DESARROLLO DE UN SISTEMA COMPUTACIONAL PARA EL APRENDIZAJE DE PROGRAMACIÓN ESTRUCTURADA "SCAPE"

JUAN DAVID ARIAS PATIÑO
MIGUEL ANGEL LOZANO ISAZA
ANDRES FELIPE MARIN RODRIGUEZ

UNIVERSIDAD TECNOLOGICA DE PEREIRA
FACULTAD DE INGENIERIAS
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PEREIRA
SEPTIEMBRE DE 2011

DESARROLLO DE UN SISTEMA COMPUTACIONAL PARA EL APRENDIZAJE DE PROGRAMACIÓN ESTRUCTURADA "SCAPE"

JUAN DAVID ARIAS PATIÑO
MIGUEL ANGEL LOZANO ISAZA
ANDRES FELIPE MARIN RODRIGUEZ

INFORME DE PROYECTO DE GRADO DE PREGRADO

UNIVERSIDAD TECNOLOGICA DE PEREIRA
FACULTAD DE INGENIERIAS
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PEREIRA
SEPTIEMBRE DE 2011

Nota de aceptación

_____

_____

_____

_____

_____

Firma del presidente del jurado

_____

Firma del jurado

DEDICATORIA

A Dios quien nos da fortaleza y salud, y a nuestros padres por su cariño, estímulo y apoyo constante durante el desarrollo de nuestra carrera.

JUAN DAVID ARIAS
MIGUEL ANGEL LOZANO

"A mi padre Dios quien siempre ha estado apoyándome y sosteniéndome en todo momento, quien ha estado en las buenas y en las malas, y quien me llevó de su mano a alcanzar este importante objetivo de mi vida, a Él todos mis agradecimientos.

A mi padre Humberto Antonio Marin, que aunque ya fallecido sé que siempre soñó con este momento y dio hasta su último esfuerzo para que así fuera, quiero hacerle sentir orgulloso desde donde me ve.

A mi madre quien con empeño y sacrificio estuvo dándome todo su apoyo para culminar mi carrera.

A mi hermana, quien fue mi inspiración y motivo para seguir adelante y no desfallecer en mi meta.

Y a todos aquellos amigos y conocidos que estuvieron animándome en continuar.

A todos, Gracias."

ANDRES FELIPE MARIN RODRIGUEZ

AGRADECIMIENTOS

Al semillero de investigación INSILICO por su constante apoyo y ayuda en el desarrollo del proyecto.

Al ingeniero Andrés Guillermo Velásquez Gómez y al grupo de investigación SIRIUS por su asesoría con los temas relacionados sobre dispositivos programables y por facilitarnos los elementos físicos necesarios para el desarrollo del proyecto.

Al ingeniero Omar Iván Trejos Buriticá por su interés en el proyecto y constante asesoría en el desarrollo del informe del proyecto y temas de pedagogía.

Al ingeniero Hugo Humberto Morales Peña por el apoyo y por facilitarnos los elementos necesarios para el desarrollo del proyecto.

Finalmente a la Universidad Tecnológica de Pereira y a los profesores del programa que durante el transcurso de la carrera nos brindaron conocimientos profesionales para la vida.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE IMÁGENES

GLOSARIO

- Ámbito: El termino Ámbito es aquel que se utiliza para referirse a un perímetro o espacio determinado, ya sea un espacio que se refiera a un lugar, o un espacio que se refiera a una terminología.

- Aprendizaje: El aprendizaje es el proceso a través del cual se adquieren o modifican habilidades, destrezas, conocimientos, conductas o valores como resultado del estudio, la experiencia, la instrucción, el razonamiento y la observación. Este proceso puede ser analizado desde distintas perspectivas, por lo que existen distintas teorías del aprendizaje. El aprendizaje es una de las funciones mentales más importantes en humanos, animales y sistemas artificiales.

- FPGA: Un FPGA (del inglés Field Programmable Gate Array) es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada 'in situ' mediante un lenguaje de descripción especializado.

- Hardware: Corresponde a todas las partes tangibles de una computadora: sus componentes eléctricos, electrónicos, electromecánicos y mecánicos; sus cables, gabinetes o cajas, periféricos de todo tipo y cualquier otro elemento físico involucrado; contrariamente, el soporte lógico es intangible y es llamado software.

- Hipermedia: Hipermedia es el término con el que se designa al conjunto de métodos o procedimientos para escribir, diseñar o componer contenidos que integren soportes tales como: texto, imagen, video, audio, mapas y otros soportes de información emergentes, de tal modo que el resultado obtenido, además tenga la posibilidad de interactuar con los usuarios.

- IDE: Un entorno de desarrollo integrado (en inglés integrated development environment) es un programa informático compuesto por un conjunto de herramientas de programación. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador y un constructor de interfaz gráfica (GUI).

- Intérprete (informática): En ciencias de la computación, intérprete o interpretador es un programa informático capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel.

- Lenguaje (Programación): Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.

- Multimedia: El término multimedia se utiliza para referirse a cualquier objeto o sistema que utiliza múltiples medios de expresión (físicos o digitales) para presentar o comunicar información. De allí la expresión «multimedios». Los medios pueden ser variados, desde texto e imágenes, hasta animación, sonido, video, etc.

- Programación Estructurada: La programación estructurada es una técnica para escribir programas (programación de computadora) de manera clara.

- Prototipo: Un prototipo también se puede referir a cualquier tipo de máquina en pruebas, o un objeto diseñado para una demostración de cualquier tipo. Éstos permiten testar el objeto antes de que entre en producción, detectar errores, deficiencias, etcétera. Cuando el prototipo está suficientemente perfeccionado en todos los sentidos requeridos y alcanza las metas para las que fue pensado, el objeto puede empezar a producirse.

- Robot: Un robot es una entidad virtual o mecánica artificial. En la práctica, esto es por lo general un sistema electromecánico que, por su apariencia o sus movimientos, ofrece la sensación de tener un propósito propio. La palabra robot puede referirse tanto a mecanismos físicos como a sistemas virtuales de software, aunque suele aludirse a los segundos con el término de bots.

- Simulación por computadora: Un modelo de simulación por computador o un modelo informatizado es un programa informático o una red de ordenadores cuyo fin es crear una simulación de un modelo abstracto de un determinado sistema.

- Sistema Computacional o Sistema Informático: Un sistema informático como todo sistema, es el conjunto de partes interrelacionadas, hardware, software y de recurso humano (humanware) que permite almacenar y procesar información.

- Software: Se conoce como software al equipamiento lógico o soporte lógico de una computadora digital; comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados hardware.

- Taxonomía: La taxonomía, en su sentido más general, es la ciencia de la clasificación.

- Token: Un token o también llamado componente léxico es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación. Ejemplos de tokens, podrían ser palabras clave (if, else, while, int, ...), identificadores, números, signos, o un operador de varios caracteres, (por ejemplo, :=). Son los elementos más básicos sobre los cuales se desarrolla toda traducción de un programa, surgen en la primera fase, llamada análisis léxico, sin embargo se siguen utilizando en las siguientes fases (análisis sintáctico y análisis semántico) antes de perderse en la fase de síntesis.

RESUMEN

En el presente documento se propone un sistema computacional que sirve como alternativa en la enseñanza de la programación estructurada a un usuario tecnológico promedio.

A lo largo del documento se muestran el proceso de análisis y diseño de los diferentes elementos que componen la propuesta, y como se construyó el prototipo del sistema que integrara hardware y software.

Finalmente se presentan las conclusiones del trabajo y algunas recomendaciones sobre el funcionamiento y proyectos futuros que pueden derivarse del sistema.

1. ANTEPROYECTO

1.1 TITULO

Desarrollo de un Sistema computacional para el aprendizaje de programación estructurada.

1.2 FORMULACIÓN DEL PROBLEMA

En los últimos 10 años se ha desarrollado la utilización de lenguajes con el fin específico de introducir a la programación, especialmente para niños y jóvenes, y algunos de estos lenguajes están siendo utilizados en las aulas universitarias. Estos lenguajes están basados en la lengua inglesa y algunos cuentan con simulaciones, entre estos están Scratch, Alice, y algunas derivaciones de Logo.[1]

En la tabla 1.1 se presenta un rango de los lenguajes de programación con más popularidad en el mundo, se resaltan los orientados a la enseñanza.

Position	Programming Language	Ratings
21	Transact-SQL	0.500%
22	JavaFX Script	0.485%
23	Lisp/Scheme	0.481%
24	Scratch	0.476%
25	D	0.463%
26	Bourne Shell	0.437%
27	Scala	0.433%
28	Erlang	0.420%
29	COBOL	0.398%
30	RPG (OS/400)	0.379%
31	S-lang	0.372%
32	Logo	0.361%
33	Ada	0.357%

1 http://www.tecnun.es/departamentos/doi/personal/nserrano/proyectos-final-de-carrera/analisis-de-lenguajes-visuales-para-la-ensenanza-de-programacion-en-entorno-universitario.html

34	Fortran	0.350%
35	NXT-G	0.321%
36	Tcl/Tk	0.319%
37	Alice	0.317%
38	Forth	0.296%
39	FoxPro/xBase	0.291%
40	Caml/F#	0.280%
41	PowerShell	0.268%
42	CL (OS/400)	0.259%
43	Prolog	0.258%
44	Groovy	0.238%
45	Smalltalk	0.221%
46	VHDL	0.220%
47	J	0.213%
48	C Shell	0.198%
49	MAX/MSP	0.196%
50	VBScript	0.186%

Tabla 1.1: Lenguajes de programación más populares, Tomado de TIOBE

Actualmente hay tendencias de utilizar los lenguajes que se utilizan en el mundo profesional o que imponen una disciplina de diseño para la enseñanza de la programación, estos presentan al programador una gran variedad de características y usos que pueden ser confusos a la hora de adquirir conocimientos básicos debido a que estos están hechos para el desarrollo comercial o investigativo y no para la enseñanza. En la imagen 1.1 se presenta una gráfica de los lenguajes de programación más populares en el mundo, todos basados en el idioma inglés.

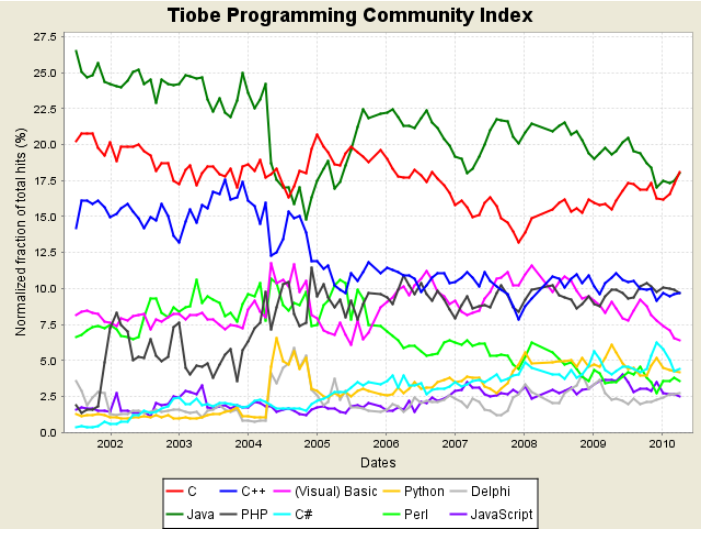


Imagen 1.1: Idiomas más populares en el mundo, todos basados en el idioma ingles

Aunque en el siglo XXI, la programación orientada a objetos se ha convertido en la tecnología de software más utilizada; el conocimiento profundo de algoritmos y estructuras de datos, en muchos casos con el enfoque estructurado, facilita al programador los fundamentos técnicos necesarios para convertirse en un brillante programador orientado a objetos.

Por lo anterior se podría decir que, la programación estructurada sirve de fundamentación para los desarrollos de POO. No es que sea mejor una que otra.[2]

Según lo expuesto anteriormente se plantea: ¿Será posible que con un Sistema computacional, un usuario tecnológico promedio asimile fácilmente los conceptos básicos de programación estructurada?

Entendiéndose a sistema computacional como la combinación de software y hardware para ejecutar aplicaciones didácticas; los estudiantes de Programación I de la Universidad Tecnológica de Pereira del programa de Ingeniería de Sistemas y Computación se entenderán como usuarios tecnológicos promedio; y los conceptos básicos de programación estructurada se entienden como manejo de:

- Funciones
- Ciclos
- Condicionales
- Variables
- Instrucciones

2 http://www.mcgraw-hill.es/bcv/guide/capitulo/8448146433.pdf

1.3 JUSTIFICACIÓN

Según el Plan Nacional Decenal de Educación (PNDE) 2006-2016 presentado por el Ministerio de Educación Nacional, en el capítulo 1 según los desafíos de la educación en Colombia de la renovación pedagógica y uso de las TIC en la educación, y de la Ciencia y tecnología integradas a la educación, se plantea el fortalecimiento de una cultura de ciencia, tecnología e innovación; formando el talento humano necesario para el desarrollo de la ciencia, la tecnología y la innovación; fortaleciendo la educación técnica y tecnológica, de tal manera que responda a las necesidades del mercado laboral, el sector productivo y la sociedad; el fortalecimiento de procesos pedagógicos a través de las TIC; facilitar el aprendizaje autónomo, colaborativo y el pensamiento crítico y creativo; entre otras que sirva de ruta y horizonte para el desarrollo educativo del país.[3]

El desarrollo de herramientas para la construcción del conocimiento, donde los estudiantes puedan aprender con ellas, no de ellas, permite que los estudiantes actúen como diseñadores, y la tecnología opere como herramienta de la mente para interpretar y organizar su creatividad.[4]

1.4 OBJETIVO GENERAL

Implementar un Sistema computacional, con el que un usuario tecnológico promedio asimile fácilmente los conceptos básicos de programación estructurada

1.5 OBJETIVO ESPECÍFICOS

- Especificar formalmente un lenguaje de programación estructurada mediante una gramática que defina el comportamiento del prototipo físico y el prototipo simulado.
- Desarrollar el interpretador del lenguaje diseñado para el comportamiento del prototipo físico (robot) y el prototipo simulado.
- Construir un prototipo físico (robot) que ejecute un subconjunto del código programado por el usuario.
- Construir una simulación en 2D para el prototipo simulado en la cual se ejecute el código programado por el usuario.
- Implementar un entorno de desarrollo gráfico.
- Documentar los manuales.

