado("La variable "+v4+" no está declarada");
      return null;
  }
  if(Semantico.esDecimal(getVector(), v4) == false){
      setErrorEncontrado("Incompatibilidad de tipos de datos, la variable
      <"+v4+"> no es de tipo decimal");
      return null;
  }
}
| DECIMAL_S:ds3 operadores_relacionales:op4 DECIMAL_S:ds4 {: RESULT=ds3+"
  "+GetSym.get(Integer.parseInt(op4.toString()))+" "+ds4; ;}
;
operadores_relacionales ::=
    MAYQ {: RESULT=sym.MAYQ; ;}
    | MENQ {: RESULT=sym.MENQ; ;}
    | IGUALIGUAL {: RESULT=sym.IGUALIGUAL; ;}
    | DIFERENTE {: RESULT=sym.DIFERENTE; ;}
    | MAYIGUAL {: RESULT=sym.MAYIGUAL; ;}
    | MENIGUAL {: RESULT=sym.MENIGUAL; ;}
;
operador_mod ::=
    RES {: RESULT=sym.RES; ;}
;
declaracion_for ::=
    ENTERO VARIABLE:v IGUAL ENTERO_S:es1 {: RESULT=" int "+v+" = "+es1+" ";
    Object[] o=Semantico.verificarVariablesRepetidas(getVector(), "entero", v);
    setVector((Vector<Object[]> o[0]);
    if(Boolean.parseBoolean(o[1].toString()) == false){
        setErrorEncontrado("Variable repetida\nentero "+v+" = "+es1);
        return null;
    }
}
;
incrementos ::=
    VARIABLE:v MASMAS {: RESULT=v+" "+GetSym.get(sym.MASMAS);
    if(Semantico.verificarDeclaracion(getVector(), v) == false){
        setErrorEncontrado("La variable "+v+" no está declarada");
        return null;
    }
    if(Semantico.esEntero(getVector(), v) == false){
        setErrorEncontrado("Expresión de incremento no válida, la variable
        <"+v+"> no es de tipo entero");
        return null;
    }
}
| VARIABLE:v1 MENOSMENOS {: RESULT=v1+"
  "+GetSym.get(sym.MENOSMENOS);
  if(Semantico.verificarDeclaracion(getVector(), v1) == false){
      setErrorEncontrado("La variable "+v1+" no está declarada");
      return null;
  }
}

```

```

    }
    if(Semantico.esEntero(getVector(), v1) == false){
        setErrorEncontrado("Expresión de decremento no válida, la variable
        <"+v1+"> no es de tipo entero");
        return null;
    }
    :}
| VARIABLE:v2 IGUAL VARIABLE:v3 MAS ENTERO_S:e1 {: RESULT=v2+"
  "+GetSym.get(sym.IGUAL)+" "+v3+" "+GetSym.get(sym.MAS)+" "+e1;
  if(Semantico.verificarCondiciones(getVector(), v2, v3, "=") == false){
      return null;
  }
  :}
| VARIABLE:v4 IGUAL VARIABLE:v5 MENOS ENTERO_S:e2 {: RESULT=v4+"
  "+GetSym.get(sym.IGUAL)+" "+v5+" "+GetSym.get(sym.MENOS)+" "+e2;
  if(Semantico.verificarCondiciones(getVector(), v4, v5) == false){
      return null;
  }
  :}
;
expresion_for ::=
  VARIABLE:v1 operadores_relacionales:op1 VARIABLE:v2 {: RESULT=v1+"
  "+GetSym.get(Integer.parseInt(op1.toString()))+" "+v2;
  if(Semantico.verificarCondiciones(getVector(), v1, v2) == false){
      return null;
  }
  :}
| VARIABLE:v3 operadores_relacionales:op2 ENTERO_S:es1 {: RESULT=v3+"
  "+GetSym.get(Integer.parseInt(op2.toString()))+" "+es1;
  if(Semantico.verificarDeclaracion(getVector(), v3) == false){
      setErrorEncontrado("La variable "+v3+" no está declarada");
      return null;
  }
  if(Semantico.esEntero(getVector(), v3) == false){
      setErrorEncontrado("Incompatibilidad de tipos de datos, la variable
      <"+v3+"> no es de tipo entero");
      return null;
  }
  :}
| ENTERO_S:es2 operadores_relacionales:op3 VARIABLE:v4 {: RESULT=es2+"
  "+GetSym.get(Integer.parseInt(op3.toString()))+" "+v4;
  if(Semantico.verificarDeclaracion(getVector(), v4) == false){
      setErrorEncontrado("La variable "+v4+" no está declarada");
      return null;
  }
  if(Semantico.esEntero(getVector(), v4) == false){
      setErrorEncontrado("Incompatibilidad de tipos de datos, la variable
      <"+v4+"> no es de tipo entero");
      return null;
  }
  :}
| ENTERO_S:es3 operadores_relacionales:op4 ENTERO_S:es4 {: RESULT=es3+"
  "+GetSym.get(Integer.parseInt(op4.toString()))+" "+es4; :}
;
expresiones_aritmeticas ::=
  expresiones_aritmeticas:ea2 MAS factor:f3 {: RESULT=ea2+" + "+f3; :}
| expresiones_aritmeticas:ea MENOS factor:f2 {: RESULT=ea+" - "+f2; :}
| expresiones_aritmeticas:ea RES factor:f2 {: RESULT=ea+" % "+f2; :}
| factor:f {: RESULT=f; :}
| poten:p {: RESULT=p; :}

```

```

| LEER PARENTI concatenar_variable:covs PARENTF {;
|   RESULT="Integer.parseInt(javax.swing.JOptionPane.showInputDialog(null,"+
|   covs+","\PSGRAM",javax.swing.JOptionPane.INFORMATION_MESSAGE));
|   };
;

factor ::=
| factor:f POR termino:t1 {; RESULT=f+" * "+t1; ;}
| factor:f1 DIVIDIR termino:t2 {; RESULT=f1+" / "+t2; ;}
| termino:t {; RESULT=t; ;}
;

poten ::=
| POTENCIA PARENTI termino:bas PUNTOCOMA termino:pt PARENTF {;
|   RESULT= "Math.pow ( "+bas+" , "+pt+" ) "; ;}
;

termino ::=
| PARENTI expresiones_aritmeticas:ea PARENTF {; RESULT="( "+ea+" )"; ;}
| ENTERO_S:es {; RESULT=es; ;}
| DECIMAL_S:ds {; RESULT=ds; ;}
| VARIABLE:ve {; RESULT=ve;
|   if(Semantico.verificarDeclaracion(getVector(), ve) == false){
|     setErrorEncontrado("La variable "+ve+" no está declarada");
|     return null;
|   }
|   if(Semantico.esEnteroDecimal(getVector(), ve) == false){
|     setErrorEncontrado("Incompatibilidad de tipos de datos en la expresión,\nla
|     variable <"+ve+"> no es de tipo entero o decimal");
|     return null;
|   }
| };}
;

```

Tabla 4 . Parte de código del archivo jcup.txt, perteneciente a PsGram

1.2.3. ANALIZADOR SEMÁNTICO

☀ Son métodos que, validan que la semántica esté correcta. A continuación el código de la clase Semántico.java es:

```

package edu.psg.dominio.analiza;

import com.psg.dominio.analiza.Analizador;
import java.util.Vector;

public class Semantico {
    private static Analizador ada = new Analizador();
    /**
     * Verifica que no exista declaración de variables repetidas
     * si esta contenida en el vector no se agrega, para que no hayan repeticiones
     (inconsistencia)
     * @param Vector de Object[] v, Object tipo, Object nombre
     * @return Object[] vector con todas las variables declaradas, true si se agreg
     */
    @SuppressWarnings("unchecked")
    public static Object [] verificarVariablesRepetidas(Vector<Object[]> v, Object tipo,
    Object nombre){
        Object[] objetoRetorno=new Object[2];
        for (int i = 0; i < v.size(); i++) {
            if(v.get(i)[1].equals(nombre)){
                objetoRetorno[0]=v;
                objetoRetorno[1]=false;
                return objetoRetorno;
            }
        }
        Object[] objetoInterno=new Object[2];
        objetoInterno[0]=tipo;
        objetoInterno[1]=nombre;
        v.add(objetoInterno);
        objetoRetorno[0]=v;
        objetoRetorno[1]=true;
        return objetoRetorno;
    }
    /**
     * Verifica que una variable utilizada está previamente declarada
     * @param Vector de Object[] v, Object tipo, Object nombreVariable
     * @return true si la encuentra declarada, de lo contrario false
     */
    public static boolean verificarDeclaracion(Vector<Object[]> v, Object nombreVar){
        for (int i = 0; i < v.size(); i++) {
            if(v.get(i)[1].equals(nombreVar)){
                return true;
            }
        }
        return false;
    }
    /**
     * Permite comprobar si una variable es de tipo Entero
     * @param Vector de Object[] v, Object tipo, Object nombre
     * @return true si la variable es de tipo entero, de lo contrario false
     */
}

```

```

public static boolean esEntero(Vector<Object[]> v, Object nombreVar){
    for (int i = 0; i < v.size(); i++) {
        if(v.get(i)[1].equals(nombreVar)){
            if(v.get(i)[0].equals("entero")){
                return true;
            }else{
                return false;
            }
        }
    }
    return false;
}
/**
 * Permite comprobar si una variable es de tipo afirmacion
 * @param Vector de Object[] v, Object tipo, Object nombre
 * @return true si la variable es de tipo afirmacion, de lo contrario false
 */
public static boolean esAfirmacion(Vector<Object[]> v, Object nombreVar){
    for (int i = 0; i < v.size(); i++) {
        if(v.get(i)[1].equals(nombreVar)){
            if(v.get(i)[0].equals("afirmacion")){
                return true;
            }else{
                return false;
            }
        }
    }
    return false;
}
public static boolean esDecimal(Vector<Object[]> v, Object nombreVar){
    for (int i = 0; i < v.size(); i++) {
        if(v.get(i)[1].equals(nombreVar)){
            if(v.get(i)[0].equals("decimal")){
                return true;
            }else{
                return false;
            }
        }
    }
    return false;
}
public static boolean esEnteroDecimal(Vector<Object[]> v, Object nombreVar){
    for (int i = 0; i < v.size(); i++) {
        if(v.get(i)[1].equals(nombreVar)){
            if(v.get(i)[0].equals("entero")){
                return true;
            }else if(v.get(i)[0].equals("decimal")) {
                return true;
            }else
                return false;
        }
    }
    return false;
}
/**
 * Permite comprobar si una variable es de tipo cadena
 * @param Vector de Object[] v, Object tipo, Object nombre

```

```

* @return true si la variable es de tipo cadena, de lo contrario false
*/
public static boolean esCadena(Vector<Object[]> v, Object nombreVar){
    for (int i = 0; i < v.size(); i++) {
        if(v.get(i)[1].equals(nombreVar)){
            if(v.get(i)[0].equals("cadena")){
                return true;
            }else{
                return false;
            }
        }
    }
    return false;
}

/**
 * Devuelve el tipo de dato de una variable
 * @param Vector de Object[] v, Object tipo, Object nombre
 * @return Object
 */
public static Object getTipoDato(Vector<Object[]> v, Object nombreVar){
    for (int i = 0; i < v.size(); i++) {
        if(v.get(i)[1].equals(nombreVar)){
            return v.get(i)[0];
        }
    }
    return null;
}

/**
 * Se la utiliza para hacer comparaciones entre dos variables
 * @param Vector de Object[] v, Object nombreVar1, Object nombreVar2, Object
operador
 * @return true si la variable es de tipo afirmacion, de lo contrario false
 */
public static boolean verificarCondiciones(Vector<Object[]> v, Object nombreVar1,
Object nombreVar2, Object operador){
    if(verificarDeclaracion(v, nombreVar1) == false){
        ada.guardarError("ERROR AL COMPILAR:\nError Semantico
encontrado\n"+"La variable "+nombreVar1+" no se encuentra
declarada");
        return false;
    }
    if(verificarDeclaracion(v, nombreVar2) == false){
        ada.guardarError("ERROR AL COMPILAR:\nError Semantico
encontrado\n"+"La variable "+nombreVar2+" no se encuentra
declarada");
        return false;
    }
    Object tipo1 = getTipoDato(v, nombreVar1);
    Object tipo2 = getTipoDato(v, nombreVar2);
    if(tipo1.equals(tipo2) == false){
        ada.guardarError("ERROR AL COMPILAR:\nError Semantico
encontrado\n"+"Tipos de datos no compatibles "+nombreVar1+"
tipo: "+tipo1+" y "+nombreVar2+" tipo: "+tipo2);
        return false;
    }else{
        if(tipo1.equals("afirmacion") && tipo2.equals("afirmacion")){
            if(operador.equals("<") || operador.equals(">")){

```

1.3. MÓDULO DE DIAGRAMA DE FLUJO

OBJETIVO 3: Elaborar un módulo gráfico que permita implementar los diferentes componentes del diagrama de flujo.

Para elaborar el modulo grafico se utilizó la librería JGraphX, que es la mejor biblioteca en java para la visualizacion de graficos.

✿ La clase EditorGraf.java, contiene la pantalla del DIAGRAMA DE FLUJO PSGRAM. (Ver en manual del progamador pág. 12).

✿ Cada uno de los componentes ha sido dibujado en clases, por ejemplo la clase *Entrada_Salida.java*. (Ver en manual del progamador pág.16)

✿ Al diagrama de flujo una vez creado se lo guarda como .png pero dentro tiene una codificación xml, la cual permite cargarlo nuevamente al gráfico con todos sus elementos para que estos puedan ser editados nuevamente; en la clase *AccionesEditor.java* de PsGram está el método que realiza la acción de guardar. (Ver en manual del progamador pág.16)

1.4. DESARROLLO DE PsGram

OBJETIVO 4: Desarrollar un módulo que permita convertir un lenguaje de pseudocódigo a diagrama de flujo y de un diagrama de flujo a pseudocódigo.

PSEUDOCODIGO A DIAGRAMA DE FLUJO

El Pseudo-Código que se escribe en el la interfaz *Editor* de PsGram, después de compilar y no tener ningún error da la opción de generar su respectivo Diagrama de Flujo; para eso se utilizó la clase DibujarAction.java.

A continuación se presenta el código de la clase Styles.java y DibujarAction.java:

- Clase Styles.java: aquí constan las descripciones de las variables estáticas que almacenan los elementos del diagrama de flujo.

```
package edu.psg.graficador.editor;
public class Styles {

    public static final String OPERACIONES =
        "shape=operaciones;fillColor=white;fontColor=blue;fontSize=15";
    public static final String ENTRADA_SALIDA = "shape=entrada_salida;fontSize=15";
    public static final String MIENTRAS =
        "shape=mientras;fillColor=green;fontColor=black;fontSize=15";
    public static final String SWITCH =
        "shape=switch;fillColor=pink;fontColor=black;strokeColor=#5d65df;fontSize=15";
    public static final String PARA = "shape=para;fillColor=yellow;fontColor=black;fontSize=15";
    public static final String CONDICIONAL =
        "shape=condicional;fillColor=#adc5ff;fontColor=white;fontSize=15;gradientColor=#7d85d
        f;strokeColor=#5d65df";
    public static final String CONECTOR =
        "shape=conector;fillColor=black;fontColor=white;fontSize=15";
    public static final String INICIO_FIN =
        "shape=inicio_fin;fillColor=gray;gradientColor=#7d85df;strokeColor=#5d65df;fontColor=w
        hite;fontSize=15";
}

```

Tabla 6 .Clase Styles.java

- Clase DibujarAction.java

```
public class DibujarAction extends AbstractAction {
    private String pseudocodigo;
    public static final EditorPSGram getEditor(ActionEvent e) {
        if (e.getSource() instanceof Component) {
            Component component = (Component) e.getSource();
            while (component != null && !(component instanceof EditorPSGram)) {
                component = component.getParent();
            }
            return (EditorPSGram) component;
        }
    }
}

```

```

    } return null;
}
public void actionPerformed(ActionEvent e) {
    EditorPSGram editor = getEditor(e);
    if (editor != null) {
        dibujar(editor.getGraphComponent());
    }
}
public List<String> obtenerList(String s){
    List<String> res = new ArrayList<String>();
    while(s.length() > 0){
        String s1 = "";
        s=s.trim();
        if(s.startsWith("\n")){
            if(s.length()>1){
                s = s.substring(1).trim();
            }else{
                s="";
            }
        }
        if(s.contains("\n")){
            s1= s.substring(0, s.indexOf("\n")).trim();
        }else{
            s1 = s.trim();
        }
        if(s1.trim().length() > 0)res.add(s1);
        s = s.replaceFirst(Pattern.quote(s1),Matcher.quoteReplacement(""));
    }
    return res;
}
List<Object> conectores;
List<Object> recorridos;
List<Object> elementosMientras;
List<Object> elementosPara;
List<Object> elementosIf;
List<Object> elementosSwitch;

public void dibujar(mxGraphComponent graphComponent) {
    final mxGraph grafico = graphComponent.getGraph();
    JTextPane ac= new JTextPane();
    ac.setText(pseudocodigo);
    //Se recorre el String para obtener uns List<String> en donde cada elemento es una línea
    de texto
    List<String> lineas = obtenerList(ac.getText());
    conectores = new ArrayList<Object>();
    Long conect = 0L;
    elementosIf = new ArrayList<Object>();
    elementosSwitch = new ArrayList<Object>();
    elementosPara = new ArrayList<Object>();
    elementosMientras = new ArrayList<Object>();
    //Se procede a graficar
    grafico.getModel().beginUpdate();
    try {
        Object ob= null;
        double x = 0;
        double y = 0;
        String textoLineaAux = "";
        //Se recorre linea por linea del String enviado desde el editor de pseudocodigo para ir
        graficando
        for(int i=0; i<lineas.size();i++){
            String style = "";
            String s1 = lineas.get(i);
            String textoLinea = "";

```

```

if(!textoLineaAux.equals("")){
    textoLinea = textoLineaAux;
    textoLineaAux = "";
}
if((s1.startsWith("inicio") || s1.startsWith("fin")) && !s1.startsWith("finsi")&&
    !s1.startsWith("finmientras")&& !s1.startsWith("finparacaso")&&
    !s1.startsWith("finpara")){
    style = Styles.INICIO_FIN;
}
if(s1.startsWith("escribir") || s1.startsWith("leer")){
    style = Styles.ENTRADA_SALIDA;
}
if(s1.startsWith("mientras") && (s1.charAt(s1.indexOf("mientras")+8) > 'z' ||
    s1.charAt(s1.indexOf("mientras")+8) < 'a')){
    style = Styles.MIENTRAS;
    s1 = s1.replace("hacer", "").trim();
    textoLineaAux = "Si";
}
if(s1.startsWith("para") && (s1.charAt(s1.indexOf("para")+4) > 'z' ||
    s1.charAt(s1.indexOf("para")+4) < 'a')){
    style = Styles.PARA;
    s1 = s1.replace("hacer", "").trim();
    textoLineaAux = "Si";
}
if(s1.startsWith("si") && !s1.startsWith("sino")){
    style = Styles.CONECTOR;
    conect++;
    conectores.add(dibujarElementos(""+conect , null, grafico,style,x+30,y,20,20,""));
    style = Styles.CONDIONAL;
    s1 = s1.substring(2).replace("entonces", "");
    textoLineaAux = "Si";
}
if(s1.startsWith("paracaso")){
    style = Styles.CONECTOR;
    conect++;
    conectores.add(dibujarElementos(""+conect , null, grafico,style,x+30,y,20,20,""));
    style = Styles.SWITCH;
    s1 = s1.replace("hacer", "").trim();
}
if(s1.startsWith("caso") || s1.startsWith("default")){
    ob=elementosSwitch.get(elementosSwitch.size()-1);
    textoLineaAux = s1.replace(":", "").trim();
}
if(s1.startsWith("sino")){
    grafico.insertEdge(grafico.getDefaultParent(), null, "", ob,
        conectores.get(conectores.size()-1));
    ob=elementosIf.get(elementosIf.size()-1);
    textoLineaAux = "No";
}
if(s1.startsWith("parar")){
    grafico.insertEdge(grafico.getDefaultParent(), null, "", ob,
        conectores.get(conectores.size()-1));
}
if(s1.startsWith("finsi")){
    grafico.insertEdge(grafico.getDefaultParent(), null, "", ob,
        conectores.get(conectores.size()-1));
    ob=conectores.get(conectores.size()-1);
    conectores.remove(conectores.size()-1);
    elementosIf.remove(elementosIf.size()-1);
}
if(s1.startsWith("finparacaso")){
    ob=conectores.get(conectores.size()-1);
}

```



```

        conectores.remove(conectores.size()-1);
        elementosSwitch.remove(elementosSwitch.size()-1);    }
    if(s1.startsWith("finmientras")){
        grafico.insertEdge(grafico.getDefaultParent(), null, "", ob,
        elementosMientras.get(elementosMientras.size()-1));
        ob = elementosMientras.get(elementosMientras.size()-1);
        elementosMientras.remove(elementosMientras.size()-1);
        textoLineaAux = "No";    }
    if(s1.startsWith("finpara") && !s1.startsWith("finparacaso")){
        grafico.insertEdge(grafico.getDefaultParent(), null, "", ob,
        elementosPara.get(elementosPara.size()-1));
        ob = elementosPara.get(elementosPara.size()-1);
        elementosPara.remove(elementosPara.size()-1);
        textoLineaAux = "No";    }
    if(style == null || style.trim().equals("")){
        style=Styles.OPERACIONES;    }
    if(!s1.startsWith("sino") && !s1.startsWith("finsi") && !s1.startsWith("finmientras") &&
    !s1.startsWith("finpara")&& !s1.startsWith("parar")&& !s1.startsWith("finparacaso")&&
    !s1.startsWith("default") && !s1.startsWith("caso")){
        ob = dibujarElementos(s1, ob, grafico,style,x,y,250,70,textoLinea);    }
    if(style.equals(Styles.CONDICIONAL)){
        elementosIf.add(ob);    }
    if(style.equals(Styles.MIENTRAS)){
        elementosMientras.add(ob);    }
    if(style.equals(Styles.PARA)){
        elementosPara.add(ob);    }
    if(style.equals(Styles.SWITCH)){
        elementosSwitch.add(ob);    }
        y = y + 90;
    }
    } finally {
        grafico.getModel().endUpdate();
    } }
public Object dibujarElementos(String valor, Object anterior, mxGraph grafico,String style,
double x, double y, double ancho,double alto,String tituloLinea){
    Object graficoActual = grafico.insertVertex(grafico.getDefaultParent(), null, valor, x, y,
    ancho,alto, style);
    if(anterior != null){
        grafico.insertEdge(grafico.getDefaultParent(), null, tituloLinea, anterior, graficoActual);
    }
    return graficoActual;
}
/**
 * @return the pseudocodigo
 */
public String getPseudocodigo() {
    return pseudocodigo;
}
/**
 * @param pseudocodigo the pseudocodigo to set
 */
public void setPseudocodigo(String pseudocodigo) {
    this.pseudocodigo = pseudocodigo;
} }

```

Tabla 7. Clase DibujarAction de PsGram

DIAGRAMA DE FLUJO A PSEUDOCODIGO

A partir del Diagrama de Flujo graficado se genera el Pseudo-Código respectivo y se ubica en el editor del PsGram, para ahí poder compilarlo, en caso de existir algún tipo de error, corregirlo. A continuación está la clase VerificarAction.java:

```

public class VerificarAction extends AbstractAction {
    private boolean solamenteRevision = true;
    public VerificarAction(boolean revisar) {
        solamenteRevision = revisar;
    }
    public static final EditorPSGram getEditor(ActionEvent e) {
        if (e.getSource() instanceof Component) {
            Component component = (Component) e.getSource();
            while (component != null
                && !(component instanceof EditorPSGram)) {
                component = component.getParent();
            }
            return (EditorPSGram) component;
        }
        return null;
    }
    //accion para pasar del diagrama al editor de pseudocodigo
    public void actionPerformed(ActionEvent e) {
        EditorPSGram editor = getEditor(e);
        if (editor != null) {
            String res = revisarGrafico(editor.getGraphComponent());
            if (res != null && !res.trim().equals("")) {
                com.psg.vista.Editor ed = new Editor();
                ed.ponLaAyuda();
                String path = ed.nuevoProyecto();
                ed.nuevo("Diagram", 0, path);
                ed.actualizationCopy(res);
                ed.setVisible(true);
            }
        }
    }
    public boolean erroresAlRevisarInicioFin(List<mxICell> elementos) {
        int inicio = 0;
        int fin = 0;
        int inicioFinEnOtrasFiguras = 0;
        int formasElipse = 0;
        for (mxICell m : elementos) {
            //Del Style obtiene solo la parte anterior del primer punto y coma, es decir solo quedaria
            (style:inicio_fin)
            String style = m.getStyle().substring(0, m.getStyle().indexOf(";"));
            if (style.equals("shape=inicio_fin")) {
                formasElipse++;
            }
            if (m.getValue() != null && m.getValue().toString().trim().equalsIgnoreCase("inicio") &&
                style.equals("shape=inicio_fin")) {
                inicio++;
            }
            if (m.getValue() != null && m.getValue().toString().trim().equalsIgnoreCase("fin") &&
                style.equals("shape=inicio_fin")) {
                fin++;
            }
            if (m.getValue() != null && (m.getValue().toString().trim().equalsIgnoreCase("inicio") ||
                m.getValue().toString().trim().equalsIgnoreCase("fin")) &&
                !style.equals("shape=inicio_fin")) {
                inicioFinEnOtrasFiguras++;
            }
        }
    }
}

```

```

if (inicio == 0 || fin == 0) {
    Mensajes.sinInicioFin();
    return true;
}
if (inicio > 1 || fin > 1) {
    Mensajes.soloUnInicioFin();
    return true;
}
if (inicioFinEnOtrasFiguras > 0) {
    Mensajes.inicioFinPalabrasReservadas();
    return true;
}
if (formasElipse > 2) {
    Mensajes.elipseSoloInicioFin();
    return true;
}
}
return false;
}
}

public boolean numeroDeConectoresCorrectos(List<mxICell> elementos) {
    int conector = 0;
    int ciclos = 0;
    for (mxICell m : elementos) {
        String style = m.getStyle().substring(0, m.getStyle().indexOf(";"));
        if (style.equals("shape=conector")) {
            conector++;
        }
        if (style.equals("shape=condicional") || style.equals("shape=switch")) {
            ciclos++;
        }
    }
    if (conector == ciclos) {
        return true;
    }
    Mensajes.revisarConectores();
    return false;
}

public boolean erroresEnConecciones(List<mxICell> elementos, mxGraphComponent gc) {
    for (mxICell m : elementos) {
        if (gc.getGraph().getConnections(m).length == 0) {
            Mensajes.elementosSuetos();
            return true;
        }
        String style = m.getStyle().substring(0, m.getStyle().indexOf(";"));
        if (style.equalsIgnoreCase("shape=switch")) {
            if (!salidasCorrectasSwitch(gc.getGraph().getOutgoingEdges(m))) {
                Mensajes.definirCaseSwitch();
                return true;
            }
        }
        if ((m.getValue() == null || m.getValue().toString().trim().isEmpty()) &&
            !style.equalsIgnoreCase("shape=conector")) {
            Mensajes.elementosVacios();
            return true;
        }
        if (m.getValue().toString().equalsIgnoreCase("inicio") &&
            gc.getGraph().getIncomingEdges(m).length > 0) {
            Mensajes.inicioConeccionesEntrada();
            return true;
        }
        if ((style.equalsIgnoreCase("shape=mientras") || style.equalsIgnoreCase("shape=para"))
            && (gc.getGraph().getIncomingEdges(m).length != 2 ||
            gc.getGraph().getOutgoingEdges(m).length != 2)) {
            Mensajes.conexionesMientras();
            return true;
        }
        //Revisa el numero de conecciones de salida de elementos FIN
        if (m.getValue().toString().equalsIgnoreCase("fin") &&
            gc.getGraph().getOutgoingEdges(m).length > 0) {
            Mensajes.finConeccionesSalida();
            return true;
        }
    }
}

```

```

//Revisa el numero de conecciones de entrada de los elementos
if ((gc.getGraph().getIncomingEdges(m).length >= 2 &&
!style.equalsIgnoreCase("shape=condicional") &&
!style.equalsIgnoreCase("shape=mientras") && !style.equalsIgnoreCase("shape=para")
&& !style.equalsIgnoreCase("shape=conector")) ||
(gc.getGraph().getIncomingEdges(m).length >= 3 &&
style.equalsIgnoreCase("shape=condicional"))) {
    Mensajes.revisarConexionesEntrada(m);
    return true;
}
//Revisa el numero de conecciones de SALIDA de los elementos
if ((gc.getGraph().getOutgoingEdges(m).length >= 2 &&
!style.equalsIgnoreCase("shape=switch") &&
!style.equalsIgnoreCase("shape=condicional") &&
!style.equalsIgnoreCase("shape=mientras") && !style.equalsIgnoreCase("shape=para"))
|| (gc.getGraph().getOutgoingEdges(m).length >= 3 &&
style.equalsIgnoreCase("shape=condicional")) ||
(gc.getGraph().getOutgoingEdges(m).length >= 3 &&
style.equalsIgnoreCase("shape=mientras")) ||
(gc.getGraph().getOutgoingEdges(m).length >= 3 &&
style.equalsIgnoreCase("shape=para"))) {
    Mensajes.revisarConexionesSalida(m);
    return true;
}
//Revisa las salidas de CONDICIONAL, MIENTRAS y PARA
if (style.equalsIgnoreCase("shape=condicional") ||
style.equalsIgnoreCase("shape=mientras") || style.equalsIgnoreCase("shape=para")) {
    if (errorEnSalidas(m, gc)) {
        return true;
    }
}
} return false;
}
public boolean errorEnSalidas(mxICell elemento, mxGraphComponent gc) {
    String style = elemento.getStyle().substring(0, elemento.getStyle().indexOf(";"));
    if (gc.getGraph().getOutgoingEdges(elemento).length != 2) {
        if (style.equalsIgnoreCase("shape=mientras")) {
            Mensajes.corregirSalidasMientras();
            return true;
        } else {
            if (style.equalsIgnoreCase("shape=para")) {
                Mensajes.corregirSalidasPara();
                return true;
            } else {
                Mensajes.corregirSalidasCondicional();
                return true;
            }
        }
    }
}
int si = 0;
int no = 0;
for (Object o : gc.getGraph().getOutgoingEdges(elemento)) {
    mxCell mc = (mxCell) o;
    if (mc.getValue() == null || (!mc.getValue().toString().trim().equalsIgnoreCase("si") &&
!mc.getValue().toString().trim().equalsIgnoreCase("no"))) {
        if (style.equalsIgnoreCase("shape=mientras")) {
            Mensajes.corregirSalidasMientras();
            return true;
        } else {
            if (style.equalsIgnoreCase("shape=para")) {
                Mensajes.corregirSalidasPara();
            }
        }
    }
}

```

```

        return true;
    } else {
        Mensajes.corregirSalidasCondicional();
        return true;
    }
}
if (mc.getValue() != null && mc.getValue().toString().trim().equalsIgnoreCase("si")) {
    si++;
}
if (mc.getValue() != null && mc.getValue().toString().trim().equalsIgnoreCase("no")) {
    no++;
}
if (si > 1 || no > 1) {
    if (style.equalsIgnoreCase("shape=mientras")) {
        Mensajes.corregirSalidasMientras();
        return true;
    } else {
        if (style.equalsIgnoreCase("shape=para")) {
            Mensajes.corregirSalidasPara();
            return true;
        } else {
            Mensajes.corregirSalidasCondicional();
            return true;
        }
    }
}
return false;
}
}

//Este metodo realiza todas las validaciones posibles sobre el grafico
public String revisarGrafico(mxGraphComponent c) {
    //obtiene todos los elementos que se han graficado excepto las lineas
    List<mxICell> elementos = encontrarElementos(c.getGraph());
    //Revisa todos los tipos de errores conciernientes a Inicio y Fin
    if (erroresAlRevisarInicioFin(elementos)) {
        return null;
    }
    //revisa que el numero de conectores sea equitativo
    if (!numeroDeConectoresCorrectos(elementos)) {
        return null;
    }
    //Revisa todas las conecciones de los elementos
    if (erroresEnConecciones(elementos, c)) {
        return null;
    }
    nuevo = true;
    elementoslf = new ArrayList<List<mxICell>>();
    mxICell inicio = encontrarInicio(elementos);
    //Se encarga de unir los elementos con sus conectores SWITCH o CONDICIONAL
    unirElementos(inicio, c.getGraph());
    if (!conectoreslfCorrectos()) {
        return null;
    }
    if(solamenteRevision) return null;
    conectores = new ArrayList<mxICell>();
    readyItems = new ArrayList<mxICell>();
    elementos = new ArrayList<mxICell>();
    celdas = new ArrayList<mxICell>();
    recorridos = new ArrayList<mxICell>();
    String dev = generarCodigo(inicio, c.getGraph());
    dev = dev.replaceAll("\n\n", "\n");
    return dev;
}

public List<mxICell> encontrarElementos(mxGraph g) {
    List<mxICell> elementos = new ArrayList<mxICell>();
    mxCell cel = ((mxCell) g.getDefaultParent());

```

```

int h = ((mxCell) g.getDefaultParent()).getChildCount();
for (int i = 0; i < h; i++) {
    if (cel.getChildAt(i).isVertex()) {
        elementos.add(cel.getChildAt(i));
    }
}
return elementos;
}
}

public List<mxICell> encontrarAristas(mxGraph g) {
    List<mxICell> aristas = new ArrayList<mxICell>();
    mxCell cel = ((mxCell) g.getDefaultParent());
    int h = ((mxCell) g.getDefaultParent()).getChildCount();
    for (int i = 0; i < h; i++) {
        if (cel.getChildAt(i).isEdge()) {
            aristas.add(cel.getChildAt(i));
        }
    }
    return aristas;
}

public static boolean nuevo = true;
public boolean salidasCorrectasSwitch(Object[] salidas) {
    boolean res = true;
    for (Object o : salidas) {
        mxCell mx = (mxCell) o;
        if (mx.getValue() == null || mx.getValue().toString().trim().equals("")) {
            return false;
        }
    }
    return res;
}

public static List<mxICell> ordenarSalidasCondicional(Object[] salidas, boolean esCondicional)
{
    ArrayList<mxICell> ordenados = new ArrayList<mxICell>();
    if (esCondicional) {
        mxCell m1 = (mxCell) salidas[0];
        mxCell m2 = (mxCell) salidas[1];
        if (m1.getValue().toString().equalsIgnoreCase("si")) {
            ordenados.add(m1.getTarget());
            ordenados.add(m2.getTarget());
        } else {
            ordenados.add(m2.getTarget());
            ordenados.add(m1.getTarget());
        }
        return ordenados;
    } else {
        for (Object o : salidas) {
            mxCell mx = (mxCell) o;
            ordenados.add(mx.getTarget());
        }
        return ordenados;
    }
}

List<List<mxICell>> elementosIf;
public boolean containsCell(mxICell c) {
    if (elementosIf == null) {
        elementosIf = new ArrayList<List<mxICell>>();
    }
    for (List<mxICell> l : elementosIf) {
        if (l.get(0).equals(c)) {
            return true;
        }
    }
    return false;
}

public boolean addInElemento(mxICell m) {
    String style = m.getStyle().substring(0, m.getStyle().indexOf(";"));
}

```

```

if ((style.equalsIgnoreCase("shape=condicional") || style.equalsIgnoreCase("shape=switch"))
&& !containsCell(m)) {
    List<mxICell> aux = new ArrayList<mxICell>();
    aux.add(m);
    elementosf.add(aux);
    return true;
}
if (style.equalsIgnoreCase("shape=conector")) {
    if (elementosf.size() == 0) {
        return false;
    }
    for (int i = elementosf.size() - 1; i >= 0; i--) {
        if (elementosf.get(i).size() == 1) {
            elementosf.get(i).add(m);
            return true;
        }
    }
    return false;
}
return false;
}
public mxICell obtenerConector(mxICell m) {
    for (List<mxICell> l : elementosf) {
        if (l.get(0).equals(m)) {
            return l.get(1);
        }
    }
    return null;
}
//Indica si cada CONDICIONAL tiene su CONECTOR correspondiente
public boolean conectoresfCorrectos() {
    if (elementosf == null) {
        return true;
    }
    for (List<mxICell> l : elementosf) {
        if (l.size() <= 1) {
            Mensajes.revisarConectores();
            return false;
        }
    }
    return true;
}
List<mxICell> conectores;
List<mxICell> recorridos;
List<mxICell> celdas = new ArrayList<mxICell>();
public String generarCodigo(mxICell elemento, mxGraph g) {
    String res = "";
    if (celdas.contains(elemento)) {
        return res;
    }
    celdas.add(elemento);
    //Guarda las salidas del elemento
    Object[] lista = g.getOutgoingEdges(elemento);
    String style = elemento.getStyle().substring(0, elemento.getStyle().indexOf(";"));
    if (style.equalsIgnoreCase("shape=condicional")) {
        //Devuelve la lista de salidas del condicional, en la posicion 0 ubica la salida que corresponde al
        Si y en la posicion 1 ubica la salida que corresponde al No
        List<mxICell> conexiones = ordenarSalidasCondicional(g.getOutgoingEdges(elemento),
            true);
        //Obtengo el conector que corresponde al condicional actual
        mxICell conector = obtenerConector(elemento);
        mxICell si = (mxCell) conexiones.get(0);
        mxICell no = (mxCell) conexiones.get(1);
        conectores.add(conector);
    }
}

```