3 Plan Decenal Nacional De Educación.

4 http://www.eduteka.org/modulos.php?catx=9&idSubX=272&ida=78&art=1

1.6 MARCO REFERENCIAL

1.6.1 MARCO DE ANTECEDENTES

En lo últimos 10 años se ha propuesto diversos conjuntos de habilidades para la educación del pensamiento estructurado, lenguajes mezclados con filosofías de enseñanza con el fin específico de introducir a la programación para la resolución de problemas, donde podemos encontrar lenguajes de programación orientados a la enseñanza. A continuación se describe brevemente algunos de ellos.

El Lenguaje de Programación Logo fue diseñado por Danny Bobrow, Wally Feurzeig y Seymour Papert. Es un dialecto de Lisp, este fue diseñado como una herramienta para el aprendizaje, cuenta con las siguientes características: modularidad, extensibilidad, interactividad, flexibilidad, siguiendo el objetivo del aprendizaje.[5]

Logo no es solamente un lenguaje de computación o un software muy difundido en Educación, sino que es además un recurso que somete al usuario a una prueba sobre la coherencia de sus ideas al tratar de ordenar algo a la computadora, o a un robot. El uso frecuente de este programa pone en evidencia el proceso intelectual realizado por los alumnos y facilita la auto corrección en el razonamiento lógico. El aprendizaje con Logo es activo, exploratorio y vivencial.[6]

Scratch desarrollado por el Lifelong Kindergarten Group del MIT Media Lab, es un lenguaje gráfico de programación que aprovecha los avances en poder de computación y en diseño de interfaces para hacer que la programación sea más atractiva y accesible para niños, adolescentes y todo aquel que esté aprendiendo a programar. Las características claves de este lenguaje son: programación con bloques de construcción, manipulación de medios y compartir y colaborar.

A medida que los estudiantes trabajan en proyectos de Scratch, tienen oportunidades para aprender conceptos de computación importantes, tales como iteración, condicionales, variables, tipos de datos, eventos y procedimientos. Scratch se ha usado para presentar por primera vez estos conceptos a estudiantes de diferentes edades, desde la escuela elemental (primaria) hasta educación superior. Algunos estudiantes hacen la transición a lenguajes

5 http://el.media.mit.edu/Logo-foundation/logo/index.html

6 http://www.rmm.cl/index_sub.php?id_contenido=5349&id_seccion=3437&id_portal= 520

tradicionales basados en texto, después de haberse iniciado en la programación con Scratch.[7]

Algunos de los anteriores lenguajes de enseñanza siguen el modelo constructivista, su idea central es que el aprendizaje humano se construye, que la mente de las personas elabora nuevos conocimientos a partir de la base de enseñanzas anteriores. El aprendizaje de los estudiantes debe ser activo, deben participar en actividades en lugar de permanecer de manera pasiva observando lo que se les explica. [8]

Los estudiantes tienen la oportunidad de ampliar su experiencia de aprendizaje al utilizar las nuevas tecnologías como herramientas para el aprendizaje constructivista. Estas herramientas le ofrecen opciones para lograr que el aula tradicional se convierta en un nuevo espacio, en donde tienen a su disposición actividades innovadoras de carácter colaborativo y con aspectos creativos que les permiten afianzar lo que aprenden al mismo tiempo que se divierten. Estas características dan como resultado que el propio alumno sea capaz de construir su conocimiento con el profesor como un guía y mentor, otorgándole la libertad necesaria para que explore el Sistema tecnológico, pero estando presente cuando tenga dudas o le surja algún problema.

Los programas de computador que representan modelos del mundo real y permiten a los estudiantes explorar, manipular y experimentar con esos modelos son llamados micromundos.

Estos Sistemas permiten elaborar simulaciones restringidas de fenómenos del mundo real, controlables por los estudiantes. Además, aportan la funcionalidad exploratoria (herramientas de observación, manipulación y objetos de prueba) necesaria para examinar dichos fenómenos. Los Micromundos son quizás el más reciente ejemplo de Sistemas de aprendizaje activo, en que los estudiantes pueden ejercer bastante control sobre sus creaciones. [9]

7 http://www.eduteka.org/modulos.php?catx=9&idSubX=284&ida=908&art=1

8 http://www.uoc.edu/rusc/5/2/dt/esp/hernandez.pdf

9 http://www.eduteka.org/modulos/9/286

1.6.2 MARCO CONCEPTUAL

INTERPRETE

Es un programa informático capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel. Los intérpretes se diferencian de los compiladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los primeros (los intérpretes) sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.[10]

Las primeras tres fases son de análisis: análisis léxico, sintáctico y semántico, fases en las que se leen los caracteres del código fuente, se analizan, se comprueban si son válidos y se van reagrupando en secuencias lógicas y frases gramaticales. Esta primera parte es la que se conoce como Front End o Análisis.

ANÁLISIS LÉXICO

En el llamado análisis lexicográfico o léxico, el intérprete revisa y controla que las "palabras" estén bien escritas y pertenezcan a algún tipo de token (cadena) definido dentro del lenguaje, como por ejemplo que sea algún tipo de palabra reservada, o si es el nombre de una variable que este escrita de acuerdo a las pautas de definición del lenguaje. En esta etapa se crea la tabla de símbolos, la cual contiene las variables y el tipo de dato al que pertenece, las constantes literales, el nombre de funciones y los argumentos que reciben etc.[11]

ANÁLISIS SINTÁCTICO

En el análisis sintáctico como su nombre lo indica se encarga de revisar que los tokens estén ubicados y agrupados de acuerdo a la definición del lenguaje. Dicho de otra manera, que los tokens pertenezcan a frases gramaticales validas, que el compilador utiliza para sintetizar la salida. Por lo general las frases gramaticales son representadas por estructuras jerárquicas, por medio de árboles de análisis sintáctico. En esta etapa se completa la tabla de símbolos con la dimensión de los identificadores y los atributos necesarios etc.

10 http://www.mastermagazine.info/termino/4368.php

11 http://www.investigacion.frc.utn.edu.ar/labsis/Publicaciones/InvesDes/Compiladores/rxc.htm

ANÁLISIS SEMÁNTICO

El análisis semántico se encarga de revisar que cada agrupación o conjunto de token tenga sentido, y no sea un absurdo. En esta etapa se reúne la información sobre los tipos para la fase posterior, en esta etapa se utiliza la estructura jerárquica de la etapa anterior y así poder determinar los operadores, y operandos de expresiones y preposiciones.

MANEJADOR DE ERRORES

Durante todo este proceso, se va ejecutando en forma permanente el manejador de errores, que se encarga de analizar en cada una de las fases los posibles errores que puede haber durante el proceso de la traducción. No es una etapa del proceso, sino que es una función, muy importante, pues al ocurrir un error esta función debe tratar de alguna forma el error para así seguir con el proceso de compilación.

FPGA

Una FPGA (Field Programmable Gate Array) es un circuito integrado que contiene bloques de lógica cuya interconexión es configurada, la configuración es generalmente especificada usando un lenguaje de descripción de hardware (HDL). Las FPGAs tienen una jerarquía de interconexiones programables que permite a los bloques lógicos de un FPGA ser interconectados según la necesidad del diseñador del sistema. Estos bloques lógicos e interconexiones pueden ser programados después del proceso de manufactura por el usuario/diseñador, así que el FPGA puede desempeñar cualquier función lógica necesaria.[12]

MICROBLAZE

Es un soft-procesor descrito con el lenguaje de programación HDL diseñado por xilinx, está diseñado bajo una arquitectura RISC de 32 bits, es implementado totalmente en la memoria de propósito general de la FPGA, esta optimizado para desarrollar aplicaciones embebidas y es totalmente flexible en la integración con periféricos, además soporta la configuración de las diferentes partes que componen el procesador: tamaño de cache, pipeline, periféricos, unidad de memoria y las interfaces de sus buses. Soporta la capacidad de correr un pequeño kernel de Linux o correr en modo stand-alone. El compilador para microblaze está basado en el gcc desarrollado por el proyecto GNU, por lo cual no tiene ninguna

12 http://www.xilinx.com/company/gettingstarted/index.htm

restricción al escribir código para un sistema microblaze. [13] En la imagen 1.2 se muestra la arquitectura del procesador Microblaze.

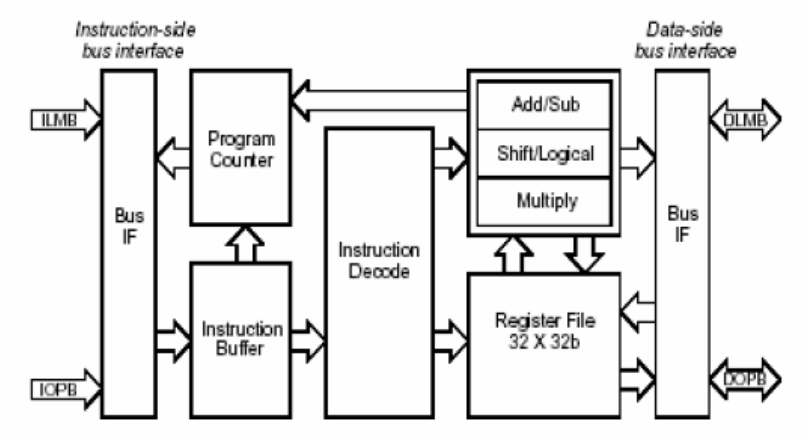


Imagen 1.2: Procesador Microblaze

EDK

EDK de Xilinx (Embedded Development Kit) es el paquete de desarrollo e implementación de MicroBlaze, este consta de dos entornos el XPS y SDK.

Los diseñadores utilizan XPS (Xilinx Platform Studio) para configurar y construir la especificación de hardware de su sistema integrado (núcleo del procesador, la memoria de controlador, I / O periféricos, etc). La XPS convierte las especificaciones del diseño de la plataforma en una descripción RTL sintetizable (Verilog o VHDL), y escribe un conjunto de scripts para automatizar la aplicación del sistema integrado Para el núcleo MicroBlaze, el EDK normalmente genera una conexión encriptada.

El SDK controla el software que se ejecuta en el sistema embebido. Desarrollado por GNU (GNU Compiler Collection, depurador de GNU), el SDK permite a los programadores escribir, compilar, depurar aplicaciones para su sistema embebido. Xilinx incluye un simulador de sistema de instrucción ciclo con precisión (ISS), dando a los programadores la elección de probar el software en una simulación, o el uso de una FPGA adecuada para descargar y ejecutar en el sistema real.[14]

13 http://www.xilinx.com/tools/microblaze.html

14 http://en.wikipedia.org/wiki/MicroBlaze

MODELOS DE CALIDAD DE SOFTWARE

MOSCA: Es un modelo para estimar la calidad del software, este responde a las necesidades de garantizar productos de calidad en una competencia abierta y mundial, sobre la base de las características propias de cada país y de cada tipo de Sistema de Software. Este integra el modelo de Calidad del Producto y el modelo de Calidad del Proceso.[15]

ADECUACIÓN DE MOSCA PARA SOFTWARE EDUCATIVO

En el trabajo “Instrumento de Evaluación de software educativo bajo un enfoque sistémico” (DíazAntón y otros.) se hace una adaptación del modelo original de MOSCA (Modelo Sistémico de Calidad del Software. Mendoza y otros (2002)) para la evaluación del software educativo específicamente.

La estructura del modelo consta de cuatro niveles.

Nivel 0: Dimensiones. Efectividad Producto.

Nivel 1: Categorías. Se contemplan tres categorías:

- Funcionalidad (FUN): Es la capacidad del producto del software para proveer funciones que cumplan con necesidades específicas o implícitas, cuando el software es utilizado bajo ciertas condiciones.
- Usabilidad (USA): Esta categoría se refiere a la capacidad del producto de software para ser atractivo, entendido, aprendido y utilizado por el usuario bajo condiciones específicas.
- Fiabilidad (FIA): La fiabilidad es la capacidad del producto de software para mantener un nivel especificado de rendimiento cuando es utilizado bajo condiciones especificadas.

Nivel 2: Características. Cada categoría tiene asociado un conjunto de características.

Nivel 3: Subcaracterísticas. Para algunas de las características se asocian un conjunto de subcaracterísticas.

15 http://redalyc.uaemex.mx/src/inicio/ArtPdfRed.jsp?iCve=61580304

Nivel 4: Métricas. Para cada característica se propone una serie de métricas utilizadas para medir la calidad sistémica. Se evalúan métricas adicionales relacionadas con Funcionalidad, Usabilidad y Fiabilidad, que permitan adaptar MOSCA en el área de software educativo.

En la tabla 1.2 se muestra una serie de categorías, características y subcaracterísticas del modelo de evaluación de calidad.

CATEGORÍAS	CARACTERÍSTICAS	SUBCARACTERÍSTICAS
FUNCIONALIDAD (FUN)	FUN 1. Ajuste a los propósitos	FUN.1.1 General FUN.1.2 Objetivos de aprendizaje FUN.1.3 Contenidos de aprendizaje FUN.1.4 Actividades de aprendizaje FUN.1.5 Ejemplos FUN.1.6 Motivación FUN.1.7 Retroalimentación FUN 1.8 Ayudas FUN.1.9 Metodología de enseñanza
	FUN.2 Seguridad	
USABILIDAD (USA)	USA.1 Facilidad de comprensión	USA.1.1 General USA.1.2 Interactividad USA.1.3 Diseño de la interfaz
	USA.2 Capacidad de uso USA.3 Interfaz Gráfica USA.4 Operabilidad	
FIABILIDAD (FIA)	FIA.1 Recuperación FIA.2 Tolerancia a fallas	

Tabla 1.2: Criterios de Evaluación seleccionados para nuestro software basados en el modelo adaptado de MOSCA, tomado de http://developyourdream.net/tutoriales/tesis/tomo_pdf/CAPITULO%203%20-%20METODOLOG%CDA.pdf

ALGORITMO PARA LA EVALUACIÓN DE LA CALIDAD DE SOFTWARE EDUCATIVO

El algoritmo consta de dos fases, la primera fase describe los pasos a seguir para la preselección del software, en caso de que sea adquirido por vía comercial. Para este caso en particular, no se tomará en cuenta dicha fase, por lo que se describe a continuación la segunda fase del algoritmo.

Fase 2. Estimar la calidad del producto de software educativo a través de un enfoque sistémico.

En esta fase se evalúa la calidad del producto siguiendo tres actividades:

1. Estimar la calidad de la funcionalidad del producto.

- Para la categoría de FUNCIONALIDAD (FUN) se debe cumplir la característica de “Ajuste a los propósitos” y al menos una de las dos restantes, “Precisión” o “Seguridad”. (Díaz y otros. 2001)

2. Estimación de la calidad para cada categoría.

Para las dos (2) categorías restantes, USABILIDAD y FIABILIDAD, se debe:

- Aplicar las métricas propuestas en el submodelo del producto para las categorías seleccionadas y normalizar los resultados de las métricas a una escala de 1 al 5. (DíazAntón y otros.)
- Verificar que el 75% de las métricas se encuentran dentro de los valores óptimos (mayor o igual a 4) para cada una de las subcaracterísticas y características. Si no se cumple el 75% de las métricas asociadas, entonces esa subcaracterística o característica tendrá calidad nula.
- Evaluar la categoría. Una categoría es satisfecha si el número de características es altamente satisfecho; es decir, si satisface el 75% de las características asociadas a la categoría. Tratándose de software educativo, donde existen características que son imprescindibles que estén presentes, se proponen las características mínimas satisfechas que debe tener cada categoría del producto para que ésta pueda ser satisfecha.

3. Estimar la calidad del producto partiendo de las categorías evaluadas.

Para que un software educativo obtenga calidad intermedia, deben ser satisfechas las categorías de FIABILIDAD y USABILIDAD. Para alcanzar la calidad avanzada todas las categorías deben ser satisfechas. (Díaz y otros. 2001).[16]

1.6.3 MARCO TEÓRICO

TEORÍAS Y MÉTODOS DE ENSEÑANZA EN EL USO DE SOFTWARE EDUCATIVOS

La relación entre Software Educativo y Aprendizaje puede articularse considerando los principales modelos de aprendizaje que han orientado la acción e investigación pedagógica a lo largo de este siglo.[17]

16 http://www.academia-interactiva.com/evaluacion.pdf

17 http://www.enlaces.cl/doc/evaluacion_recursos_educativos.doc.

TAXONOMÍA DEL SOFTWARE EDUCATIVO

La taxonomía propuesta por el pedagogo Sánchez (1998) que se observa en la imagen 1.3, permite relacionar a los modelos de aprendizaje y su influencia en el diseño de software educativos. Así, tenemos software que permiten, según su orientación, la Presentación, Representación y Construcción de información y conocimiento.

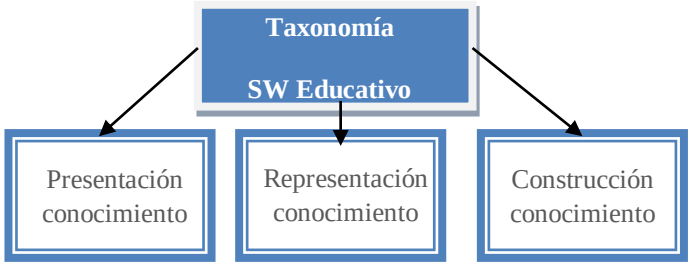


Imagen 1.3: Taxonomía del software educativo

PRESENTACIÓN

Es un programa que presenta información y conocimientos bajo un modelo tutorial de aprendizaje, donde usualmente la modalidad de interacción con el usuario se basa en un ciclo, contenido - preguntas - presentación - preguntas. Si bien el formato de presentación comienza a ser variado, impactante y algo motivador, el modelo que subyace sigue siendo conductista. Ha cambiado la tecnología pero el contenido y las formas de interacción del usuario permanecen intactas. Su modelo implícito es que con sólo presentar la información y los conocimientos, éstos serán idealmente incorporados por el aprendiz. En este modelo, la acción, el control, el ritmo y la interacción están determinados más por el software que por el usuario.

REPRESENTACIÓN

Trata la información y conocimiento de la misma forma como éstos hipotéticamente se organizan y representan en las estructuras mentales de los usuarios. Es decir, la forma de organizar los contenidos se asemeja a modelos de organización de información en memoria. La estructura del software, su navegación y la interacción con el usuario intentan imitar la forma como se almacenaría la información en la memoria. La idea es que la información pueda ser representada mediante una comparación metafórica de la relación estructural entre conceptos del programa y posibles estructuras mentales formadas por el aprendiz.

CONSTRUCCIÓN

Está centrado en el aprendiz y entrega herramientas, materiales, elementos y estrategias para que éste construya y reconstruya su conocimiento. Esto es principalmente sustentado por el hecho que el aprendiz, para trabajar con el software, debe hacer cosas, construir, reconstruir, resolver, crear, corregir y reparar los errores. El aprendiz hace cosas con el software y no el software hace cosas con él. En este tipo de software existe una intencionalidad de desarrollar o estimular el uso de algún proceso cognitivo y su transferencia al aprender. Aquí el aprendiz juega, se entretiene, resuelve complejidades, controla variables, se enfrenta a situaciones inciertas, resuelve problemas, etc.

En general, los distintos tipos de software responden a un modelo de aprender. Los software de presentación tienden a responder a un modelo de estímulo-respuesta, los de representación a un modelo de estructuración en memoria semántica o conceptual y los de construcción a un modelo activo de aprender y conocer.

TIPOS DE SOFTWARE EDUCATIVOS

En la tabla 1.3 se presenta una breve descripción de los diferentes tipos de software educativo.

Tipo	Definición	Ejemplo
Ejercitación	Se refiere a programas que intentan reforzar hechos y conocimientos que han sido analizados en una clase expositiva o de laboratorio.	Ven a Jugar con Pipo
Tutorial	Esencialmente presenta información, que se plasma en un diálogo entre el aprendiz y el computador. Utiliza un ciclo de presentación de información, respuesta a una o más preguntas o solución de un problema. Esto se hace para que la información presentada motive y estimule al alumno a comprometerse en alguna acción relacionada con la información.	Viaje hacia la vida
Simulación	Son principalmente modelos de algunos eventos y procesos de la vida real, que proveen al aprendiz de medio ambientes fluidos, creativos y manipulativos. Las simulaciones son utilizadas para examinar sistemas que no pueden ser estudiados a través de experimentación natural, debido a que involucra largos períodos, grandes poblaciones, aparatos de alto costo o materiales con un cierto peligro en su	Modellus

	manipulación.	
Juego Educativo	Es muy similar a las simulaciones, la diferencia radica en que incorpora un nuevo componente: la acción de un competidor, el cual puede ser real o virtual.	Estrategias del Mundo
Material de Referencia Multimedial	Usualmente presentado como enciclopedias interactivas. La finalidad de estas aplicaciones reside en proporcionar el material de referencia e incluyen tradicionalmente estructura hipermedial con clips de vídeo, sonido, imágenes, etc.	Enciclopedia Encarta 98
Edutainment	Es un tipo de software que integra elementos de educación y entretenimiento, en el cual cada uno de estos elementos juega un rol significativo y en igual proporción. Estos programas son interactivos por excelencia, utilizan colores brillantes, música y efectos de sonido para mantener a los aprendices interesados mientras se les introduce en algún concepto o idea.	¿Dónde en el mundo está Carmen San Diego?
Historias y cuentos	Son aplicaciones que presentan al usuario una historia multimedial, la cual se enriquece con un valor educativo.	La tortuga y la liebre
Editores	El objetivo de estos productos no es dar respuesta a preguntas del usuario, sino dar un marco de trabajo donde el alumno pueda crear y experimentar libremente en un dominio gráfico o similar.	Fine Artist
Hiperhistoria	Es un tipo de software donde a través de una metáfora de navegación espacial se transfiere una narrativa interactiva. Su característica principal reside en que combina activamente un modelo de objetos reactivos en un marco de ambiente virtual navegable. Tiene cierta semejanza con los juegos de aventuras.	HiperZoo, AudioDoom

Tabla 1.3: Tipos de Software Educativos, tomado de http://www.enlaces.cl/doc/evaluacion_recursos_educativos.doc.

TEORÍAS DEL APRENDIZAJE

CONDUCTISMO

El aprendizaje humano se produce porque un determinado estímulo provoca una respuesta conductual. [18]

18 Teorías del aprendizaje y métodos de enseñanza con ordenadores. Manuel Area Moreira.

Este modelo explica el aprendizaje a través de una dinámica estímulo-respuesta. El aprendizaje, dentro de esta teoría, está más centrado en el tipo y calidad de estímulo del profesor y en la respuesta observable del alumno.

- La tarea del profesor consiste en seleccionar el estímulo adecuado y presentarlo al alumnado.
- Se asocia a aprendizaje por repetición (Aprender la tabla de multiplicar).

CONSTRUCTIVISMO

El aprendizaje humano es una actividad que el sujeto realiza a través de su experiencia con el entorno.

El énfasis de este modelo está dado en cómo los aprendices construyen conocimientos en función a sus experiencias previas, estructuras mentales, creencias o ideas que usan para interpretar objetos y eventos. La teoría constructivista postula que el conocimiento, sea este de cualquier naturaleza, se construye a través de acciones que realiza el aprendiz sobre la realidad, esto implica que la construcción es interna (mental) y que el aprendiz es quién construye e interpreta su vida.

- La tarea del profesor consiste en crear situaciones de aprendizaje para el alumno construya el conocimiento a través de la actividad.
- Se asocia a aprendizaje por descubrimiento.

Las teorías del aprendizaje ofrecen una explicación sistemática y coherente sobre el método del aprendizaje, pero son los modelos de enseñanza - aprendizaje los que determinan cómo se debe aprender.

MODELOS DE ENSEÑANZA – APRENDIZAJE

APRENDIZAJE POR DESCUBRIMIENTO

Se aprende a través de la actividad sobre un problema y a partir de la misma el alumno “construye” el conocimiento.

El papel del profesor es plantear problemas, ofrecer orientaciones y proporcionar los medios para que el alumno “descubra” las respuestas. El profesor crea las condiciones para que el alumno construya el conocimiento por sí mismo.
Los materiales educativos son materiales constructivistas:

1. Plantea un problema (en formato texto, audio y/o visual).
2. Ofrece orientaciones y recursos para resolver el problema.
3. Solicita al alumno la realización de actividades de búsqueda, selección y elaboración de la información.
4. Comprueba si el alumno ha aprendido la información.

APRENDIZAJE SIGNIFICATIVO

Para Ausubel psicólogo y pedagogo estadounidense, aprender es sinónimo de comprender e implica una visión del aprendizaje basada en los procesos internos del alumno y no solo en sus respuestas externas. Con la intención de promover la asimilación de los saberes, el profesor utilizará organizadores previos que favorezcan la creación de relaciones adecuadas entre los saberes previos y los nuevos. Los organizadores tienen la finalidad de facilitar la enseñanza receptivo significativa, con lo cual, sería posible considerar que la exposición organizada de los contenidos, propicia una mejor comprensión.[19]

De acuerdo al aprendizaje significativo, los nuevos conocimientos se incorporan en forma sustantiva en la estructura cognitiva del alumno. Esto se logra cuando el estudiante relaciona los nuevos conocimientos con los anteriormente adquiridos; pero también es necesario que el alumno se interese por aprender lo que se le está mostrando.

VENTAJAS DEL APRENDIZAJE SIGNIFICATIVO: [20]

- Produce una retención más duradera de la información.
- Facilita el adquirir nuevos conocimientos relacionados con los anteriormente adquiridos de forma significativa, ya que al estar claros en la estructura cognitiva se facilita la retención del nuevo contenido.
- La nueva información al ser relacionada con la anterior, es guardada en la memoria a largo plazo.
- Es activo, pues depende de la asimilación de las actividades de aprendizaje por parte del alumno.
- Es personal, ya que la significación de aprendizaje depende los recursos cognitivos del estudiante.

19 http://ausubel.idoneos.com/index.php/368873

20 http://www.omerique.net/twiki/pub/Main/TrabajoSegundoPaloma/Ausubel.pdf

1.7 MÉTODO O ESTRUCTURA DE LA UNIDAD DE ANÁLISIS, CRITERIOS DE VALIDEZ Y CONFIABILIDAD

La unidad de muestreo es la universidad tecnológica de Pereira y la unidad de análisis de estudio son los expertos en informática, profesores y estudiantes.
El instrumento de medición son los cuestionarios de valoración cualitativa.