```

if(elemento.getValue().toString().startsWith("si")){
    res = res + elemento.getValue() + " entonces \n" + generarCodigo(si, g) + "\n";
}else{
    res = res + "si(" + elemento.getValue() + ") entonces \n" + generarCodigo(si, g) + "\n";
}
res = res + "sino \n" + generarCodigo(no, g) + "\n fin" + "\n";
//Obtenemos las salidas del conector
Object[] lista2 = g.getOutgoingEdges(conector);
mxCell mc = ((mxCell) lista2[0]).getTarget();
res = res + generarCodigo(mc, g);
} else {
if (style.equalsIgnoreCase("shape=conector") && conectores.size() > 0 &&
elemento.equals(conectores.get(conectores.size() - 1))) {
    conectores.remove(conectores.size() - 1);
    return res;
} else {
if (style.equalsIgnoreCase("shape=mientras")) {
    List<mxCell> conexiones =
ordenarSalidasCondicionales(g.getOutgoingEdges(elemento), true);
mxCell si = (mxCell) conexiones.get(0);
mxCell no = (mxCell) conexiones.get(1);
if(elemento.getValue().toString().toLowerCase().startsWith("mientras")){
    res = res + elemento.getValue().toString().toLowerCase() + " hacer \n" +
generarCodigo(si, g) + "\n";
}else{
    res = res + "mientras (" + elemento.getValue() + ") hacer \n" + generarCodigo(si,
g) + "\n";
}
res = res + "finmientras \n" + generarCodigo(no, g) + "\n";
} else {
if (style.equalsIgnoreCase("shape=para")) {
    List<mxCell> conexiones =
ordenarSalidasCondicionales(g.getOutgoingEdges(elemento), true);
mxCell si = (mxCell) conexiones.get(0);
mxCell no = (mxCell) conexiones.get(1);
if(elemento.getValue().toString().toLowerCase().startsWith("para")){
    res = res + elemento.getValue().toString().toLowerCase() + " hacer \n" +
generarCodigo(si, g) + "\n";
}else{
    res = res + "para (" + elemento.getValue() + ") hacer \n" + generarCodigo(si, g)
+ "\n";
}
res = res + "finpara \n" + generarCodigo(no, g) + "\n";
} else {
if (style.equalsIgnoreCase("shape=switch")) {
    List<mxCell> conexiones =
ordenarSalidasCondicionales(g.getOutgoingEdges(elemento), false);
mxCell conector = obtenerConector(elemento);
conectores.add(conector);
if(elemento.getValue().toString().startsWith("paracaso")){
    res = res + elemento.getValue().toString().toLowerCase() + " hacer\n";
}else{
    res = res + "paracaso(" + elemento.getValue() + ")hacer\n";
}
for (mxCell mxi : conexiones) {
    Object[] entradas = g.getIncomingEdges(mxi);
    mxCell mx = (mxCell) entradas[0];
    res = res + mx.getValue() + ": \n" + generarCodigo(mxi, g) + "\nparar\n";
}
Object[] lista2 = g.getOutgoingEdges(conector);

```


1.5. PRUEBAS

OBJETIVO 5: Realizar pruebas de la aplicación realizada.


1.5.1. PRUEBA DE VALIDACIÓN

El proceso de pruebas de la aplicación de PsGram, durante o al final del proceso de desarrollo, es muy importante, porque, permite determinar si satisface los requisitos del análisis, si el diseño es amigable y entendible para el usuario que en este caso serán los alumnos de la carrera de Ingeniería en Sistemas.

Para demostrar el rendimiento de la aplicación se realizó una encuesta a los alumnos, la cuál permitió descubrir errores existentes y realizar las correcciones respectivas. A continuación se presenta la encuesta realizada a los estudiantes, el análisis de los resultados obtenidos y un informe de las pruebas de validación.



HERRAMIENTA PARA VALIDACIÓN DEL SOFTWARE



UNIVERSIDAD NACIONAL DE LOJA
Área de la Energía, las Industrias y los Recursos Naturales No Renovables
Carrera de Ingeniería en Sistemas
CUESTIONARIO PARA VALIDACIÓN SE SOFTWARE

1. ¿PsGram es amigable con el usuario?
Si () No ()
¿Por qué?.....
.....
2. ¿Para realizar fácilmente sus actividades académicas, es ventajosa la interfaz de PsGram?
Si () No ()
¿Por qué?.....
.....
3. ¿Para desarrollar éstas actividades, es suficiente la información presentada en PsGram?
Si () No ()
¿Por qué?.....
.....
4. ¿Es rápido, seguro y confiable el almacenamiento de información de PsGram?



Si () No ()
¿Por qué?.....
.....

5. ¿Al utilizar PsGram se le presento cierto inconveniente?
Si () No ()
¿Por qué?.....
.....

6. ¿El tiempo de ejecución de PsGram es razonable?
Si () No ()
¿Por qué?.....
.....

7. ¿PsGram es rápido, eficiente y confiable en el proceso de corrida de Pseudo-Códigos y detección de errores?
Si () No ()
¿Por qué?.....
.....

8. ¿PsGram es rápido, eficiente y confiable en el proceso de corrida de Diagrama de Flujos y detección de errores?
Si () No ()
¿Por qué?.....
.....

9. ¿Cree usted que con PsGram podrá practicar y mejorar los conocimientos adquiridos en el aula?
Si () No ()
¿Por qué?.....
.....

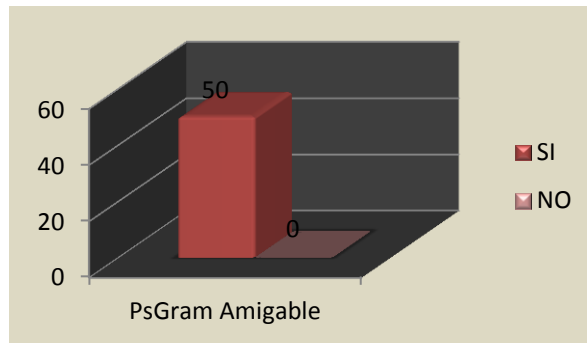
10. ¿Tiene usted alguna recomendación para el software?
Si () No ()
En caso de ser positivo ¿Cuál es su recomendación?.....
.....
.....
.....

Gracias Por Su Colaboración

Tabla 9 . Modelo de Encuesta para pruebas de PsGram

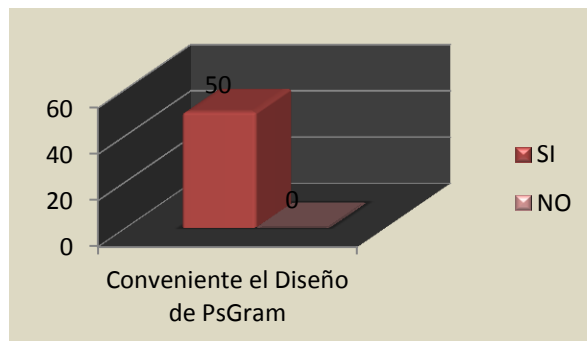
ANÁLISIS DE RESULTADOS DE VALIDACIÓN

1. ¿PsGram es amigable con el usuario?



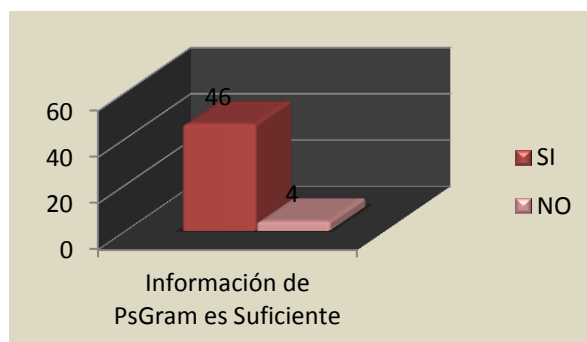
11. Ilustración: PsGram Amigable

2. ¿Para realizar fácilmente sus actividades académicas, es ventajosa la interfaz de PsGram?



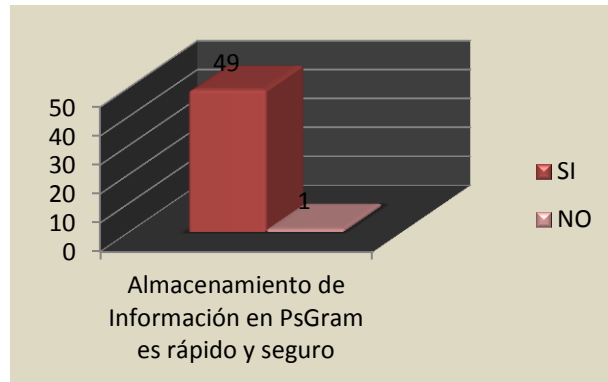
12. Ilustración: Conveniente el Diseño de Psgram

3. ¿Para desarrollar éstas actividades, es suficiente la información presentada en PsGram?



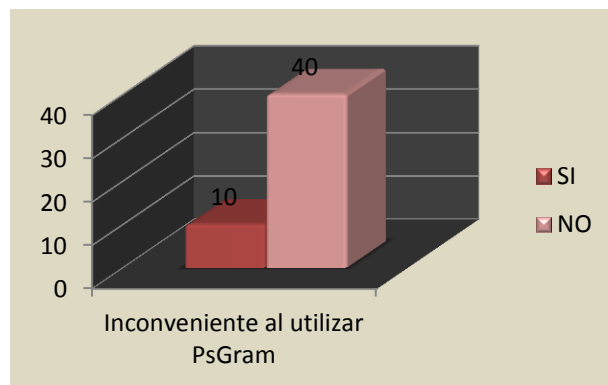
13. Ilustración: Información de PsGram es Suficiente

4. ¿Es rápido, seguro y confiable el almacenamiento de información de PsGram?



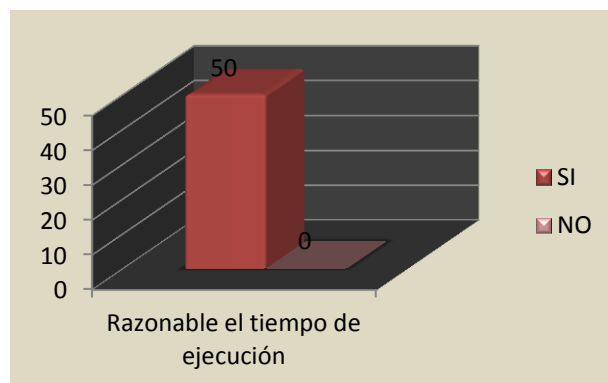
14. Ilustración: Almacenamiento de Información en PsGram es rápido y seguro.

5. ¿Al utilizar PsGram se le presento cierto inconveniente?



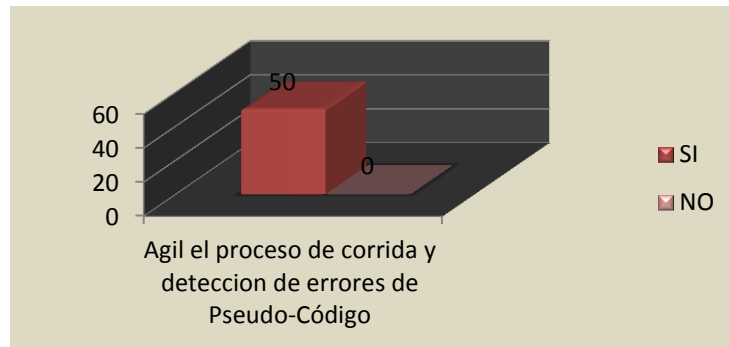
15. Ilustración: Inconveientes al utilizar PsGram.

6. ¿El tiempo de ejecución de PsGram es razonable?



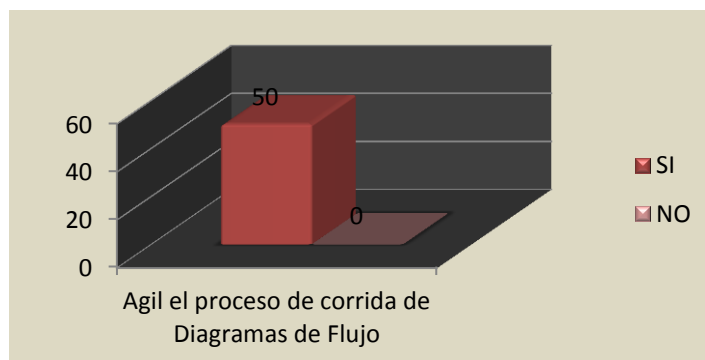
16. Ilustración: Razonable el tiempo de ejecución.

7. ¿PsGram es rápido, eficiente y confiable en el proceso de corrida de Pseudo-Códigos y detección de errores?



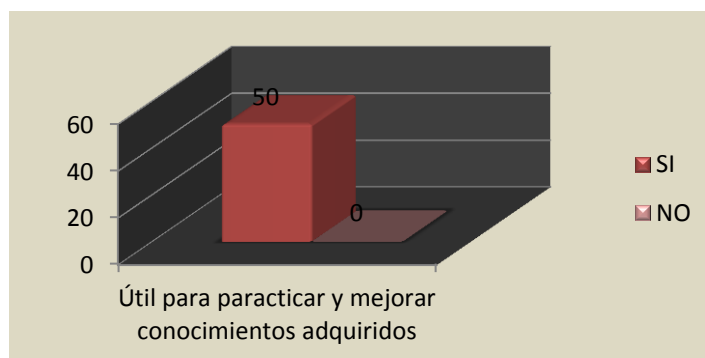
17. Ilustración: Agil el proceso de corrida de Pseudo-Código y detección de errores.

8. ¿PsGram es rápido, eficiente y confiable en el proceso de corrida de Diagrama de Flujos?



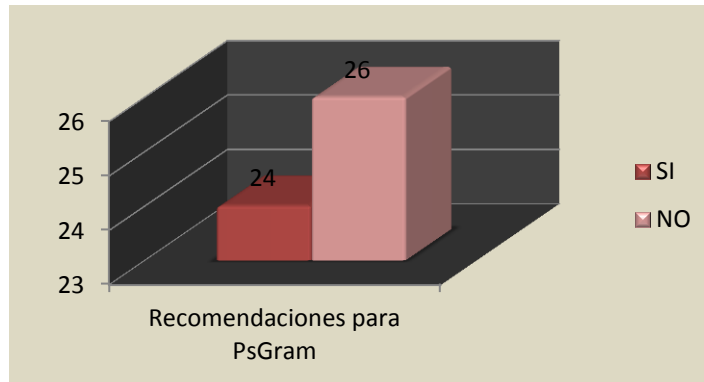
18. Ilustración: Agil el proceso de corrida de Diagramas de Flujo.

9. ¿Cree usted que con PsGram podrá practicar y mejorar los conocimientos adquiridos en el aula?



19. Ilustración: Util para practicar y mejorar conociemitos adquiridos.

10. ¿Tiene usted alguna recomendación para el software?



20. Ilustración: Recomendaciones para PsGram.

INFORME DE RESULTADOS DE PRUEBAS DE VALIDACIÓN

Las pruebas de validación se las realizó bajo la supervisión de la Ingeniera Mireya Erreyes, quién aprobó el software PsGram y es docente de los alumnos de cuarto módulo paralelo “A” y “B” de la unidad de Metodología de la Programación y Estructura de Datos de la carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja.

Informe de: Capacitación, funcionamiento de PsGram.

Fecha: 11 de Junio de 2012.

IDENTIFICADOR	FUNCIONAMIENTO DE PsGram
RESUMEN	Las pruebas se las realizó bajo la supervisión de la Ingeniera Mireya Erreyes, docente de las unidades de Metodología de la Programación y Estructura de Datos, y con la participación total de 50 alumnos que pertenecen al Cuarto módulo paralelo “A” y paralelo “B”; el día Lunes 11 de Junio de 2012 de 08H00 a 09H00 de la mañana se capacitó y realizó las pruebas a 27 alumnos que pertenecían al paralelo “A” y el mismo día a las 09H45 a 10H45 de la mañana capacite y realice las pruebas a 23 alumnos que pertenecían al paralelo “B”.
VARIACIONES	El día Lunes antes de realizar las pruebas se hizo una

	<p>capacitación sobre el funcionamiento de PsGram.</p>
<p>RESUMEN DE LOS RESULTADOS</p>	<p>El día Viernes 08 de Junio de 2012 a las 08H00 de la mañana, junto con la Ingeniera Mireya Erreyes se procedió a entregarles a los estudiantes dos cd's que contenían el instalador de PsGram y se les indicó a los alumnos algunas instrucciones para poder instalar, cómo el software es amigable y tiene un cierto parecido un módulo del PsGram a la aplicación PSeInt que ellos utilizan en la Metodología de Programación, los alumnos todo el fin de semana realizaron pruebas al software para ver si funcionaba correctamente o no.</p> <p>El día Lunes en cada paralelo se hizo una capacitación, debido a que PsGram además de convertir de Pseudo-Código a Diagrama de Flujo (los alumnos tienen conocimiento de cómo manejar esta parte de PsGram, porque la aplicación PSeInt solo hace eso) transforma de Diagrama de Flujo a Pseudo-Código, ésta última parte es algo novedoso para los alumnos por ello se les indicó las funcionalidades y que nomás hace PsGram.</p> <p>Durante la capacitación se realizó algunos ejemplos prácticos con el proyector para que todos observaran y puedan comprobar su funcionamiento, de la misma manera, los alumnos practicaban en sus portátiles haciendo las demostraciones y los ejemplos de cada uno de ellos.</p> <p>Seguidamente, una vez contestada las inquietudes y escuchado las sugerencias de la Ingeniera y de los alumnos, se procedió a realizar una encuesta a cada uno de los estudiantes para así poder terminar con las pruebas de PsGram.</p>
<p>APROBACIÓN</p>	<p>Luego de hacer las pruebas correspondientes, la Ingeniera Mireya Erreyes aprobó el software PsGram y se obtuvo una total aceptación por parte de los estudiantes, los cuales</p>

	<p>piensan que a PsGram deberían de ponerla de herramienta de apoyo para los futuros alumnos que van a ingresar a Cuarto Módulo para mayores conocimientos.</p>
--	---

Tabla 10 . Informe de resultados de Pruebas de Validacion.

Dado los resultados obtenidos de las pruebas de validación, no hubo fallos ni incumplimiento de requerimientos, no se tuvo mayores sugerencias que sean imposibles de agregarlas a PsGram, debido a esto la fase de pruebas queda concluida y PsGram se da por aceptado (ANEXO C).

2. VALORACIÓN TÉCNICA ECONÓMICA AMBIENTAL

El sistema fue desarrollado de manera satisfactoria por lo que se contó con todos los recursos necesarios como: humanos, materiales, técnicos, tecnológicos. Todas las herramientas utilizadas para el desarrollo de PsGram son de libre distribución, están disponibles para cualquier persona en su respectiva página de internet. En el ámbito económico no existió problemas porque todos los recursos materiales fueron adquiridos por la Tesista.

Finalmente se puede decir que la construcción y ejecución del presente software culminó de manera exitosa, contribuyendo al enriquecimiento académico de los estudiantes que hagan uso de la misma, facilitando la elaboración de programas básicos. A continuación se detalla los recursos utilizados para el desarrollo de PsGram:

DESCRIPCIÓN	CANTIDAD			COSTO	COSTO TOTAL
	Unitario	Horas c/u	Hojas c/u		
RECURSOS HUMANOS					
Director de Tesis	-	-	-	-	-
Desarrolladores	1	1200		\$3.00	\$3600.00
TOTAL					\$3600.00
RECURSOS MATERIALES.					
Resma de Papel.	5	-	-	\$3.30	\$16.50
Cartuchos de tinta negra.	2	-	-	\$20.00	\$40.00
Cartucho de tinta a color.	1	-	-	\$22.00	\$22.00
Kit de recarga de cartuchos.	2	-	-	\$7.00	\$14.00
Internet	1	100	-	\$1.00	\$100.00
Flash Memory (1GB) Kingston.	1	-	-	\$7.00	\$7.00
CD	6	-	-	0.70	\$4.20
TOTAL					\$203.70
RECURSOS TÉCNICOS					
Computadores	1	1200	-	0.50	\$600.00

Impresora	1	-	2500	0.05	\$125.00
Alquiler de Proyector	1	2	-	10.00	\$20,00
TOTAL					\$745.00
RECURSOS TECNOLÓGICOS.					
Software Libre	1	-	-	Gratuito	\$0.00
My SQL 5.0	1	-	-	Gratuito	\$0.00
Enterprise Architect 3.6	1	-	-	Gratuito	\$0.00
Open Office 3.2	1	-	-	Gratuito	\$0.00
TOTAL					\$0.00

Tabla 11 . Valoración técnica económica ambiental.

RESUMEN	COSTO TOTAL
Recursos Humanos	\$3600.00
Recursos Materiales	\$203.70
Recursos Técnicos	\$745.00
Recursos Tecnológicos	\$0.00
SUBTOTAL	\$4548.70
Imprevistos 10 %	\$454.87
TOTAL	\$5003.57

Tabla 12 . Resumen valoración técnica económica ambiental.

G. DISCUSIÓN

1. METODOLOGÍA ICONIX

1.1. REQUERIMIENTOS

1.1.1. Panorama General

Este proyecto tiene por objeto desarrollar una herramienta que permita graficar Diagramas de Flujo generando su respectivo Pseudocódigo y que a partir de un Pseudocódigo genere su Diagrama de Flujo, para los alumnos de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja.

1.1.2. Metas

La meta es:

- Generar rápidamente el Pseudocódigo a partir de un Diagrama de flujo.
- Graficar rápidamente un Diagrama de Flujo a partir de un Pseudocódigo.
- Análisis inmediato de errores existentes en el Pseudocódigo o Diagrama de Flujo ingresado por el usuario.

1.1.3. Funciones del Sistema

El sistema permitirá:

1. Al usuario ingresar Pseudocódigo (tipos de datos, operadores, sentencias), que sigan las reglas del Pseudocódigo. Evidente
2. Realizar un análisis léxico del Pseudocódigo ingresado por el usuario. Oculto
3. Al usuario visualizar los errores léxicos encontrados en el Pseudocódigo ingresado por el usuario. Evidente
4. Realizar un análisis sintáctico del Pseudocódigo ingresado por el usuario. Oculto
5. Al usuario visualizar los errores sintácticos encontrados en el Pseudocódigo ingresado por el usuario. Evidente
6. Realizar un análisis semántico del Pseudocódigo ingresado por el usuario. Oculto

7. Al usuario visualizar los errores semánticos encontrados en el Pseudocódigo ingresado por el usuario. Evidente
8. Al usuario visualizar el diagrama de flujo generado a partir del Pseudocódigo ingresado. Evidente
9. Al usuario ingresar formas gráficas siguiendo un esquema de un diagrama de flujo. Evidente
10. Realizar un análisis de errores del diagrama ingresado de acuerdo a la estructura de un diagrama de flujo.
11. Al usuario visualizar errores del diagrama ingresado. Evidente
12. Visualizar el Pseudocódigo generado a partir del diagrama ingresado por el usuario. Evidente.

El sistema permitirá:

CODIGO	DESCRIPCION	CATEGORIA	VALIDADO
RF001	Al usuario ingresar Pseudocódigo (tipos de datos, operadores, sentencias), que sigan las reglas del Pseudo-Código.	Evidente	X
RF002	Realizar un análisis léxico del Pseudo-Código ingresado por el usuario.	Oculto	X
RF003	Al usuario visualizar los errores léxicos encontrados en el Pseudo-Código ingresado por el usuario.	Evidente	X
RF004	Realizar un análisis sintáctico del Pseudo-Código ingresado por el usuario.	Oculto	X
RF005	Al usuario visualizar los errores sintácticos encontrados en el Pseudo-Código ingresado por el usuario.	Evidente	X
RF006	Realizar un análisis semántico del Pseudo-Código ingresado por el usuario.	Oculto	X
RF007	Al usuario visualizar los errores semánticos encontrados en Pseudo-Código ingresado por el usuario.	Evidente	X

RF008	Al usuario visualizar el diagrama de flujo generado a partir del Pseudo-Código ingresado.	Evidente	X
RF009	Al usuario ingresar formas gráficas siguiendo un esquema de un diagrama de flujo.	Evidente	X
RF010	Realizar un análisis de errores del diagrama ingresado de acuerdo a la estructura de un diagrama de flujo.	Oculto	X
RF011	Al usuario visualizar errores del diagrama ingresado.	Evidente	X
RF012	Visualizar el Pseudo-Código generado a partir del diagrama ingresado por el usuario.	Evidente.	X

Tabla 13 . Requerimientos Funcionales.

1.1.4. Atributos del Sistema

CODIGO	DESCRIPCION	VALIDADO
RNF001	El sistema será desarrollado bajo la plataforma java.	X
RNF002	El sistema será desarrollado en el IDE NETBEANS	X
RNF003	El sistema contará con una interfaz de usuario fácil de utilizar.	X
RNF004	El sistema se ejecutará en múltiples plataformas.	X
RNF005	El sistema tendrá ayudas en pantallas.	X

Tabla 14 . Requerimientos No Funcionales.

1.2. MODELO DEL DOMINIO

1.2.1. GLOSARIO DE TÉRMINOS

El glosario de términos es:

Usuario: La persona que va a hacer uso del sistema.

Pseudo-Código: Se describe algoritmos utilizando una mezcla de lenguaje común, con instrucciones de programación, palabras claves, etc.

Análisis Lexico: Identifica lexemas válidos y genera tokens.

Análisis Sintáctico: Verifica que se cumplan las reglas establecidas em PsGram.

Análisis Semántico: Verifica que el Pseudo-Código tenga sentido.

Diagrama de Flujo: Es la representación gráfica del algoritmo o proceso.

Ayuda: Ofrece ayuda al usuario.

1.2.2. MODELO CONCEPTUAL DEL DOMINIO

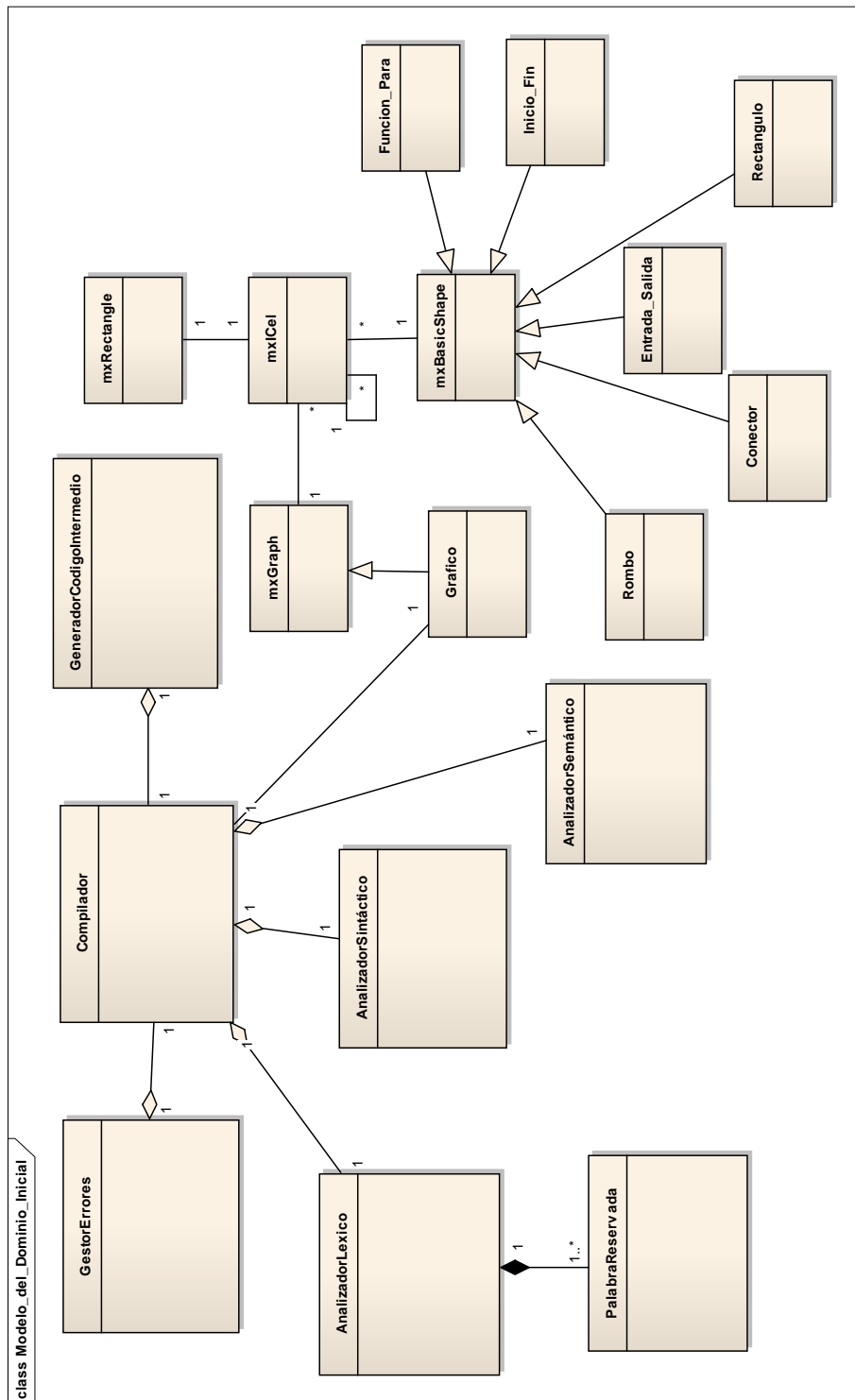


Diagrama 1 . Modelo Inicial del Dominio

1.3. MODELO DE CASOS DE USO

1.3.1. DETERMINACIÓN DE CASOS DE USO

ACTORES	META	CASO DE USO
USUARIO	Ingresar Pseudocódigo.	Ingresar Pseudo-Código
	Realizar un análisis léxico del Pseudo-Código.	Analizar Lexicamente
	Visualizar los errores léxicos encontrados en el Pseudo-Código.	Gestionar Errores
	Realizar un análisis sintáctico del Pseudo-Código.	Analizador Sintactico
	Visualizar los errores sintácticos encontrados en el Pseudo-Código.	Gestionar Errores
	Realizar un análisis semántico del Pseudo-Código.	Analizador Semantico
	Visualizar los errores semánticos encontrados en el Pseudo-Código.	Gestionar Errores
	Visualizar el diagrama de flujo generado a partir del Pseudo-Código.	Crear Diagrama
	Ingresar formas gráficas siguiendo un esquema de un diagrama de flujo.	Dibujar Diagrama
	Realizar un análisis de errores del diagrama ingresado de acuerdo a la estructura de un diagrama de flujo.	Revisar Errores
	Visualizar errores del diagrama ingresado.	Revisar Errores
	Visualizar el Pseudo-Código generado a partir del Diagrama de Flujo.	Generar Pseudo-Codigo

Tabla 15. Determinación de Casos de Uso

1.3.2. DIAGRAMA DE CASOS DE USO

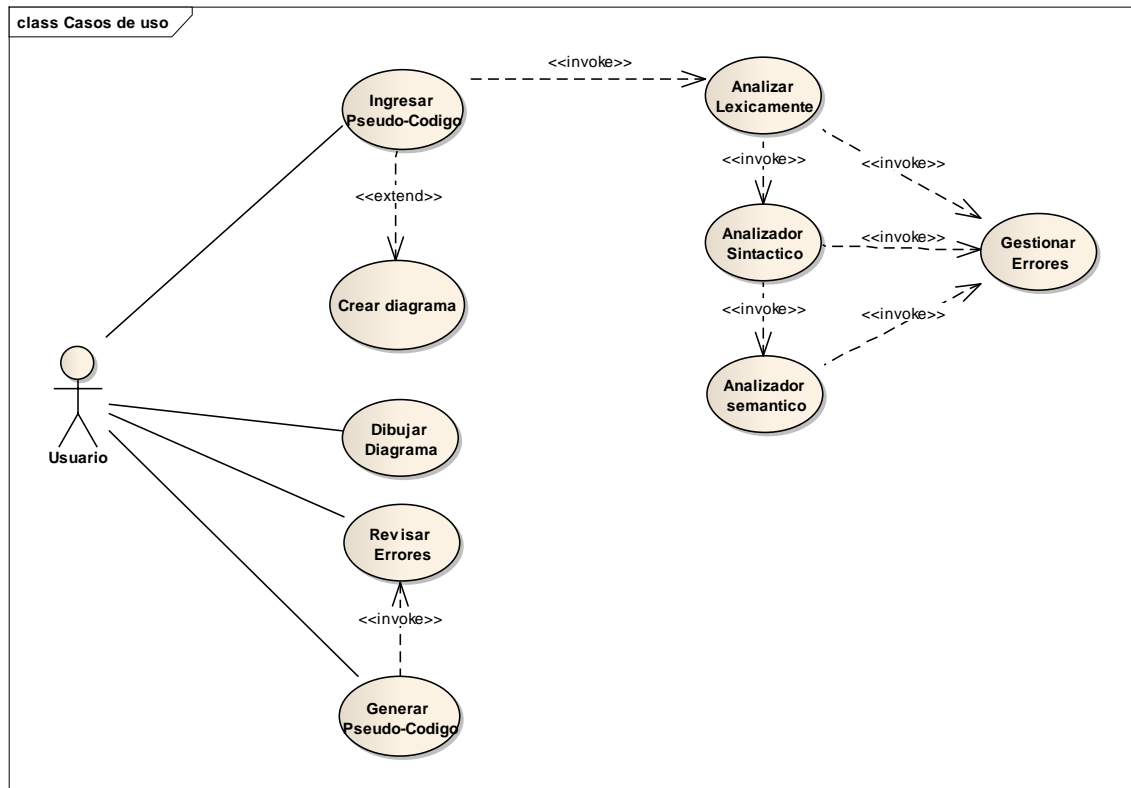


Diagrama 2 . Diagrama de Casos de Uso

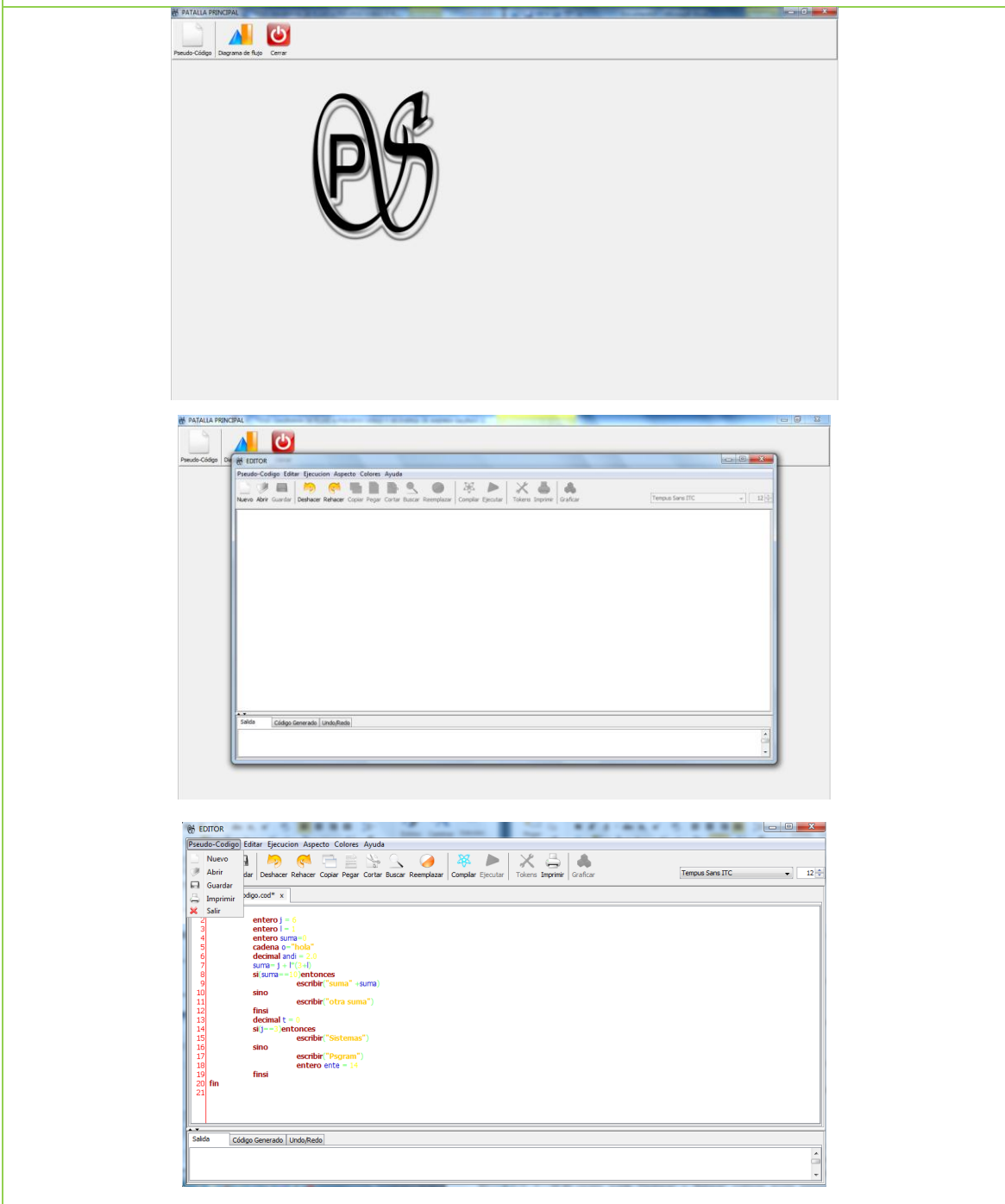
1.4. PROTOTIPOS DE PANTALLA, DESCRIPCION DE CASOS DE USO, DIAGRAMA DE SECUENCIA

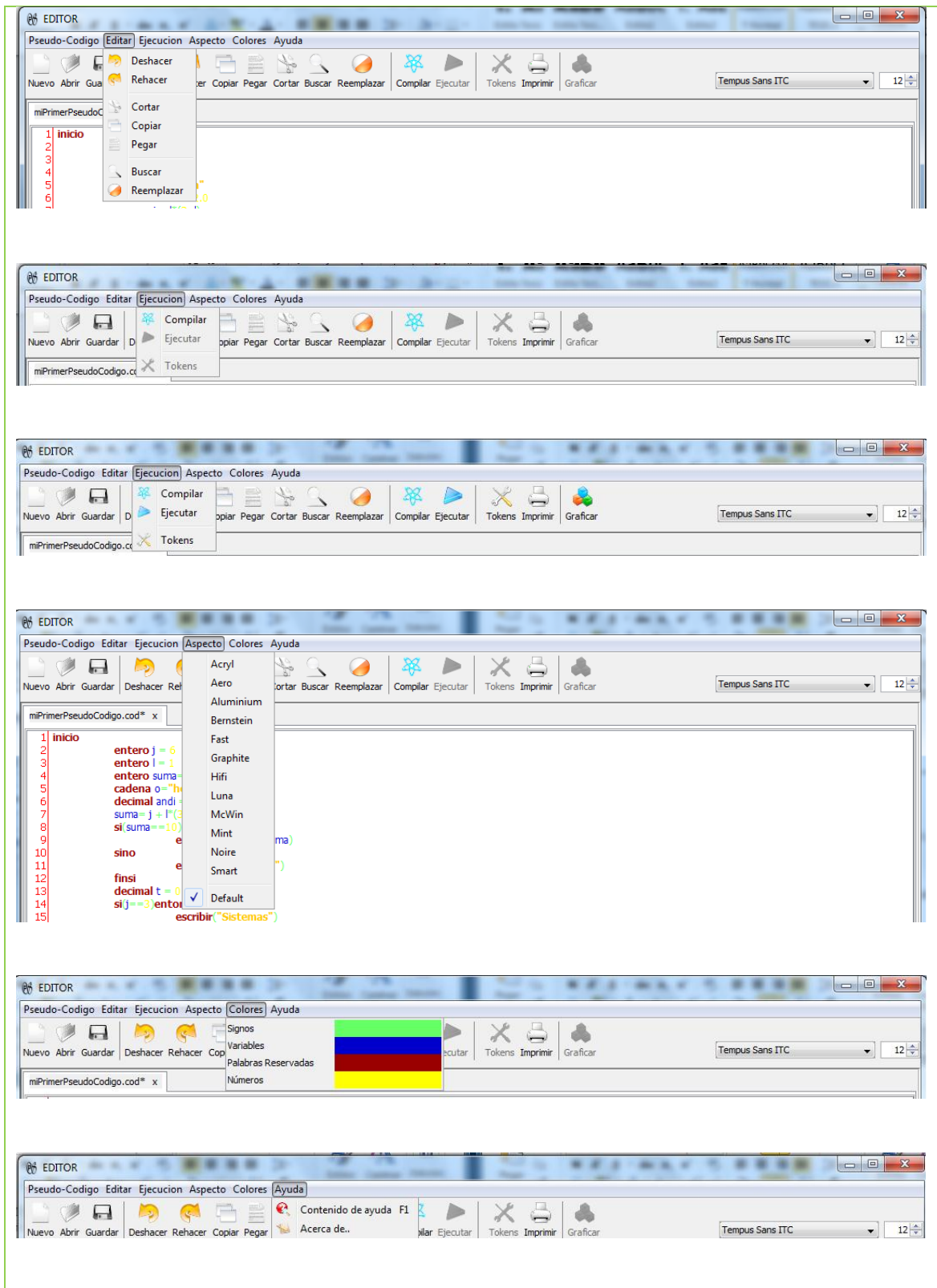
1.4.1. CASO DE USO: Ingresar Pseudo-Código

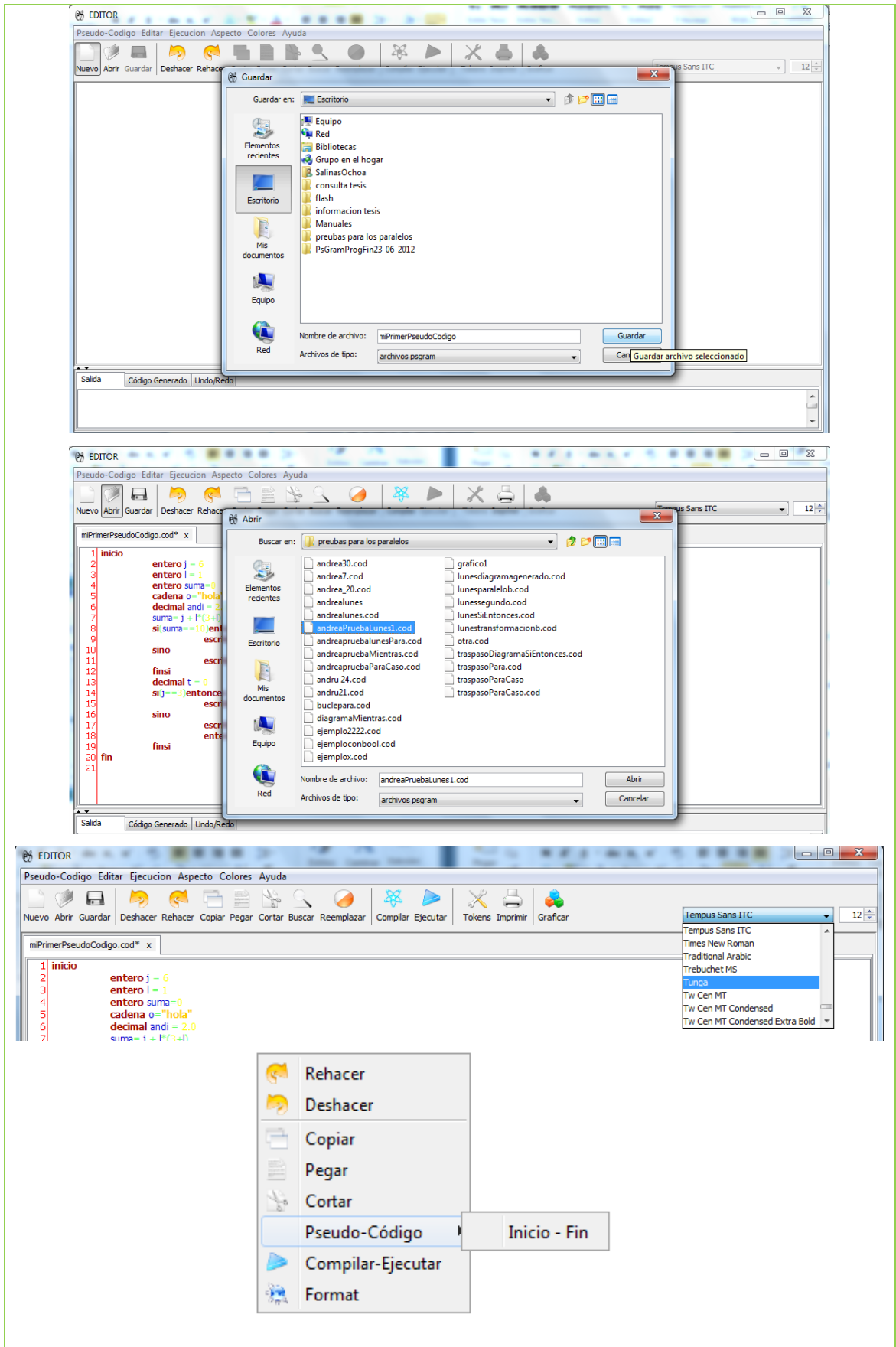
Nombre Pantalla: PANTALLA PRINCIPAL, EDITOR	Código: PP001
---	---------------

Tipo Interfaz Gráfica: JFrame

Caso de Uso: Ingresar Pseudo-Código







The image displays the PsGram IDE interface with a code editor and several dialog boxes. The code editor shows a pseudo-code program for calculating the sum of numbers from 1 to 6. The code is as follows:

```

1 inicio
2 entero i = 6
3 entero i = 1
4 entero suma = 0
5 cadena o = "hola"
6 suma = i + (i-1)
7 si suma = 10 hacer
8     escribir("suma")
9 sino escribir("otra suma")
10 fin
11 decimal t = 0
12 si j = 3 hacer
13     escribir("Sistemas")
14 sino escribir("PsGram")
15 fin
16 entero ente = 14
17 fin
18 fin
19 fin
20 fin
  
```

The dialog boxes shown are:

- Buscar:** A search dialog with the text "Buscar: suma" and buttons "BUSCAR" and "CERRAR".
- REEMPLAZAR:** A replace dialog with "Palabra actual: hola" and "Reemplazar por: adioj", with buttons "ACEPTAR" and "CERRAR".
- Mensaje:** A message dialog with the text "No hay coincidencias" and an "Aceptar" button.
- PSGRAM:** A dialog titled "PSGRAM" with the text "ingrese el numero" and a text input field containing "6", with "Aceptar" and "Cancelar" buttons.
- PSGRAM:** A dialog titled "PSGRAM" with the text "Campos vacios" and an "Aceptar" button.
- PSGRAM:** A dialog titled "PSGRAM" with the text "No se han hecho reemplazos" and an "Aceptar" button.
- Mensaje:** A message dialog with the text "ERROR AL COMPILAR: Error léxico encontrado Caracter no válido" and an "Aceptar" button.
- GUARDAR:** A dialog titled "GUARDAR" with the text "Desea guardar los cambios" and buttons "Sí", "No", and "Cancelar".
- GUARDAR:** A dialog titled "GUARDAR" with the text "Desea guardar los cambios a miPrimerPseudoCodigo.cod ?" and buttons "Sí", "No", and "Cancelar".
- SALIR:** A dialog titled "SALIR" with the text "Salir del sistema?" and buttons "Sí" and "No".

Realizado por:
 Andrea Natasha Salinas Ochoa.

Fecha:
 09/06/2011

Tabla 16 . Prototipo de Pantalla del Caso de Uso: Ingresar Pseudo-Código.

Nombre C.U: Ingresar Pseudo-Codigo		Código C.U: 0001
Req. Funcional:	RF001	
Objetivo(s):	Ingresar las sentencias que sigan las reglas del Pseudo-Código.	
Descripción:	<p>El usuario podrá ingresar el Pseudo-Código que desee ejecutar, siempre que éste se encuentre enmarcado dentro de las reglas establecidas, tanto de sintaxis como de semántica.</p> <p>El editor permite guardar, abrir y modificar el bloque de código, además de algunas opciones de edición como cortar, copiar, pegar, deshacer y rehacer.</p>	
Actor(s):	Usuario	
Tipo Caso Uso:	Sistema	
Pre-condiciones:	<p>El usuario haya instalado PsGram.</p> <p>El usuario haya ejecutado la aplicación.</p>	
Post-condiciones:	El usuario ingresará el Pseudo-Código a ejecutar o abrirá uno previamente guardado.	

Flujo Normal de Eventos:

1. En la barra de herramientas de la ventana "PANTALLA PRINCIPAL" el usuario selecciona la opción "Pseudo-Código".
2. El sistema muestra la ventana "EDITOR".
3. El usuario selecciona el botón "Nuevo" de la barra de herramientas o la opción "Nuevo" del menú "Pseudo-Codigo".
4. El sistema muestra el cuadro de diálogo "Guardar".
5. El usuario ingresa en nombre del archivo y selecciona la ubicación donde va almacenar.
6. El usuario presiona el botón "Guardar".
7. El sistema guarda el archivo en la dirección especificada.
8. El usuario ingresará el Pseudo-Código que desee ejecutar.

Opciones de Edición

9. El usuario puede "Copiar" [Ctrl+c], "Cortar" [Ctrl+x], "Pegar" [Ctrl+v], cualquier Pseudo-Código o cualquier parte de éste; presionando las opciones "Copiar", "Cortar", "Pegar", del menú "Editar" o de los botones de la barra de herramientas, como también con el teclado haciendo clic en los comandos mencionados.
10. El usuario puede "Deshacer" y "Rehacer" cualquier acción presionando las

opciones “Deshacer”, “Rehacer”, del menú “Editar” o de los botones de la barra de herramientas

11. El usuario puede dar “Formato” al Pseudo-Código ingresado, haciendo clic derecho sobre el sector donde se ingresa el Pseudo-Código y presionando la opción “Format”.
12. El usuario puede “Buscar” una palabra presionando las opciones “Buscar”, del menú “Editar” o del botón de la barra de herramientas.
13. El sistema muestra la ventana “Buscar”.
14. El usuario ingresa la palabra que se desee buscar en el campo “Buscar”.
15. El usuario presiona el botón “Buscar”.
16. El sistema en el Pseudo-Código, resalta la palabra buscada.
17. El usuario presiona nuevamente el botón “Buscar”.
18. El sistema en el Pseudo-Código, resalta la siguiente palabra buscada.
19. El usuario puede “Reemplazar” una palabra presionando las opciones “Reemplazar”, del menú “Editar” o del botón de la barra de herramientas.
20. El sistema muestra la ventana “Reemplazar”.
21. El usuario en el campo “Palabra actual”, ingresa la palabra que se desee reemplazar.
22. El usuario en en el campo “Reemplazar por”, ingresa la(s) palabra(s) que va o van a reemplazar a la anterior.
23. El usuario hace clic en “Aceptar”.
24. El sistema reemplaza la palabra.
25. El usuario hace clic derecho en el campo donde se escribe el Pseudo-Código, se despliega un menú, se selecciona la opción “Pseudo-Código” y seguidamente se hace clic en la opción “Inicio - Fin”.
26. El sistema presenta las palabras “Inicio” y “Fin” en el campo donde se ingresa el Pseudo-Código.
27. El usuario selecciona la opción “Guardar” de la barra de herramientas o del menú “Pseudo-Codigo”.
28. El sistema sobre escribe el mismo archivo.

ABRIR

29. En la barra de herramientas de la ventana “PANTALLA PRINCIPAL” el usuario selecciona la opción “Pseudo-Código”.
30. El sistema muestra la ventana “EDITOR”.
31. El usuario selecciona el botón “Abrir” de la barra de herramientas o la opción

“Abrir” del menú “Pseudo-Codigo”.

32. El sistema muestra el cuadro de diálogo “Abrir”
33. El usuario selecciona el archivo de Pseudo-Código que desea abrir, de una dirección especificada
34. El sistema, en una pestaña nueva, carga el archivo en la ventana “EDITOR”.
35. El usuario puede realizar lo detallado a partir del paso 8 o 36.

NUEVO

36. En la barra de herramientas de la ventana “PANTALLA PRINCIPAL” el usuario selecciona la opción “Pseudo-Código”.
37. El sistema muestra la ventana “EDITOR”.
38. El usuario selecciona el botón “Nuevo” de la barra de herramientas o la opción “Nuevo” del menú “Pseudo-Codigo”, de la ventana “EDITOR”.
39. El sistema muestra el cuadro de diálogo “Guardar”.
40. El usuario ingresa en nombre del archivo y selecciona la ubicación donde va almacenar.
41. El usuario presiona el botón “Guardar”.
42. El sistema guarda el archivo en la dirección especificada y presenta en la ventana “EDITOR” una nueva pestaña con su campo limpio y con el nombre del nuevo archivo.
43. El usuario puede realizar lo detallado a partir del paso 8 o 29.
44. El usuario selecciona la opción “Salir” presionando el botón “X” de la pestaña del archivo de Pseudo-Código.
45. El usuario selecciona la opción “Salir” del menú “Pseudo-Codigo” o del botón cerrar de la ventana “EDITOR”.

OPCIONES ADICIONALES

46. El usuario puede cambiar de “Aspecto” a la ventana “EDITOR” presionando cualquiera de las opciones del menú “Aspecto” de la misma ventana.
47. El sistema muestra la ventana “EDITOR” con el “Aspecto” seleccionado.
48. El usuario puede cambiar de “Colores” a los “Simbolos”, “Variables”, “Palabras Reservadas”, “Números” del Pseudo-Código que se ingrese en la ventana “EDITOR”; presionando cualquiera de los capos de colores que están en las opciones del menú “Colores”.
49. El sistema muestra el Pseudo-Código con los colores seleccionados.
50. El usuario puede imprimir el Pseudo-Código, presionando las opciones “Imprimir”, del menú “Pseudo-Codigo” o del botón de la barra de herramientas.

51. El sistema presenta la ventana "Imprimir".
52. El usuario selecciona las opciones respectivas para imprimir.
53. El sistema da la orden de imprimir el Pseudo-Código.
54. El usuario puede seleccionar "Ayuda" para saber el funcionamiento de el "EDITOR" de PsGram, presionando la opción "Contenido de Ayuda" del menú "Ayuda" de "EDITOR", ó presionando la tecla "F1".
55. El sistema presenta la ventana "Contenidos PSGRAM".
56. El usuario puede seleccionar "Acerca de.." para saber sobre la función del programa, correo electrónico del desarrollador de PsGram y los derechos. Presionando la opción "Acerca de.." del menú "Ayuda" de la ventana "EDITOR".
57. El sistema presenta la ventana "ADMINISTRADOR".
58. El usuario puede cambiar el tipo de letra del Pseudo-Código, haciendo clic en la primera flechita (con dirección hacia abajo) del lado izquierdo del programa, se despliega un menú y se podrá escoger cualquier tipo de letra que ahí indique.
59. El sistema muestra el Pseudo-Código con el tipo de letra seleccionado.
60. El usuario puede cambiar el tamaño de letra del Pseudo-Código, haciendo clic en las segundas flechitas (con dirección hacia arriba y abajo) del lado izquierdo del programa, se podrá escoger hasta el tamaño de letra número 100.
61. El sistema muestra el Pseudo-Código con el tamaño de letra seleccionado.
62. El caso de uso finaliza

Flujo alterno de Eventos:

A. ERROR AL COMPILAR (Mensaje)

A.8. El sistema muestra el mensaje "**ERROR AL COMPILAR: Error léxico encontrado Carácter no válido**".

A.9. El caso de uso continúa en el numeral 8 del flujo normal de eventos.

B. PALABRA NO EXISTE (Mensaje)

B.15. El sistema muestra el mensaje de "**No hay coincidencias**".

B.16. El caso de uso continúa en el numeral 14 del flujo normal de eventos.

C. YA NO ENCONTRAMOS MAS PALABRAS (Mensaje)

C.17. El sistema muestra el mensaje de "**No hay coincidencias**".

C.18. El caso de uso continúa en el numeral 14 del flujo normal de eventos.

D. CAMPOS VACIOS(PSGRAM)

D.23. El sistema muestra el mensaje de "**Campos vacíos**".

D.24. El caso de uso continúa en el numeral 21 del flujo normal de eventos.

E. NO EXISTIERON REEMPLAZOS (PSGRAM)

E.23. El sistema muestra el mensaje de “No se han hecho reemplazos”.

E.24. El caso de uso continúa en el numeral 21 del flujo normal de eventos.

F. ARCHIVO INCORRECTO (PSGRAM)

F.33. El sistema muestra el mensaje de “Archivo incorrecto”.

F.34. El caso de uso continúa en el numeral 32 del flujo normal de eventos.

G. GUARDAR CAMBIOS AL CERRAR (GUARDAR)

G.44. El sistema muestra el mensaje “Desea guardar los cambios”.

G.45. (SI) El caso de uso continúa en el numeral 2 del flujo normal de eventos

G.46. (NO) El sistema cierra la pestaña de la pantalla EDITOR sin guardar ningún cambio.

G.47. (CANCELAR) El sistema deja intacta la pestaña de la ventana EDITOR, el caso de uso continúa en el numeral 8 del flujo normal de eventos.

H. GUARDAR CAMBIOS AL CERRAR (GUARDAR)

H.45. El sistema muestra el mensaje “Desea guardar los cambios a “*nombre del archivo*””.

H.46. (SI) El caso de uso continúa en el numeral 1 del flujo normal de eventos

H.47. (NO) El sistema cierra la aplicación sin guardar ningún cambio.

H.48. (CANCELAR) El sistema deja intacto el EDITOR, el caso de uso continúa en el numeral 8 del flujo normal de eventos.

I. SALIR DEL SISTEMA (Salir)

I.45. El sistema muestra el mensaje “Salir del Sistema?”.

I.46. SI El sistema cierra la ventana “EDITOR”.

I.47.NO El sistema deja intacto el “EDITOR”

Tabla 17. Descripción del Caso de Uso: Ingresar Pseudo-Código.

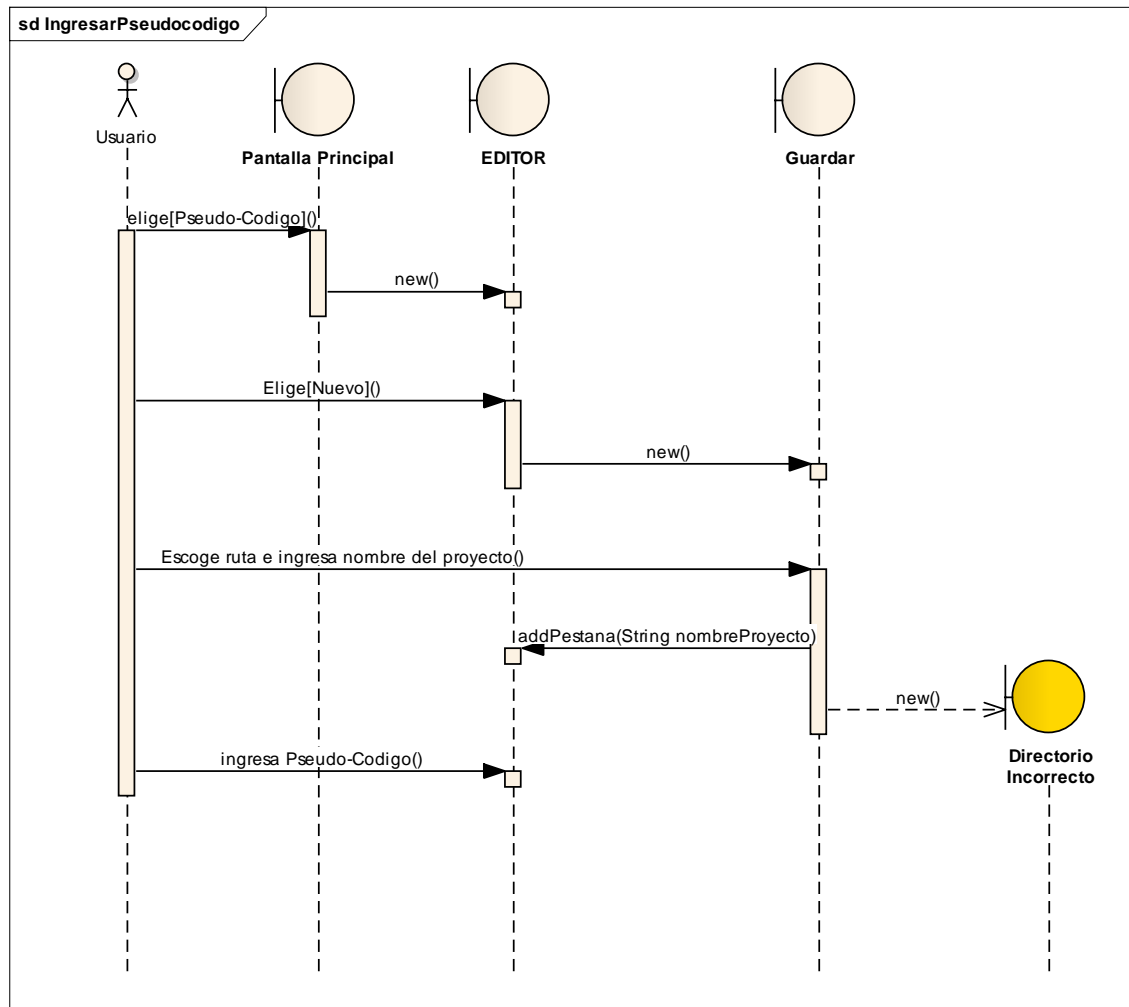


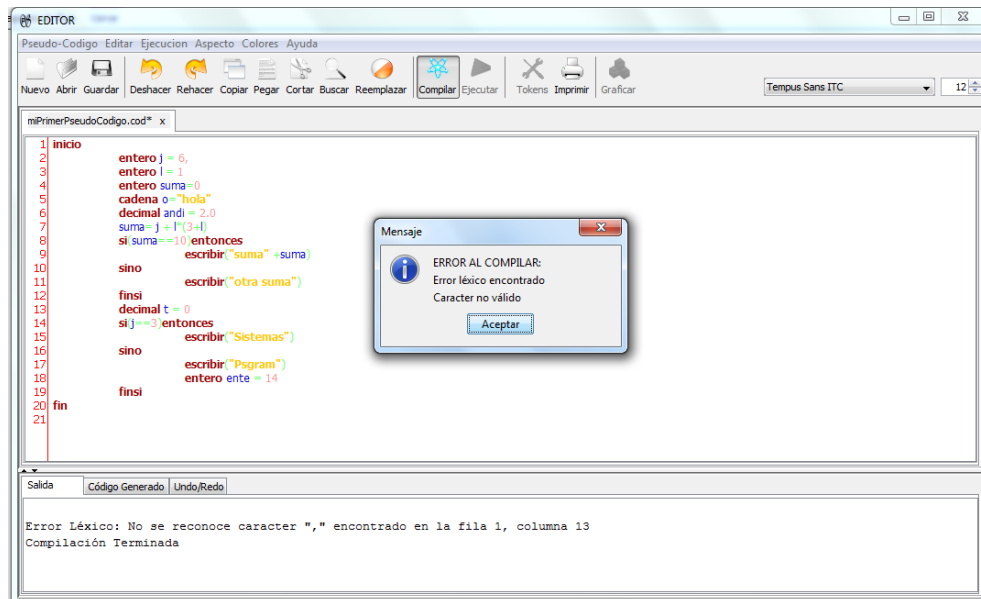
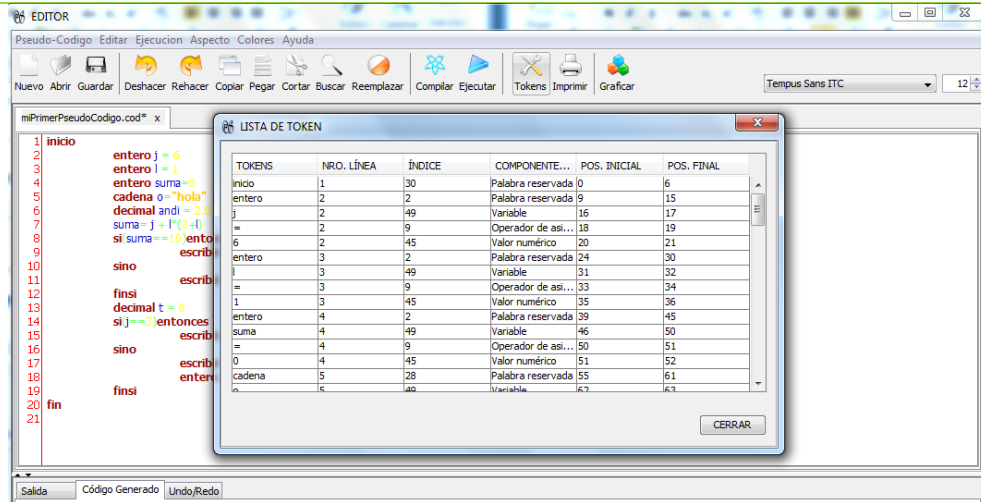
Diagrama 3 . Diagrama de Secuencia del Caso de Uso: Ingresar Pseudo-Código

1.4.2. CASO DE USO: Analizar Lexicamente

Nombre Pantalla: EDITOR **Código: PP002**

Tipo Interfaz Gráfica: JFrame

Caso de Uso: Analizar Lexicamente



Realizado por:

Andrea Natasha Salinas Ochoa.

Fecha:

09/06/2011

Tabla 18 . Prototipo de Pantalla del Caso de Uso: Analizar Lexicamente.

Nombre C.U: Analizar Lexicamente		Código C.U: 0002
Req. Funcional:	RF002, RF003	
Objetivo(s):	Analizar lexicamente el Pseudocódigo ingresado.	
Descripción:	El Pseudo-Código ingresado por el usuario será analizado lexicamente, si hubiese algún error el caso de uso "Gestionar Error" presentará los errores léxicos en la pantalla "EDITOR". Si todo esta coorrecto se construirá un archivo con los tokens (palabras o cadenas) generados.	
Actor(s):	Usuario	
Tipo Caso Uso:	Sistema	
Pre-condiciones:	Tener cargado sobre el "EDITOR" el Pseudo-Código.	
Post-condiciones:	Se obtendrá un archivo analizado lexicográficamente y visualizar los tokens de este archivo.	
Flujo Normal de Eventos:		
<ol style="list-style-type: none"> 1. El usuario selecciona con un el botón "Compilar" de la barra de herramientas o la opción "Compilar" del menú "Ejecucion". 2. El sistema analiza lexicamente el Pseudo-Código ingresado por el usuario. 3. El sistema ejecuta el caso de uso Gestionar Errores. 4. El sistema construye un archivo que contiene el conjunto de tokens cada con: número de línea, índice, componente léxico, posición inicial y su posición final. 5. El usuario selecciona el botón "Tokens" de la barra de herramientas o la opción "Tokens" del menú "Ejecucion", de la ventana "EDITOR". 6. El sistema carga los tokens en una tabla, en la ventana "LISTA DE TOKEN" (TOKENS, NRO. LÍNEA, ÍNDICE, COMPONENTE LÉXICO, POS. INICIAL, POS. FINAL) 7. El caso de uso finaliza. 		
Flujo alterno de Eventos:		
A. ERROR AL COMPILAR (Mensaje)		
A.3. El sistema muestra el mensaje "ERROR AL COMPILAR: Error léxico encontrado Carácter no válido" .		
A.4. El caso de uso regresa al caso de uso "Ingresar Pseudo-Código".		
B. ERROR AL COMPILAR		
B.3. El sistema carga sobre la pestaña "Salida" de la ventana "EDITOR" el mensaje "Error Léxico: No se reconoce caracter "[caracter]" encontrado en la fila		

[número fila], columna [número columna] **Compilación Terminada**".

B.4. El caso de uso regresa al caso de uso "Ingresar Pseudo-Código".

Tabla 19 . Descripción del Caso de Uso: Analizar Léxicamente.

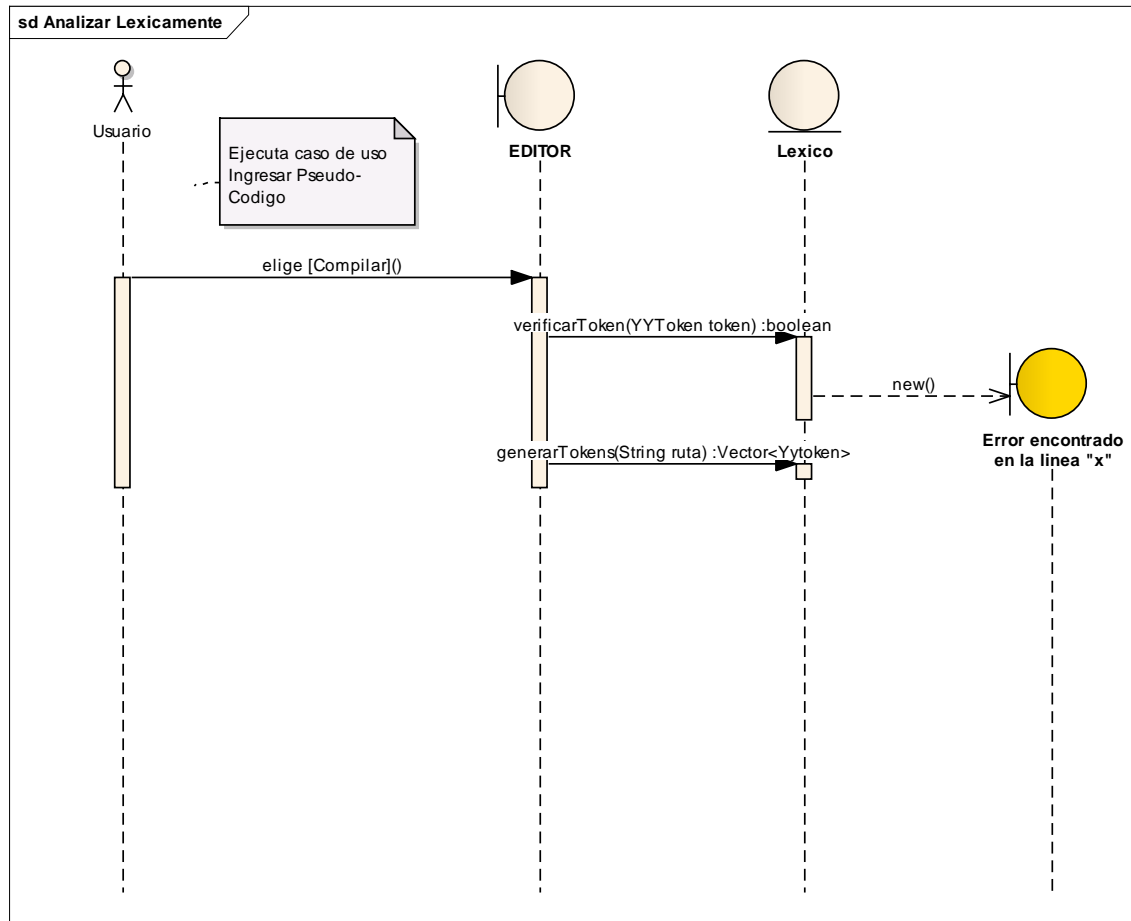


Diagrama 4 .Diagrama de Secuencia del Caso de Uso: Analizar Léxicamente

1.4.3. CASO DE USO: Analizador Sintáctico

Nombre Pantalla: EDITOR	Código: PP003
Tipo Interfaz Gráfica: JFrame	
Caso de Uso: Analizador Sintactico	
Realizado por: Andrea Natasha Salinas Ochoa.	Fecha: 09/06/2011

Tabla 20 . Prototipo de Pantalla del Caso de Uso: Analizador Sintactico.

Nombre C.U: Analizador Sintactico		Código C.U: 0003
Req. Funcional:	RF004, RF005	
Objetivo(s):	Realizar el análisis sintáctico de la sentencias ingresadas por el usuario gracias a la definición de la gramática.	
Descripción:	El usuario podrá ingresar Pseudo-Código que desee ejecutar, siempre que éste se encuentre enmarcado dentro de las reglas establecidas. El Pseudo-Código será analizado sintácticamente, si hubiese algún error el caso de uso "Gestionar Error" presentará los errores sintácticos en la pantalla "EDITOR".	
Actor(s):	Usuario	
Tipo Caso Uso:	Sistema	
Pre-condiciones:	Tener el Pseudo-Código cargado sobre la ventana "EDITOR" y analizado léxicamente, en caso de presentar errores léxicos corregirlos.	
Post-condiciones:	Se obtendrá un Pseudo-Código analizado sintácticamente y listo para ser analizado semánticamente.	
Flujo Normal de Eventos:		
<ol style="list-style-type: none"> 1. Una vez ejecutado el CU Analizar Léxicamente el sistema analiza sintácticamente el Pseudo-Código ingresado. 2. El sistema ejecuta el caso de uso Gestionar Errores. 3. El caso de uso finaliza. 		
Flujo alterno de Eventos:		
A. ERROR AL COMPILAR		
A.2. El sistema carga sobre la pestaña "Salida" de la ventana "EDITOR" el mensaje "ERROR AL COMPILAR: Error de sintaxis palabras reservada 'inicio' falta en el programa" o "Compilando... ERROR AL COMPILAR: Error de sintáctico entre las líneas [número línea] y [número línea], '[letra o palabra que se encuentra haciendo error]' " .		
A.3. El caso de uso regresa al caso de uso "Ingresar Pseudo-Código".		

Tabla 21 . Descripción del Caso de Uso: Analizador Sintactico.

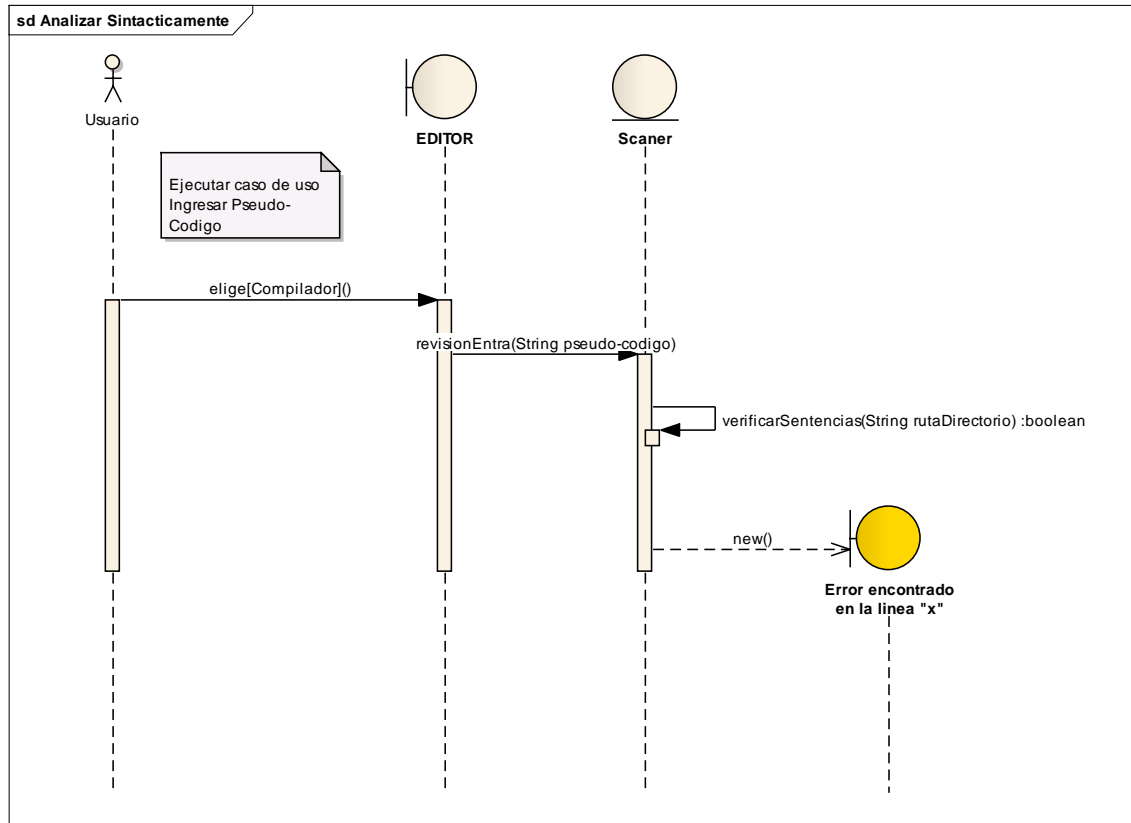


Diagrama 5 .Diagrama de Secuencia del Caso de Uso: Analizador Sintactico

1.4.4. CASO DE USO: Analizador Semántico

Nombre Pantalla: EDITOR	Código: PP004
Tipo Interfaz Gráfica: JFrame	
Caso de Uso: Analizador Semantico	

```

miPrimerPseudoCodigo.cod x
1 inicio
2     entero j = 6
3     entero l = 1
4     entero suma=0
5     cadena o="hola"
6     decimal andi = 2.0
7     suma = j + l*(3+l)
8     si (suma==10) entonces
9         escribir("suma" +suma)
10
11     sino
12         escribir("otra suma")
13     fin
14     decimal t = 0
15     si(j==3) entonces
16         escribir("Sistemas")
17     sino
18         escribir("Psgram")
19     fin
20 fin
21
  
```

Salida Código Generado Undo/Redo

Compilando...
 Compilación Correcta

```

miPrimerPseudoCodigo.cod x
1 inicio
2     entero j = 6
3     entero l = 1
4     entero suma=0
5     cadena o="hola"
6     decimal andi = 2.0
7     suma = j + l*(3+l)
8     si (suma==10) entonces
9         escribir("suma" +suma)
10
11     sino
12         escribir("otra suma")
13     fin
14     decimal t = 0
15     si(j==3) entonces
16         escribir("Sistemas")
17     sino
18         escribir("Psgram")
19     fin
20 fin
21
  
```

Salida Código Generado Undo/Redo

Salida del programa:
 suma10
 Psgram
 Fin de la Ejecución

```

miPrimerPseudoCodigo.cod x
1 inicio
2     entero j = 6
3     entero l = 1
4     entero suma=0
5     cadena o="hola"
6     decimal andi = 2.0
7     suma = j + l*(3+l)
8     si (suma==10) entonces
9         escribir("suma" +suma)
10
11     sino
12         escribir("otra suma")
13     fin
14     decimal t = 0
15     si(j==3) entonces
16         escribir("Sistemas")
17     sino
18         escribir("Psgram")
19     fin
20 fin
21
  
```

Salida Código Generado Undo/Redo

```

public class CodigoGenerado {
public void cuerpoPrograma(){
int j = 6;
int l = 1;
int suma = 0;
String o = "hola";
double andi = 2.0;
suma = j + l * (3 + l);
if (suma == 10) {
System.out.println("suma" + suma);
} else {
  
```