Para asegurar la confiabilidad del proyecto se utiliza la adaptación de MOSCA (Modelo Sistémico de Calidad del Software) anteriormente descrita para la evaluación del software educativo.

Para asegurar la validez se utiliza el procedimiento de validación por expertos donde se consulta a investigadores familiarizados con el tema.

1.8 DISEÑO METODOLOGICO

HIPÓTESIS

Es posible que, con un sistema computacional independiente del computador, un usuario tecnológico promedio pudiera asimilar fácilmente los conceptos básicos de la programación estructurada.

TIPO DE INVESTIGACIÓN

Enfoque Cualitativo.

VARIABLES

- Funcionalidad (FUN): Es la capacidad del producto del software para proveer funciones que cumplan con necesidades específicas o implícitas, cuando el software es utilizado bajo ciertas condiciones.

- Usabilidad (USA): Esta categoría se refiere a la capacidad del producto de software para ser atractivo, entendido, aprendido y utilizado por el usuario bajo condiciones específicas.

- Fiabilidad (FIA): La fiabilidad es la capacidad del producto de software para mantener un nivel especificado de rendimiento cuando es utilizado bajo condiciones especificadas.

Estas variables se medirán mediante cuestionarios de valoración en base a la descripción de adaptación de MOSCA para software educativo antes mencionado.

POBLACIÓN

Estudiantes de Programación I de la Universidad Tecnológica de Pereira del programa de Ingeniería de Sistemas Y Computación que se entenderán como usuarios tecnológicos promedio.

MUESTRA

5 estudiantes, un profesor y un experto en informática de la Universidad Tecnológica de Pereira.

DISEÑO DE INSTRUMENTOS PARA TOMA DE INFORMACIÓN

No aplica dado que corresponde a una fase posterior de la presente propuesta.

1.9 ESQUEMA TEMATICO

- Características del Sistema computacional
  - Funcionales
  - Educativas
- Especificación del lenguaje
  - Comportamiento Prototipo
- Estructura del interprete
  - Análisis Léxico
  - Análisis Sintáctico
  - Análisis Semántico
- Simulación
- Comunicación Computador - Prototipo Físico
- Prototipo Físico
  - Características funcionales
  - Características físicas
  - Arquitectura del Procesador
- IDE
  - Panel de diseño del entorno de simulación.
  - Editor de texto del código del lenguaje
- HCI (Human Computer Interaction)

1.10 NOMBRE DE LAS PERSONAS QUE PARTICIPAN EN EL PROYECTO

Universidad Tecnológica de Pereira. Programa Ingeniería de Sistemas y Computación.

Andrés Guillermo Velásquez Gómez
Ingeniero de Sistemas y Computación
Asesor en electrónica y FPGA's.

Juan David Arias Patiño
Estudiante Ingeniería de sistemas y computación
Electrónica.

Miguel Angel Lozano Isaza
Estudiante Ingeniería de sistemas y computación
Documentador.

Andrés Felipe Marín Rodríguez
Estudiante Ingeniería de sistemas y computación
Programador.

Sebastián Gómez González
Estudiante Ingeniería de sistemas y computación
Asesor en electrónica y programación.

Omar Iván Trejos
Ingeniero de Sistemas y Computación
Asesor en pedagogía.

1.11 RECURSOS DISPONIBLES

- Universidad Tecnológica de Pereira, Programa de Ingeniería de Sistemas y Computación, préstamo de FPGA'S, asesorías y bibliografía.
- Software libre.
- La parte económica corre por cuenta del grupo.

1.12 CRONOGRAMA

	Meses	1				2				3				4				5			
	Semanas	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Objetivos																					
A		X	X																		
B				X	X	X	X														
C					X	X	X	X	X	X	X	X									
D										X	X	X	X								
E													X	X	X						
F																X	X	X			
G			X		X		X		X		X		X		X		X				

Tabla 1.4: Cronograma del proyecto.

Los objetivos que aparecen en la tabla 1.4 son los objetivos específicos antes mencionados y están en orden cronológico.

1.13 POSIBILIDADES DE PUBLICACION

Revista indexada Scientia et technica.
Portal web Semillero InSilico.

2. FUNDAMENTACIÓN

2.1 APRENDIZAJE SIGNIFICATIVO

La teoría del aprendizaje significativo tiene como premisa el hecho que el estudiante tiene algún conocimiento y que puede utilizarlo como un medio para insertar información nueva en su estructura cognitiva, entendiéndose por "estructura cognitiva", al conjunto de conceptos, ideas que un individuo posee en un determinado campo del conocimiento, así como su organización.[1]

El aprendizaje significativo ocurre cuando una nueva información "se conecta" con un concepto relevante pre existente en la estructura cognitiva, esto implica que, las nuevas ideas, conceptos y proposiciones pueden ser aprendidos significativamente en la medida en que otras ideas, conceptos o proposiciones relevantes estén adecuadamente claras y disponibles en la estructura cognitiva del individuo y que funcionen como un punto de "anclaje" a las primeras. [2]

Como se observa en la imagen 2.1, las estructuras de conocimiento a través del cual los nuevos conceptos se producen en la teoría significativa se denominan inclusores. Estos actúan como una estructura de apoyo para unificar el nuevo concepto formando un nuevo inclusor. La siguiente figura muestra esta conceptualización. [1]

Imagen 2.1: Formación de conceptos basado en la existencia de inclusores.

[1] http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5492549&queryText%3Dmeaningful+learning%26openedRefinements%3D*%26searchField%3DSearch+All

[2] http://www.utemvirtual.cl/plataforma/aulavirtual/assets/asigid_745/contenidos_ara/39247_david_ausubel.pdf

2.1.1 APRENDIZAJE SIGNIFICATIVO Y APRENDIZAJE MECÁNICO

El aprendizaje puede ser dividido en aprendizaje mecánico y aprendizaje significativo de acuerdo a los contenidos de aprendizaje. También se puede dividir las actividades de aprendizaje en aprendizaje receptivo y aprendizaje por descubrimiento.

El aprendizaje mecánico es un tipo de aprendizaje de memoria, en donde la nueva información es almacenada arbitrariamente, sin interactuar con conocimientos pre-existentes. Por el contrario el aprendizaje significativo es favorecido debido a que los estudiantes deben tratar de construir una conexión entre los conocimientos antiguos y nuevos.[3]

2.1.2 REQUISITOS PARA EL APRENDIZAJE SIGNIFICATIVO

El alumno debe manifestar una disposición para relacionar sustancial y no arbitrariamente el nuevo material con su estructura cognoscitiva, por lo que el material que aprende es potencialmente significativo para él, es decir, relacionable con su estructura de conocimiento sobre una base no arbitraria.[4]

Lo anterior presupone que debe haber disposición para el aprendizaje significativo, es decir, que el alumno muestre una disposición para relacionar de manera sustantiva y no literal el nuevo conocimiento con su estructura cognitiva.

Pero independientemente de cuanto significado potencial posea el material a ser aprendido, si la intención del alumno es memorizar arbitraria y literalmente, tanto el proceso de aprendizaje como sus resultados serán mecánicos; de manera inversa, sin importar lo significativo de la disposición del alumno, ni el proceso, ni el resultado serán significativos, si el material no es potencialmente significativo, y si no es relacionable con su estructura cognitiva.[2]

2.1.3 TIPOS DE APRENDIZAJE SIGNIFICATIVO

Ausubel distingue tres tipos de aprendizaje significativo: representaciones, conceptos y proposiciones.

---

3 http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5236308&queryText%3Dmeaningful+learning%26openedRefinements%3D*%26searchField%3DSearch+All

4 AUSUBEL-NOVAK-HANESIAN (1983) Psicología Educativa: Un punto de vista cognoscitivo. 2° Ed. TRILLAS México

APRENDIZAJE DE REPRESENTACIONES

Ocurre cuando se igualan en significado símbolos arbitrarios con sus referentes (objetos, eventos, conceptos) y significan para el alumno cualquier significado al que sus referentes aludan.[4]

Es el aprendizaje más elemental del cual dependen los demás tipos de aprendizaje. Consiste en la atribución de significados a determinados símbolos.

Este tipo de aprendizaje se presenta generalmente en los niños, por ejemplo, el aprendizaje de la palabra "Pelota", ocurre cuando el significado de esa palabra pasa a representar, o se convierte en equivalente para la pelota que el niño está percibiendo en ese momento, por consiguiente, significan la misma cosa para él; no se trata de una simple asociación entre el símbolo y el objeto sino que el niño los relaciona de manera relativamente sustantiva y no arbitraria, como una equivalencia representacional con los contenidos relevantes existentes en su estructura cognitiva.[2]

APRENDIZAJE DE CONCEPTOS

Los conceptos se definen como "objetos, eventos, situaciones o propiedades que poseen atributos de criterios comunes y que se designan mediante algún símbolo o signos".[4]

Partiendo de esto se puede afirmar que en cierta forma también es un aprendizaje de representaciones. Los conceptos son adquiridos a través de dos procesos: formación y asimilación.

En la formación de conceptos, los atributos de criterio (características) del concepto se adquieren a través de la experiencia directa, en sucesivas etapas de formulación y prueba de hipótesis, del ejemplo anterior podemos decir que el niño adquiere el significado genérico de la palabra "pelota", ese símbolo sirve también como significante para el concepto cultural "pelota", en este caso se establece una equivalencia entre el símbolo y sus atributos de criterios comunes. De allí que los niños aprendan el concepto de "pelota" a través de varios encuentros con su pelota y las de otros niños.

El aprendizaje de conceptos por asimilación se produce a medida que el niño amplía su vocabulario, pues los atributos de criterio de los conceptos se pueden definir usando las combinaciones disponibles en la estructura cognitiva por ello el

niño podrá distinguir distintos colores, tamaños y afirmar que se trata de una "Pelota", cuando vea otras en cualquier momento.[2]

APRENDIZAJE DE PROPOSICIONES

Este tipo de aprendizaje va más allá de la simple asimilación de lo que representan las palabras, combinadas o aisladas, puesto que exige captar el significado de las ideas expresadas en forma de proposiciones.

El aprendizaje de proposiciones implica la combinación y relación de varias palabras cada una de las cuales constituye un referente unitario, luego estas se combinan de tal forma que la idea resultante es más que la simple suma de los significados de las palabras componentes individuales, produciendo un nuevo significado que es asimilado a la estructura cognoscitiva.[2]

2.2 APRENDIZAJE POR DESCUBRIMIENTO

El aprendizaje por descubrimiento ha sido promovido en literatura educativa desde principios de la década de 1960. Este toma lugar en situaciones de resolución de problemas en donde el alumno se basa en su propia experiencia y conocimientos previos. Este es un método de enseñanza a través del cual los estudiantes interactúan con su entorno explorando y manipulando objetos, luchando con las preguntas y controversias, o realizando experimentos.[5]

Según Jerome Seymour Bruner el aprendizaje por descubrimiento es un método en el cual el alumno atentamente participa, e iniciativa y activamente explora con el fin de acceder al conocimiento y al enfoque de resolución de problemas. En las circunstancias del aula, el aprendizaje por descubrimiento es también un método de enseñanza importante, debido a que los estudiantes exploran los conceptos requeridos de estudio y encuentran estrategias y soluciones al problema bajo el estímulo de los profesores.

Los profesores seleccionan el problema y los estudiantes encuentran la solución por ellos mismos. Mediante el establecimiento de preguntas de fondo, los estudiantes encuentran conflictos en las circunstancias y luego plantean solución a sus preguntas. Los estudiantes proponen la hipótesis por actividades de

[5] http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=4755792&queryText%3Ddiscovery+learning%26openedRefinements%3D*%26searchField%3DSearch+All

aprendizaje, luego prueban la hipótesis y extraen leyes generales y conceptos debatiendo.[6]

Jerome Bruner argumenta que la práctica de descubrir por sí mismo le enseña al sujeto a adquirir una información que resulta más asequible y útil para la solución de problemas. Bruner también sugiere que los estudiantes son más propensos a recordar conceptos, si los descubren por sí mismos.[5]

2.2.1 FORMAS DE DESCUBRIMIENTO

Según Bruner, podemos hablar de tres tipos de descubrimiento: [7]

El descubrimiento inductivo implica la colección y reordenación de datos para llegar a una nueva categoría, concepto o generalización.
El descubrimiento deductivo implicaría la combinación o puesta en relación de ideas generales, con el fin de llegar a enunciados específicos, como en la construcción de un silogismo.
El descubrimiento o pensamiento transductivo el individuo relaciona o compara dos elementos particulares y advierte que son similares en uno o dos aspectos.

2.2.2 CONDICIONES DE APRENDIZAJE POR DESCUBRIMIENTO

Las condiciones que se deben presentar para que se produzcan un aprendizaje por descubrimiento son: [7]

- El ámbito de búsqueda debe ser restringido, ya que así el individuo se dirige directamente al objetivo que se planeó en un principio.
- Los objetivos y los medios estarán bastante especificados y serán atrayentes, ya que así el individuo se incentivara a realizar este tipo de aprendizaje.
- Se debe contar con los conocimientos previos de los individuos para poder así guiarlos adecuadamente, ya que si se presenta un objetivo del cual este no tiene la base, no va a poder llegar a su fin.
- Los individuos deben estar familiarizados con los procedimientos de observación, búsqueda, control y medición de variables, o sea, tiene el

6 http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5593719&queryText%3Ddiscovery+learning%26openedRefinements%3D*%26searchField%3DSearch+All

7 http://www.csi-csif.es/andalucia/modules/mod_ense/revista/pdf/Numero_18/ OLGA_ZARZA_CORTES01.pdf

individuo que tener conocimiento de las herramientas que se utilizan en el proceso de descubrimiento para así poder realizarlo.