```

miPrimerPseudoCodigo.cod* x
1 inicio
2     entero j = 6
3     entero l = 1
4     entero suma=0
5     cadena o="hola"
6     decimal andi = 2 + o
7     suma = j + l*(3+l)
8     si (suma==10) entonces
9         escribir("suma" +andi)
10
11     sino
12         escribir("otra suma")
13     fin
14     decimal t = 0
15     si(j==3) entonces
16         escribir("Sistemas")
17     sino
18         escribir("Psgram")
19     fin
20 fin
21
  
```

Salida Código Generado Undo/Redo

ERROR AL COMPILAR:
 Error Semantico encontrado
 Incompatibilidad de tipos de datos en la expresión,
 la variable <o> no es de tipo entero o decimal

	<p>Salida Código Generado Undo/Redo</p> <p>ERROR AL COMPILAR: Error Semantico encontrado La variable j no está declarada</p>
<p>Salida Código Generado Undo/Redo</p> <p>ERROR AL COMPILAR: Error Semantico encontrado Incompatibilidad de tipos de datos, variable <suma> = 'a + 2.5'</p>	
<p>Salida Código Generado Undo/Redo</p> <p>ERROR AL COMPILAR: Error Semantico encontrado Variable repetida entero j = 6</p>	<p>Salida Código Generado Undo/Redo</p> <p>ERROR AL COMPILAR: Error Semantico encontrado Variable repetida afirmacion j = v</p>
<p>Salida Código Generado Undo/Redo</p> <p>ERROR AL COMPILAR: Error Semantico encontrado Variable repetida entero j = 4</p>	<p>Salida Código Generado Undo/Redo</p> <p>ERROR AL COMPILAR: Error Semantico encontrado La variable suma no está declarada</p>
<p>Salida Código Generado Undo/Redo</p> <p>ERROR AL COMPILAR: Error Semantico encontrado Incompatibilidad de tipos de datos, la variable <l> no es de tipo afirmacion</p>	
<p>Realizado por: Andrea Natasha Salinas Ochoa.</p>	<p>Fecha: 09/06/2011</p>

Tabla 22 . Prototipo de Pantalla del Caso de Uso: Analizador Semantico.

Nombre C.U: Analizador Semantico		Código C.U: 0004
Req. Funcional:	RF006, RF007	
Objetivo(s):	Analizar semánticamente el Pseudo-Código ingresado.	
Descripción:	El usuario podrá ingresar el bloque de código que desee ejecutar, siempre que éste se encuentre enmarcado dentro de las reglas establecidas. El Pseudo-Código será analizado semánticamente, si hubiese algún error el caso de uso "Gestionar Error" presentará los errores semánticos en la pantalla "EDITOR". Si todo esta correcto se construirá un archivo con código intermedio (.java).	
Actor(s):	Usuario	
Tipo Caso Uso:	Sistema	
Pre-condiciones:	<ul style="list-style-type: none"> -Tener cargado sobre el editor el Pseudo-Código. -Se haya ejecutado el caso de uso Analizar Léxicamente. -Se encuentre en ejecución el caso de uso analizador Sintactico. 	
Post-condiciones:	-Código analizado semánticamente, generación de código intermedio.	
Flujo Normal de Eventos:		
<ol style="list-style-type: none"> 1. La ejecución del analizador Sintáctico y Semántico se realiza simultáneamente. 2. El sistema ejecuta el caso de uso Gestionar Errores. 3. El sistema construye un archivo que contiene el código intermedio (.java). 4. El sistema carga el código intermedio (.java) en la pestaña "Código Generado" de la ventana "EDITOR". 8. El sistema activa: los los botones "Ejecutar", "Tokens", "Graficar" de la barra de herramientas y las opciones "Ejecutar", "Tokens" del menú "Ejecucion". <p>Opciones de Ejecución</p> <ol style="list-style-type: none"> 5. El usuario puede seleccionar con un clic una de estas tres opciones: el botón "Ejecutar" de la barra de herramientas, la opción "Ejecutar" del menú "Ejecucion" o en su defecto hacer clic derecho sobre el editor opción "Compilar-Ejecutar" de la ventana "EDITOR". 6. El sistema guarda automáticamente sobre escribiendo el mismo archivo. 7. El sistema carga sobre la pestaña "Salida" de la ventana "EDITOR" el mensaje 		

<p>“Salida del programa: [resultados de la ejecución] Fin de la Ejecución”.</p> <p>8. El caso de uso finaliza.</p>
<p>Flujo alterno de Eventos:</p>
<p>A. ERROR AL COMPILAR</p> <p>A.2. El sistema muestra el mensaje “ERROR AL COMPILAR: Error Semantico encontrado [Tipo de error].</p> <p>Los tipos de error pueden ser:</p> <p style="padding-left: 40px;">“Variable repetida/ncadena”</p> <p style="padding-left: 40px;">“Variable repetida/nafirmacion”</p> <p style="padding-left: 40px;">“Variable repetida/nentero”</p> <p style="padding-left: 40px;">“La variable <variable> no está declarada”</p> <p style="padding-left: 40px;">“Incompatibilidad de tipo de datos en la expresión, la variable <variable> no es de tipo entero o decimal”</p> <p style="padding-left: 40px;">“Incompatibilidad de tipo de datos, la variable <nombreVariable> no es de tipo afirmacion”</p> <p>A.3. El caso de uso regresa al caso de uso “Ingresar Pseudo-Código”.</p> <p>B. NO SE HA COMPILADO (PSGRAM)</p> <p>B.5. El sistema muestra e mensaje “Primero Compile el programa”.</p> <p>B.6. El caso de uso regresa al caso de uso “Analizar Lexicamente”.</p>

Tabla 23 . Descripción del Caso de Uso: Analizador Semantico.

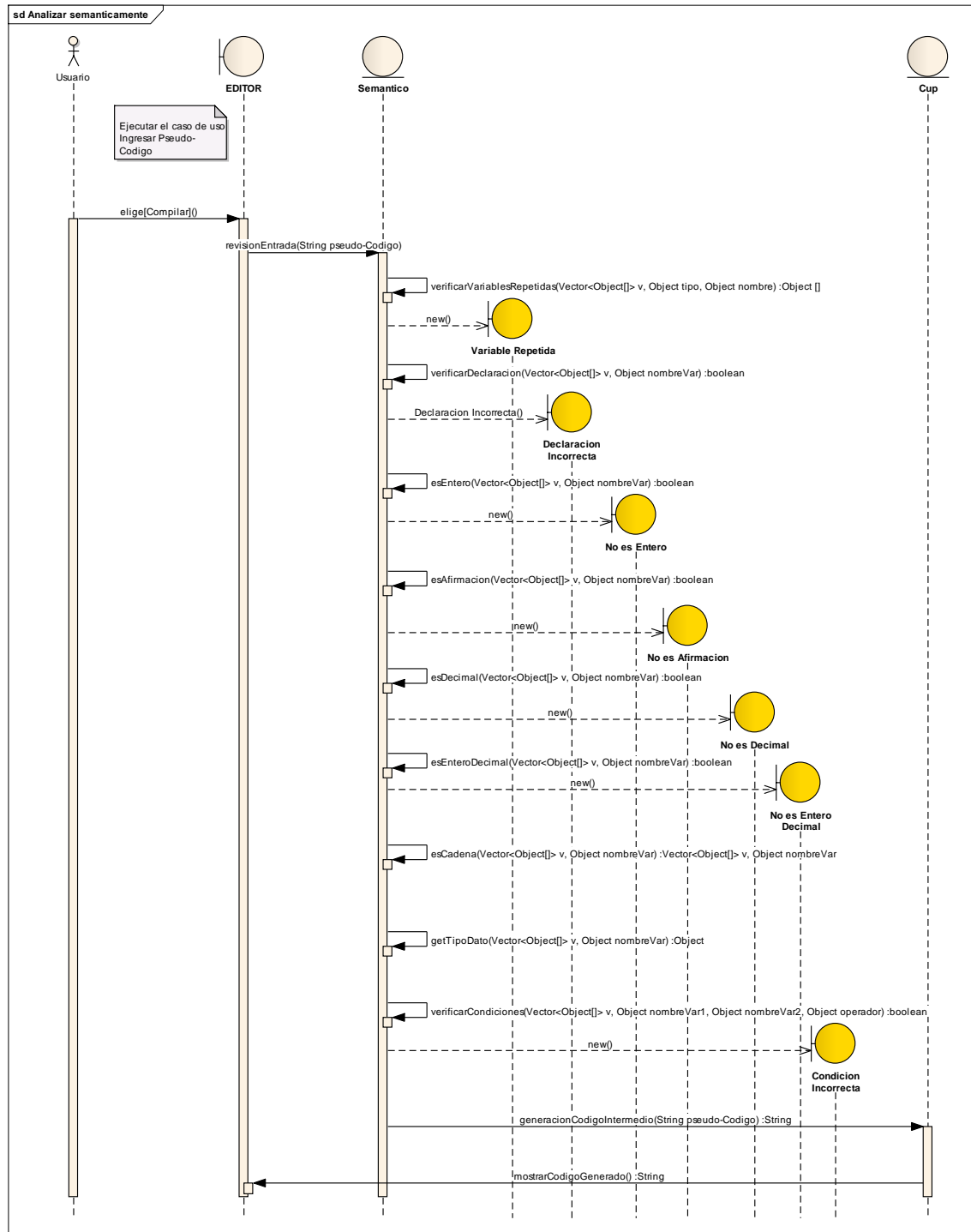


Diagrama 6 .Diagrama de Secuencia del Caso de Uso: Analizador Semantico

1.4.5. CASO DE USO: Gestionar Errores

Nombre Pantalla: EDITOR	Código: PP005
Tipo Interfaz Gráfica: JFrame	
Caso de Uso: Gestionar Errores	
Realizado por: Andrea Natasha Salinas Ochoa.	Fecha: 09/06/2011

Tabla 24 . Prototipo de Pantalla del Caso de Uso: Gestionar Errores.

Nombre C.U: Gestionar Errores	Código C.U: 0005
Req. Funcional:	RF003, RF005, RF007
Objetivo(s):	Gestionar los errores del Pseudo-Código ingresado.
Descripción:	El usuario podrá ingresar el bloque de código que desee ejecutar, siempre que éste se encuentre enmarcado dentro de las reglas establecidas. El Pseudo-Código será analizado lexicamente, sintácticamente y semánticamente; en el momento que va pasando por cada análisis se presentarán los errores correspondientes a cada uno si los hubiese. Este caso de uso hace la parte de visualizar los errores en la ventana "EDITOR".
Actor(s):	Usuario
Tipo Caso Uso:	Sistema
Pre-condiciones:	-Tener cargado sobre el editor el Pseudo-Código. -Se encuentre en ejecución el caso de uso analizar Lexicamente.
Post-condiciones:	-Presentar el la ventana "EDITOR" "Compilacion correcta" del Pseudo-Código.
Flujo Normal de Eventos:	
<ol style="list-style-type: none"> 1. El sistema gestiona los errores del análisis léxico del Pseudo-Código ingresado. 2. El sistema gestiona los errores del análisis sintáctico del Pseudo-Código ingresado. 3. El sistema gestiona los errores del análisis semántico del Pseudo-Código ingresado. 4. El sistema carga sobre la pestaña "Salida" de la ventana "EDITOR" el mensaje "Compilando... Compilación Correcta" 5. El caso de uso finaliza. 	
Flujo alterno de Eventos:	
A. ERROR LÉXICO	
<p>A.1. Si el análisis léxico encuentra algún error. En éste caso de uso, el sistema se encarga de comparar que clase de error es?, y presentarlo en la ventana "EDITOR". Para que el usuario sepa que tipo de error es, el caso de uso regresa al "Flujo alterno de Eventos" del caso de uso "Analizar Lexicamente".</p>	

B. ERROR SINTÁCTICO

B.2. Si el análisis sintáctico encuentra algún error. En éste caso de uso, el sistema se encarga de comparar que clase de error es?, y presentarlo en la ventana "EDITOR". Para que el usuario sepa que tipo de error es, el caso de uso regresa al "**Flujo alternativo de Eventos**" del caso de uso "Analizador Sintactico".

C. ERROR SEMÁNTICO

C.3. Si el análisis semántico encuentra algún error. En éste caso de uso, el sistema se encarga de comparar que clase de error es?, y presentarlo en la ventana "EDITOR". Para que el usuario sepa que tipo de error es, el caso de uso regresa al "**Flujo alternativo de Eventos, literal A**" del caso de uso "Analizador Semantico".

Tabla 25 . Descripción del Caso de Uso: Gestionar Errores.

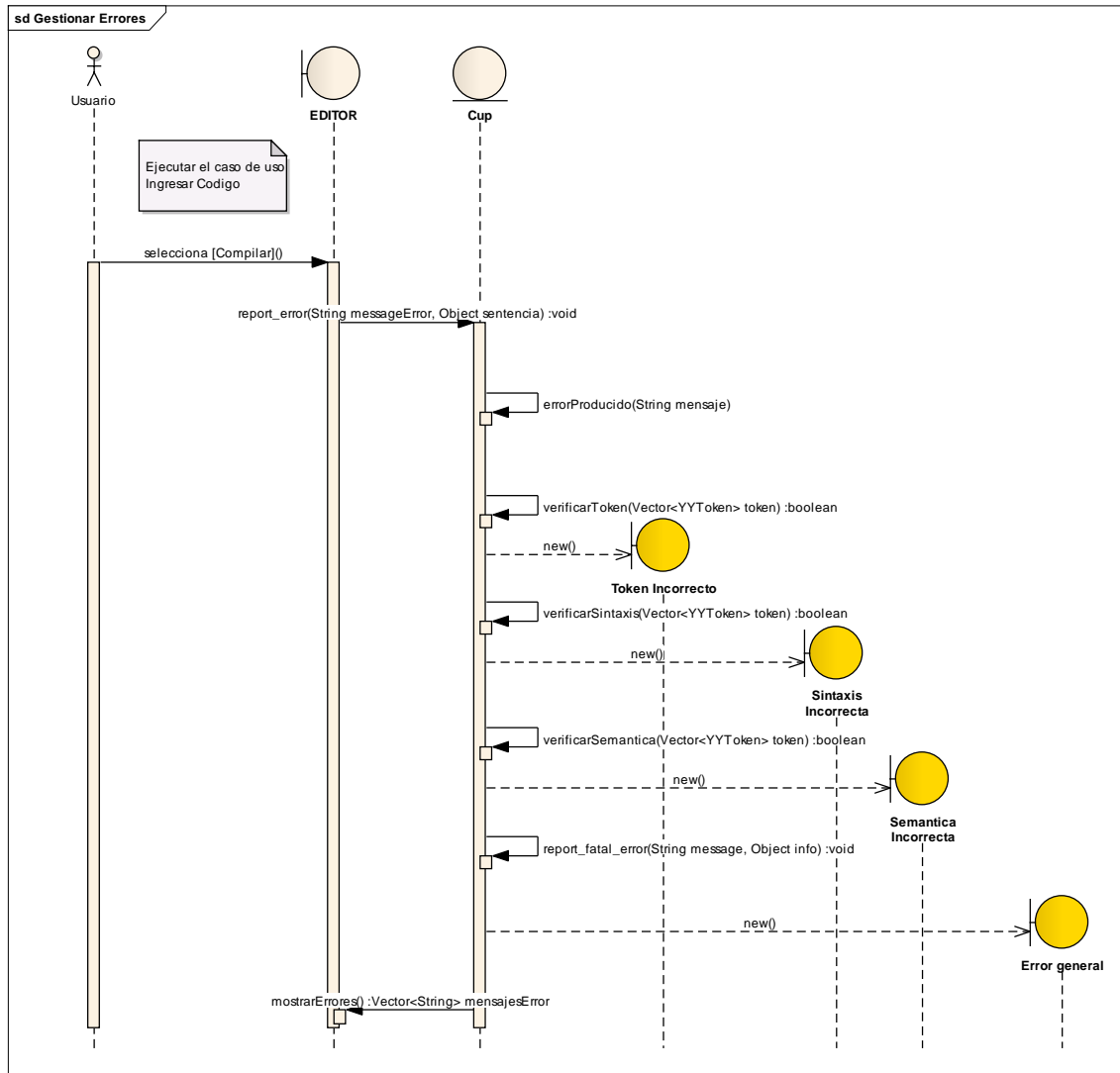


Diagrama 7 . Diagrama de Secuencia del Caso de Uso: Gestionar Errores

1.4.6. CASO DE USO: Crear Diagrama

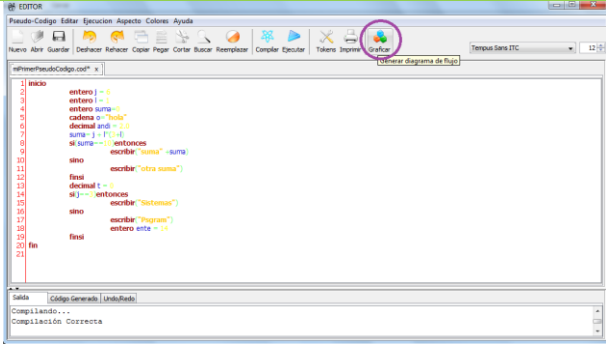
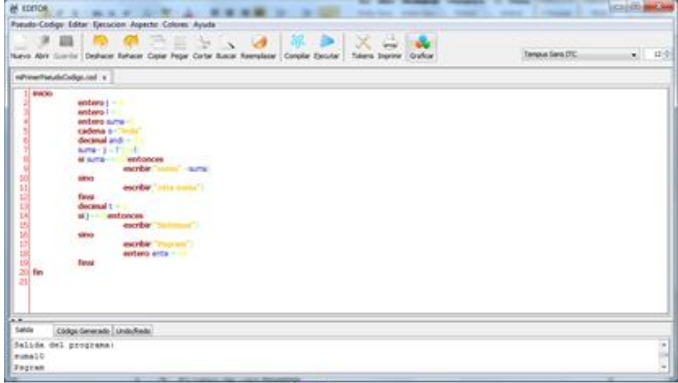
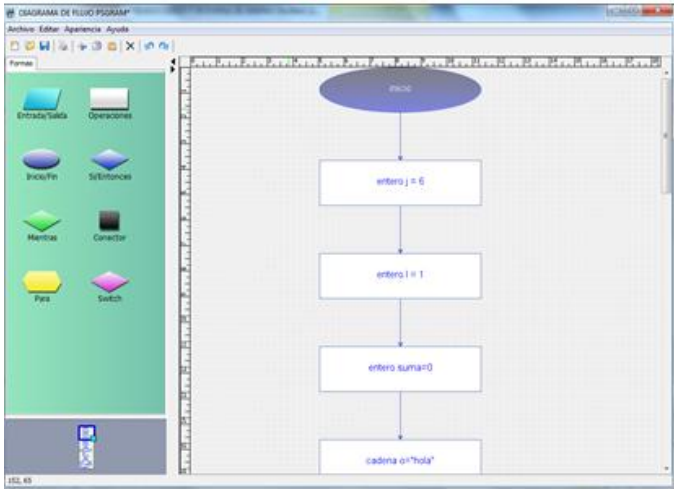
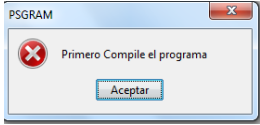
Nombre Pantalla:	Código: PP006
Tipo Interfaz Gráfica: JFrame	
Caso de Uso: Crear Diagrama	
<div style="display: flex; justify-content: space-between;"> <div style="width: 60%;">   <p style="text-align: center; color: purple; font-size: 2em;">↓ se genera</p>  </div> <div style="width: 35%; text-align: right;">  <p style="color: purple; font-size: 1.5em; margin-top: 20px;">Ventana EDITOR</p> <p style="color: purple; font-size: 1.5em; margin-top: 20px;">Ventana DIAGRAMA DE FLUJO PSGRAM</p> </div> </div>	
Realizado por: Andrea Natasha Salinas Ochoa.	Fecha: 09/06/2011

Tabla 26 . Prototipo de Pantalla del Caso de Uso: Crear Diagrama.

Nombre C.U: Crear Diagrama		Código C.U: 0006
Req. Funcional:	RF008	
Objetivo(s):	Crear el Diagrama de Flujo de su respectivo Pseudo-Código ingresado.	
Descripción:	Después de que está correctamente compilado el Pseudo-Código ingresado, el usuario selecciona la opción crear Diagrama de Flujo. Y éste diagrama se generará y será visible para el usuario.	
Actor(s):	Usuario	
Tipo Caso Uso:	Sistema	
Pre-condiciones:	-Se haya ejecutado el caso de uso Analizador Semantico.	
Post-condiciones:	-Creado el Diagrama de Flujo.	
Flujo Normal de Eventos:		
<ol style="list-style-type: none"> 1. El usuario selecciona el botón “Graficar” de la barra de herramientas de la ventana “EDITOR”. 2. El sistema visualiza en la ventana “DIAGRAMA DE FLUJO PSGRAM” el diagrama de flujo generado. 3. El caso de uso finaliza. 		
Flujo alterno de Eventos:		
A. NO SE HA COMPILADO (PSGRAM)		
A.1. El sistema muestra e mensaje “ Primero Compile el programa ”.		
A.1. El caso de uso regresa al caso de uso “Analizar Lexicamente”.		

Tabla 27 . Descripción del Caso de Uso: Crear Diagrama.

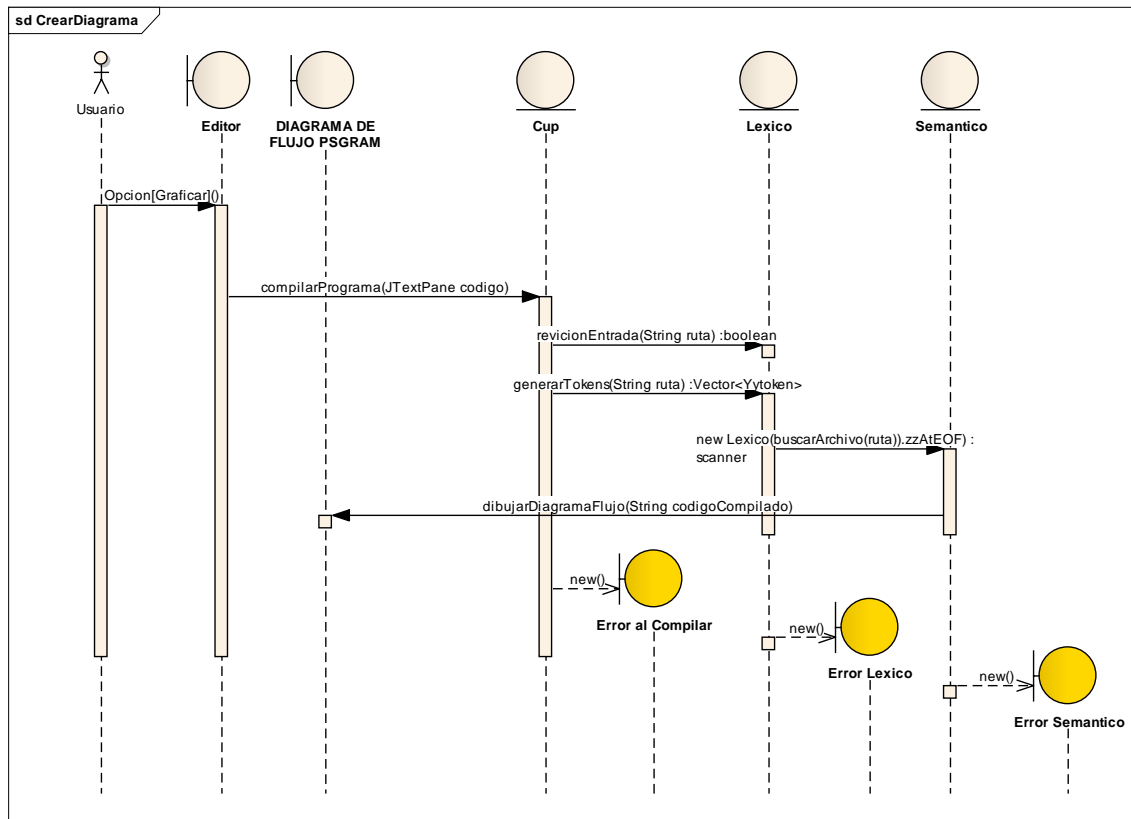
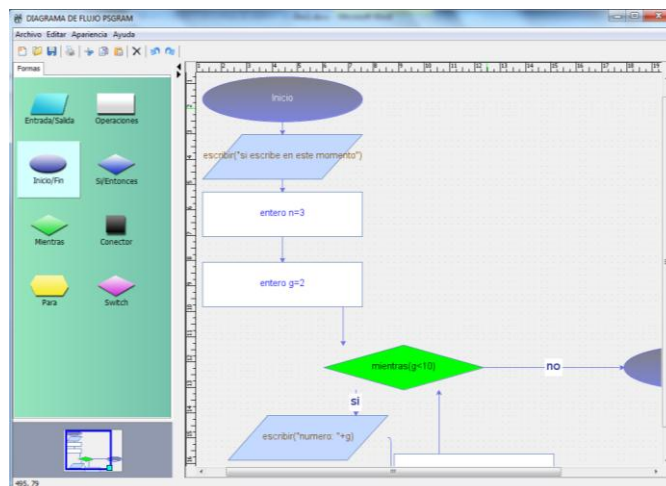
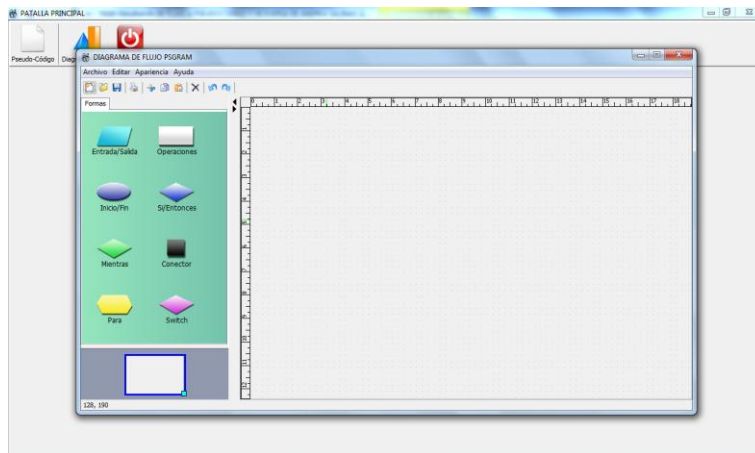
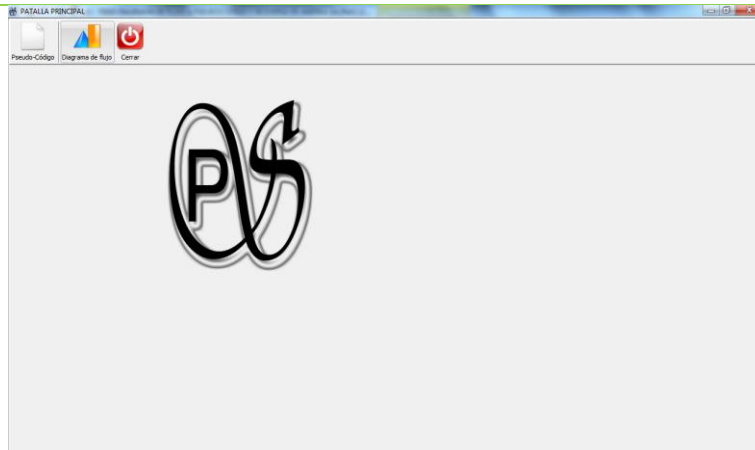
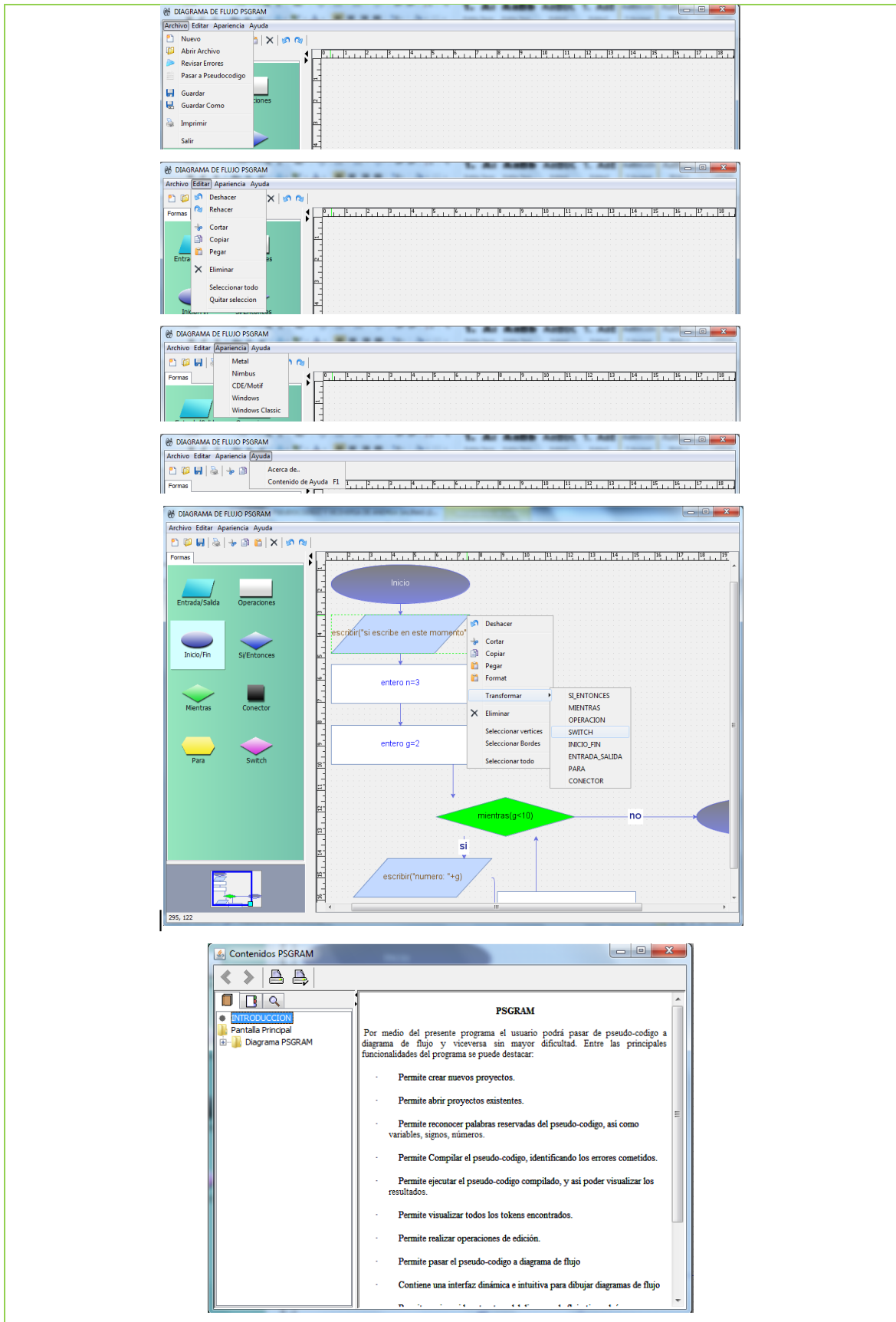


Diagrama 8 . Diagrama de Secuencia del Caso de Uso: Crear Diagrama

1.4.7. CASO DE USO: Dibujar Diagrama

Nombre Pantalla: DIAGRAMA DE FLUJO PSGRAM	Código: PP007
Tipo Interfaz Gráfica: JFrame	
Caso de Uso: Dibujar Diagrama	





 		
		
		
		
<p>Realizado por: Andrea Natasha Salinas Ochoa.</p>		<p>Fecha: 09/06/2011</p>

Tabla 28 . Prototipo de Pantalla del Caso de Uso: Dibujar Diagrama.

Nombre C.U: Dibujar Diagrama		Código C.U: 0007
Req. Funcional:	RF009	
Objetivo(s):	Idibujar nuestro propio Diagrama de Flujo.	
Descripción:	El usuario podrá ingresar los distintos elementos del diagrama de flujo en el campo destinada a esta función. El usuario podrá: administrar (cortar, copiar, pegar, eliminar), dar formato, imprimir los elementos, deshacer y rehacer las acciones, cambiar de aspecto a la ventana "DIAGRAMA DE FLUJO PSGRAM", y entre otras mas opciones que describiré a continuación.	
Actor(s):	Usuario	
Tipo Caso Uso:	Sistema	
Pre-condiciones:	Se haya ejecutado la ventana "PANTALLA PRINCIPAL".	
Post-condiciones:	El usuario ingresará el Diagrama de Flujo a ejecutar o abrirá uno previamente guardado.	
Flujo Normal de Eventos:		
<ol style="list-style-type: none"> 1. En la barra de herramientas de la ventana "PANTALLA PRINCIPAL" el usuario selecciona la opción "Diagrama de flujo". 2. El sistema muestra la ventana "DIAGRAMA DE FLUJO PSGRAM". 3. El usuario ingresará el Diagrama de Flujo que desee. <p>Opciones de Edición</p> <ol style="list-style-type: none"> 4. El usuario puede "Seleccionar todo" o "Quitar seleccion" a cualquier Diagrama de Flujo; presionando las siguientes opciones: - "Seleccionar todo" [Ctrl+a], "Quitar seleccion", del menú "Editar"; - "Seleccionar todo" haciendo clic derecho sobre el sector donde se ingresa el Diagrama de Flujo. 5. El usuario puede "Copiar" [Ctrl+c], "Cortar" [Ctrl+x], "Pegar" [Ctrl+v], cualquier Diagrama de Flujo o parte de éste; presionando las siguientes opciones: - "Copiar", "Cortar", "Pegar", del menú "Editar"; - En la barra de herramientas haciendo clic en los botones respectivos; - Haciendo clic derecho (sobre el sector donde se ingresa el Diagrama de Flujo) en "Copiar", "Cortar", "Pegar". 6. El usuario puede "Deshacer" y "Rehacer" cualquier acción presionando las siguientes opciones: -"Deshacer" [Ctrl+z], "Rehacer" [Ctrl+y], del menú "Editar"; - En los botones de la barra de herramientas; -Para deshacer haciendo clic derecho (sobre el sector donde se ingresa el Diagrama de Flujo) en la opción "Deshacer". 		

7. El usuario puede “Eliminar” cualquier Diagrama de Flujo o parte de éste; presionando las siguientes opciones: -“Eliminar” del menú “Editar”, -El botón “X” de la barra de herramientas, - Haciendo clic derecho (sobre el sector donde se ingresa el Diagrama de Flujo) en la opción “Eliminar”.
 8. El usuario puede dar “Formato” al diagrama de flujo ingresado, haciendo clic derecho sobre el sector donde se ingresa el diagrama y presionando la opción “Format”.
 9. El usuario puede seleccionar los vértices del diagrama de flujo, haciendo clic derecho sobre el sector donde se ingresa el diagrama y presionando la opción “Seleccionar vertices”.
 10. El usuario puede seleccionar los bordes del diagrama de flujo, haciendo clic derecho sobre el sector donde se ingresa el diagrama y presionando la opción “Seleccionar bordes”.
 11. El usuario puede transformar los vértices a otra figura, haciendo clic derecho sobre el vértice a modificar, se selecciona la opción “Transformar” ahí se despliegan las opciones y se selecciona una de ellas.
 12. El usuario selecciona la opción “Guardar” de la barra de herramientas o del menú Archivo, de la ventana “DIAGRAMA DE FLUJO PSGRAM”.
 13. El sistema muestra el cuadro de diálogo de “Guardar”.
 14. El usuario ingresa en nombre del archivo y selecciona la ubicación donde va almacenar.
 15. El usuario presiona el botón “Guardar”.
 16. El sistema guarda el archivo en la dirección especificada.
 17. Si el usuario ya guardo el diagrama con anterioridad y presiona la opción “Guardar”. El sistema sobre escribe el mismo archivo.
 18. El usuario selecciona la opción “Guardar Como” del menú Archivo, de la ventana “DIAGRAMA DE FLUJO PSGRAM”.
 19. El sistema muestra el cuadro de diálogo de “Guardar”.
 20. El usuario ingresa en nombre del archivo y selecciona la ubicación donde va almacenar.
 21. El usuario presiona el botón “Guardar”.
 22. El sistema guarda el archivo en la dirección especificada.
- ABRIR**
23. En la ventana “DIAGRAMA DE FLUJO PSGRAM”, el usuario selecciona el botón “Abrir” de la barra de herramientas o la opción “Abrir Archivo” del menú

“Archivo”.

24. El sistema muestra el cuadro de diálogo “Abrir”
25. El usuario selecciona el archivo de Diagrama de Flujo que desea abrir, de una dirección especificada
26. El usuario selecciona “Abrir”.
27. El sistema carga el archivo en la ventana “DIAGRAMA DE FLUJO PSGRAM”.
28. El usuario puede edita el Diagrama de Flujo.

Opciones de Edición

29. El usuario puede realizar lo detallado a partir del paso 3, 29 o 32.

NUEVO

30. En la ventana “DIAGRAMA DE FLUJO PSGRAM”, el usuario selecciona el botón “Nuevo” de la barra de herramientas o la opción “Nuevo” del menú “Archivo”.
31. El sistema limpia la ventana “DIAGRAMA DE FLUJO PSGRAM” para crear un nuevo diagrama de flujo.
32. El usuario puede realizar lo detallado a partir del paso 3, 23 o 32.

OTRAS OPCIONES

33. El usuario puede imprimir el diagrama de flujo, presionando las opciones “Imprimir”, del menú “Archivo” o del botón de la barra de herramientas.
34. El sistema presenta la ventana “Imprimir”.
35. El usuario selecciona las opciones respectivas para imprimir.
36. El sistema da la orden de imprimir el diagrama de flujo.
37. El usuario puede cambiar de “Aspecto” a la ventana “DIAGRAMA DE FLUJO PSGRAM” presionando cualquiera de las opciones del menú “Apariencia” de la misma ventana.
38. El sistema muestra la ventana “DIAGRAMA DE FLUJO PSGRAM” con la “Apariencia” seleccionada.
39. El usuario puede seleccionar “Ayuda” para saber el funcionamiento del “DIAGRAMA DE FLUJO PSGRAM” de PsGram, presionando la opción “Contenido de Ayuda F1” del menú “Ayuda” de la ventana “DIAGRAMA DE FLUJO PSGRAM”, ó presionando la tecla “F1”.
40. El sistema presenta la ventana “Contenidos PSGRAM”.
41. El usuario puede seleccionar “Acerca de..” para saber sobre el nombre y correo electrónico del desarrollador de PsGram, como de la licencia que tiene éste programa. Presionando la opción “Acerca de..” del menú “Ayuda” de la

ventana “DIAGRAMA DE FLUJO PSGRAM”.

42. El sistema presenta la ventana “ACERCA DE.. PSGRAM”.

43. El usuario selecciona la opción “Salir” del menú “Archivo” o el botón cerrar de la ventana “DIAGRAMA DE FLUJO PSGRAM”.

44. El caso de uso finaliza.

Flujo alterno de Eventos:

A. ARCHIVO A GUARDAR ESTA VACÍO (ERROR)

A.15. El sistema muestra el mensaje de error “**java.lang.NullPointerException**”.

A.7. El caso de uso continúa en el numeral 3 del flujo normal de eventos.

B. NOMBRE DEL ARCHIVO A GUARDAR YA EXISTE (Seleccionar una opción)

B.15. El sistema muestra el mensaje de error “**Sobreescribir el archivo existente**”.

B.17.(SI) El sistema sobreescribe el archivo existente.

B.18.(NO) El caso de uso continúa en el numeral 3 del flujo normal de eventos.

B.19.(CANCELAR) El caso de uso continúa en el numeral 3 del flujo normal de eventos.

C. NOMBRE DEL ARCHIVO A GUARDAR YA EXISTE (Seleccionar una opción)

C.21. El sistema muestra el mensaje de error “**Sobreescribir el archivo existente**”.

C.22.(SI) El sistema sobreescribe el archivo existente.

C.23.(NO) El caso de uso continúa en el numeral 3 del flujo normal de eventos.

C.24.(CANCELAR) El caso de uso continúa en el numeral 3 del flujo normal de eventos.

D. PERDER LOS CAMBIOS DEL ARCHIVO ACTUAL (Seleccionar una opción)

D.23. El sistema muestra el mensaje de error “**Perder los cambios del archivo actual**”.

D.24.(SI) El caso de uso continúa en el numeral 24 del flujo normal de eventos.

D.25.(NO) El caso de uso continúa en el numeral 3 del flujo normal de eventos.

D.26.(CANCELAR) El caso de uso continúa en el numeral 3 del flujo normal de eventos.

E. TIPO DE ARCHIVO NO VALIDO PARA ABRIR (Error)

E.26. El sistema muestra el mensaje de error “**Archivo no soportado**” o “**La imagen no contiene información válida**”.

E.27. El caso de uso continúa en el numeral 3 del flujo normal de eventos.

F. PERDER LOS CAMBIOS DEL ARCHIVO ACTUAL (Seleccionar una opción)

F.30. El sistema muestra el mensaje de error “**Perder los cambios del archivo**”

actual”.

F.31.(SI) El caso de uso continúa en el numeral 31 del flujo normal de eventos.

F.32.(NO) El caso de uso continúa en el numeral 3 del flujo normal de eventos.

F.33.(CANCELAR) El caso de uso continúa en el numeral 3 del flujo normal de eventos.

G. SALIR DEL GRAFICADOR (PSGRAM)

G.43. En caso de no modificar el diagrama desde la última vez que se lo guardo o solo se abrió la ventana DIAGRAMA DE FLUJO PSGRAM, el sistema muestra el mensaje de “**Salir del graficador?**”.

G.44.(SI) El caso de uso continúa en el numeral 1 del flujo normal de eventos.

G.45.(NO) El sistema deja intacta la pantalla DIAGRAMA DE FLUJO PASGRAM.

H. GUARDAR ULTIMOS CAMBIOS (PSGRAM)

H.43. El sistema muestra el mensaje “**Desea guardar los últimos cambios?**”.

H.44.(SI) El sistema guarda los últimos cambios y cierra la ventana DIAGRAMA DE FLUJO PSGRAM; el caso de uso continúa en el numeral 1 del flujo normal de eventos.

H.45.(NO) El sistema no guarda los últimos cambios y cierra la ventana DIAGRAMA DE FLUJO PSGRAM; el caso de uso continúa en el numeral 1 del flujo normal de eventos.

Tabla 29 . Descripción del Caso de Uso: Dibujar Diagrama.

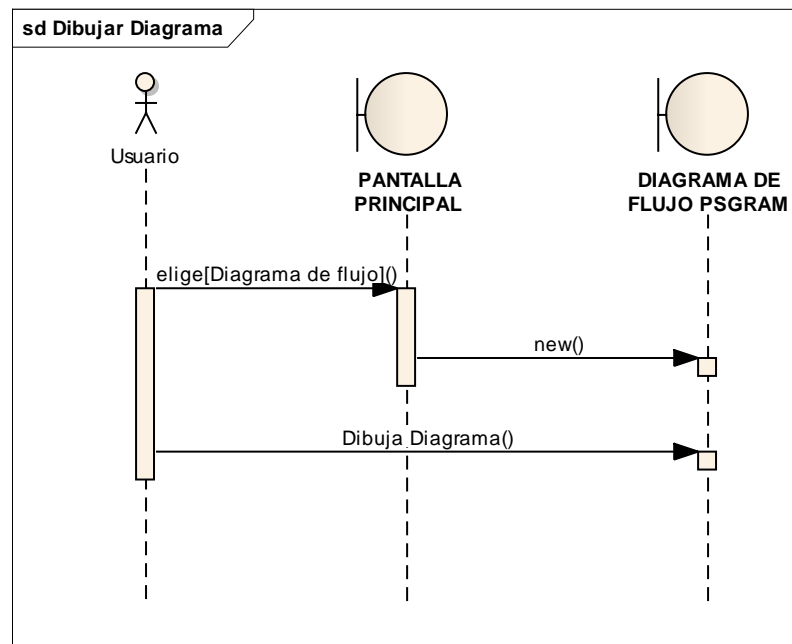


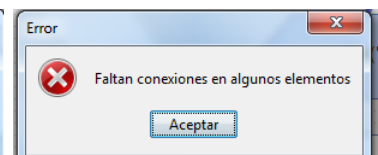
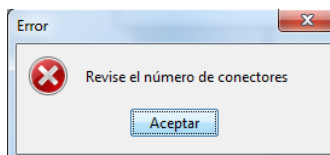
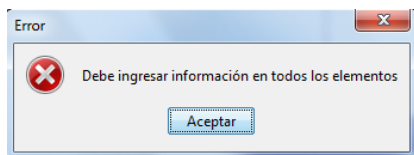
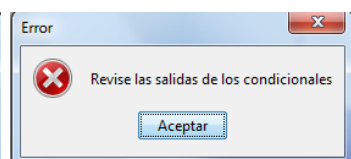
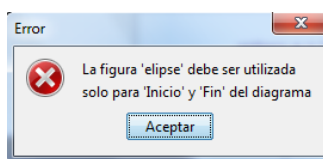
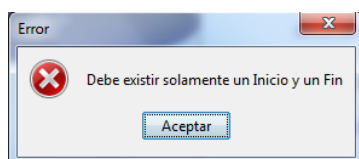
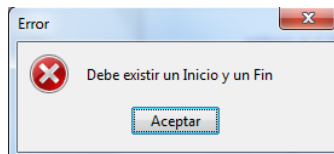
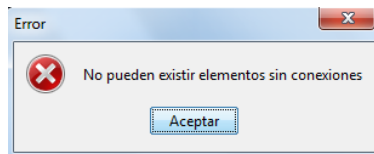
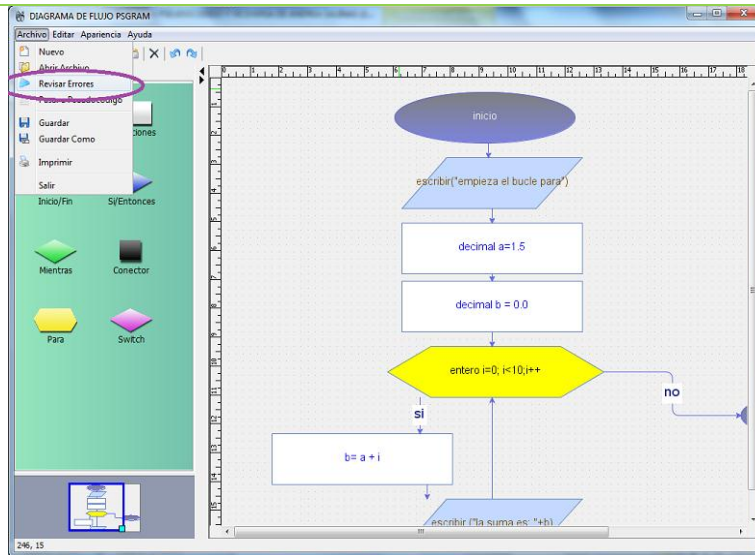
Diagrama 9 . Diagrama de Secuencia del Caso de Uso: Dibujar Diagrama

1.4.8. CASO DE USO: Revisar Errores

Nombre Pantalla: DIAGRAMA DE FLUJO PSGRAM	Código: PP008
--	----------------------

Tipo Interfaz Gráfica: JFrame

Caso de Uso: Revisar Errores



Realizado por:
 Andrea Natasha Salinas Ochoa.

Fecha:
 09/06/2011

Tabla 30 . Prototipo de Pantalla del Caso de Uso: Revisar Errores.

Nombre C.U: Revisar Errores		Código C.U: 0008
Req. Funcional:	RF011	
Objetivo(s):	Idibujar nuestro propio Diagrama de Flujo.	
Descripción:	El usuario mientras esta desarrollando su diagrama de flujo podrá revisar si el diagrama tiene algún error como: la falta de Inicio/Fin, que todos sus vértices estén conectados, escrito "si" o "no" en los bordes correspondientes cuando sea necesario, determinados vertices no pueden ser usados para fines diferentes, entre otros. Al existir un error será notificado al usuario, y nose podrá pasar al caso de uso "Generar Pseudo-Codigo" hasta no corregirlos.	
Actor(s):	Usuario	
Tipo Caso Uso:	Sistema	
Pre-condiciones:	Se haya ejecutado el caso de uso "Dibujar Diagrama".	
Post-condiciones:	El usuario tendrá un diagrama de flujo sin errores entre las conecciones de sus vértices y bordes.	
Flujo Normal de Eventos:		
<ol style="list-style-type: none"> 1. En la ventana "DIAGRAMA DE FLUJO PSGRAM", el usuario selecciona la opción "Revisar Errores" del menú "Archivo". 2. El sistema revisa que no hayan errores existentes. 3. El sistema regresa al paso 3 del fujo normal de eventos del caso de uso "Dibujar Diagrama". 4. El caso de uso finaliza. 		
Flujo alterno de Eventos:		
A. EERORES GENERADOS (Error)		
A.2. El sistema muestra los siguientes mensajes:		
<p>"No pueden existir elementos sin conecciones".</p> <p>"Debe existir un Inicio y un Fin"</p> <p>"Debe existir solamente un Inicio y un Fin"</p> <p>"La figura "elipse" debe ser utilizada solo para "Inicio" y "Fin" del diagrama"</p> <p>"Revise las salidas de los condicionales"</p> <p>"Debe ingresar la información en todos los elementos"</p> <p>"Revise el número de conectores"</p> <p>"Falta conexiones en algunos elementos"</p>		

1.4.9. CASO DE USO: Generar Pseudo-Codigo

Nombre Pantalla: DIAGRAMA DE FLUJO PSGRAM	Código: PP009
Tipo Interfaz Gráfica: JFrame	
Caso de Uso: Generar Pseudo-Codigo	
<p style="text-align: center;"> Genera </p>	
Realizado por: Andrea Natasha Salinas Ochoa.	Fecha: 09/06/2011

Tabla 32 . Prototipo de Pantalla del Caso de Uso: Generar Pseudo-Codigo.

Nombre C.U: Generar Pseudo-Codigo		Código C.U: 0009
Req. Funcional:	RF012	
Objetivo(s):	Generar el Pseudo-Código del Diagrama de flujo ingresado.	
Descripción:	El usuario después de realizar el caso de uso “Revisar Errores”, podrá pasar el Diagrama de Flujo a su Pseudo-Código correspondiente y una vez este código se encuentre en la ventana “EDITOR” podrá analizarlo (lexicamente, sintácticamente y semanticamente) para ver si el código que ingresé en el diagrama de flujo esta correcto.	
Actor(s):	Usuario	
Tipo Caso Uso:	Sistema	
Pre-condiciones:	Se haya ejecutado el caso de uso “Revisar Errores”.	
Post-condiciones:	El usuario tendrá un Pseudo-Código a partir del Diagrama de Flujo que ingresó.	
Flujo Normal de Eventos:		
<ol style="list-style-type: none"> 1. En la ventana “DIAGRAMA DE FLUJO PSGRAM”, el usuario selecciona la opción “Pasar a Pseudocodigo” del menú “Archivo”. 2. El sistema solo ejecuta el paso 2 del flujo normal de eventos del caso de uso “Revisar Errores” 3. El sistema muestra el cuadro de diálogo de “Guardar”. 4. El usuario ingresa en nombre del archivo y selecciona la ubicación donde va almacenar. 5. El usuario presiona el botón “Guardar”. 6. El sistema guarda el archivo en la dirección especificada. 7. El sistema carga el Pseudo-Código generado del Diagrama de Flujo en la ventana “EDITOR”. 8. El caso de uso finaliza. 		
Flujo alterno de Eventos:		
A. ERORES GENERADOS (Error)		
A.2. Si existe Errores el sistema muestra los errores del <i>Fujo Alterno de Eventos</i> del caso de uso “Revisar Errores”.		
A.3. El sistema regresa al paso 3 del fujo normal de eventos del caso de uso “Dibujar Diagrama”.		

Tabla 33 . Descripción del Caso de Uso: Generar Pseudo-Codigo.

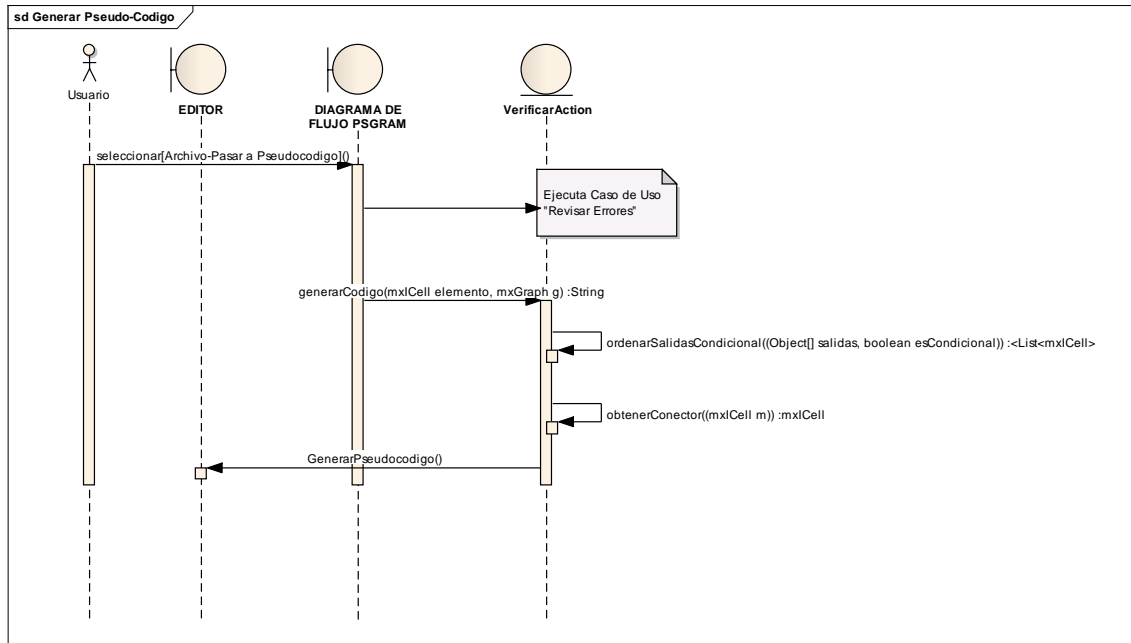
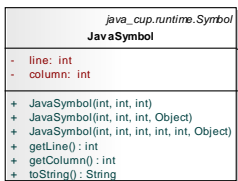
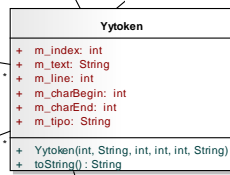
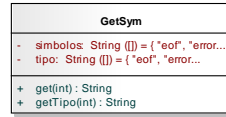
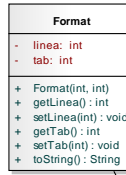
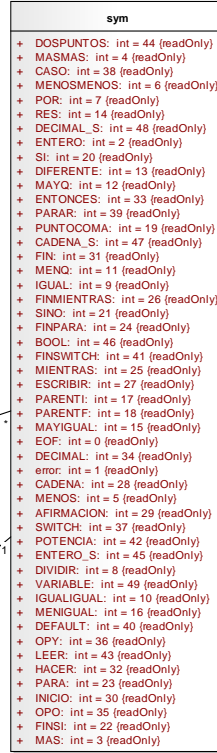
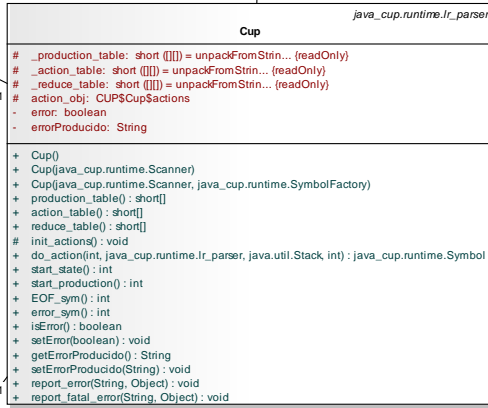
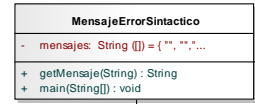
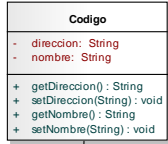
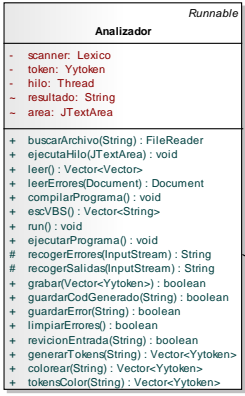


Diagrama 11 . Diagrama de Secuencia del Caso de Uso: Generar Pseudo-Codigo

1.8. DIAGRAMA DE CLASES FINAL

edu.psg.editor.dominio

Diagrama 13. Diagrama de clases del paquete edu.psg.editor.dominio





edu.psg.editor.negocio

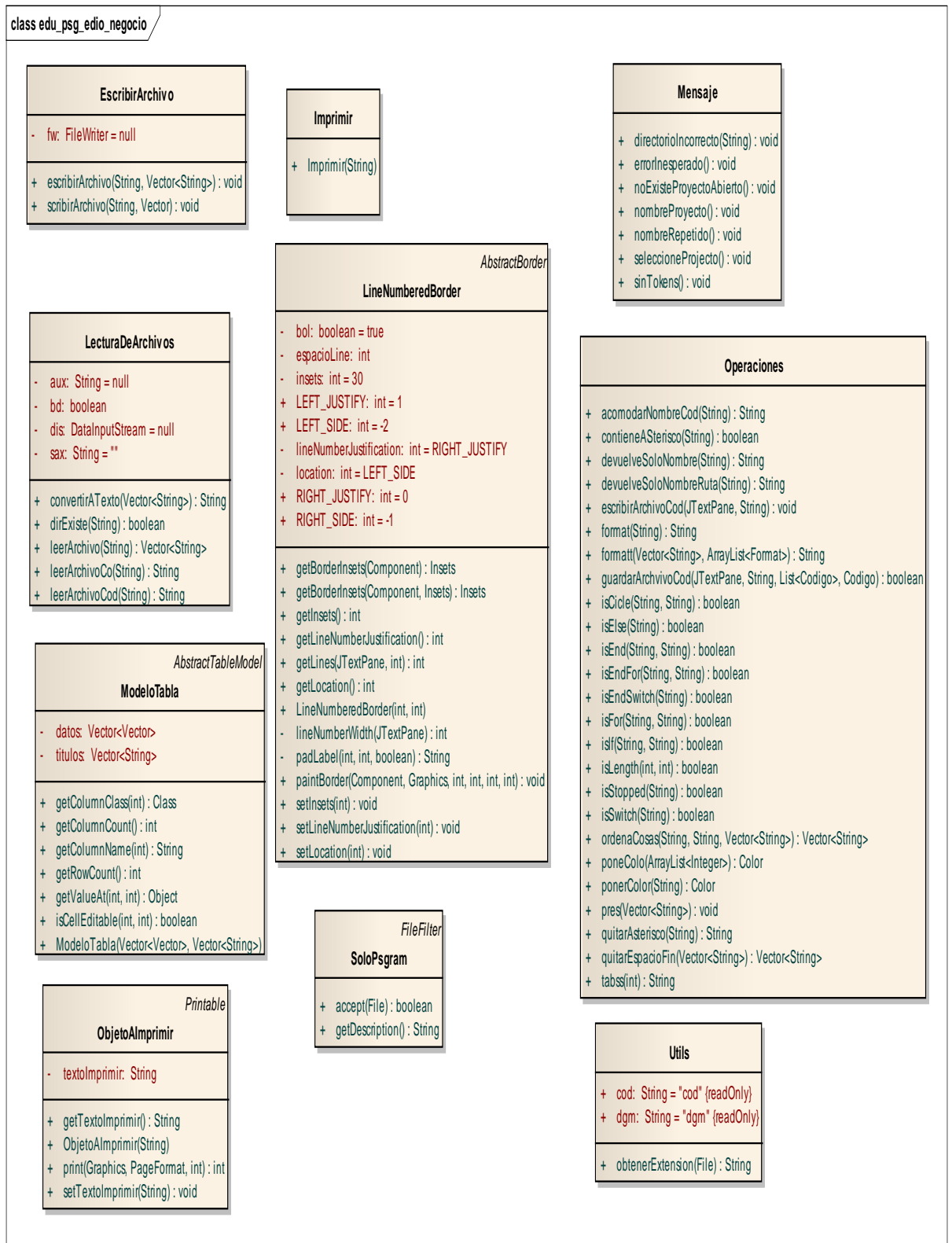


Diagrama 14 .Digrama de clases del paquete edu.psg.editor.negocio

edu.psg.editor.vista

Diagrama 15. Diagrama de clases del paquete edu.psg.editor.vista



edu.psg.graficador.
dominio

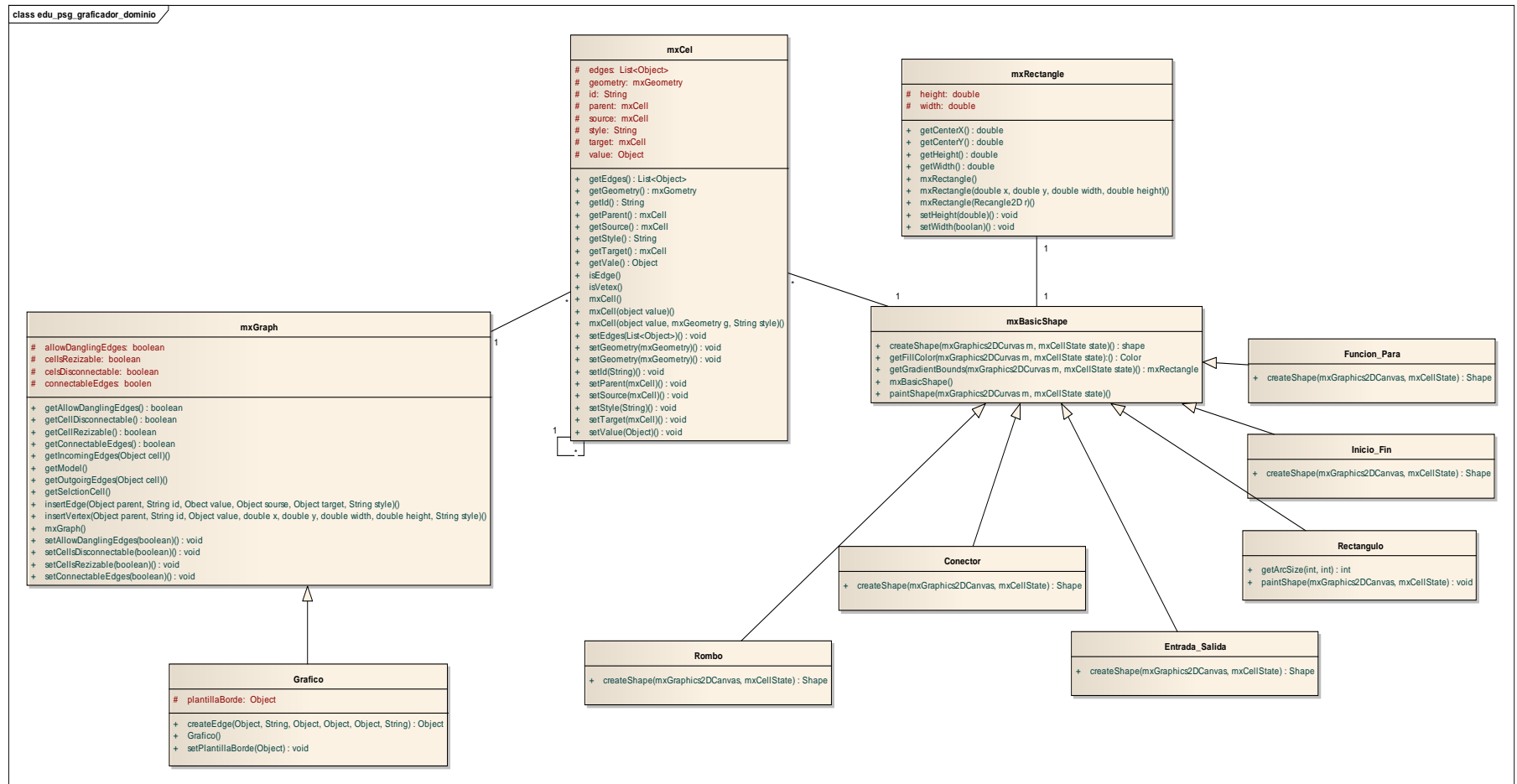


Diagrama 16 . Digrama de clases del paquete edu.psg.graficador.dominio



edu.psg.graficador. negocio



Diagrama 17 . Digrama de clases del paquete edu.psg.graficador.negocio



edu.psg.graficador. vista

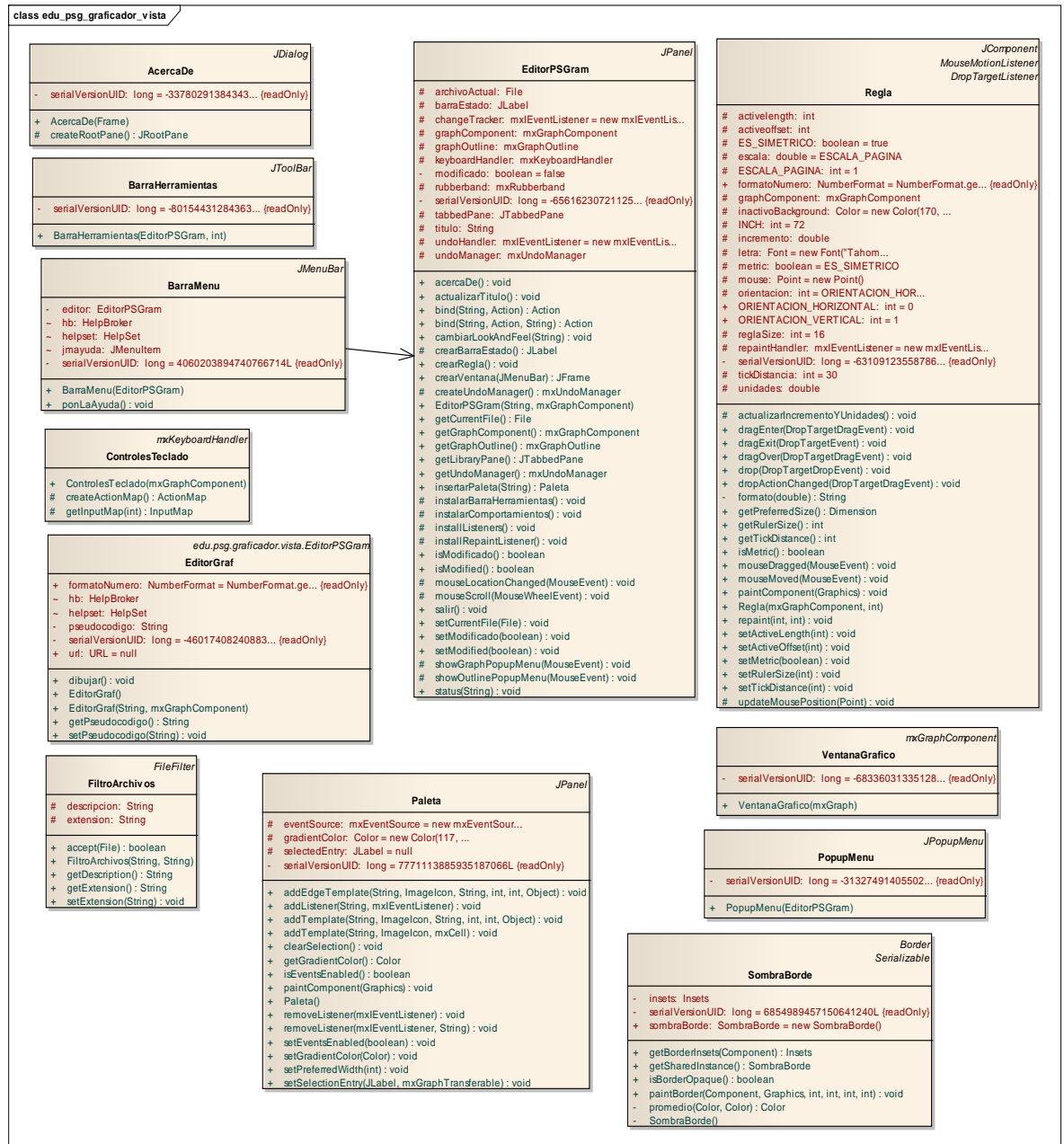


Diagrama 18 . Digrama de clases del paquete edu.psg.graficador.vista

1.9. DIAGRAMA DE PAQUETES

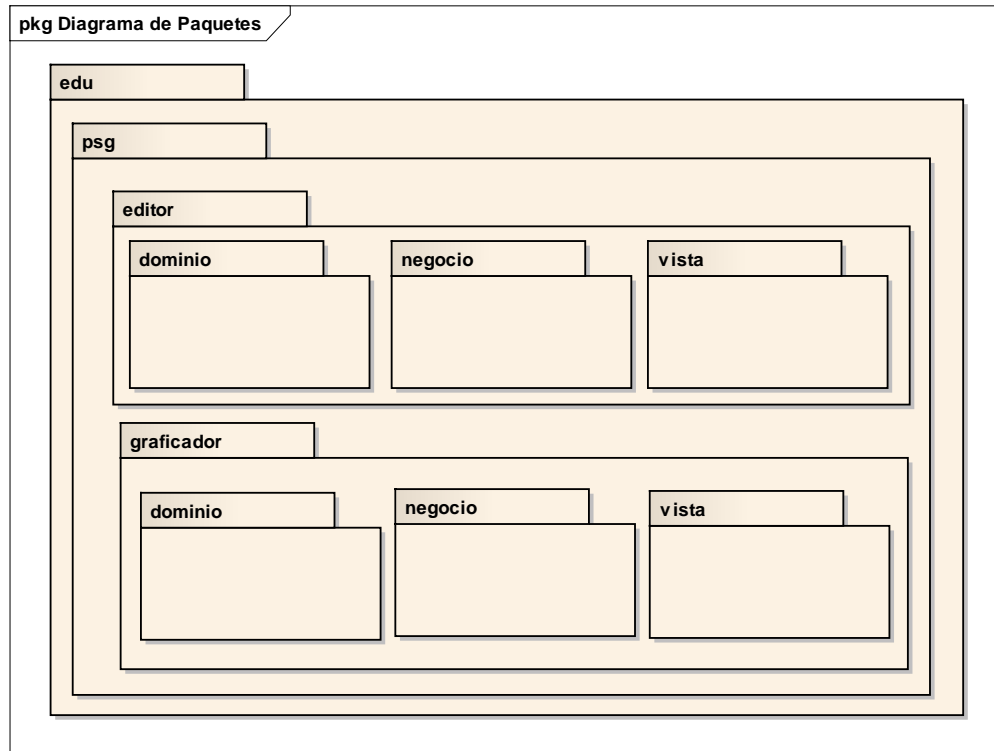


Diagrama 19 .Diagrama de Paquetes.

1.10. MODELO DE ARQUITECTURA

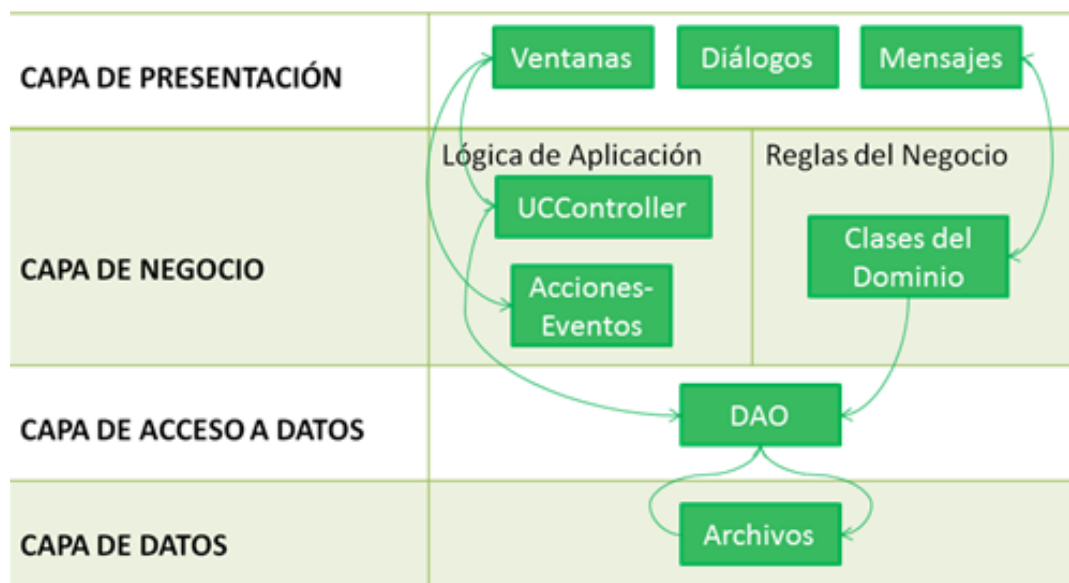


Diagrama 20 . Modelo de Arquitectura.

1.11. DIAGRAMA DE COMPONENTES

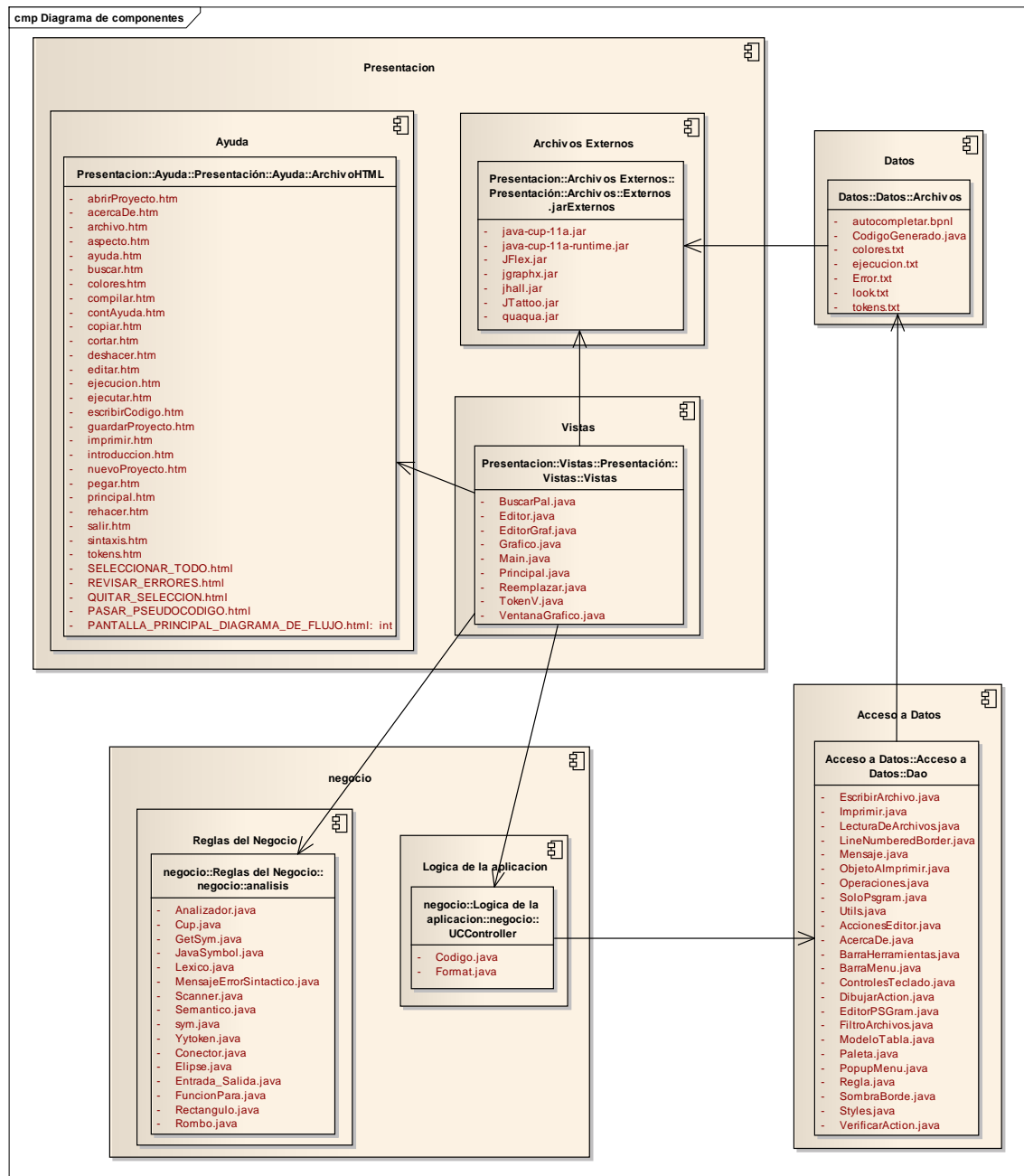


Diagrama 21 .Diagrama de Componentes

H. CONCLUSIONES

Luego de haber realizado el presente proyecto se puede concluir lo siguiente:

- El uso de encuestas permite determinar los requerimientos que el software debe cumplir.
- La utilización herramientas como: JFlex y Java_Cup facilitan el desarrollo del compilador.
- La correcta definición de los tokens en el desarrollo del análisis léxico, la estructura de la gramática en el análisis sintáctico y los respectivos controles semánticos son la base fundamental para la construcción de un compilador.
- El JGraphX es una librería de java que facilita trabajar con gráficos.
- En el programa PsGram los diagramas de flujo fueron diseñados con figuras geométricas, favoreciendo la comprensión a simple vista.
- El software permite obtener un Diagrama de Flujo a partir de su respectivo Pseudo-Código; presentando información clara y concisa, ayudando a comprender su flujo lógico.
- PsGram genera el Pseudo-Código de su respectivo Diagrama de Flujo, permitiendo hacer su compilación y ejecución.
- PsGram permite crear, abrir, almacenar y corregir los archivos tanto de Pseudo-Código como Diagrama de Flujo, mejorando el aprovechamiento de la información por parte del usuario.
- PsGram será un facilitador entre el estudiante y docente, ayudando a obtener buenos conocimientos para un futuro(a) programador(a).

I. RECOMENDACIONES

- ✿ Investigar minuciosamente el funcionamiento y uso de nuevas herramientas que faciliten el desarrollo de compiladores definidos en lenguaje natural.
- ✿ En el desarrollo de compiladores para Pseudo-códigos, definir concretamente el alfabeto a utilizar, la gramática del contexto y los controles semánticos necesarios para su buen funcionamiento.
- ✿ Investigar el funcionamiento y uso de nuevas herramientas que faciliten el desarrollo de aplicaciones para Diagrama de Flujo.
- ✿ Se debe tener en cuenta los requerimientos mínimos de software y hardware que se encuentran especificados en el manual de usuario, para el correcto funcionamiento de PsGram.
- ✿ La entidad universitaria facilite a los alumnos este software como herramienta de apoyo para afianzar sus conocimientos.

J. BIBLIOGRAFÍA

1. LIBROS:

- ✿ DEAN, Kelley. 1995. Teoría de Autómatas y Lenguajes Formales. Editorial PRENTICE HALL.
- ✿ GÁLVEZ ROJAS, Sergio Y MORA MATA, Miguel Ángel. .2005, Java a Tope: Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC [Consulta: 17 enero 2011].
- ✿ WEITZENFELD, Alfredo. 2004, Ingeniería de Software Orientada a Objetos con Uml. Java e Internet. Thomson Editores.

2. RECURSOS DE INTERNET:

- ✿ AHO, Alfred y ULLMAN, Jeffrey.2005, Compiladores Principios, Técnicas y Herramientas, Edición Electrónica,[Consulta: 17 enero 2011].
- ✿ TREJO AVILA, Mary Carmen. 2 de septiembre de 2004, RELIPMOC: Construcción de un Compilador Básico haciendo uso de las herramientas JLex y CUP, Edición Electrónica, [Consulta: 17 enero 2011].

3. SITIOS WEB:

- ✿ GNU. La Definicion de Software Libre. [en línea]. [http://www.gnu.org/philosophy/free-sw.es.html], [Consulta: 19 enero 2011].
- ✿ PONS VIVANCO, Ramon.2003-08-12.[http: //laurel.datsi.fi.upm. es/~rpons/ gjsc /sintactico/node5.html],[Consulta: 17 enero 2011]
- ✿ MILLÁN, Antonio ,FERMÍN ,Gerardo y CHACÓN, José. Diagrama de Flujo. [en línea]. Puerto Ordaz, [http://www.monografias.com/trabajos59/diagrama-flujo/diagrama-flujo.shtml], [Consulta: 17 enero 2011].
- ✿ BONILLA,Oscar. [en línea] Compiladores, Universidad Galileo, [http://74.125.45.104/search?q=cache:A9YLY0RcmuUJ:oscarbonilla.com/courses/compilers/materials/06_Analisis_Sintactico.ppt+analizador+sint%C3%A1ctico&hl=es &ct=clnk&cd=11], [Consulta: 17 enero 2011].

-
- ✿ CARRETO, Julio, Ing. Símbolos Gráficos. [en línea] Produccion e Inventarios, [<http://uproprod.blogspot.com/2007/08/aprenda-crear-diagramas-de-flujo.html>], [Consulta: 17 enero 2011].
 - ✿ Programación de computadores. Pseudocódigo. [en línea]. Bogota-Colombia, Universidad Nacional de Colombia. [http://www.virtual.unal.edu.co/cursos/ingenieria/2001839/modulo1/cap_02/leccion1022.htm], [Consulta: 17 enero 2011].

K. ANEXOS

ANEXO A. Análisis de Resultados

Análisis de Resultados


Modelo de la encuesta que realice para obtener los requerimientos, a los alumnos de Cuarto Módulo de la Carrera de Ingeniería en Sistemas.



UNIVERSIDAD NACIONAL DE LOJA

Área de la Energía, las Industrias y los Recursos Naturales No Renovables

Carrera de Ingeniería en Sistemas

ENCUESTA

Encuesta de recolección de información necesaria, para el desarrollo e una herramienta de software que permita graficar un diagrama de flujo generando su pseudo-Código y de un pseudo-Código obtener su diagrama de flujo.

1. ¿Los diagramas de flujo por ser diseñados con símbolos o figuras, favorece la comprensión del proceso del programa a simple vista?
 Si ()
 No ()
2. ¿Los diagramas de flujo son una excelente herramienta para capacitarse y realizar mejoras en el proceso?
 Si ()
 No ()
3. ¿los diagramas complejos y detallados suelen ser laboriosos en su planteamiento y diseño a mano: por eso cree necesario un software en el cuál exista facilidad de modificaciones tanto del diagrama de flujo como del Pseudo-código?
 Si ()
 No ()
4. ¿El Pseudo-código ayuda a interpretar el algoritmo al programador facilitando su conversión en lenguaje de alto nivel; desea que el software sea en español para su mayor comprensión?
 Si ()
 No ()
 ¿En caso de ser esta pregunta su respuesta negativa, que idioma desearía?

5. ¿Desearía que el software tenga una opción de “Guardar” sus diagramas de flujo y pseudo-código ingresados?
 Si ()
 No ()
Alguna sugerencia:

6. ¿Desea que la interfaz del software sea amigable y fácil de entender el manejo?
 Si ()
 No ()

Alguna sugerencia:

7. ¿Desea que el software detecte los errores antes de hacer la conversión de diagrama de flujo a pseudo-código y viceversa?
 Si ()
 No ()

Alguna sugerencia:

8. ¿Necesitaría que el software permita deshacer acciones?
 Si ()
 No ()

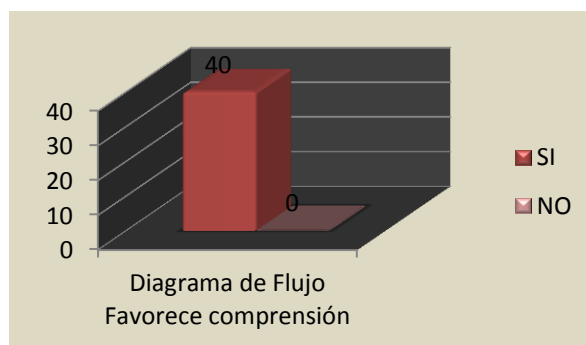
Tabla 34 . Modelo de encuesta para obtener requerimientos.

Tabulación de la encuesta que realice para obtener los requerimientos, a los alumnos de Cuarto Módulo de la Carrera de Ingeniería en Sistemas.

1. ¿Los diagramas de flujo por ser diseñados con símbolos o figuras, favorece la comprensión del proceso del programa a simple vista?

Respuestas	Valor	Porcentaje
SI	40	100%
NO	0	0%
TOTAL	40	100%

Tabla 35 . Diagrama de Flujo favorece comprensión.

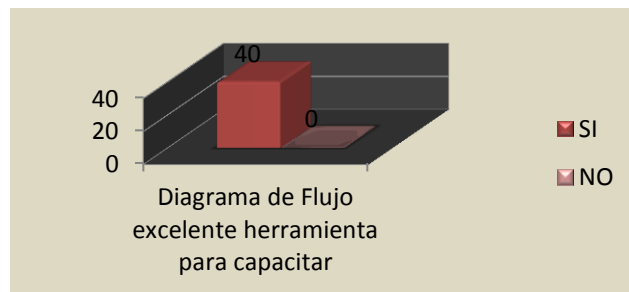


21. Ilustración: Diagrama de Flujo favorece comprensión.

2. ¿Los diagramas de flujo son una excelente herramienta para capacitarse y realizar mejoras en el proceso?

Respuestas	Valor	Porcentaje
SI	40	100%
NO	0	0%
TOTAL	40	100%

Tabla 36 . Diagrama de Flujo excelente herramienta para capacitar.

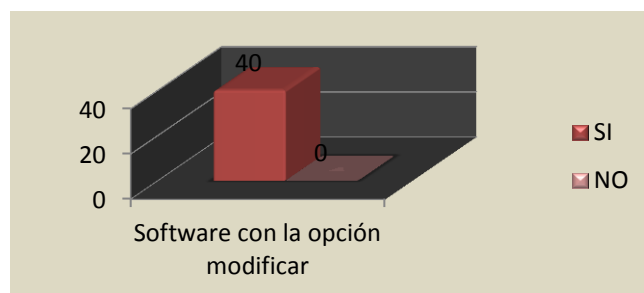


22. Ilustración: Diagrama de Flujo excelente herramienta para capacitar.

3. ¿Los diagramas complejos y detallados suelen ser laboriosos en su planteamiento y diseño a mano: por eso cree necesario un software en el cuál exista facilidad de modificaciones tanto del diagrama de flujo como del Pseudo-código?

Respuestas	Valor	Porcentaje
SI	40	100%
NO	0	0%
TOTAL	40	100%

Tabla 37 . Software con la opción modificar.

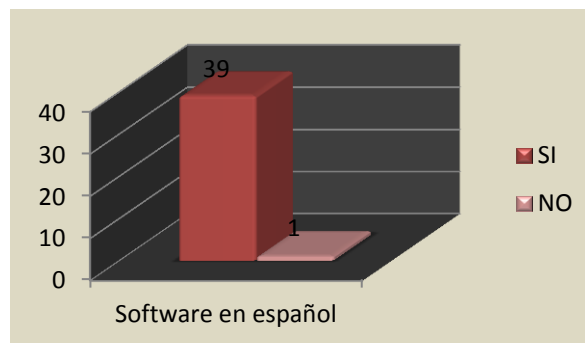


23. Ilustración: Software con la opción modificar.

4. ¿El Pseudo-código ayuda a interpretar el algoritmo al programador facilitando su conversión en lenguaje de alto nivel; desea que el software sea en español para su mayor comprensión?

Respuestas	Valor	Porcentaje
SI	39	97,5%
NO	1	2,5%
TOTAL	40	100%

Tabla 38 . Software en español.



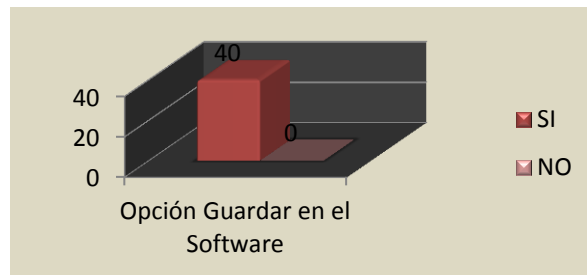
24. Ilustración: Software en español.

La mayor parte de los encuestados respondieron que sean el software en español y una persona aspiró que el software sea en inglés.

5. ¿Desearía que el software tenga una opción de “Guardar” sus diagramas de flujo y pseudo-código ingresados?

Respuestas	Valor	Porcentaje
SI	40	100%
NO	0	0%
TOTAL	40	100%

Tabla 39 . Opción Guardar en el Software.

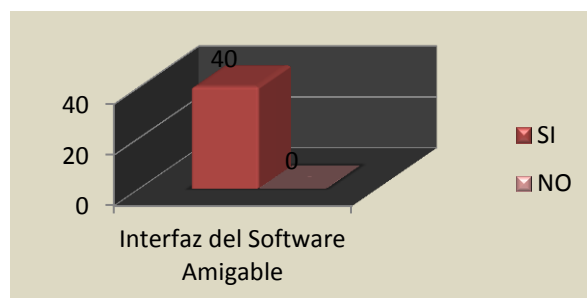


25. Ilustración: Opción Guardar en el Software.

6. ¿Desea que la interfaz del software sea amigable y fácil de entender el manejo?

Respuestas	Valor	Porcentaje
SI	40	100%
NO	0	0%
TOTAL	40	100%

Tabla 40 . Interfaz del Software Amigable.

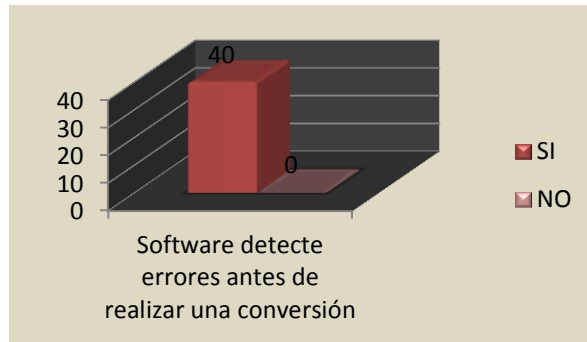


26 Ilustración: Interfaz del Software Amigable.

7. ¿Desea que el software detecte los errores antes de hacer la conversión de diagrama de flujo a pseudo-código y viceversa?

Respuestas	Valor	Porcentaje
SI	40	100%
NO	0	0%
TOTAL	40	100%

Tabla 41 . Software detecte errores antes de realizar una conversión.

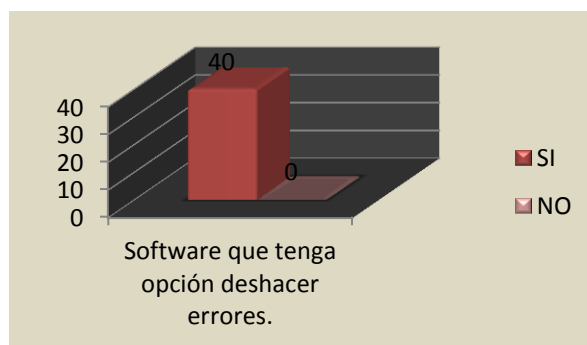


27. Ilustración: Software detecte errores antes de realizar una conversión.

8. ¿Necesitaría que el software permita deshacer acciones?

Respuestas	Valor	Porcentaje
SI	40	100%
NO	0	0%
TOTAL	40	100%

Tabla 42 . Software que tenga opción deshacer errores.



28. Ilustración: Software que tenga opción deshacer errores.



ANEXO B. Anteproyecto

Anteproyecto



UNIVERSIDAD NACIONAL DE LOJA

ÁREA DE LA ENERGÍA, LAS INDUSTRIAS Y LOS RECURSOS NATURALES NO RENOVABLES

Ingeniería en Sistemas

Anteproyecto

TÍTULO:

“DESARROLLO DE UNA HERRAMIENTA DE SOFTWARE QUE PERMITA GRAFICAR UN DIAGRAMA DE FLUJO GENERANDO SU PSEUDOCÓDIGO Y DE UN PSEUDOCÓDIGO OBTENER SU DIAGRAMA DE FLUJO, PARA LOS ALUMNOS DE LA CARRERA DE INGENIERÍA EN SISTEMAS DE LA UNIVERSIDAD NACIONAL DE LOJA.”

AUTOR:

Andrea Natasha Salinas Ochoa

Loja – Ecuador

2012



1. TITULO

“DESARROLLO DE UNA HERRAMIENTA DE SOFTWARE QUE PERMITA GRAFICAR UN DIAGRAMA DE FLUJO GENERANDO SU PSEUDOCÓDIGO Y DE UN PSEUDOCÓDIGO OBTENER SU DIAGRAMA DE FLUJO, PARA LOS ALUMNOS DE LA CARRERA DE INGENIERÍA EN SISTEMAS DE LA UNIVERSIDAD NACIONAL DE LOJA.”

2. PROBLEMÁTICA

2.1. Situación Problemática

En la actualidad los alumnos cuando empiezan a seguir la especialidad en su bachillerato o en su carrera universitaria de informática, los primeros conocimientos que se les otorgan es la unidad de metodología de programación, en el que encuentran las herramientas de programación como son el pseudocódigo y los diagramas de flujo.

Los diagramas de flujo han sido la herramienta de programación por excelencia, y en la actualidad son muy empleados debido a que son fáciles de diseñar porque el flujo lógico del algoritmo se muestra en un dibujo en lugar de palabras. Pero el Diagrama de Flujo presenta un problema que son difíciles de actualizar en lápiz y papel precisamente por su carácter gráfico; como su alternativa surgió el Pseudocódigo, que es un lenguaje algorítmico que permite un seguimiento y una redacción rápida de la lógica de un algoritmo y es más fácil de mantener y de convertir en programas escritos en lenguaje de programación específicos.

Teniendo de esta manera los alumnos problemas en desarrollar algunos ejercicios que son impartidos en clases, también el no poder de manera correcta interpretar de un pseudocódigo a diagrama de flujo y viceversa; y el no saber si lo que hizo esta correcto.

En Loja la mayor parte de establecimientos que se imparte esta unidad tiene el problema que los alumnos no captan inmediatamente, y es por eso que necesitan una ayuda para saber si van por buen camino o no.

En este caso en la Universidad Nacional de Loja, los alumnos que están empezando la carrera de Ingeniería en Sistemas se encuentran con este inconveniente de no tener una herramienta para saber si lo están desarrollando bien o no, debido a que todo lo hacen con lápiz y papel y no tienen una herramienta informática que les presente los errores y sugerencias ya sean léxicos, sintácticos, semánticos, mal utilización de los diagramas, entre otros.



Es por esto que surge la necesidad de crear un una herramienta de software que les permita traducir el pseudocódigo de ejercicios informáticos en diagramas de flujo y viceversa. Por lo problemas que a continuación les presento: no presentan sugerencias para la corrección de errores; los alumnos no tienen una herramienta especializada fuera del aula para saber si su diagrama o su pseudocódigo se encuentra bien desarrollado.

2.2. Problema General de investigación

“Falta de una herramienta informática que coadyuve a los alumnos que están iniciando la unidad de metodología de programación, a generar el pseudocódigo a partir de un diagrama de flujo y viceversa, en la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja.”.

2.3. Delimitación

- Módulo de PSEUDOCÓDIGO
- Módulo de DIAGRAMA DE FLUJO.
- Módulo de DIAGRAMA DE FLUJO a PSEUDOCÓDIGO y viceversa.

2.3.1. Problemas específicos de investigación

Luego del análisis realizado a los alumnos que recién están cursando la materia de metodología de programación en la Carrera de Ingeniería en Sistema perteneciente a la Universidad Nacional de Loja he podido determinar las siguientes problemáticas:

- Inexistencia de una aplicación que permita a los alumnos ingresar su pseudocódigo y verificar si tiene algún: error léxico, error sintáctico o error semántico.
- Los alumnos tienen problemas al realizar un diagrama de flujo porque no hay una aplicación que les permita diseñar e indique si tiene algún error de escritura o esta sin sentido su estructura.
- Los alumnos tienen problemas al convertir de un diagrama de flujo a pseudocódigo y viceversa, porque no pueden darse cuenta rápidamente en lápiz y papel donde se encuentra el error.

2.3.2. Espacio

El presente proyecto de investigación tiene como escenario a los alumnos, que están empezando a tener conocimientos básicos de la programación en adelante, de la carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja, perteneciente a la provincia y ciudad de Loja.

2.3.3. Tiempo

El presente proyecto de investigación, tiene una planificación de acuerdo a los parámetros a realizarse que se encuentran detallados en el cronograma de actividades



con duración de 12 meses. Además en este tiempo están incluidas las aprobaciones del proyecto.

2.3.4. Unidades de Observación

Las unidades de observación para realizar este proyecto son los alumnos de tercer modulo de la carrera de Ingeniería en Sistemas, los cuales nos ayudaran a identificar los requerimientos para poderles dar solución; y el uso de software libre.

3. JUSTIFICACION

3.1. Justificación

3.1.1. Justificación Académica.

La Universidad Nacional de Loja como centro de educación superior, con una notable importancia en la Región Sur del País, brinda a sus egresados y profesionales la oportunidad de aportar, los conocimientos adquiridos durante el transcurso de la carrera, involucrando consigo a la sociedad.

Este proyecto se justifica académicamente en la necesidad innata que tenemos como egresados de poner en práctica todos los conocimientos adquiridos durante nuestra vida de formación profesional en la carrera de Ingeniería en Sistemas, de esta manera tomando en consideración los problemas que presentan los alumnos de no tener una herramienta de ayuda en la transformación de diagramas de flujo a pseudocódigo y viceversa buscamos plantear soluciones en conceptos informáticos, aportando de esta manera al desarrollo de la sociedad.

Con el desarrollo del presente proyecto además de ayudar a solucionar problemas para la sociedad me ayuda a profundizar y a adquirir nuevos conocimientos.

3.1.2. Justificación Técnica.

Para la realización del presente proyecto cuento con todos los medios técnicos (computadoras, impresora, etc.) y herramientas (software libre, etc.) necesarias, que usare a lo largo de su desarrollo. Además es factible de realizar porque puedo acceder a diversos medios de consultas bibliográficas como libros, recursos de internet, etc. Con la finalidad de obtener la información necesaria que me permita sustentar este proyecto.

3.1.3. Económica.

Económicamente de justifica este proyecto porque cuento con los medios económicos necesarios para solventar el desarrollo de este proyecto, para ello utilizare software gratuito, fuentes internas, y demás, las cuales se encuentran detalladas a continuación



3.2. Viabilidad

Este proyecto de investigación es viable, debido a que cuento con los medios: técnicos, tecnológicos y económicos, que se requiere para su realización; además de contar con el apoyo de los alumnos y de la asesoría de los docentes de la Carrera de Ingeniería en Sistemas.

4. OBJETIVOS

4.1. General

Desarrollar una herramienta que permita graficar Diagramas de Flujo generando su respectivo Pseudocódigo y que a partir de un Pseudocódigo genere su Diagrama de Flujo, para los alumnos de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja.

4.2. Específicos

- Analizar y recolectar información para el desarrollo de la herramienta.
- Desarrollar un analizador léxico, sintáctico y semántico para el pseudocódigo.
- Elaborar un módulo gráfico que permita implementar los diferentes componentes del diagrama de flujo.
- Desarrollar un módulo que permita convertir un lenguaje de pseudocódigo a diagrama de flujo y de un diagrama de flujo a pseudocódigo.
- Realizar pruebas de la aplicación realizada.

5. MARCO TEORICO

CAPITULO I

1. COMPILADORES

- 1.1. Definición
- 1.2. Estructura
- 1.3. Analizador Léxico
 - 1.3.1. Definición
 - 1.3.2. Funciones
 - 1.3.3. Tokens, lexemas y patrones
- 1.4. Analizador Sintáctico
 - 1.4.1. Definición
 - 1.4.2. Visión General
 - 1.4.3. El papel de Analizador sintáctico
 - 1.4.4. Manejo de errores sintácticos
 - 1.4.5. Gramáticas independientes del contexto
- 1.5. Analizador Semántico
 - 1.5.1. Introducción
 - 1.5.2. Definición
 - 1.5.3. Atributos y acciones semánticas
 - 1.5.4. Ejecución de una acción semántica
 - 1.5.5. Análisis dirigido por sintaxis
 - 1.5.5.1. Gramática con atributos
 - 1.5.5.2. Métodos de evaluación de los atributos
 - 1.5.6. Alfabetos, Palabras, Lenguajes
- 1.6. Lenguajes Regulares
- 1.7. Autómata finito determinista
- 1.8. Autómata finito no determinista

CAPITULO II

2. INTÉRPRETE

- 2.1. Diferencias entre Intérpretes y Compiladores.

CAPITULO III

3. DIAGRAMAS DE FLUJO

- 3.1. Concepto
- 3.2. Características
- 3.3. Ventajas de los Diagramas de Flujo
- 3.4. Símbolos para la creación de diagramas de flujo y su significado
- 3.5. Símbolos Gráficos

CAPITULO IV

4. PSEUDOCÓDIGO

- 4.1. Definición
- 4.2. Características y partes
- 4.3. Funciones y operaciones
- 4.4. Estructuras de control
 - 4.4.1. Estructuras secuenciales
 - 4.4.2. Estructuras selectivas
 - 4.4.2.1. Selectiva doble (alternativa)
 - 4.4.2.2. Selectiva múltiple
 - 4.4.2.3. Selectiva múltiple-Casos
 - 4.4.3. Estructuras iterativas
 - 4.4.3.1. Bucle mientras
 - 4.4.3.2. Bucle repetir
 - 4.4.3.3. Bucle para
 - 4.4.3.4. Bucle para cada

CAPITULO V

5. HERRAMIENTAS TENTATIVAS A UTILIZAR

- 5.1. Software Libre

CAPITULO I

1. Compiladores⁶

1.1. Definición

Un compilador es un programa que lee un programa escrito en un lenguaje, el lenguaje fuente, y lo traduce a un programa equivalente en otro lenguaje, el lenguaje objeto. Como parte importante de éste proceso de traducción, el compilador informa a su usuario de la presencia de errores en el programa fuente.

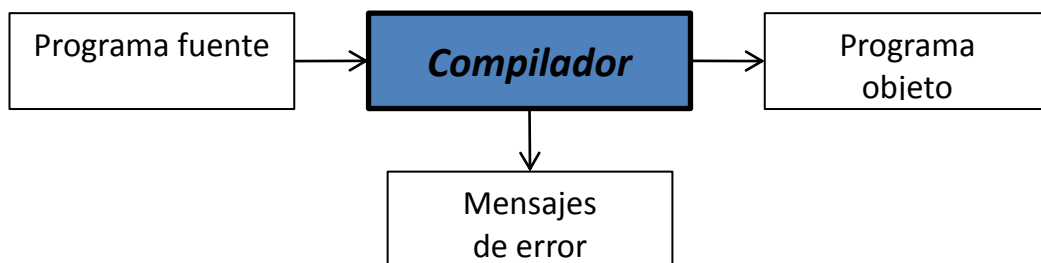


Fig.1. Funcionamiento del compilador

Hay miles de lenguajes fuente, desde los lenguajes de programación tradicionales, hasta los lenguajes especializados que han surgido virtualmente en todas las áreas de aplicación de la información.

Lenguajes objeto son igualmente variados; un lenguaje objeto puede ser otro lenguaje de programación o el lenguaje de máquina de cualquier computador entre un microprocesador y un supercomputador. Los compiladores a menudo se clasifican como una pasada, de múltiples pasadas, de carga y ejecución, de depuración o de optimización, dependiendo de cómo hayan sido construidos o de qué función se suponen que realizan. A pesar de esta aparente complejidad, las tareas básicas que debe realizar cualquier compilador son esencialmente las mismas.

⁶BONILLA, Oscar. [http://74.125.45.104/search?q=cache:A9YLY0RcmuUJ:oscarbonilla.com/courses/compilers/materials/06_Analisis_Sintactico.ppt+analizador+sint%C3%A1ctico&hl=es&ct=clnk&cd=11]

1.2. Estructura

La estructura de un compilador puede listarse de la siguiente manera:

1. Análisis léxico.
2. Análisis sintáctico.
3. Análisis semántico.
4. Generación de código intermedio.
5. Optimización de código intermedio.
6. Generación de código objeto.

Con cada una de estas fases interactúa la Tabla de símbolos y la Gestión de errores.

1.3. Analizador Léxico⁷

1.3.1. Definición

El analizador léxico (scanner), lee un texto fuente y lo transforma en una secuencia ordenada de elementos léxicamente válidos. Un carácter o conjunto de estos que constituya un componente léxico se llama lexema (token). Como componentes léxicos consideramos: palabras reservadas, separadores, operadores, identificadores, constantes y signos de puntuación.

1.3.2. Funciones

Las principales funciones de un analizador léxico son:

- Manejar el archivo fuente (abrirlo, leerlo, cerrarlo).
- Generar y entregar tokens bajo la petición del analizador sintáctico.
- Rechazar un carácter o conjunto de estos que no concuerden con patrones especificados.
- Entendamos como patrón una expresión regular que se define en el lenguaje.
- Ignorar comentarios, espacios en blanco y tabuladores.

⁷ AHO, Alfred y ULLMAN, Jeffrey.2005, Compiladores Principios, Técnicas y Herramientas, Edición Electrónica

- Reconocer las palabras reservadas del lenguaje.
- Gestionar errores, contando los saltos de línea y asociando los mensajes de error con el número de la línea del archivo fuente donde se producen.
- Guardar tokens junto con su atributo en una tabla de símbolos. Este atributo es información adicional relevante, habitualmente con relación a los identificadores.

1.3.3. Tokens, lexemas y patrones

Algunas definiciones:

- **Token:** “nombre “que se da a cada componente léxico.
- **Lexema:** secuencia de caracteres de la entrada que corresponden a un token.
- **Patrón:** forma compacta de describir conjuntos de lexemas.