- Por último, los individuos deben percibir que la tarea tiene sentido y merece la pena, esto lo incentivara a realizar el descubrimiento, que llevara a que se produzca el aprendizaje.

2.2.3 PRINCIPIOS DE APRENDIZAJE POR DESCUBRIMIENTO

Los principales principios que rigen este tipo de aprendizaje son los siguientes: [7]

- Todo el conocimiento real es aprendido por el mismo, es decir, que el individuo adquiere conocimiento cuando lo descubre por el mismo o por su propio discernimiento.
- El significado es producto exclusivo del descubrimiento creativo y no verbal, es decir, que el significado que es la relación e incorporación de forma inmediata de la información a su estructura cognitiva tiene que ser a través del descubrimiento directo y no verbal, ya que los verbalismos son vacíos.
- La capacidad para resolver problemas es la meta principal de la educación, es decir, la capacidad de resolver problemas es la finalidad educativa legítima, para esto es muy razonable utilizar métodos científicos de investigación. En un sentido contradictorio, se encuentra que la capacidad de resolver problemas no es una función primaria en la educación.
- Cada niño debería ser un pensador creativo y crítico, es decir, se puede mejorar y obtener niños pensadores, creativos y críticos mejorando el sistema de educación y así obtendríamos alumnos capaces de dominar el ámbito intelectual, así como un incremento del entendimiento de las materias de sus estudios.
- El descubrimiento organiza de manera eficaz lo aprendido para emplearlo posteriormente, es decir, ejecuta una acción basada en los conocimientos cuando está estructurada, simplificada y programada para luego incluir varios ejemplares del mismo principio en un orden de dificultad.
- El descubrimiento es el generador único de motivación y confianza en sí mismo, es decir, que la exposición diestra de ideas puede ser también la estimación intelectual.
- El descubrimiento es una fuente primaria de motivación intrínseca, es decir, el individuo sin estimulación intrínseca adquiere la necesidad de ganar símbolos (elevadas calificaciones y la aprobación del profesor) como también la gloria y el prestigio asociados con el descubrimiento independiente de nuestra cultura.

- El descubrimiento asegura la conservación del recuerdo, es decir, que a través de este tipo de aprendizaje es más probable que el individuo conserve la información.

2.3 APRENDIZAJE BASADO EN PROBLEMAS

El ABP es un enfoque de aprendizaje activo en el cual la resolución de problemas proporciona un contexto para que los estudiantes apliquen conocimientos previos y adquieran nuevos. Los miembros de la Facultad de Medicina de la Universidad de Mc Master en Ontario se acreditan el desarrollo del ABP en la década de 1960. El aprendizaje basado en problemas es ampliamente reconocido en el cultivo del pensamiento creativo de los estudiantes, debido a que mejora el análisis, la habilidad de resolver problemas y el aprendizaje permanente.[8]

ABP es un contextual, colaborativo y constructivista ambiente de aprendizaje. Uno de los elementos clave de la ABP es que está centrado en los estudiantes. Los alumnos emiten su propio juicio sobre lo que deben aprender. El ABP generalmente funciona bajo orientación de un maestro. El profesor diseña simulaciones de problemas y guía a los estudiantes en el desarrollo de su aprendizaje. Los problemas son resueltos a través de procesos colaborativos en grupos pequeños. Los profesores juegan el rol de facilitadores, estos aportan información y guían el grupo a través del proceso de aprendizaje.[9]

2.3.1 MODELO DE ENFOQUE DE ABP

Tal como se muestra en la imagen 2.2 en el proceso de desarrollo del enfoque de ABP, hay implícita una dinámica de trabajo que permite facilitar el proceso de aprendizaje y desarrollar habilidades, actitudes y valores importantes para mediar en la formación del estudiante.[10]

---

8 http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5529601&queryText%3Dproblem+based+learning%26openedRefinements%3D*%26pageNupage%3D2%26searchField%3DSearch+All

9 http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5460038&queryText%3Dproblem+based+learning%26openedRefinements%3D*%26searchFsear%3DSearch+All

10 https://tspace.library.utoronto.ca/bitstream/1807/8986/1/rc01037.pdf

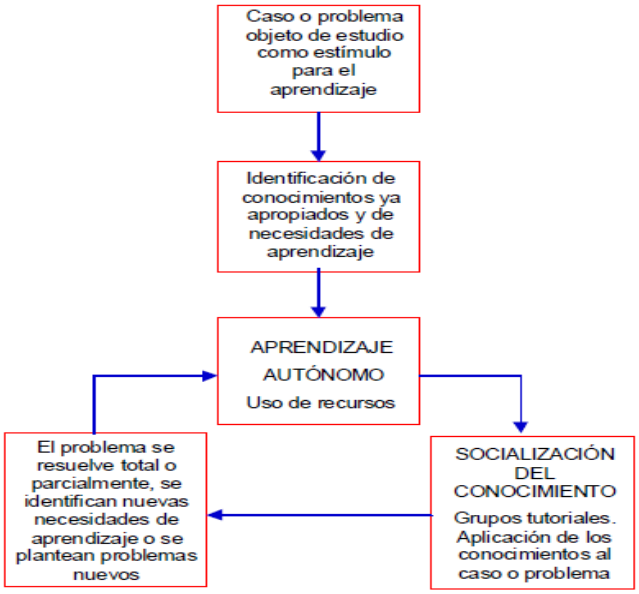


Imagen 2.2: Modelo de enfoque de ABP

2.3.2 CARACTERÍSTICAS DEL ABP

Las características fundamentales del aprendizaje basado en problemas son:[11]

- El aprendizaje centrado en el alumno: Los estudiantes deben tomar la responsabilidad de su propio aprendizaje bajo la guía de un tutor que se convierte en consultor del alumno, identificando los elementos necesarios para tener un mejor entendimiento y manejo del problema en el cual se trabaja, y detectando dónde localizar la información necesaria (libros, revistas, profesores, Internet, etc. De esta manera se logra la personalización del aprendizaje del alumno, ya que le permite concentrarse en las áreas de conocimiento, centrando su interés en áreas específicas que le sean significativas.

- Generación del aprendizaje en grupos pequeños: Los grupos de trabajo se conforman por 5 a 8 estudiantes. Al finalizar cada unidad programática los estudiantes cambian, en forma aleatoria, de grupo y trabajan con un nuevo tutor. Permitiéndoles adquirir práctica en el trabajo intenso y efectivo, con una variedad de diferentes personas.

11 http://www.rieoei.org/deloslectores/1460Santillan.pdf

- El docente adquiere el papel de facilitador: Al profesor se le denomina facilitador o tutor. El rol del tutor es plantear preguntas a los estudiantes que les ayude a cuestionarse y encontrar por ellos mismos la mejor ruta de entendimiento y manejo del problema. Conforme el ciclo escolar avanza los estudiantes asumen este rol ellos mismos, exigiéndose unos a otros.

- El núcleo de generación organizacional y aprendizaje radica en la generación de problemas: El ABP debe utilizar situaciones del mundo real. Los problemas deben estar bien estructurados, deben ser multifacéticos y deben ser presentados como ocurrirían en la realidad. En el contexto de ABP, un problema multifacético es complejo, no tiene soluciones sencillas, y refleja las situaciones que los estudiantes pueden encontrarse. Los alumnos utilizan el problema para establecer lo que se conoce, y detallar que necesitan conocer. Como los estudiantes trabajan con problemas de la vida real, su conocimiento acerca de estos temas debería aumentar. Ellos determinan sus problemas de aprendizaje, los cuales pueden ser comparados con los resultados del aprendizaje.

- Los problemas generan habilidades: Como se dijo anteriormente, los estudiantes deben de trabajar en equipo como base para el ABP. Las habilidades necesarias para el éxito del trabajo de equipo incluyen: toma de decisiones consensuadas, dialogo, conservación de la discusión del equipo, manejo de conflicto, y habilidades de liderazgo de equipo. Los graduados que tengan estas habilidades tendrán mejores oportunidades en la vida. Los problemas ABP deben ser elaborados y presentados para asegurar que los estudiantes encuentren situaciones en las que estas habilidades sean desarrolladas.

- El aprendizaje autodirigido genera nuevo conocimiento: Finalmente, se espera que los estudiantes aprendan a partir del conocimiento del mundo real y de la acumulación de experiencia por virtud de su propio estudio e investigación. Durante este aprendizaje autodirigido, los estudiantes trabajan juntos, discuten, comparan, revisan y debaten permanentemente lo que han aprendido.

2.4 HUMAN-COMPUTER INTERACTION

Human-Computer Interaction (HCI) es la interacción entre los hombres y maquinas. En esencia, esto significa gente interactuando con computadores o maquinas que contienen computadoras. El objetivo principal de la investigación de

HCI es explorar como diseñar equipo para ayudar a personas a completar las tareas necesarias más segura y eficientemente. [12]

Desde que el primer computador digital nació en 1946, la tecnología informática se ha desarrollado increíblemente. Pero el computador es todavía una herramienta, es la expansión de los cerebros, manos y ojos humanos, entonces este sigue siendo controlado por personas. En este caso, las personas y el computador necesitan comunicarse entre sí, esto es llamado Interacción Humano-Computador, y el sistema que incluye hardware y software para alcanzar la comunicación entre hombre y computador es llamado sistema interactivo.

2.4.1 FORMA INTERACTIVA

HCI es un proceso de entrada y salida, en el cual se ponen comandos en el computador a través de la interfaz humano-computador y el computador muestra la salida al usuario. Los medios de entrada y salida son diversos, por lo que se define la forma de interacción:

- Interacción de datos: Es poner e intercambiar datos con el computador. Esta es una importante forma de interacción humano-computador, el proceso general de interacción es así: Primero el sistema le dice al usuario que introduzca los datos y cómo hacerlo; entonces, en respuesta a la entrada del usuario, el sistema hace retroalimentación, y la muestra en la pantalla o en otro medio. Al mismo tiempo, el sistema valida la entrada del usuario, si hay algún error, el sistema solicita volver a introducir la información. Diferentes formas de entradas de datos determinan diferentes formas de datos interactivos, y los datos pueden ser todo tipo de símbolos de información; como figuras, gráficos, colores y signos.

- Interacción de Imagen: Los humanos generalmente transmiten mensajes a través de tres formas: lenguaje, palabras e imágenes, y más del 70% de la información que ellos consiguen de un sistema visual es obtenido a través de las imágenes. La interacción de imágenes, para ponerlo de una manera simple, es la comprensión de la imagen por la computadora basada en el comportamiento humano, para luego reflejar esto. En el presente, el sistema de visión artificial puede ser dividido en tres niveles: Procesamiento de imagen (nivel más bajo), Reconocimiento de imagen (nivel superior), y Percepción de imagen (nivel más alto). El procesamiento de imagen es

[12] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4959073

principalmente llevar a cabo una variedad de procesamiento de imágenes para mejorar los efectos visuales, y convertirlas en las imágenes deseadas. El reconocimiento de imagen es probar y medir las partes de interés de la imagen y establecer una descripción objetiva de esta. Finalmente la percepción de imagen se centra en el reconocimiento de la imagen e investigar los personajes y la mutua relación entre ellos.

- Interacción inteligente: Las interacciones inteligentes hombres-computador representan el lema "fácil de aprender y usar", y serán la tecnología de las siguientes generaciones. Con estas los computadores podrán predecir lo que los usuarios desean hacer a través del reconocimiento inteligente del comportamiento del usuario, en consecuencia se cumplirán sus necesidades y eventualmente los computadores se adaptaran a los humanos.

2.5 ENLACE ENTRE FUNDAMENTACIÓN Y PROPUESTA

En el currículo de la formación de docentes, además de las asignaturas básicas, se agregan algunas relacionadas con la pedagogía y la educación, como las didácticas y se realizan unas prácticas pedagógicas supervisadas por docentes con los mismos criterios tradicionales. Pero no existen campos de estudio ni de práctica en investigación e innovaciones educativas; no se elaboran recursos didácticos para favorecer la educación activa, ni hay procesos de desarrollo del pensamiento y la creatividad. "En realidad, la preparación de los maestros se hace a espaldas del sistema educativo en el cual van a trabajar." Según el MEN (2004), falta una mayor integración entre los componentes científicos, humanísticos y pedagógicos, para evitar que las universidades manejen discursos académicos formalistas y tradicionalistas en este tema.[13]

Como consecuencia de lo anterior se evidencia poca formación de los docentes en pedagogía en la educación superior, enseñan cómo les enseñaron, a través de clases expositivas donde se da primero la teoría, unos ejemplos y finalmente se evalúa. Esta modalidad de enseñanza normalmente está focalizada en los contenidos, priorizando los conceptos abstractos sobre los ejemplos concretos y las aplicaciones. Las técnicas de evaluación se limitan a comprobar la memorización de información y de hechos, ocupándose muy rara vez de desafiar al estudiante a alcanzar niveles cognitivos más altos de comprensión.

13 Situación de la formación docente inicial y en servicio en Colombia, Ecuador y Venezuela - Eduardo Fabara Garzón - 2004

Según Jerome Bruner desarrollador de la teoría constructivistas del aprendizaje y uno de los impulsores de la psicología cognitiva, en su libro Actos de significado "para adquirir el lenguaje, el niño requiere mucha más ayuda e interacción con los adultos que le cuidan de lo que había supuesto Chomsky, el lenguaje se adquiere utilizándolo y no adoptando el papel de mero espectador", y en este trabajo se piensa que este postulado puede ser transferido a la enseñanza de la programación debido a que los profesores como alumnos refuerzan la idea de que en el proceso de enseñanza-aprendizaje el profesor es el responsable de transferir contenidos y los estudiantes son receptores pasivos del conocimiento.