Además:

- Un token se corresponde con un patrón
- Un token se puede corresponder con muchos lexemas

Tokens más habituales:

- Palabras reservadas
- Identificadores
- Operadores
- Constantes
- Símbolos de puntuación: ; , . :
- Símbolos especiales: () []

Pero, a la vez que el propio token, el scanner puede (debe) devolver más información:

- Si es un token CONSTANTE, su valor
- Si es un identificador, el string correspondiente
- Si es un símbolo de puntuación, cuál

Esta información, se devuelve mediante “atributos”. Pero aún puede hacer algo más:

- Puede detectar algunos (pocos) errores léxicos
 - No hay concordancia con ningún patrón
- Puede llevar a cabo algunas recuperaciones de errores
 - Filtrado de caracteres “extraños”
 - Completar algún patrón

- Reemplazar algún carácter

1.4. Analizador Sintáctico⁸

1.4.1. Definición

Es la fase del analizador que se encarga de chequear la secuencia de *tokens* que representa al texto de entrada, en base a una gramática dada. En caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce en base a una representación computacional. Este árbol es el punto de partida de la fase posterior de la etapa de análisis: el analizador semántico.

1.4.2. Visión General

Todo lenguaje de programación obedece a unas reglas que describen la estructura sintáctica de los programas bien formados que acepta. Se puede describir la sintaxis de las construcciones de los lenguajes de programación por medio de gramáticas de contexto libre.

Las gramáticas formales ofrecen ventajas significativas a los diseñadores de lenguajes y a los desarrolladores de compiladores:

- Las gramáticas son especificaciones sintácticas y precisas de lenguajes de programación.
- A partir de una gramática se puede generar automáticamente un analizador sintáctico.
- El proceso de generación automática anterior puede llevar a descubrir ambigüedades.
- Una gramática proporciona una estructura a un lenguaje de programación, siendo más fácil generar código y detectar errores.
- Es más fácil ampliar/modificar el lenguaje si está descrito con una gramática.

⁸ GÁLVEZ ROJAS, Sergio Y MORA MATA, Miguel Ángel. .2005, Java a Tope: Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC, Edición Electrónica
Universidad Nacional de Loja

El analizador sintáctico dirige el proceso de compilación, de manera que el resto de fases evolucionan a medida que el sintáctico va reconociendo la secuencia de entrada por lo que, a menudo, el árbol ni siquiera se genera realmente.

En la práctica, el analizador sintáctico también:

- Incorpora acciones semánticas en las que colocar el resto de fases del compilador (excepto el analizador léxico): desde el análisis semántico hasta la generación de código.
- Informa de la naturaleza de los errores sintácticos que encuentra e intenta recuperarse de ellos para continuar la compilación.
- Controla el flujo de *tokens* reconocidos por parte del analizador léxico.

1.4.3. El papel de Analizador sintáctico

El analizador obtiene una cadena de componentes léxicos del analizador léxico, como se muestra en la siguiente figura, y comprueba si la cadena puede ser generada por la gramática del lenguaje fuente. El analizador sintáctico informará de cualquier error de sintaxis de manera inteligente. También debería recuperarse de los errores que ocurren frecuentemente para poder continuar procesando el resto de su entrada.

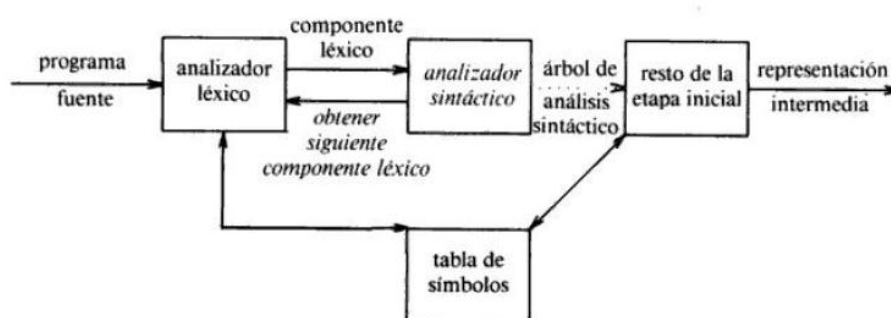


Fig. 2. Posición del Analizador sintáctico en el modelo del compilador

1.4.4. Manejo de errores sintácticos

Los errores en la programación pueden ser de los siguientes tipos:

- Léxicos, producidos al escribir mal un identificador, una palabra clave o un operador.

- Sintácticos, por una expresión aritmética o paréntesis no equilibrados.
- Semánticos, como un operador aplicado a un operando incompatible.
- Lógicos, puede ser una llamada infinitamente recursiva.
- De corrección, cuando el programa no hace lo que el programador realmente deseaba.

1.4.5. Gramáticas independientes del contexto

La gramática independiente del contexto consta de terminales, no terminales, un símbolo inicial y producciones.

- Los terminales son los símbolos básicos con que se forman las cadenas.
- Los no terminales son variables sintácticas que denotan conjuntos de cadenas. Los no terminales definen conjuntos de cadenas que ayudan a definir el lenguaje generado por la gramática.
- En una gramática, un no terminal es considerado como el símbolo inicial, y el conjunto de cadenas que representan es el lenguaje definido por la gramática.
- Las producciones de una gramática especifican cómo se pueden combinar los terminales y los no terminales para formar cadenas. Cada producción consta de un terminal, seguido por algún símbolo y seguida por una cadena de no terminales y terminales.

1.5. Analizador Semántico⁹

1.5.1. Introducción

Esta fase revisa el árbol sintáctico junto con los atributos y la tabla de símbolos para tratar de encontrar errores semánticos. Para todo esto se analizan los operadores y operandos de expresiones y proposiciones. Finalmente reúne la información necesaria sobre los tipos de datos para la fase posterior de generación de código. El componente más importante del análisis semántico es la verificación de tipos. Aquí, el compilador verifica si los operandos de cada operador son compatibles según la especificación del lenguaje fuente.

⁹ AHO, Alfred y ULLMAN, Jeffrey. 2005, *Compiladores Principios, Técnicas y Herramientas*, Edición Electrónica

Si suponemos que nuestro lenguaje solo trabaja con números reales, la salida de esta fase sería su mismo árbol, excepto porque el atributo de <NUM>, que era el entero 8 a la entrada, ahora pasaría a ser el real 8,0. Además se ha debido controlar que las variables implicadas en la sentencia, a saber, comisión, fijo y valor son compatibles con el tipo numérico de la constante 8,0.

1.5.2. Definición

El análisis semántico dota de un significado coherente a lo que se hace en el análisis sintáctico. El chequeo semántico se encarga de que los tipos que intervienen en las expresiones sean compatibles o que los parámetros reales de una función sean coherentes con los parámetros formales: por ejemplo: no suele tener mucho sentido el multiplicar una cadena de caracteres por un entero. Comenzaremos viendo un ejemplo sencillo en el que se introduce el concepto de atributo mediante la construcción del intérprete de una calculadora. Además de controlar que un programa cumple con las reglas de la gramática del lenguaje, hay que comprobar que lo que se quiere hacer tiene sentido.

1.5.3. Atributos y acciones semánticas

Un atributo es una información asociada a un terminal o a un no terminal. Una acción o regla semántica es un algoritmo que puede acceder a los atributos de los terminales y/o no terminales. Como acción semántica no sólo se puede poner una asignación a un atributo, además puede añadirse código.

1.5.4. Ejecución de una acción semántica

Hay dos formas de asociar reglas semánticas con reglas de producción:

Definición dirigida por sintaxis: consiste en asociar una acción semántica a una regla de producción, pero dicha asociación no indica cuándo se debe ejecutar dicha acción semántica. Se supone que en una primera fase se construye el árbol sintáctico completo y, posteriormente, se ejecutan las acciones semánticas en una secuencia tal que permita el cálculo de todos los atributos de los nodos del árbol.

Esquema de traducción: es igual que una definición dirigida por sintaxis excepto que, además, se asume o se suministra información acerca de cuándo se deben ejecutar las acciones semánticas.

1.5.5. Análisis dirigido por sintaxis

Un esquema de análisis dirigido por sintaxis se puede implementar construyendo el árbol de análisis sintáctico y luego realizar un recorrido en pre orden del árbol ejecutando las acciones semánticas.

Dirigida por Sintaxis

Hay dos combinaciones que funcionan muy bien:

- Una gramática LR y reglas semánticas S-atribuidas
- Una gramática LL y reglas semánticas L-atribuidas

1.5.5.1. Gramática con atributos

Una gramática con atributos es una gramática de contexto libre cuyos símbolos pueden tener asociados atributos y las producciones pueden tener asociadas reglas de evaluación de los atributos.

Cada símbolo puede tener asociado un número finito de atributos. Cada producción puede tener asociada un número finito de reglas de evaluación de los atributos. Los valores de los atributos deberán estar asociados con un dominio de valores.

Dada una regla de evaluación $b = f(c_1, \dots, c_k)$ asociado con la producción $A \rightarrow \alpha \in P$.

Atributo Heredado: Si b está asociado con algún símbolo de α

Atributo Sintetizado: Si b está asociado con el símbolo no terminal A .

1.5.5.2. Métodos de evaluación de los atributos

Se han propuesto varios métodos para la evaluación de las reglas semánticas:

Métodos basados en grafos de dependencias: en el momento de la compilación estos métodos obtienen un orden de evaluación a partir del grafo de dependencias sobre el árbol de análisis sintáctico para la entrada dada (el programa fuente). Poco eficientes (en espacio y tiempo) porque necesitan construir todo el árbol de análisis sintáctico y sobre él, el grafo de dependencias para cada entrada. Posibilidad de ciclos.

Métodos basados en reglas: en el momento de la construcción del compilador, para cada producción queda predeterminado por el diseñador del compilador el orden de evaluación de los atributos de esa construcción lingüística, y así la forma de recorrer el árbol para calcular ese atributo (prefija, infija, postfija). No siempre será necesario construir el árbol de análisis sintáctico para después recorrerlo, los atributos sintetizados se pueden calcular a la vez que se realiza el análisis sintáctico.

1.5.6. Alfabetos, Palabras, Lenguajes¹⁰

Alfabetos

Un alfabeto es un conjunto finito y no vacío de símbolos. Se representa por Σ .

Ejemplo:

El alfabeto del español. $\Sigma = \{A, B, \dots, Z\}$

$\Sigma = \{x \mid x \text{ es un token legal en Java}\}$

Si Σ_1 y Σ_2 son alfabetos entonces

$\Sigma_1 \cup \Sigma_2$ también es un alfabeto.

Aún más, si $\Sigma_1 \cap \Sigma_2$, $\Sigma_1 - \Sigma_2$, $\Sigma_2 - \Sigma_1$ son conjuntos no vacíos, entonces también son alfabetos.

Lenguajes

Un lenguaje es un conjunto de cadenas.

Ej.- $\{1, 12, 123, 1234, 12345, 12346\}$ es un lenguaje sobre el alfabeto de los dígitos.

¹⁰ DEAN, Kelley. 1995. Teoría de Autómatas y Lenguajes Formales. Editorial PRENTICE HALL

Los lenguajes pueden ser infinitos y en consecuencia difíciles de especificar, este es uno de los problemas que vamos a estudiar.

Lenguaje vacío = $\{ \} \neq \{ \epsilon \}$.

Supóngase que Σ es un alfabeto y w es una cadena sobre Σ . Si L es el lenguaje formado por algunas de las cadenas sobre Σ y si w está en L , entonces se tiene que $w \in L$ y se dice que w es un elemento de L , o w es un miembro de L .

Ej.- $121 \in \{1, 12, 121, 1212, 12121\}$

Palabras

Una palabra o cadena en una secuencia finita de símbolos del alfabeto

Ejemplo:

La palabra $w = aaabbb$ se puede formar a partir de los símbolos del alfabeto

$\Sigma_3 = \{aa, bb, ab\} \mid aaabbb \mid \Sigma_3 = 3$.

Esa misma palabra se puede considerar formada a partir del alfabeto

$\Sigma_2 = \{a, b, c, d\}$. La longitud de $w = aaabbb$, respecto a Σ_2 es $\mid aaabbb \mid \Sigma_2 = 6$.

1.6. Lenguajes Regulares

Un lenguaje recursivo sobre un alfabeto Σ dado se define recursivamente como:

- El lenguaje vacío \emptyset es un lenguaje regular
- El lenguaje cadena vacía $\{\epsilon\}$ es un lenguaje regular
- Para todo símbolo $a \in \Sigma$ $\{a\}$ es un lenguaje regular
- Si A y B son lenguajes regulares entonces $A \cup B$ (unión), $A \cdot B$ (concatenación) y A^* (clausura o estrella de Kleene) son lenguajes regulares
- Si A es un lenguaje regular entonces (A) es el mismo lenguaje regular
- No existen más lenguajes regulares sobre Σ

Todo lenguaje formal finito constituye un lenguaje regular. Otros ejemplos típicos son todas las cadenas sobre el alfabeto $\{a, b\}$ que contienen un número par de a 's o el lenguaje que consiste en varias a 's seguidas de varias b 's.

Si un lenguaje no es regular requiere una máquina con al menos una complejidad de $\Omega(\log \log n)$ (donde n es el tamaño de la entrada). En la práctica la mayoría de los problemas no regulares son resueltos con una complejidad logarítmica.

Un lenguaje formal infinito puede ser regular o no regular. El lenguaje $L = \{a^n, n > 0\}$ es regular porque puede ser representado, por ejemplo, mediante la expresión regular a^+ . El lenguaje $L = \{a^n b^n, n > 0\}$ es un lenguaje no regular dado que no es reconocido por ninguna de las formas de representación anteriormente enumeradas.

Ejemplos de Lenguajes regulares

Ej.- Dado $S = \{a, b\}$, las siguientes afirmaciones son ciertas:

- $\{a\}$ y $\{b\}$ son lenguajes regulares.
- $\{a\}$ y $\{b\}$ son lenguajes regulares.
- $\{a, b\}$ es regular porque es la unión de $\{a\}$ y $\{b\}$.
- $\{ab\}$ es regular.
- $\{a, ab, b\}$ es regular.
- $\{a^i | i \geq 0\}$ es regular.
- $\{a^i b^j | i \geq 0 \text{ y } j \geq 0\}$ es regular.
- $\{(ab)^i | i \geq 0\}$ es regular.

1.7. Autómata finito determinista

Un AFD o autómata finito determinista es aquel autómata finito cuyo estado de llegada está unívocamente determinado por el estado inicial y el carácter leído por el autómata.

Formalmente, un autómata finito determinista (AFD) es similar a un Autómata de estados finitos, representado con una 5-tupla (S, Σ, T, s, A) donde:

- Σ es un alfabeto;
- S un conjunto de estados;
- T es la función de transición: $T : S \times \Sigma \rightarrow S$;
- $s \in S$ es el estado inicial;
- $A \subseteq S$ es un conjunto de estados de aceptación o finales.

Al contrario de la definición de **autómata finito**, este es un caso particular donde no se permiten transiciones vacías, el dominio de la función T es S (con lo cual no se permiten transiciones desde un estado de un mismo símbolo a varios estados). A partir

de este autómata finito es posible hallar la expresión regular resolviendo un sistema de ecuaciones.

$$S_1 = 1 S_1 + 0 S_2 + \varepsilon$$

$$S_2 = 1 S_2 + 0 S_1$$

Siendo ε la palabra nula. Resolviendo el sistema y haciendo uso de las reducciones apropiadas se obtiene la siguiente expresión regular: $1^*(01^*01^*)^*$.

1.8. Autómata finito no determinista

Un autómata finito no determinístico es una quinta tupla (Q, S, q_0, d, F) en donde Q, S, q_0 y F (estados, entradas, estado inicial y estados finales) poseen el mismo significado que para un DFA, pero en este caso d es una transformación de $Q \times S$ a 2^Q . (Recuérdese que 2^Q es el conjunto de potencias de Q , el conjunto de todos los subconjuntos de Q). Obsérvese que puesto que d es una relación para todo par (q, s) compuesto por el estado actual y el símbolo de la entrada, $d(q, s)$, es una colección de cero o más estados [es decir, $d(q, s) \subseteq Q$]. Esto indica que, para todo estado q_1 se pueden tener cero o más alternativas a elegir como estado siguiente, todas para el mismo símbolo de entrada.

Generalmente el término autómata finito no determinístico se abrevia como NFA de sus siglas en inglés Nondeterministic Finite Automaton. Si M es un NFA, definiremos el lenguaje aceptado por M por medio de $L(M) = \{w \mid w \text{ es una cadena aceptada por } M\}$ donde una cadena w es aceptada por M , si M pasa de su estado inicial a su estado de aceptación o final al recorrer w (w es consumida en su totalidad).

CAPITULO II

2. INTÉRPRETE

El intérprete es un programa informático capaz de analizar y ejecutar otros programas. Uno de los entornos más comunes de uso de los intérpretes es Internet, debido a la posibilidad que estos tienen de ejecutarse independientemente de la plataforma.

2.1. Diferencias entre Intérpretes y Compiladores.

- Los compiladores traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema; Los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.
- Un intérprete a un solo archivo fuente puede producir resultados iguales incluso en sistemas sumamente diferentes. El compilador a un solo archivo fuente puede producir resultados iguales solo si es compilado a distintos ejecutables específicos a cada sistema.
- La interpretación de los programas es más lenta que los compilados por la necesidad de traducir el programa mientras se ejecuta, pero son más flexibles como los entornos de programación y depuración; permitiendo ofrecer al programa interpretado un entorno no dependiente de la máquina donde se ejecuta el intérprete, sino del propio intérprete.
- Haciendo una comparación con la del ser humano tenemos que:
 - Un compilador equivale a un traductor profesional que a partir de un texto, prepara otro independiente traducido a otra lengua.
 - Un intérprete corresponde al intérprete humano, que traduce de viva voz las palabras que oye, sin dejar constancia por escrito.

CAPITULO III

3. DIAGRAMAS DE FLUJO

3.1. Concepto

Un diagrama de flujo es un medio de presentación visual y gráfica del flujo de datos o de un algoritmo en proceso. La programación, la economía, los procesos industriales y la psicología cognitiva son algunas disciplinas en las que se utiliza a los diagramas. Maneja símbolos estándar en el que cada paso del algoritmo se visualiza dentro del símbolo y en el orden adecuado, su flujo de ejecución donde se enseña los puntos de inicio y de término, se indica conectando a los símbolos con flechas que se las conocen como líneas de flujo.

3.2. Características

El diagrama de flujo se caracteriza porque:

- Presenta información clara, concisa y ordenada del flujo lógico del algoritmo.
- Está formada por una serie de símbolos unidos por líneas de flujos.
- Cada símbolo representa una acción específica.
- Las flechas entre los símbolos representan el orden de ejecución de los pasos del algoritmo.

3.3. Ventajas de los Diagramas de Flujo

El diagrama de flujo tiene las ventajas que se expresan a continuación:

- Son fáciles de diseñar debido a que el flujo lógico del algoritmo se representa en un dibujo en lugar de palabras.
- Ayudan a comprender el flujo lógico del algoritmo.
- Nos ayuda a identificar los pasos redundantes, los problemas y las oportunidades de mejorar el proceso.

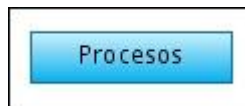
3.4. Símbolos para la creación de diagramas de flujo y su significado

Los símbolos utilizados son normalizados por las organizaciones ANSI (American National Institute) y por ISO (International Standard Organization):

Símbolos Principales



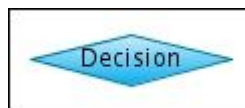
Terminal: representa el Inicio o fin del programa.



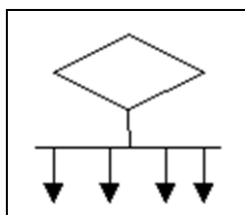
Proceso: representa cualquier tipo de operación que puede originar cambio de valor, o una instrucción.



Entrada/salida: representa operaciones de una entrada o salida de información, que sea procesada o registrada por medio de un periférico.



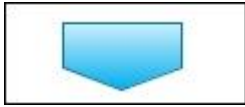
Decisión: indica operaciones lógicas o de comparación entre datos, utilizado para la toma de decisiones y ramificación



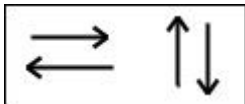
Decisión múltiple: en función del resultado de la comparación se seguirá uno de los diferentes caminos.



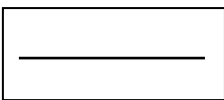
Conector: sirve para unir el flujo a otra parte del diagrama a través de un conector en la salida y otro conector en la entrada. Forma un enlace en la misma página del diagrama.



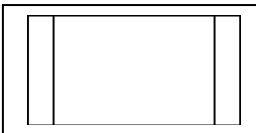
Conector: conexión de dos partes de un diagrama que se encuentre en páginas diferentes.



Indicador de dirección o línea de flujo: indica el sentido de ejecución de las operaciones.



Línea conectora: sirve para unir dos símbolos.



Llamada a subrutina o a proceso predeterminado: una rutina es un modulo independiente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y regresa al terminar.

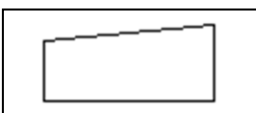
Símbolos Secundarios



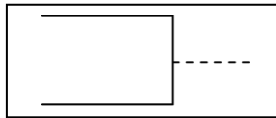
Pantalla: representa la salida o para mostrar la información por medio de el monitor.



Impresora: representa salida de datos por medio de la impresora.



Teclado: se utiliza en ocasiones en vez del símbolo E/S.



Comentario: se utiliza para añadir comentarios clasificadores a otros símbolos del diagrama de flujo.

3.5. Símbolos Gráficos

Los símbolos gráficos son utilizados para operaciones aritméticas y relaciones condicionales, se los ubica dentro de los símbolos para la creación de diagramas de flujo.

Los símbolos comúnmente más utilizados se encuentran a continuación:

+	Sumar
-	Restar
*	Multiplicación
/	División
±	Más o menos
=	Equivalente a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Diferente de
	Si
	No
	True
	False

Tabla 1.: Símbolos Gráficos

CAPITULO IV

4. PSEUDOCÓDIGO

4.1. Definición

Es un lenguaje de especificación de algoritmos. El uso de este lenguaje hace el paso de codificación final relativamente fácil.

El pseudocódigo nació como un lenguaje similar al lenguaje natural y es un medio para representar básicamente las estructuras de control, símbolos y términos de programación de lenguajes de alto nivel. Se considera un primer borrador, dado que el pseudocódigo tiene que traducirse posteriormente a un lenguaje de programación.

4.2. Características y partes

Las principales características son:

- Que en su uso en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico.
- Su representación es sencilla de utilizar y de manipular.
- Fácil de modificar el pseudocódigo si se descubren errores o anomalías en la lógica del programa.
- Traducción fácil a lenguajes de programación.
- Es independiente del lenguaje de programación que se vaya a utilizar.
- Es un método que facilita la programación y solución al algoritmo del programa.

Todo documento en pseudocódigo debe permitir la descripción de:

- Instrucciones primitivas.
- Instrucciones de proceso.
- Instrucciones de control.
- Instrucciones compuestas.
- Instrucciones de descripción.

Estructura a seguir en su realización:

1. Cabecera.
 1. Programa.
 2. Modulo.
 3. Tipos de datos.
 4. Constantes.
 5. Variables.
2. Cuerpo.
 - 1) Inicio.
 - 2) Instrucciones.
 - 3) Fin.

4.3. Funciones y operaciones

La instrucción "reemplace el valor de la variable x por el valor de la variable y " puede ser representado como:

$$x \leftarrow y \quad \text{ó} \quad x = y$$

Las operaciones aritméticas se representan de la forma usual en matemáticas.

$$\text{volumen} \leftarrow \pi r^2 h$$

4.4. Estructuras de control

Existen tres tipos de estructuras de control: las secuenciales, las selectivas y las iterativas.

4.4.1. Estructuras secuenciales

Las instrucciones siguen una secuencia fija y se ejecutan de arriba hacia abajo. Las instrucciones se ejecutan dependiendo de la condición dada dentro del algoritmo.

- Instrucción 1
- Instrucción 2
- ...
- Instrucción n

4.4.2. Estructuras selectivas

Las instrucciones selectivas representan instrucciones que pueden o no ejecutarse, según el cumplimiento de una condición.

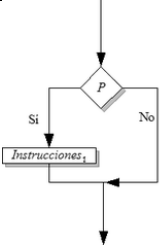
Diagrama de flujo		Instrucción condicional
	=	si condición entonces instrucciones fin si

Tabla 2.Diagrama de flujo y pseudocódigo que muestra el funcionamiento de la instrucción condicional.

4.4.2.1. Selectiva doble (alternativa)

Realiza una instrucción de dos posibles, según el cumplimiento de una condición.

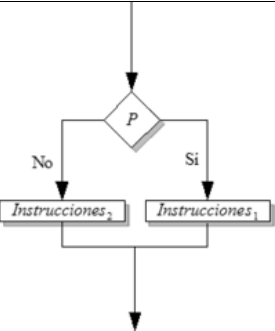
Diagrama de flujo		Instrucción Selectiva doble
	=	si condición entonces instrucciones1 si no entonces instrucciones2 fin si

Tabla 3.Diagrama de flujo y pseudocódigo que muestra el funcionamiento de la instrucción selectiva doble.

4.4.2.2. Selectiva múltiple

Equivale a anidar varias funciones de selección.

```

si condición1 entonces
  instrucciones1
sino si condición2 entonces
  instrucciones2
sino si condición3 entonces
  instrucciones3
  ...
  
```


sino entonces

instrucciones_n

fin si

4.4.2.3. Selectiva múltiple-Casos

Una construcción similar a la anterior, equivalente en algunos casos, es la que se muestra a continuación.

seleccionar *indicador*

caso *valor₁*

instrucciones₁

caso *valor₂*

instrucciones₂

caso *valor₃*

instrucciones₃

...

en otro caso

instrucciones_n

fin seleccionar

4.4.3. Estructuras iterativas

Representan la ejecución de instrucciones en más de una vez.

4.4.3.1. Bucle mientras

Se repite el bucle mientras la condición sea verdadera, si la primera vez que llega al bucle la condición es falsa, el cuerpo del bucle no se ejecuta.

mientras *condición hacer*

instrucciones

fin mientras

4.4.3.2. Bucle repetir

Se utiliza cuando es necesario que el cuerpo del bucle se ejecuten al menos una vez y hasta que se cumpla la condición:

repetir
instrucciones
hasta que *condición*

4.4.3.3. Bucle para

Se utiliza cuando se desea iterar un número conocido de veces, empleando como índice una variable que se incrementa (o decrementa):

para $i \leftarrow x$ **hasta** n **hacer**
instrucciones
fin para

4.4.3.4. Bucle para cada

Se usa cuando se tiene una lista o un conjunto L y se quiere iterar por cada uno de sus elementos:

para cada $x \in L$ **hacer**
instrucciones
fin para cada

CAPITULO V

5. HERRAMIENTAS TENTATIVAS A UTILIZAR

5.1. Software Libre¹¹

El «software libre» es una cuestión de libertad, no de precio. Para entender el concepto, debería pensar en «libre» como en «libre expresión», no como en «barra libre».

El software libre es una cuestión de la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Más precisamente, significa que los usuarios de programas tienen las cuatro libertades esenciales.

- La libertad de ejecutar el programa, para cualquier propósito (libertad 0).
- La libertad de estudiar cómo trabaja el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para que pueda ayudar al prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (la 3ª libertad). Si lo hace, puede dar a toda la comunidad una oportunidad de beneficiarse de sus cambios. El acceso al código fuente es una condición necesaria para ello.

Un programa es software libre si los usuarios tienen todas esas libertades. Entonces, debería ser libre de redistribuir copias, tanto con o sin modificaciones, ya sea gratis o cobrando una tarifa por distribución, a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir o pagar el permiso.

También debería tener la libertad de hacer modificaciones y usarlas en privado, en su propio trabajo u obra, sin siquiera mencionar que existen. Si publica sus cambios, no

¹¹ GNU. La Definición de Software Libre. [en línea]. [<http://www.gnu.org/philosophy/free-sw.es.html>], [Consulta: 17 enero 2011].

debería estar obligado a notificarlo a alguien en particular, o de alguna forma en particular.

La libertad de ejecutar el programa significa la libertad para cualquier tipo de persona u organización de usarlo en cualquier tipo de sistema de computación, para cualquier tipo de trabajo y propósito, sin estar obligado a comunicarlo a su programador, o alguna otra entidad específica. En esta libertad, el propósito de los *usuarios* es el que importa, no el propósito de los *programadores*. Como usuario es libre de ejecutar un programa para sus propósitos; y si lo distribuye a otra persona, también es libre para ejecutarlo para sus propósitos, pero usted no tiene derecho a imponerle sus propios propósitos.

Para que la 1ª y 3ª libertad, para realizar cambios y publicar versiones mejoradas, tengan sentido; debe tener acceso al código fuente del programa. Por consiguiente, el acceso al código fuente es una condición necesaria para el software libre. El «código fuente» ofuscado no es código fuente real, y no cuenta como código fuente.

En el proyecto GNU, usamos copyleft para proteger legalmente estas libertades para todos. Pero también existe software libre sin copyleft. Creemos que existen razones importantes por las que es mejor usar copyleft, pero si su programa es software libre sin copyleft, sigue siendo ético de todos modos.

Cuando se habla de software libre, es mejor evitar usar términos como «regalar» o «gratis», porque dichos términos implican que el asunto pasa por el precio, no la libertad. Algunos términos comunes como «piratería» implican opiniones con las que esperamos no concuerde.

Finalmente, tenga en cuenta que los criterios, como los establecidos en esta definición de software libre, requieren pensar con cuidado su interpretación.

6. METODOLOGIA

6.1. Matriz de consistencia general

PROBLEMA GENERAL DE INVESTIGACIÓN (ENUNCIADO):			
<p>“Falta de una herramienta informática que coadyuve a los alumnos que están iniciando la unidad de metodología de programación, a generar el pseudocódigo a partir de un diagrama de flujo y viceversa, en la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja.”.</p>			
TÍTULO	OBJETO DE INVESTIGACIÓN	OBJETIVO DE LA INVESTIGACIÓN	HIPÓTESIS DE INVESTIGACIÓN
<p>“Desarrollo de una herramienta de software que permita graficar un Diagrama de Flujo generando su Pseudocódigo y de un Pseudocódigo obtener su Diagrama de Flujo, para los alumnos de la carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja.”</p>	<p>Compiladores, Diagramas de Flujo y Pseudocódigo.</p>	<p>Desarrollar una herramienta que genere al Diagrama de Flujo a partir de un Pseudocódigo y que genere el Pseudocódigo a partir de un Diagrama de Flujo en la Universidad Nacional de Loja.</p>	<p>El desarrollo de una herramienta que permita graficar un Diagrama de Flujo y genere su Pseudocódigo y que a partir de un Pseudocódigo se genere su Diagrama de Flujo, permitirá ayudar a satisfacer las necesidades de un mayor aprendizaje para los alumnos que están empezando la unidad de Metodología de Programación.</p>

6.2. Materiales, métodos y técnicas de trabajo

Materiales.

Los materiales que voy a utilizar para desarrollar este proyecto, son los siguientes:

Resma de Papel.
Cartuchos de tinta negra.
Cartucho de tinta a color.
Kit de recarga de cartuchos.
Internet/horas
Flash Memory (1GB) Kingston.
CD
Copias
Transporte
Material Bibliográfico
Varios

Métodos.

- **Método Inductivo.-** Va de lo particular a lo general. Se lo utiliza para poder identificar los inconvenientes que se presentan al no tener una herramienta que les permita generar el Pseudocódigo a partir de un diagrama de Flujo y viceversa.
- **Método Deductivo.-** Va de lo general a lo particular. Este método nos sirve para buscar alternativas de solución para el uso de la herramienta informática que genere su Pseudocódigo a partir del Diagrama de flujo y viceversa.
- **Método Analítico.-** Sirve para realizar un análisis del objeto en estudio. Se utiliza para realizar un minucioso estudio de los problemas, causas y consecuencias que se están presentando con los alumnos que empiezan la unidad de Metodología de Programación sin una herramienta que les ayude a su mejor entendimiento.
- **Método Sintético.-** Realiza una síntesis del proceso investigativo. Se lo considera para realizar la construcción teórica de mi investigación.
- **Método ciclo de vida de un Sistema:** en este método se encuentran todas las etapas por la que tiene que pasar un sistema, guiando al desarrollador a

establecer los principales elementos que intervendrán en el desarrollo, a continuación estas son las etapas de este método:

- **Análisis:** en esta etapa recolecto toda la información necesaria, la analizo, y selecciono para la siguiente etapa.
- **Diseño:** en esta etapa desarrollo los prototipos de pantalla conforme a la información que recolecte y a las necesidades de los alumnos.
- **Desarrollo:** En esta etapa pongo como base los prototipos de pantalla que anteriormente diseñe para poder codificar y darle la funcionalidad que se esperaba.
- **Pruebas:** en esta etapa ya está terminada la aplicación pero hay que hacerle las pruebas suficientes para poder corregir los errores si existen y así entregar una herramienta funcionando correctamente.

Técnicas.

- **Lectura comprensiva.-** Consiste en obtener un conocimiento ordenado y sistemático de un aspecto de la realidad o de los acontecimientos hechos o ideas relacionadas con el tema específico. La lectura comprensiva me servirá para comprender correctamente como debo efectuar la implementación de mi aplicación.
- **La Entrevista:** Esta técnica es muy importante para el analista porque me permite obtener la información en forma verbal, a través de preguntas a los alumnos que empiezan sus estudios de programación en la Carrera de Ingeniería en Sistemas.
- **La Observación:** Esta técnica me permite observar como los alumnos se desenvuelven con los Diagramas de Flujo y los Pseudocódigo, y así poder a la herramienta que voy a desarrollar mejorarla y que fácil de interactuar con los alumnos.

Metodología.

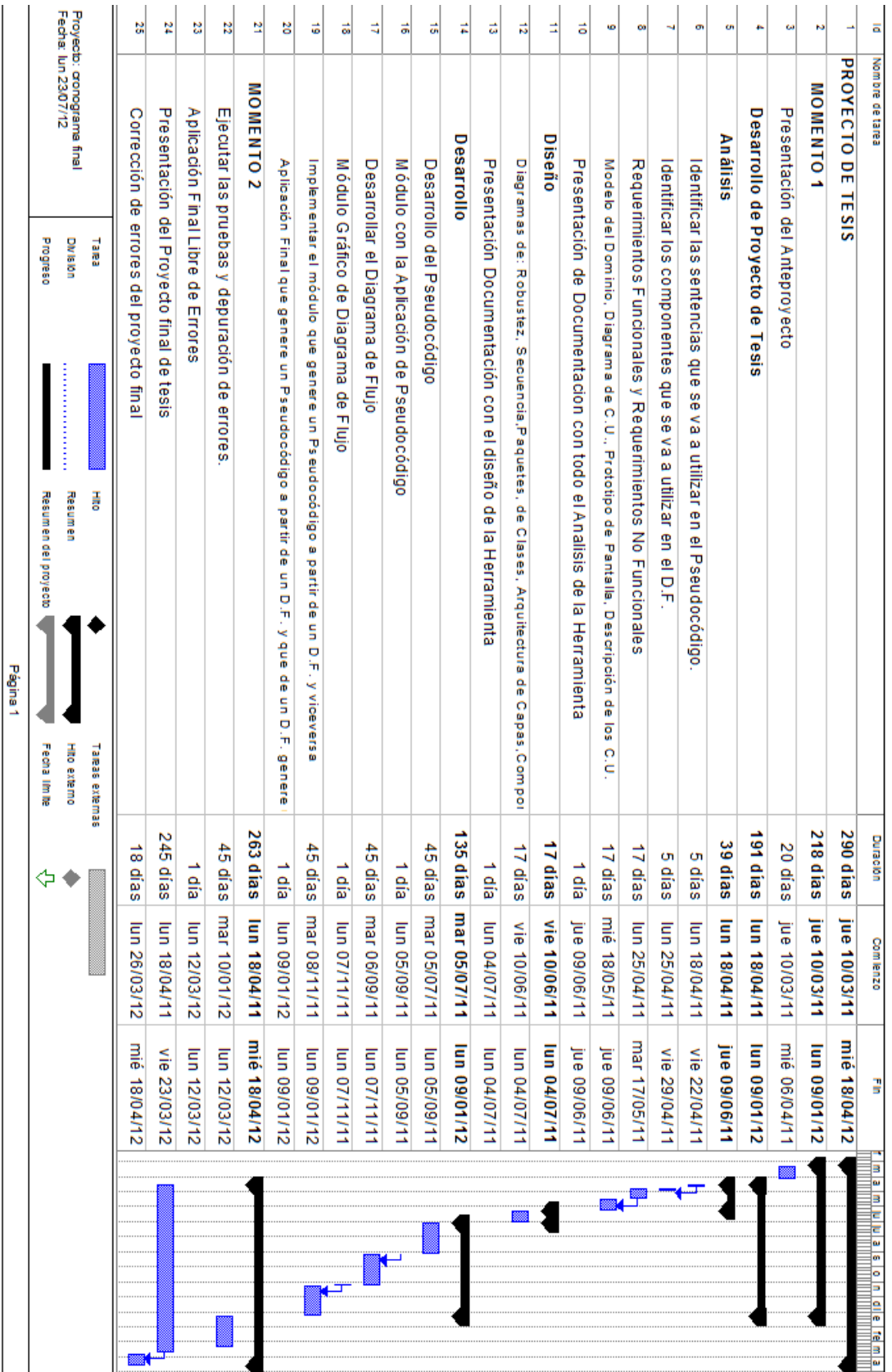
La metodología que voy a utilizar para el desarrollo de este proyecto es ICONIX, debido a que permite un desarrollo ágil, con muy buena documentación, es bastante flexible y se adapta mejor a la Programación Orientada a Objetos ya que emplea UML.



ICONIX, me permite inicializar la ejecución del presente proyecto, en la etapa de análisis con toda la información necesaria, la filosofía en que se basa este tipo de metodología es que es iterativo e incremental, esto significa que durante este proceso iremos encontrando nuevas entidades y relaciones, que no se tomaron en cuenta al iniciar nuestra investigación lo cual nos obliga a actualizar cada vez nuestro modelo del dominio o espacio del problema hasta que este queda completo.



7. CRONOGRAMA



8. PRESUPUESTO Y FINANCIAMIENTO

Recursos Humanos

Recursos Humanos	Cantidad	Horas c/u	Costo por Hora	Costo Total
Director de Tesis	----	-----	-----	----
Desarrolladores	1	1200	\$3.00	\$3600.00
TOTAL				\$3600.00

Recursos Materiales.

Recursos Materiales	Cantidad	Costo Unitario	Costo Total
Resma de Papel.	5	\$3.30	\$16.50
Cartuchos de tinta negra.	2	\$20.00	\$40.00
Cartucho de tinta a color.	1	\$22.00	\$22.00
Kit de recarga de cartuchos.	2	\$7.00	\$14.00
Internet/horas	100	\$1.00	\$100.00
Flash Memory (1GB) Kingston.	1	\$7.00	\$7.00
CD	6	0.70	\$4.20
TOTAL			\$203.70

Recursos Técnicos.

Recursos Técnicos	Cantidad	Horas	Hojas	Costo		Costo Total
				Hora	Hoja	
Computadores	1	1200		0.50		\$600.00
Impresora	1		2500		0.05	\$125.00
Alquiler de Proyector	1	2		10.00		\$20.00
TOTAL						\$745.00

Recursos Tecnológicos.

Recursos Tecnológicos	Costo Unitario	Costo Total
Software Libre	Gratuito	\$0.00
My SQL 5.0	Gratuito	\$0.00
Enterprise Architect 3.6	Gratuito	\$0.00
Open Office 3.2	Gratuito	\$0.00
TOTAL		\$0.00

Resumen del Presupuesto

Resumen del Presupuesto	Costo Total
Recursos Humanos	\$3600.00
Recursos Materiales	\$203.70
Recursos Técnicos	\$745.00
Recursos Tecnológicos	\$0.00
SUBTOTAL	\$4548.70
Imprevistos 10 %	\$454.87
TOTAL	\$5003.57

9. BIBLIOGRAFIA

LIBROS:

- ✿ DEAN, Kelley. 1995. Teoría de Autómatas y Lenguajes Formales. Editorial PRENTICE HALL.

RECURSOS DE INTERNET:

- ✿ AHO, Alfred y ULLMAN, Jeffrey. 2005, Compiladores Principios, Técnicas y Herramientas, Edición Electrónica.
- ✿ GÁLVEZ ROJAS, Sergio Y MORA MATA, Miguel Ángel. .2005, Java a Tope: Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC, Edición Electrónica.

SITIOS WEB:

- ✿ VARIOS. 2010. Diagrama de flujo. [en línea] WIKIPEDIA, La enciclopedia libre, [http://es.wikipedia.org/wiki/Diagrama_de_flujo], [Consulta: 17 enero 2011].
- ✿ GNU. La Definicion de Software Libre. [en línea]. [<http://www.gnu.org/philosophy/free-sw.es.html>], [Consulta: 19 enero 2011].
- ✿ MILLÁN, Antonio ,FERMÍN ,Gerardo y CHACÓN, José. Diagrama de Flujo. [en línea]. Puerto Ordaz, [<http://www.monografias.com/trabajos59/diagrama-flujo/diagrama-flujo.shtml>], [Consulta: 17 enero 2011].
- ✿ BONILLA, Oscar. [http://74.125.45.104/search?q=cache:A9YLY0RcmuUJ:oscarbonilla.com/courses/compilers/materials/06_Analisis_Sintactico.ppt+analizador+sint%3%A1ctico&hl=es&ct=clnk&cd=11].
- ✿ CARRETO, Julio, Ing. Símbolos Gráficos. [en línea] Produccion e Inventarios, [<http://uproprod.blogspot.com/2007/08/aprenda-crear-diagramas-de-flujo.html>], [Consulta: 17 enero 2011].
- ✿ Programación de computadores. Pseudocódigo. [en línea]. Bogota-Colombia, Universidad Nacional de Colombia. [http://www.virtual.unal.edu.co/cursos/ingenieria/2001839/modulo1/cap_02/leccion1022.htm], [Consulta: 17 enero 2011].

10. ANEXOS

10.1. MATRIZ DE CONSISTENCIA ESPECÍFICA.

PROBLEMA ESPECÍFICO:			
Inexistencia de una aplicación que satisfaga las necesidades de los alumnos.			
OBJETIVO ESPECÍFICO	HIPÓTESIS ESPECÍFICA	UNIDAD DE OBSERVACIÓN	SISTEMA CATEGORIAL