En este trabajo de grado también se considera que en la enseñanza de la programación se debe utilizar sistemas en lo que se involucre al estudiante de una forma más activa, para que el mismo construya su conocimiento de forma consciente y responsable a través de la práctica y retroalimentación. Y al mismo tiempo propiciando la motivación del estudiante.[14]

Por lo anterior se plantea la posibilidad de diseñar una herramienta en la cual se vincule el estudiante al aprendizaje de la programación, mediante procesos atractivos y que no le intimiden, que le permitan aprender a programar sin aún saber que lo está haciendo interactuando con otros tipos de hardware además del computador en el aprendizaje.

[14] http://www4.ncsu.edu/unity/lockers/users/f/felder/public/Student-Centered.html#Publications-Active

3. ANALISIS

3.1 OBSERVACIÓN DIRECTA

Se requiere un sistema en el cual un usuario pueda construir un programa siguiendo la sintaxis y reglas de un lenguaje de programación simple, el lenguaje de programación debe permitir manejar conceptos de la programación estructurada como: funciones, ciclos, bifurcaciones y variables. Se espera englobar todos esos conceptos en un sistema que combine hardware y software, por esto se propone que el lenguaje permita hacer gráficos mediante los movimientos de un cursor que represente la posición actual del robot, los movimientos del cursor serán controlados por instrucciones del lenguaje con los cuales podrá avanzar determinada distancia y girar determinados grados en una dirección, estos movimientos serán representados en la pantalla por el cursor usando un sistema de coordenadas cartesianas, el usuario podrá usar conceptos de geometría básica para construir diferentes figuras geométricas, así como usar la recursividad para hacer figuras complejas como fractales. El sistema permitirá la opción al usuario de ejecutar su programa presentando su simulación en pantalla o ejecutar el programa en un robot físico permitiéndole al usuario visualizar su ejecución en el mundo real. El sistema debe hacer un análisis del programa e informar si existen errores. Todo lo anterior será presentado en un entorno grafico intuitivo que englobara lo especificado.

El sistema a crear debe permitir:

- Escribir un programa al usuario mediante el lenguaje de programación suministrado.
- Guardar el programa escrito por el usuario en el disco duro del computador.
- Abrir un programa antes guardado por el usuario.
- Chequear si el programa del usuario tiene errores léxicos, sintácticos o de tiempo de ejecución y presentar el error en pantalla.
- Permitir al usuario hacer configuraciones sobre el sistema (simulación o programar robot, fuente de la letra, tamaño, color del lápiz).
- Permitir ejecutar el programa del usuario mostrando sus resultados en pantalla.
- Permitir ejecutar el programa del usuario en un prototipo de robot físico.

- Presentar en la ayuda del sistema un tutorial sobre el lenguaje de programación y el manejo del sistema.

3.2 REQUERIMIENTOS NO FUNCIONALES

- Usabilidad: El Sistema debe proveer una interfaz simple que facilite la utilización de la herramienta.

- Confiabilidad: Capacidad que posee el Sistema computacional para realizar sus funciones previstas sin incidentes por un período de tiempo especificado y bajo condiciones indicadas.

3.3 ESPECIFICACIÓN DE CASO DE USO

Se define la funcionalidad del sistema a través de los casos de uso que se observan en la imagen 3.1, los cuales muestran la interacción de los actores con el sistema.

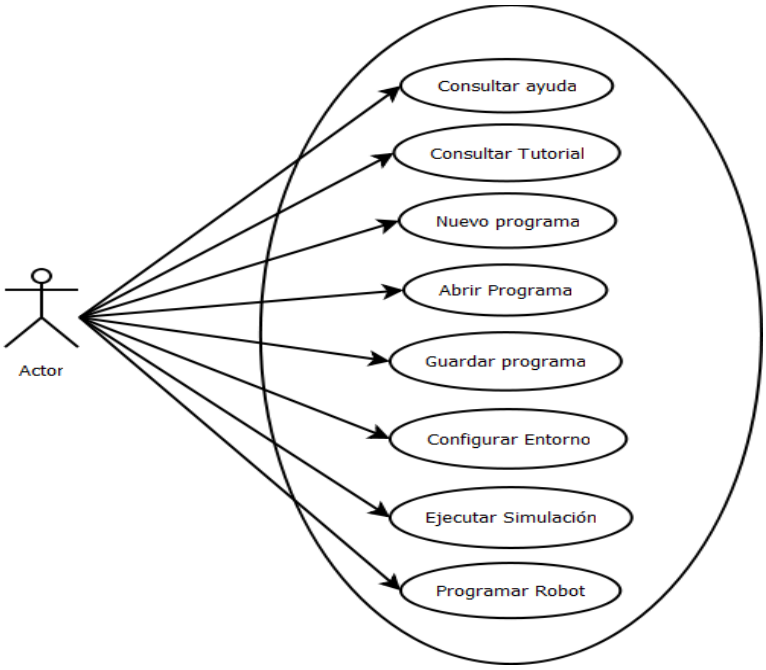


Imagen 3.1 Casos de Uso

3.3.1 ABRIR PROGRAMA

SECCIÓN PRINCIPAL	
Casos de Uso:	Abrir Programa
Actores:	Usuario (Iniciador)
Propósito:	Abrir un programa existente
Resumen:	El usuario luego de haber abierto el programa, se dirige al menú “Archivo” y presiona la opción “Abrir”, se abre una ventana que permite navegar a través de los directorios para escoger el archivo deseado y este último se carga en el área de texto.
Tipo:	Real
Referencias:	El usuario de haber abierto el programa.

CURSO NORMAL DE LOS EVENTOS	
Acción de los Actores	Respuesta del sistema
1. El usuario da click sobre el menú archivo, y luego sobre la opción abrir.	2. El sistema despliega una ventana para buscar el archivo que quiere abrir.
3. El usuario selecciona el archivo y da click en el botón abrir.	4. El sistema toma el contenido del archivo y lo escribe sobre el área de texto de la aplicación.

CURSO ALTERNATIVO
Acción 3: Si el usuario no encuentra el archivo, cierra la ventana dando click en el botón cancelar.

Tabla 3.1 Caso de uso Abrir Programa

3.3.2 CONFIGURAR ENTORNO

SECCIÓN PRINCIPAL	
Casos de Uso:	Configurar Entorno
Actores:	Usuario (Iniciador)
Propósito:	Configurar el entorno de trabajo
Resumen:	El usuario luego de haber abierto el programa, se dirige al menú “Configuración” y da click sobre el ítem “Configurar Entorno”, el sistema despliega una ventana con las opciones de configuración, el usuario hace los cambios correspondientes y da click en aceptar.
Tipo:	Real
Referencias:	El usuario de haber abierto el programa.

CURSO NORMAL DE LOS EVENTOS	
Acción de los Actores	Respuesta del sistema

1. El usuario da click sobre el menú configurar, y luego sobre la opción configurar entorno.	2. El sistema despliega una ventana con las opciones de configuración.
3. El usuario da click sobre el botón fuente.	4. El sistema despliega una ventana con las opciones de configuración de la fuente
5. El usuario selecciona el tipo de fuente, el estilo, el tamaño, los efectos, y el color de la fuente.	6. El sistema cierra esta ventana y pasa a la ventana de configuración de entorno.
7. El usuario da click sobre el botón color de fondo.	8. El sistema despliega una ventana para seleccionar el color.
9. EL usuario selecciona el color y da click en aceptar.	10. El sistema cierra esta ventana y pasa a la ventana de configuración de entorno.
11. El usuario da click sobre la lista desplegable del tamaño del lápiz.	12. El sistema despliega una lista con los tamaños de grosor del lápiz.
13. El usuario da click sobre el botón color de lápiz.	14. El sistema despliega una ventana para seleccionar el color.
15. EL usuario selecciona el color y da click en aceptar.	16. El sistema cierra esta ventana y pasa a la ventana de configuración de entorno.
17. En la parte de modo de ejecución, el usuario selecciona el modo de ejecución, simulación o programar robot.	
18. El usuario da click sobre el botón aceptar, para guardar los cambios.	19. El sistema guarda la configuración del entorno, cierra la ventana y vuelve a la pantalla principal.

CURSO ALTERNATIVO
Acción 3: si el usuario no necesita configurar la fuente, pasa a la acción 7.
Acción 7: si el usuario no necesita configurar el color de fondo, pasa a la acción 11.
Acción 11: si el usuario no necesita configurar el tamaño del lápiz, pasa a la acción 13.
Acción 13: si el usuario no necesita configurar el color del lápiz, pasa a la acción 17.
Acción 17: si el usuario no necesita configurar el modo de ejecución, pasa a la acción 18.
Acción 18: si el usuario no necesita guardar los cambios de la simulación.

Tabla 3.2 Caso de uso Configurar Entorno

3.3.3 CONSULTAR AYUDA

SECCIÓN PRINCIPAL	
Casos de Uso:	Consultar ayuda
Actores:	Usuario (Iniciador)
Propósito:	Consultar la ayuda de la aplicación
Resumen:	El usuario luego de haber abierto el programa, se dirige al menú “Ayuda” y

	presiona la opción “Ayuda”, se abre una ventana que permite navegar a través del contenido de la ayuda de la aplicación.
Tipo:	Real
Referencias:	El usuario de haber abierto el programa.

CURSO NORMAL DE LOS EVENTOS	
Acción de los Actores	Respuesta del sistema
1. El usuario da click sobre el menú ayuda, y luego sobre la opción ayuda.	2. El sistema despliega una ventana para navegar a través del contenido de la ayuda.
3. El usuario accede a la información indexada por contenidos y despliega la información disponible por cada contenido.	
4. Cuando el usuario encuentra la información requerida en la ayuda da click en el botón cerrar.	5. El sistema cierra la ventana de la ayuda y vuelve a la ventana principal.

CURSO ALTERNATIVO
Acción 3: Si el usuario desea buscar una palabra en el contenido, puede hacerlo a través de la pestaña buscar.
Acción 4: Si el usuario no encuentra la información que busca, puede enviar un correo al proyecto.scape@hotmail.com.

Tabla 3.3 Caso de uso Consultar Ayuda

3.3.4 CONSULTAR TUTORIAL

SECCIÓN PRINCIPAL	
Casos de Uso:	Consultar tutorial
Actores:	Usuario (Iniciador)
Propósito:	Consultar el tutorial para aprender a usar el lenguaje
Resumen:	El usuario luego de haber abierto el programa, se dirige al menú “Ayuda” y presiona la opción “Tutorial”, se abre una ventana que permite navegar a través del contenido del tutorial de la aplicación.
Tipo:	Real
Referencias:	El usuario de haber abierto el programa.

CURSO NORMAL DE LOS EVENTOS	
Acción de los Actores	Respuesta del sistema
1. El usuario da click sobre el menú ayuda, y luego sobre la opción tutorial.	2. El sistema despliega una ventana para navegar a través del contenido del tutorial.
3. El usuario accede al tutorial creado para	

llevar a la persona de manera sencilla a aprender los conceptos básicos de la programación estructurada.	
4. Cuando el usuario termina, da click en el botón cerrar.	5. El sistema cierra la ventana del tutorial y vuelve a la ventana principal.

CURSO ALTERNATIVO
Acción 3: Si el usuario desea buscar una palabra en el contenido, puede hacerlo a través de la pestaña buscar.
Acción 4: Si el usuario desea más información sobre el lenguaje o el tutorial, puede enviar un correo al proyecto.scape@hotmail.com.

Tabla 3.4 Caso de uso Consultar Tutorial

3.3.5 EJECUTAR SIMULACIÓN

SECCIÓN PRINCIPAL	
Casos de Uso:	Ejecutar simulación
Actores:	Usuario (Iniciador)
Propósito:	Ejecutar un programa del usuario y presentar su simulación en el panel principal.
Resumen:	El usuario luego de haber abierto la aplicación y haber construido un programa en el lenguaje, escribe en el cuadro de texto de comandos la función que quiere ejecutar y hace click en el botón ejecutar, el intérprete del lenguaje realiza las verificaciones pertinentes al código del usuario y luego ejecuta cada una de las instrucciones del programa, presentando en el panel principal una simulación de la ejecución.
Tipo:	Real
Referencias:	El usuario debe haber abierto el programa. El modo de ejecución debe estar en la opción “Simulación” (Se realiza en el caso de uso Configurar entorno)

CURSO NORMAL DE LOS EVENTOS	
Acción de los Actores	Respuesta del sistema
1. El usuario escribe un programa en el área de texto.	
2. El usuario escribe el nombre de la función con sus parámetros que desea ejecutar y hace click en el botón ejecutar.	3. El sistema hace un análisis léxico al programa escrito en el área de texto.
	4. El sistema hace un análisis sintáctico al programa.

	5. El sistema ejecuta la función que el usuario quiere ejecutar, presentando la simulación de las instrucciones de la función en el panel principal.

CURSO ALTERNATIVO
Acción 3: El sistema encuentra un error léxico en el programa, despliega en el panel de salida la línea en la cual se encuentra el error léxico.
Acción 4: El sistema encuentra un error sintáctico en el programa, despliega en el panel de salida la línea en la cual se encuentra el error sintáctico.
Acción 5: El sistema encuentra un error cuando está ejecutando el programa, despliega en el panel de salida la línea en la cual se encuentra el error con su correspondiente descripción.