Analizar y recolectar información para el desarrollo de la herramienta.	El análisis y la recolección de la información permitirán que desarrollemos y obtengamos una herramienta acorde a las necesidades de los alumnos.	Pseudocódigo, Diagramas de Flujo, alumnos de la Carrera de Ingeniería en Sistemas de la Universidad Nacional de Loja.	Pseudocódigo y Diagramas de Flujo. Entrevistas, Encuestas

PROBLEMA ESPECÍFICO:			
Inexistencia de una aplicación que permita a los alumnos ingresar su pseudocódigo y verificar si tiene algún error léxico, semántico o de sentido.			
OBJETIVO ESPECÍFICO	HIPÓTESIS ESPECÍFICA	UNIDAD DE OBSERVACIÓN	SISTEMA CATEGORIAL
Desarrollar un analizador léxico, sintáctico y semántico para el pseudocódigo.	La implementación del analizador léxico, sintáctico y semántico del pseudocódigo permitirá elaborar los tokens y patrones que utiliza el analizador sintáctico y éste permitirá que el compilador	Pseudocódigo. Analizador Léxico, Analizador Sintáctico y Analizador Semántico.	Analizador léxico: <ul style="list-style-type: none"> ➤ Definición ➤ Funciones ➤ Tokens, lexemas y patrones Analizador Sintáctico: <ul style="list-style-type: none"> ➤ Definición ➤ Visión general. ➤ El papel del Analizador Sintáctico. ➤ Manejo de errores sintácticos. ➤ Gramáticas independientes del contexto.

	<p>acepte y ejecute instrucciones del pseudocódigo, y el analizador semántico permitirá revisar que la semántica de las sentencias sea la correcta.</p>		<p>Analizador Semántico</p> <ul style="list-style-type: none"> ➤ Definición ➤ Reglas ➤ Función. ➤ Tabla de símbolos ➤ Análisis dirigido por sintaxis. ➤ Gramática con atributos. ➤ Comprobación de tipos
--	---	--	---

PROBLEMA ESPECÍFICO:

Los alumnos tienen problemas al realizar un diagrama de flujo porque no hay una aplicación que les permita diseñar e indique si tiene algún error de escritura o esta sin sentido su estructura.

OBJETIVO ESPECÍFICO	HIPÓTESIS ESPECÍFICA	UNIDAD DE OBSERVACIÓN	SISTEMA CATEGORIAL
<p>Elaborar un módulo gráfico que permita implementar los diferentes componentes del diagrama de flujo.</p>	<p>El desarrollo de un módulo gráfico que permita implementar los diferentes componentes del diagrama de flujo permitirá a los alumnos diseñar y ver si tienen algún de escritura o estructura en el diagrama de flujo.</p>	<p>Diagramas de flujo.</p>	<p>Diagramas de flujo:</p> <ul style="list-style-type: none"> ➤ Concepto ➤ Características ➤ Ventajas de los Diagramas de Flujo ➤ Simbología o sus componentes. ➤ Estructura de los diagramas de flujo.

PROBLEMA ESPECÍFICO:

Los alumnos tienen problemas al convertir de un diagrama de flujo a pseudocódigo y viceversa, porque no pueden darse cuenta rápidamente en lápiz y papel donde se encuentra el error.

OBJETIVO ESPECÍFICO	HIPÓTESIS ESPECÍFICA	UNIDAD DE OBSERVACIÓN	SISTEMA CATEGORIAL
<p>Desarrollar un</p>	<p>El desarrollar un</p>	<p>Diagramas de</p>	<p>Diagramas de flujo</p>

<p>módulo que permita convertir un lenguaje de pseudocódigo a diagrama de flujo y de un diagrama de flujo a pseudocódigo.</p>	<p>módulo que permita convertir un lenguaje de pseudocódigo a diagrama de flujo y de un diagrama de flujo a pseudocódigo, permitirá a los alumnos de la universidad que tengan una ayuda para que observen cómo quedaría la conversión y si tienen algún error.</p>	<p>Flujo. Pseudocódigo. Software libre.</p>	<ul style="list-style-type: none"> ➤ Concepto ➤ Características ➤ Ventajas de los Diagramas de Flujo ➤ Símbolos para la creación de diagramas de flujo y su significado ➤ Símbolos Gráficos Pseudocódigo ➤ Definición ➤ Características y partes ➤ Funciones y operaciones ➤ Estructuras de control
---	---	---	--

PROBLEMA ESPECÍFICO:

Inexistencia de realización de pruebas a la aplicación para verificar que cumpla con los requerimientos planteados.

OBJETIVO ESPECÍFICO	HIPÓTESIS ESPECÍFICA	UNIDAD DE OBSERVACIÓN	SISTEMA CATEGORIAL
<p>Realizar pruebas de la aplicación realizada.</p>	<p>Una adecuada prueba para la herramienta permitirá que la misma entre en funcionamiento cumpliendo cada uno de los requerimientos obtenidos durante la etapa de análisis.</p>	<p>Herramienta que genere su respectivo pseudocódigo a partir de un diagrama de flujo y de un diagrama de flujo genere su pseudocódigo.</p>	<p>Documentación de la herramienta.</p>



10.2. MATRIZ DE OPERATIVIDAD DE OBJETIVOS ESPECÍFICOS

OBJETIVO ESPECÍFICO: Analizar y recolectar información para el desarrollo de la herramienta.						
ACTIVIDAD O TAREA	METODOLOGÍA	FECHA		RESPON SABLES	PRESUP UESTO	RESULTADOS ESPERADOS
		INICIO	FINAL			
Identificar las sentencias que se va a utilizar en el pseudocódigo.	Consultas bibliográficas sobre las sentencias que utilizan los pseudocódigos	18-04-2011	22-04-2011	Andrea Salinas	100.00	Documento que contenga las sentencias a utilizar para el pseudocódigo.
Identificar los componentes que se va a utilizar el diagrama de flujo	Consultas bibliográficas sobre los componentes que utilizan los Diagramas de Flujo.	25-04-2011	29-04-2011	Andrea Salinas	100.00	Documento que contenga los componentes del diagrama de flujo a utilizar.
Recolectar la información necesaria para determinar los principales requerimientos de la herramienta de software.	Consultas bibliográficas. Entrevistas. Encuestas.	25-04-2011	17-05-2011	Andrea Salinas	300.00	Requerimientos funcionales y requerimientos no funcionales para la herramienta de software.
Realizar el análisis de la herramienta.	ICONIX	18-05-2011	09-06-2011	Andrea Salinas	300.00	Documento con: Modelo del dominio, Diagramas de caso de uso, Prototipo de pantallas. Descripción de casos de uso.
Elaborar el diseño de la herramienta.	ICONIX	10-06-2011	04-07-2011	Andrea Salinas	300.00	Documento con: Diagramas de robustez, Diagramas de secuencia, Diagramas de paquetes, Diagrama de clases por cada paquete Arquitectura de capas. Diagrama de componentes.

OBJETIVO ESPECÍFICO:						
Desarrollar un analizador léxico, sintáctico y semántico para el pseudocódigo.						
ACTIVIDAD O TAREA	METODOLOGÍA	FECHA		RESPONSABLES	PRESUPUESTO	RESULTADOS ESPERADOS
		INICIO	FINAL			
Realizar la programación del analizador léxico, sintáctico y semántico del pseudocódigo	ICONIX	05-07-2011	05-09-2011	Andrea Salinas	800,00	Aplicación del Pseudocódigo.

OBJETIVO ESPECÍFICO:						
Elaborar un módulo gráfico que permita implementar los diferentes componentes del diagrama de flujo.						
ACTIVIDAD O TAREA	METODOLOGÍA	FECHA		RESPONSABLES	PRESUPUESTO	RESULTADOS ESPERADOS
		INICIO	FINAL			
Realizar la Implementación del Diagrama de Flujo.	ICONIX	06-09-2011	07-11-2011	Andrea Salinas	800,00	Módulo gráfico de Diagrama de Flujo.

OBJETIVO ESPECÍFICO:						
Desarrollar un módulo que permita convertir un lenguaje de pseudocódigo a diagrama de flujo y de un diagrama de flujo a pseudocódigo.						
ACTIVIDAD O TAREA	METODOLOGÍA	FECHA		RESPONSABLES	PRESUPUESTO	RESULTADOS ESPERADOS
		INICIO	FINAL			
Implementar el módulo que genere un Pseudocódigo a partir de un Diagrama de Flujo y que de un Diagrama de Flujo genere un Pseudocódigo.	Convertir el diagrama de clases final a código.	08-11-2011	09-01-2012	Andrea Salinas	800.00	<ul style="list-style-type: none"> Módulo que genere un Pseudocódigo a partir de un Diagrama de Flujo y que de un Diagrama de Flujo genere un Pseudocódigo

OBJETIVO ESPECÍFICO:						
Realizar pruebas de la aplicación realizada.						
ACTIVIDAD O TAREA	METODOLOGÍA	FECHA		RESPONSABLES	PRESUPUESTO	RESULTADOS ESPERADOS
		INICIO	FINAL			
Ejecutar las pruebas y depuración de errores.	Realizar múltiples pruebas a la aplicación y las respectivas correcciones en caso de ser necesario.	10-01-2012	12-03-2012	Andrea Salinas	800.00	<ul style="list-style-type: none"> • Aplicación final libre de errores.

10.3. MATRIZ DE CONTROL DE RESULTADOS

No.	RESULTADOS	FECHA	FIRMA DEL DOCENTE
1	Documento que contenga las sentencias a utilizar para el pseudocódigo.	22-04-2011	
2	Documento que contenga los componentes del diagrama de flujo a utilizar.	29-04-2011	
3	Requerimientos funcionales y requerimientos no funcionales para la herramienta de software.	17-05-2011	
4	Documentación de Análisis del Software con: Modelo del Dominio, Diagramas de Caso de Uso, Prototipo de Pantalla y Descripción de Casos de Uso.	09-06-2011	
5	Documentación del Diseño del Software con: Diagramas de Robustez, Diagramas de secuencia, Diagramas de paquetes, Diagrama de clases por cada paquete, Arquitectura de capas, Diagrama de Componentes.	04-07-2011	
6	Módulo con la aplicación del Pseudocódigo	05-09-2011	
7	Módulo gráfico de Diagrama de Flujo.	07-11-2011	
8	Módulo que genere un Pseudocódigo a partir de un Diagrama de Flujo y que de un Diagrama de Flujo genere un Pseudocódigo.	09-01-2012	
9	Ejecutar las pruebas y depuración de errores.	12-03-2012	



10	Aplicación final libre de errores.	12-03-2012	
11	Presentación del Proyecto final de tesis	23-03-2012	
12	Corrección de errores del proyecto final	18-04-2012	

10.3. MODELO DE ENCUESTA PARA SABER SI NECESITAN UN SOFTWARE



UNIVERSIDAD NACIONAL DE LOJA

Área de la Energía, las Industrias y los Recursos Naturales No Renovables

Carrera de Ingeniería en Sistemas

ENCUESTA PARA DOCENTES

1. Cree usted que la adquisición de conocimientos de los temas: Diagramas de Flujo y Pseudocódigos, son de suma importancia para poder en el futuro programar.

Si ()

No ()

¿Por qué?:

.....
.....

2. Le parece que el tiempo que tiene para impartir su materia con los temas de Diagramas de Flujo y Pseudocódigo, es suficientemente bueno o le falta tiempo.

Si ()

No ()

¿Por qué?:

.....
.....

3. En el momento de desarrollar un pseudocódigo o un diagrama de flujo se le han presentado alguno de los siguientes problemas:

Olvido de algún signo ()

Alguna palabra mal escrita ()

Olvido de cerrar o abrir algún bucle ()

Algún símbolo del Diagrama de Flujo ()

Ninguna de las anteriores ()

Otras:.....

4. Cree usted que sería bueno tener el apoyo de una herramienta informática que permita de un Diagrama de Flujo generar el Pseudocódigo y viceversa.

Si ()

No ()

¿Por qué?:

.....
.....



UNIVERSIDAD NACIONAL DE LOJA

Área de la Energía, las Industrias y los Recursos Naturales No Renovables

Carrera de Ingeniería en Sistemas

ENCUESTA PARA ALUNMOS

1. Cree usted que la adquisición de conocimientos de los temas: Diagramas de Flujo y Pseudocódigos, son de suma importancia para poder en el futuro programar.

Si ()

No ()

¿Por qué?:

.....
.....

2. Le parece que el tiempo que tiene para la unidad con los temas de Diagramas de Flujo y Pseudocódigo, es suficientemente bueno o le falta tiempo.

Si ()

No ()

¿Por qué?:

.....
.....

3. En el momento de desarrollar un pseudocódigo o un diagrama de flujo se le han presentado alguna vez uno se los siguientes problemas:

Olvido de algún signo ()

Alguna palabra mal escrita ()

Olvido de cerrar o abrir algún bucle ()

Algún símbolo del Diagrama de Flujo ()

No saber si en el transcurso del desarrollo de el Diagrama de Flujo hay algún error ()

No saber si en el transcurso del desarrollo de el Pseudocódigo hay algún error ()

Ninguna de las anteriores ()

4. Cree usted que sería bueno tener el apoyo de una herramienta informática que permita de un Diagrama de Flujo generar el Pseudocódigo y viceversa.

Si ()

No ()

¿Por qué?:

.....
.....

5. Cree usted que sería de ayuda tener una herramienta informática que permita mostrar en que línea del pseudocódigo hay algún error.

Si ()

No ()

¿Por qué?:

.....
.....



ANEXO C. Certificado de Aprobación

Certificado de Aprobación

ANEXO D. DIAGRAMA DE ROBUSTEZ

Diagrama De Robustez

Caso de Uso: Ingresar Pseudo-Código

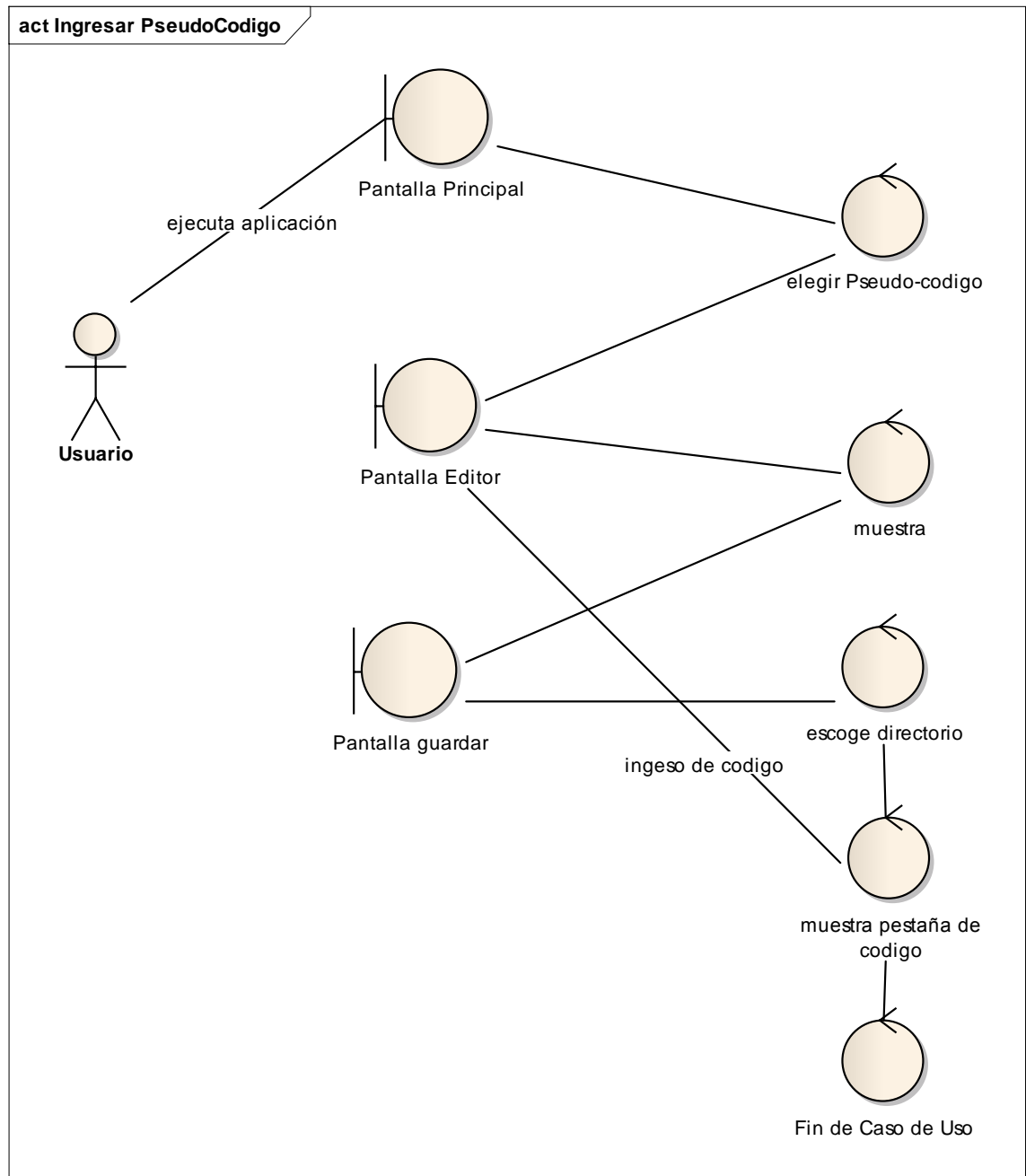


Diagrama 22 .Diagrama de Robustez del Caso de Uso: Ingresar Pseudo-Código

Caso de Uso: Analizar Lexicamente

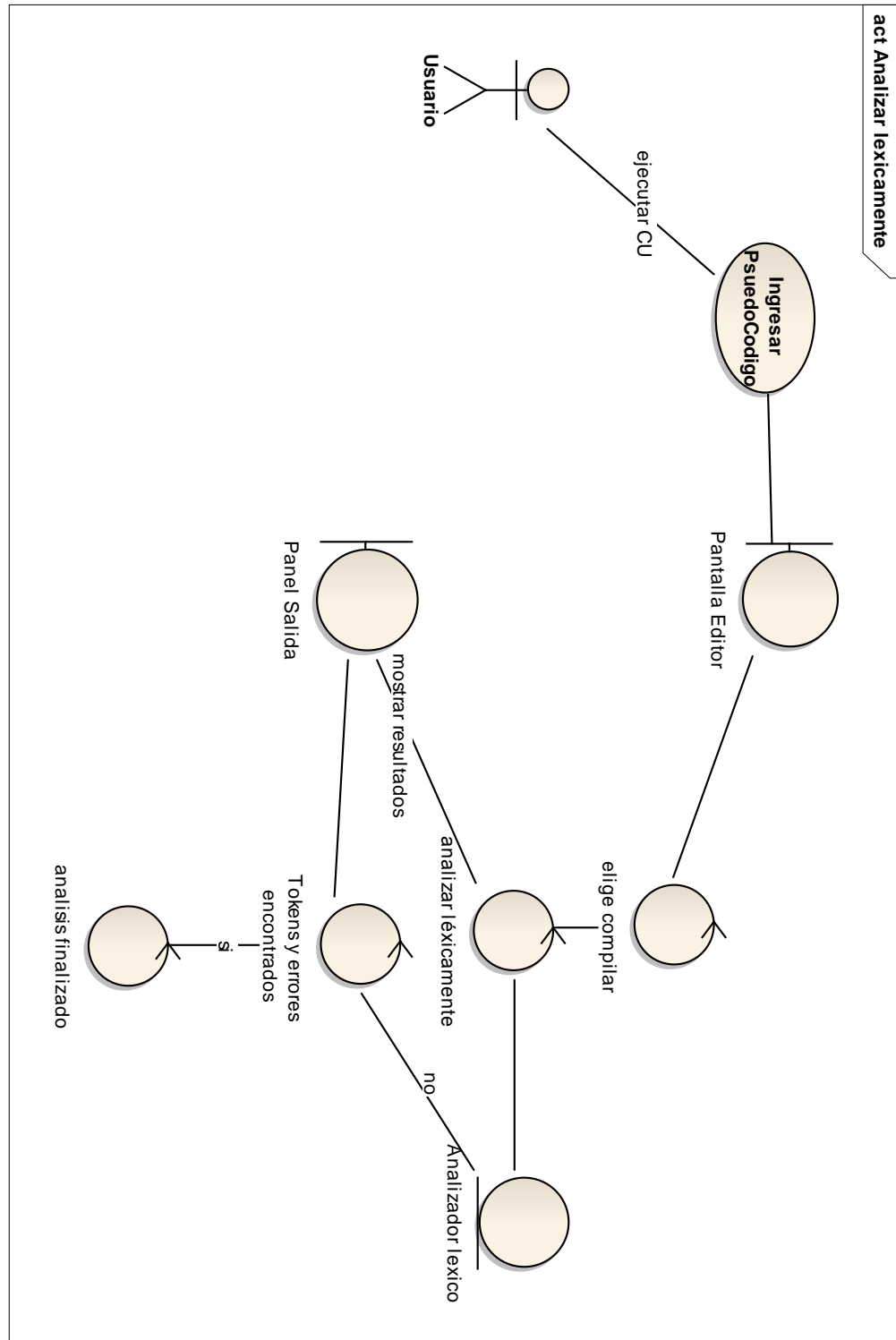


Diagrama 23. Diagrama de Robustez del Caso de Uso: Analizar Lexicamente

Caso de Uso: Analizador Sintactico

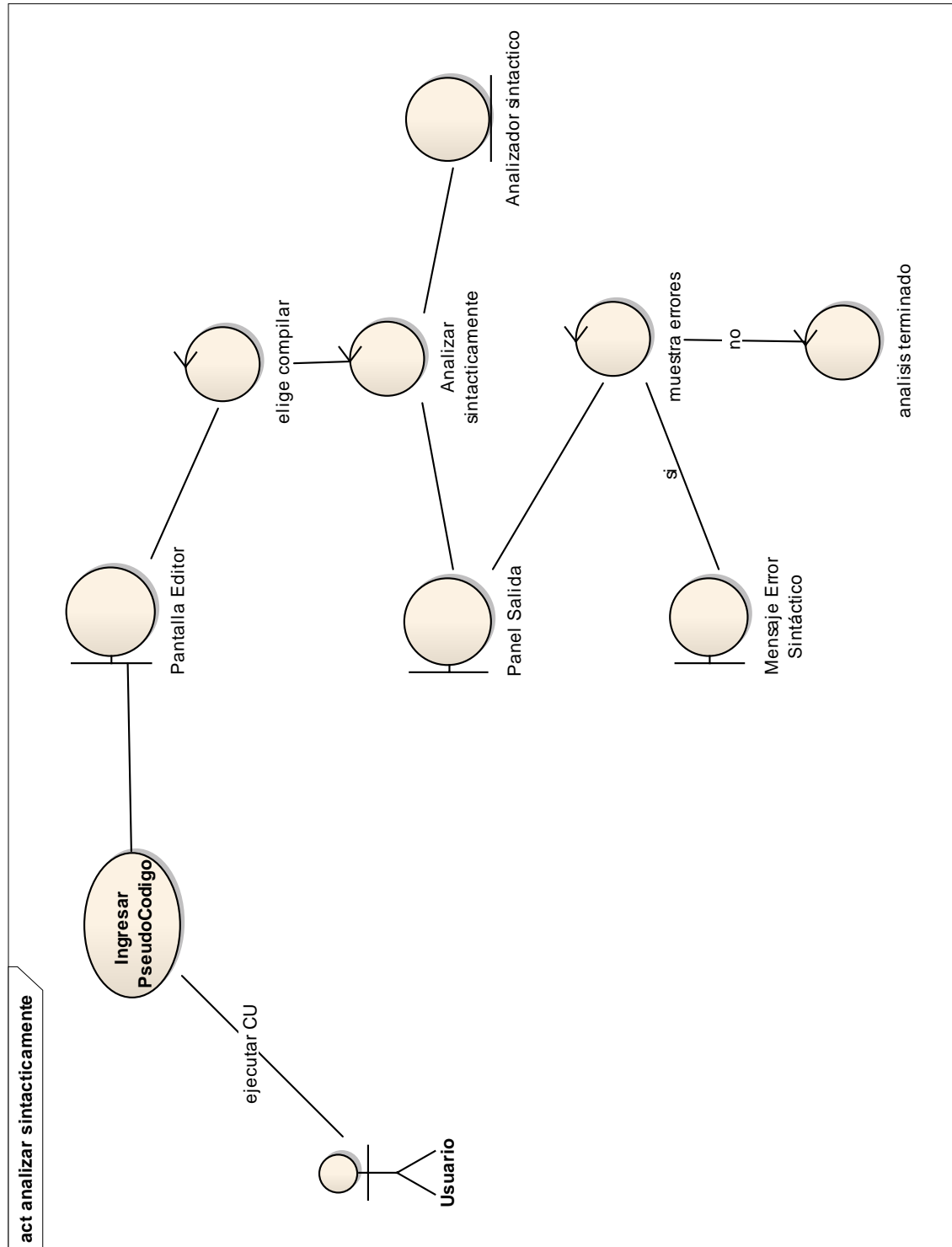


Diagrama 24. Diagrama de Robustez del Caso de Uso: Analizador Sintactico

Caso de Uso: Analizador Semantico

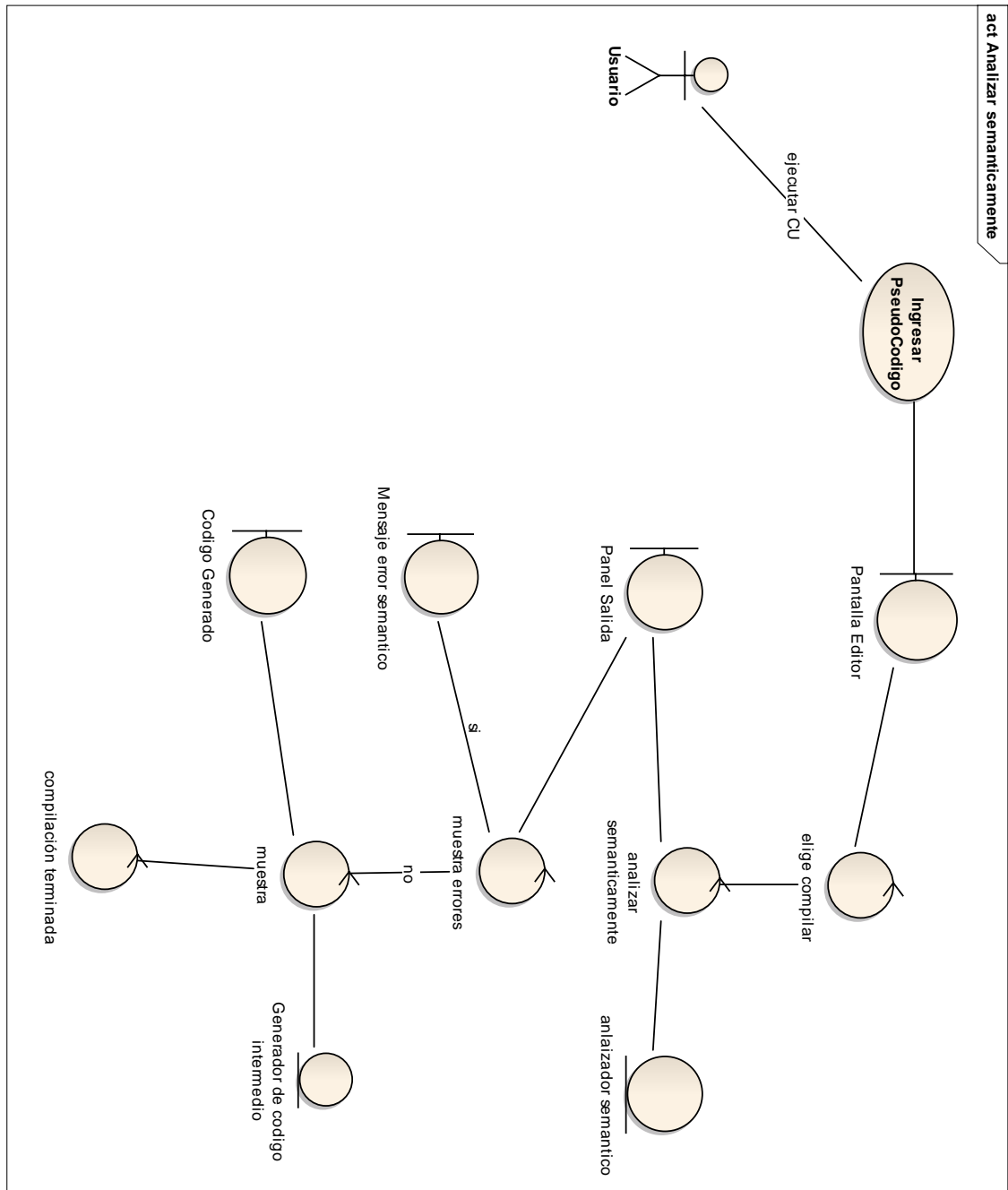


Diagrama 25. Diagrama de Robustez del Caso de Uso: Analizador Semantico

Caso de Uso: Gestionar Errores

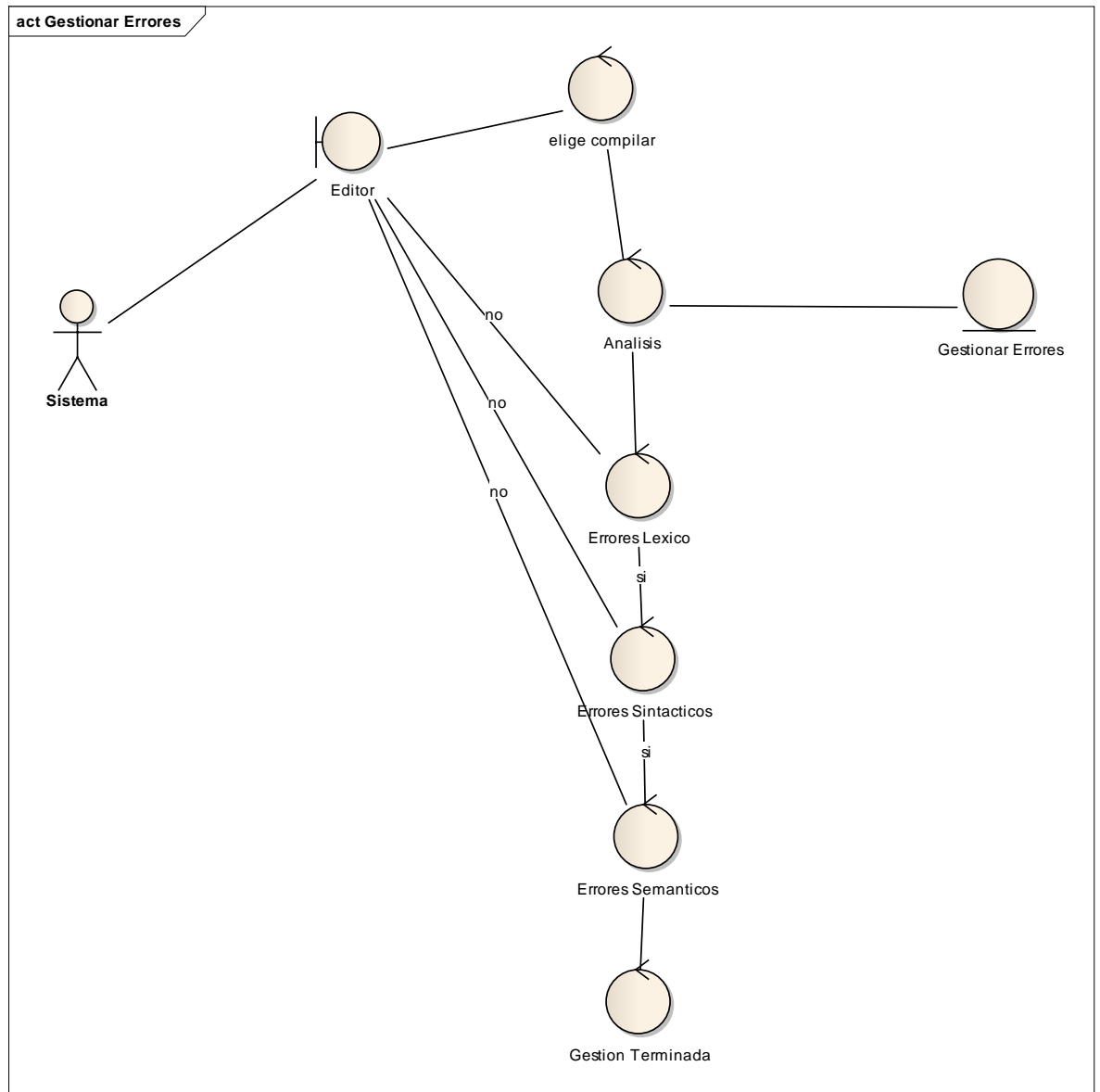


Diagrama 26. Diagrama de Robustez del Caso de Uso: Gestionar Errores

Caso de Uso: Crear Diagrama

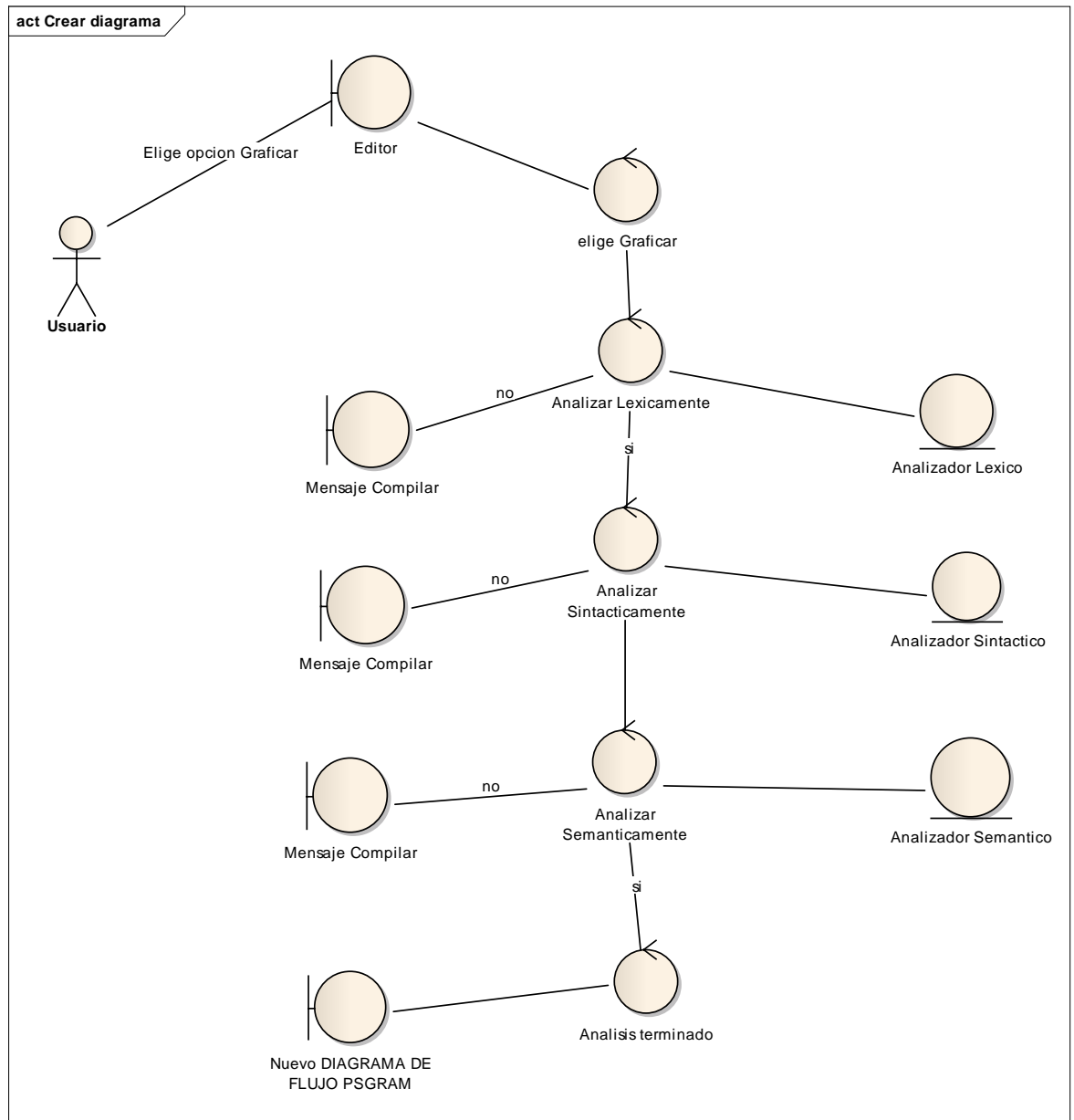


Diagrama 27. Diagrama de Robustez del Caso de Uso: Crear Diagrama

Caso de Uso: Dibujar Diagrama

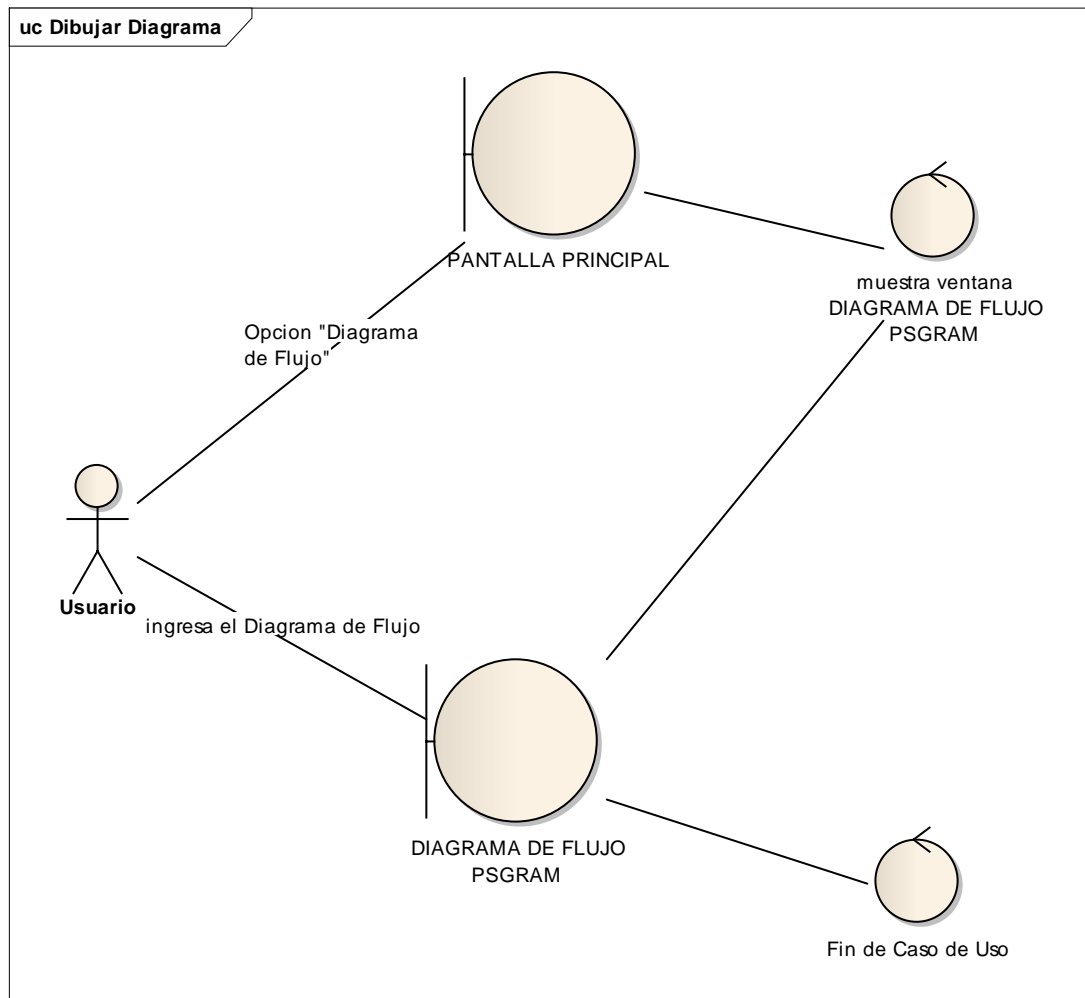


Diagrama 28. Diagrama de Robustez del Caso de Uso: Dibujar Diagrama

Caso de Uso: Revisar Errores

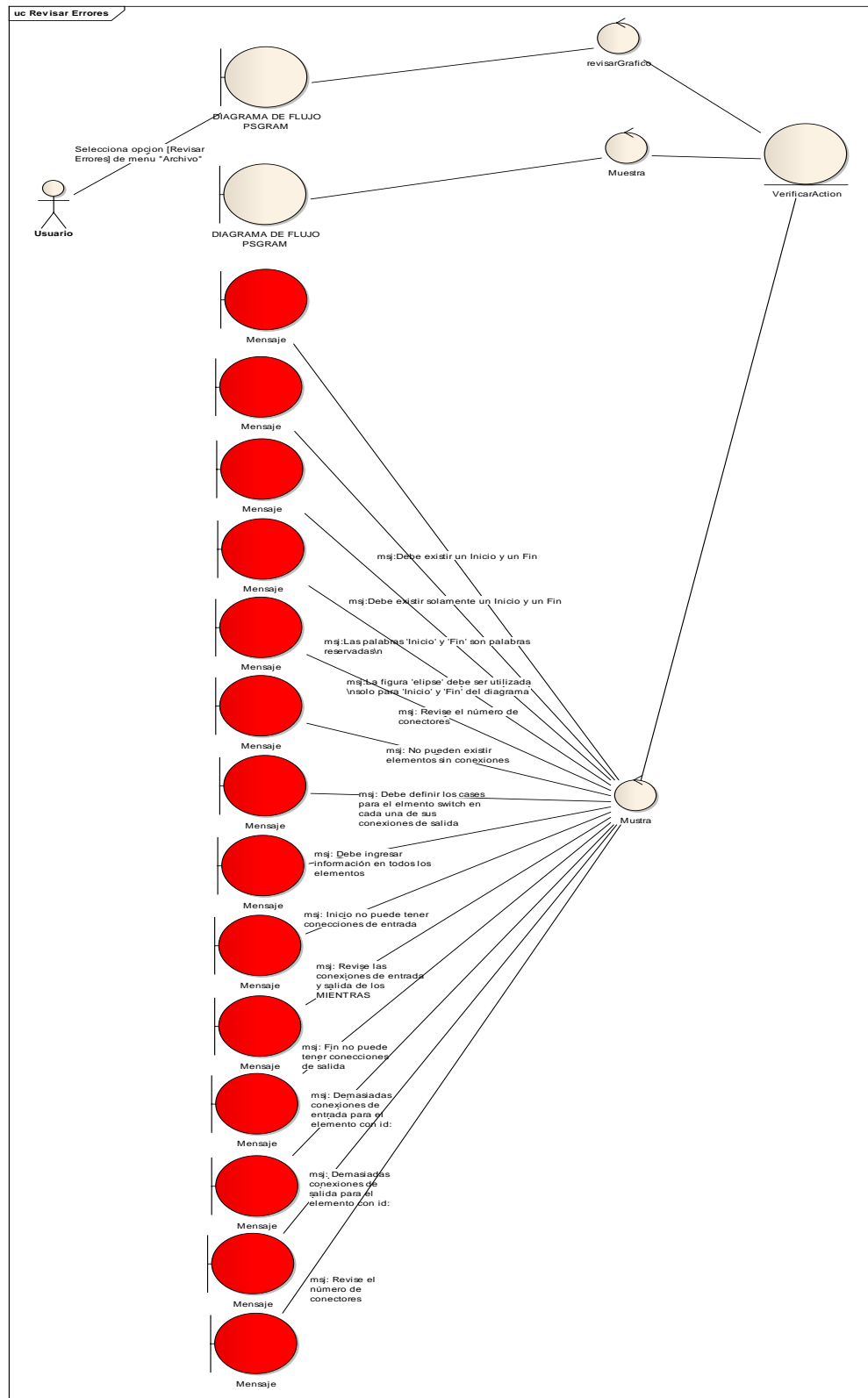


Diagrama 29. Diagrama de Robustez del Caso de Uso: Revisar Errores

Caso de Uso: Generar Pseudo-Codigo

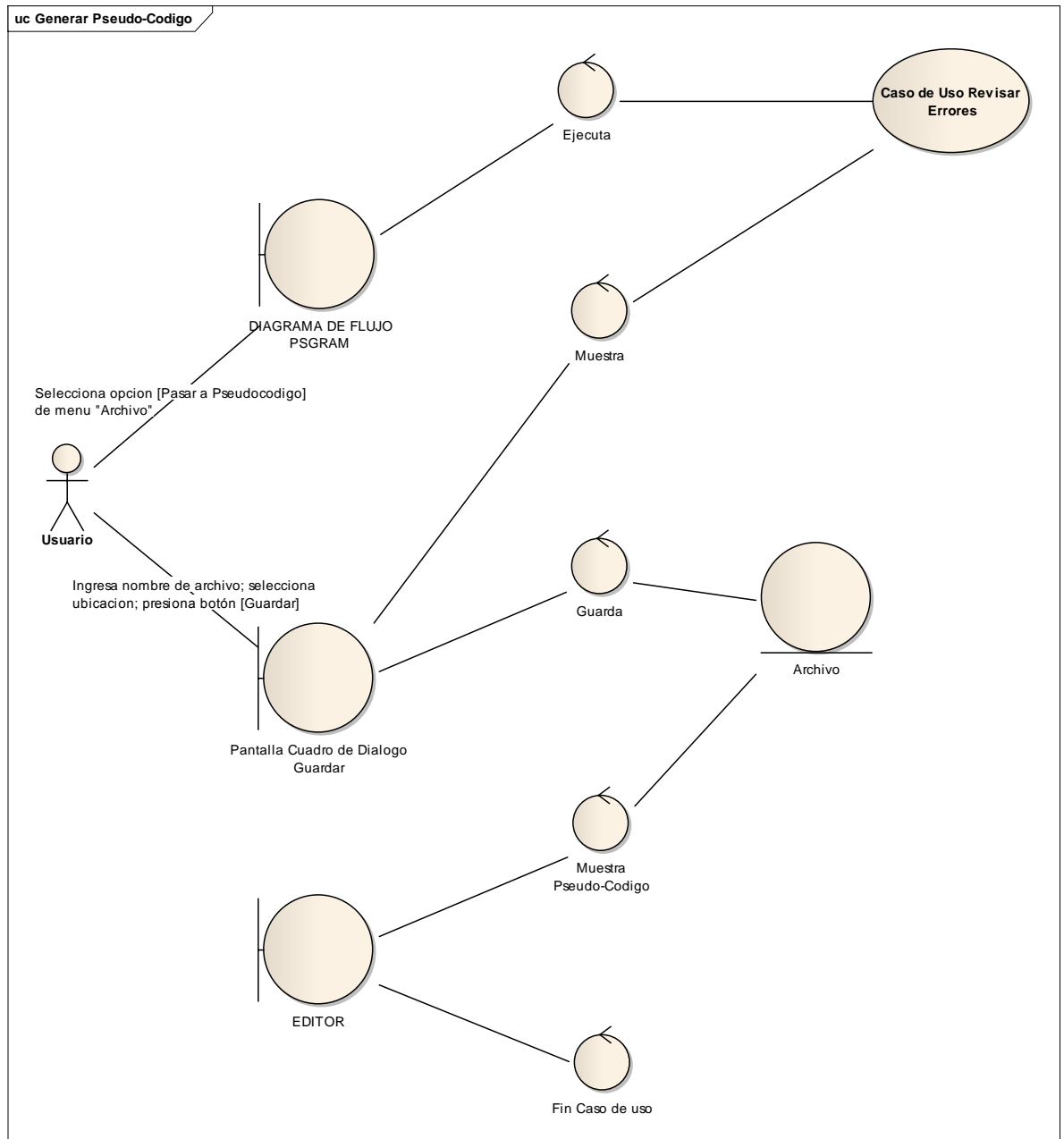


Diagrama 30. Diagrama de Robustez del Caso de Uso: Generar Pseudo-Codigo