Tabla 3.5 Caso de uso Ejecutar Simulación

3.3.6 GUARDAR PROGRAMA

SECCIÓN PRINCIPAL	
Casos de Uso:	Guardar Programa
Actores:	Usuario (Iniciador)
Propósito:	Guardar un programa escrito en un archivo txt
Resumen:	El usuario luego de haber creado un nuevo programa, se dirige al menú "Archivo" y presiona la opción "Guardar", se abre una ventana que permite navegar a través de los directorios para escoger la ubicación donde se desea guardar y el nombre del archivo.
Tipo:	Real
Referencias:	Nuevo Programa

CURSO NORMAL DE LOS EVENTOS	
Acción de los Actores	Respuesta del sistema
1. El usuario da click sobre el menú archivo, y luego sobre la opción Guardar.	2. El sistema despliega una ventana para buscar el directorio donde guardar.
3. El usuario selecciona el directorio, escribe el nombre del archivo y da click en el botón Guardar.	4. El sistema crea un archivo de texto con el nombre dado en el directorio especificado, toma el contenido del área de texto y lo escribe sobre el archivo, por último cierra la ventana.

CURSO ALTERNATIVO
Acción 1: Si el archivo ha sido abierto o guardado con anterioridad, sobrescribe el contenido y cierra la ventana.

Acción 4: Si en el directorio especificado existe un archivo con el mismo nombre que el que se le dio, el sistema pregunta si desea reemplazar, de lo contrario vaya a la acción 3 con otro nombre u otro directorio.

Tabla 3.6 Caso de uso Guardar Programa

3.3.7 NUEVO PROGRAMA

SECCIÓN PRINCIPAL	
Casos de Uso:	Nuevo Programa
Actores:	Usuario (Iniciador)
Propósito:	Crear un nuevo programa
Resumen:	El usuario luego de haber abierto el programa, se dirige al menú “Archivo” y presiona la opción “Nuevo”, se abre una ventana en la cual se pregunta si desea guardar el trabajo anterior, si el usuario hace click en sí, se abre una ventana de guardar como correspondiente al caso de uso guardar como y luego de haber guardado el trabajo se limpian todos los objetos de la ventana principal y el usuario puede hacer un nuevo trabajo.
Tipo:	Real
Referencias:	El usuario debe haber abierto el programa. El caso de uso guardar como debe estar hecho.

CURSO NORMAL DE LOS EVENTOS	
Acción de los Actores	Respuesta del sistema
1. El usuario da click sobre el menú archivo, y luego sobre la opción nuevo.	2. El sistema despliega una ventana en la cual pregunta si desea guardar el trabajo anterior en un archivo de texto.
3. El usuario hace click sobre la opción sí.	
4. Ir al caso de uso guardar.	5. El sistema limpia todos los objetos de la ventana principal.

CURSO ALTERNATIVO
Acción 3: Si el usuario hace click sobre la opción no, no se guarda el trabajo anterior y continúa al paso 5.

Tabla 3.7 Caso de uso Nuevo programa

3.3.8 PROGRAMAR ROBOT

SECCIÓN PRINCIPAL	
Casos de Uso:	Programar robot
Actores:	Usuario (Iniciador)

Propósito:	Enviar el código intermedio al servidor del robot
Resumen:	El usuario luego de haber abierto la aplicación y haber construido un programa, escribe la función que desea ejecutar en el cuadro de texto de comandos y hace click en ejecutar, el sistema crea el código intermedio del programa del usuario y lo envía al servidor del robot para que este lo ejecute sobre el robot.
Tipo:	Real
Referencias:	El usuario debe haber abierto el programa. El modo de ejecución debe estar en la opción "Programar Robot" (Se realiza en el caso de uso Configurar entorno)

CURSO NORMAL DE LOS EVENTOS	
Acción de los Actores	Respuesta del sistema
1. El usuario escribe un programa en el área de texto.	
2. El usuario escribe el nombre de la función con sus parámetros que desea ejecutar y hace click en el botón ejecutar.	3. El sistema hace un análisis léxico al programa escrito en el área de texto.
	4. El sistema hace un análisis sintáctico al programa.
	5. El sistema crea el código intermedio del programa del usuario y lo envía al servidor del robot.
	6. El sistema presenta el estado del envío en la barra de estado de la ventana principal.

CURSO ALTERNATIVO
Acción 3: El sistema encuentra un error léxico en el programa, despliega en el panel de salida la línea en la cual se encuentra el error léxico.
Acción 4: El sistema encuentra un error sintáctico en el programa, despliega en el panel de salida la línea en la cual se encuentra el error sintáctico.
Acción 5: El usuario escribió una función que no existe, el sistema despliega un mensaje en la salida de consola advirtiendo del error y cancela el envio del código intermedio del programa al servidor del robot. Si no hay conexión con el robot el sistema pasa a la acción 6 con estado de error.

Tabla 3.8 Caso de uso Programar Robot

3.4 MODELO DINÁMICO

Se muestra el comportamiento del sistema en el tiempo a través de los diagramas de secuencia.

3.4.1 ABRIR PROGRAMA

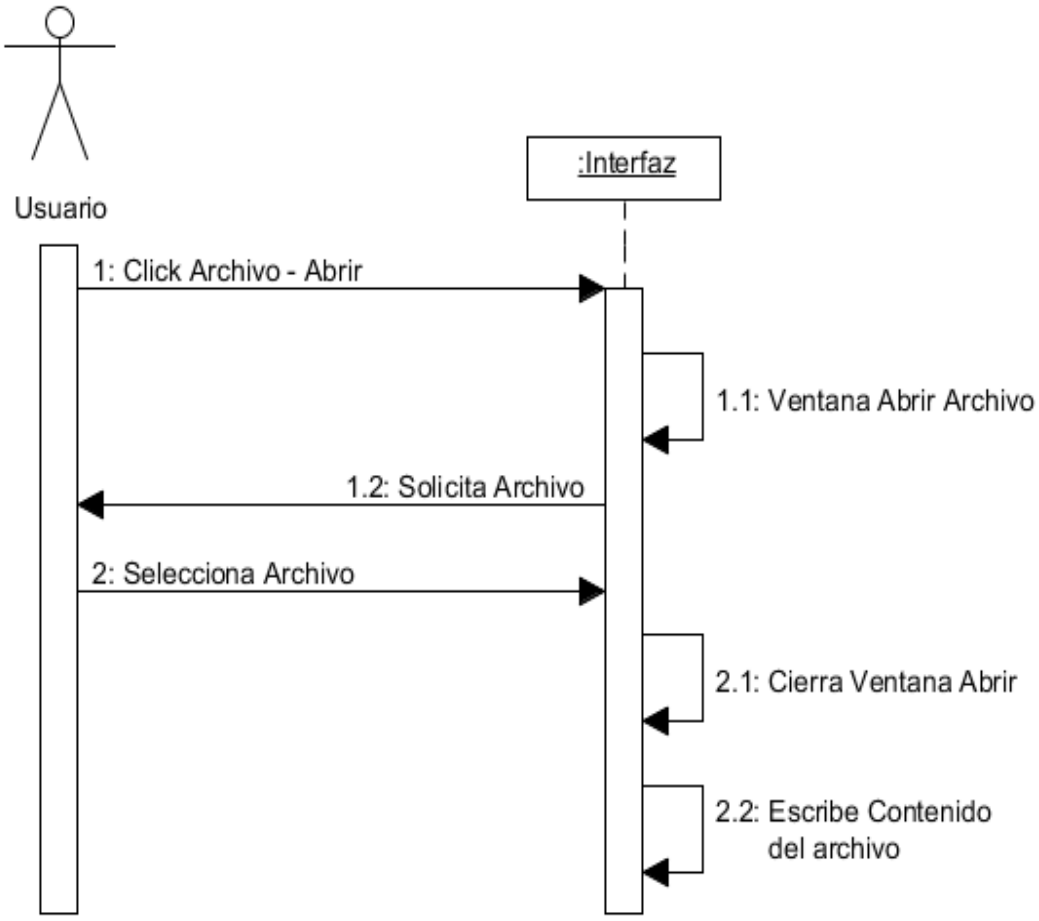


Imagen 3.2 Diagrama de Secuencia Abrir Programa

3.4.2 CONFIGURAR ENTORNO

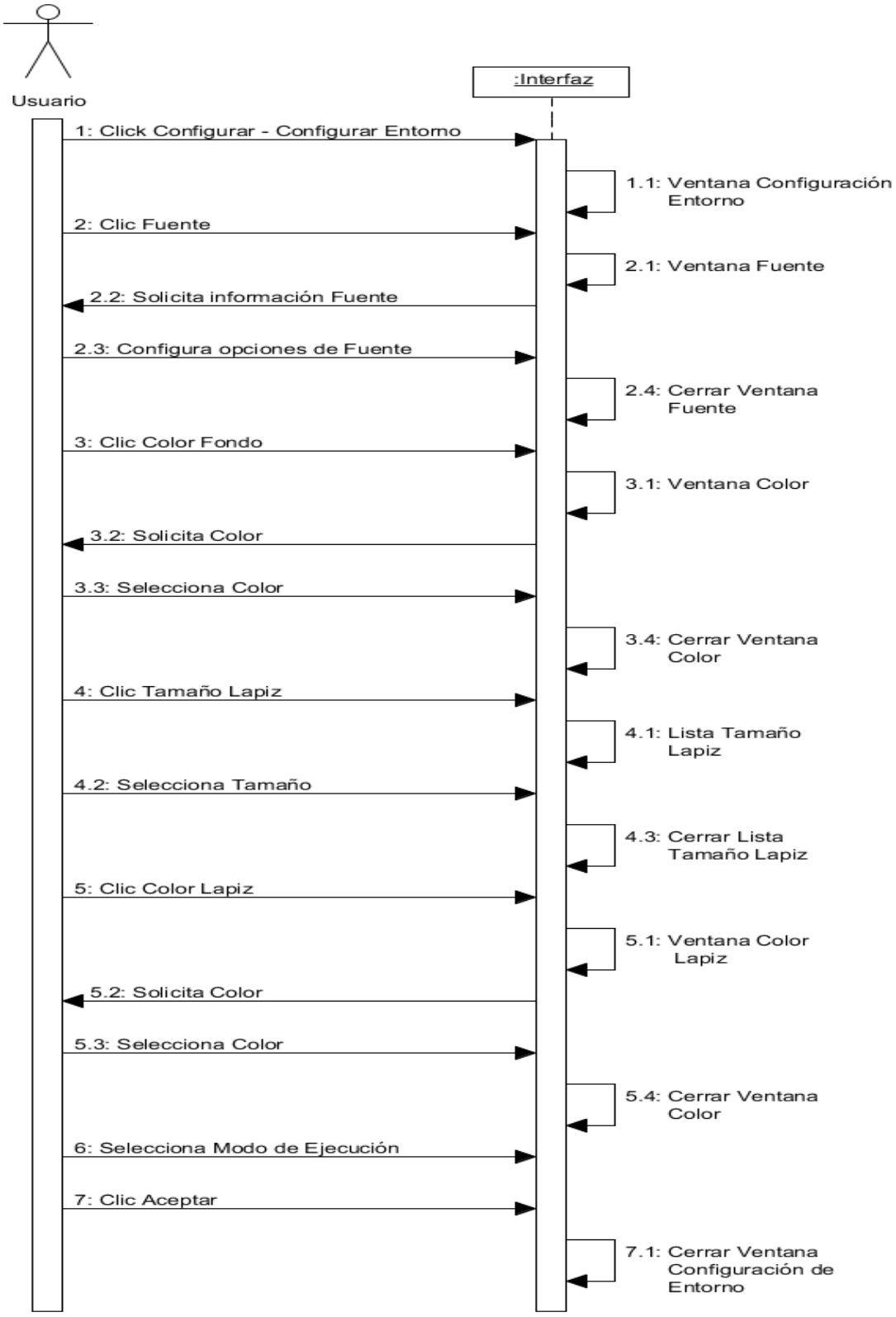


Imagen 3.3 Diagrama de Secuencia Configurar Entorno

3.4.3 CONSULTAR AYUDA

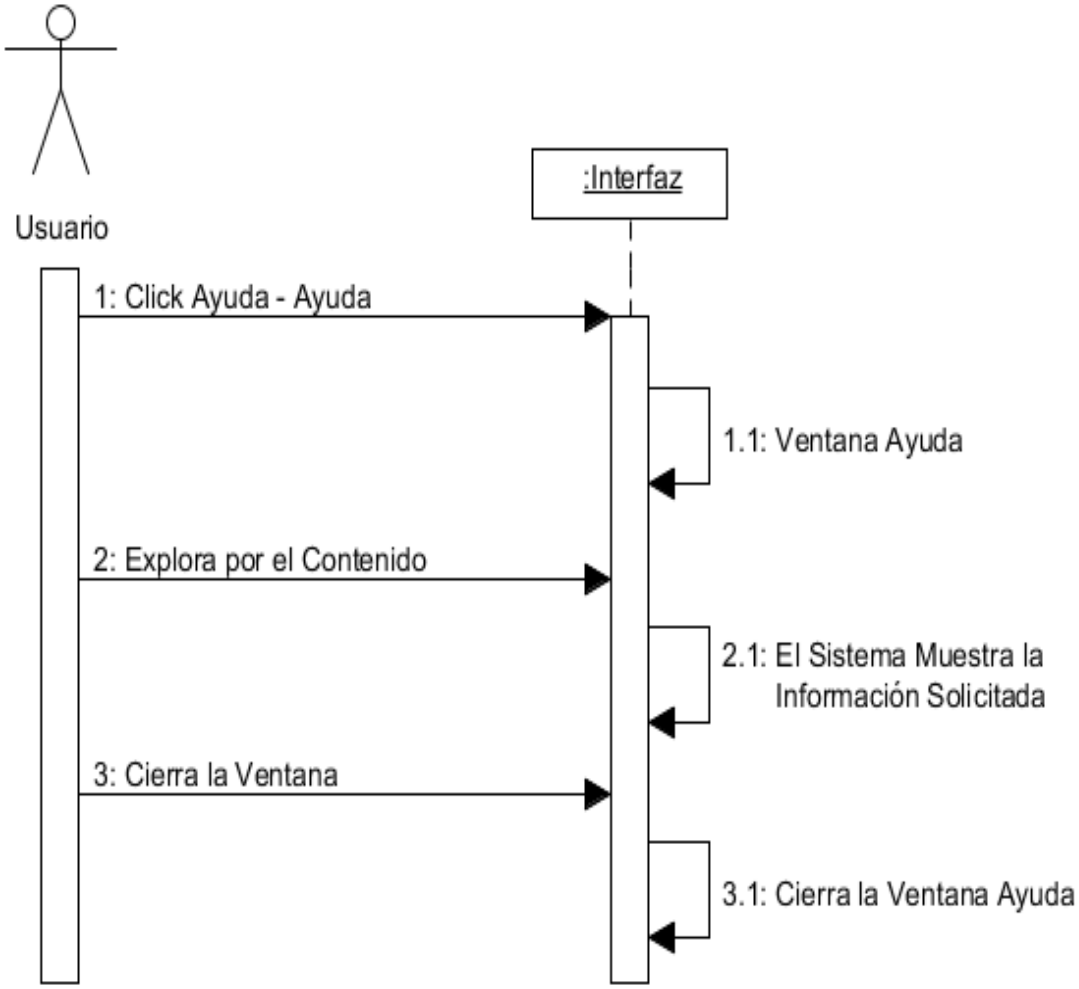


Imagen 3.4 Diagrama de Secuencia Consultar Ayuda

3.4.4 CONSULTAR TUTORIAL

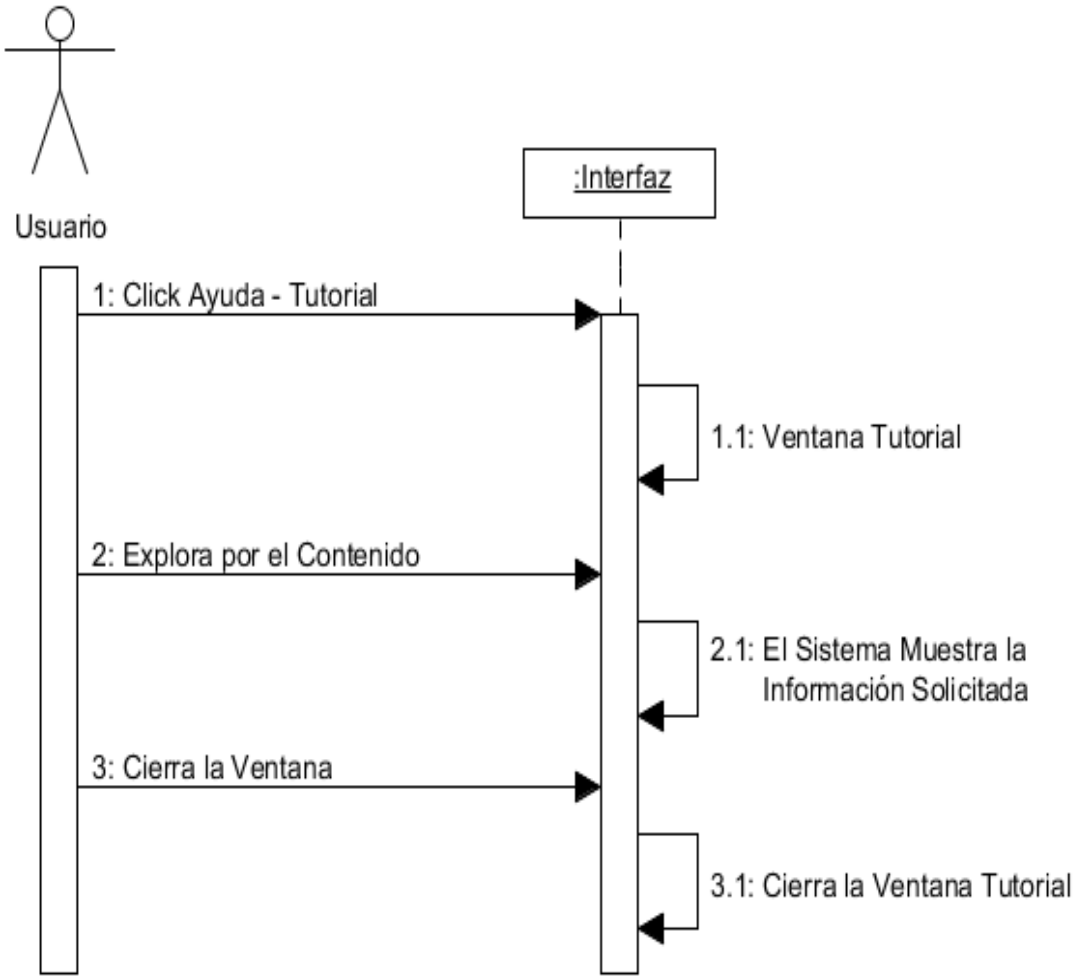


Imagen 3.5 Diagrama de Secuencia Consultar Tutorial

3.4.5 EJECUTAR SIMULACIÓN

Para empezar el usuario escribe un programa en el editor de texto del sistema, luego en la consola de comandos hace el llamado a la función que desee y da clic en el botón ejecutar, el objeto Interfaz capta esta solicitud y llama al objeto interprete con la función ejecutar, el intérprete hace un llamado al objeto Lexer que realiza un análisis léxico al programa y el cual produce una secuencia de símbolos (números, identificadores, palabras reservadas, entre otros) que sirven para la posterior etapa. Luego el intérprete hace un llamado al objeto Parser que realiza un análisis sintáctico donde se comprueba las reglas de la gramática y devuelve una estructura de árbol que es útil para la siguiente etapa. Finalmente el intérprete

con esa estructura de árbol ejecuta el programa del usuario y hace un llamado a la interfaz para que presente la simulación.

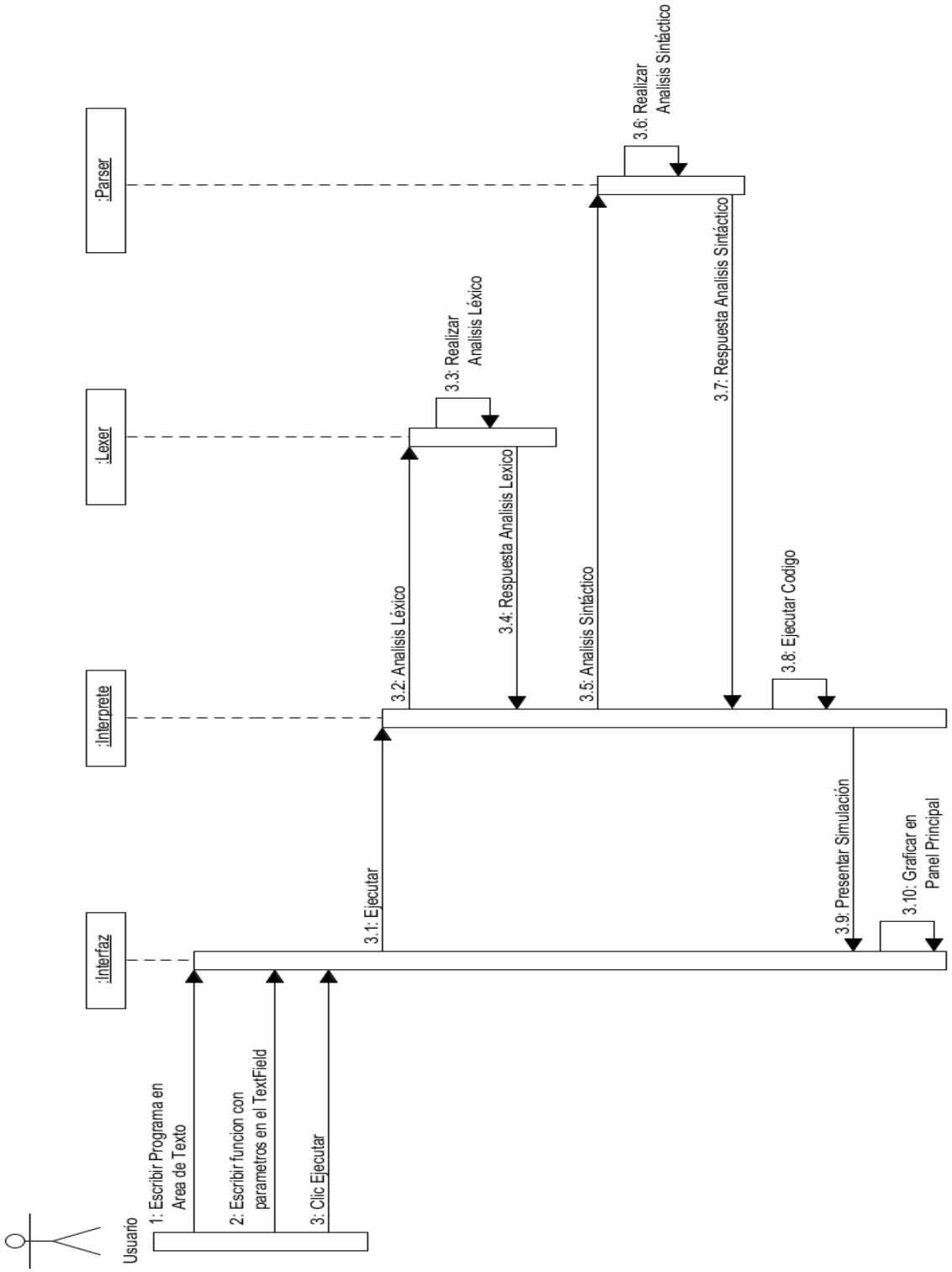


Imagen 3.6 Diagrama de Secuencia Ejecutar Simulación

3.4.6 GUARDAR ARCHIVO

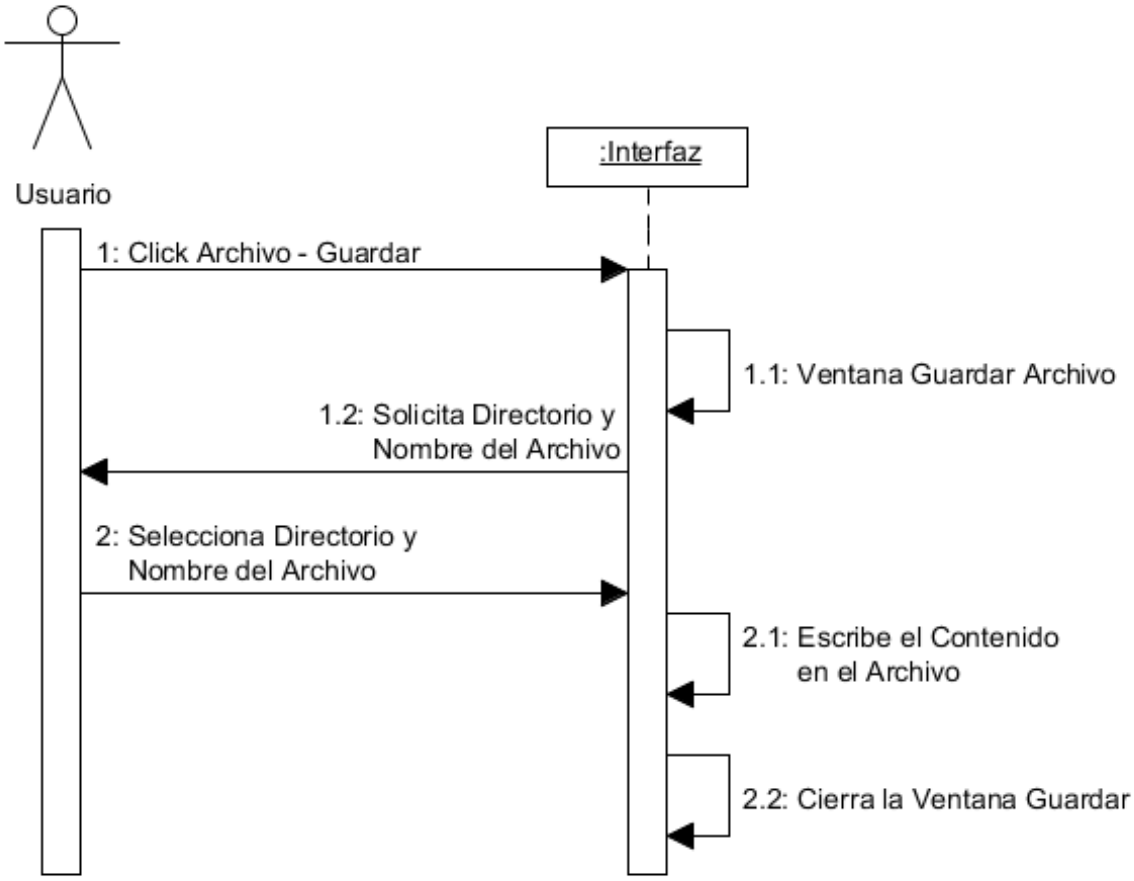
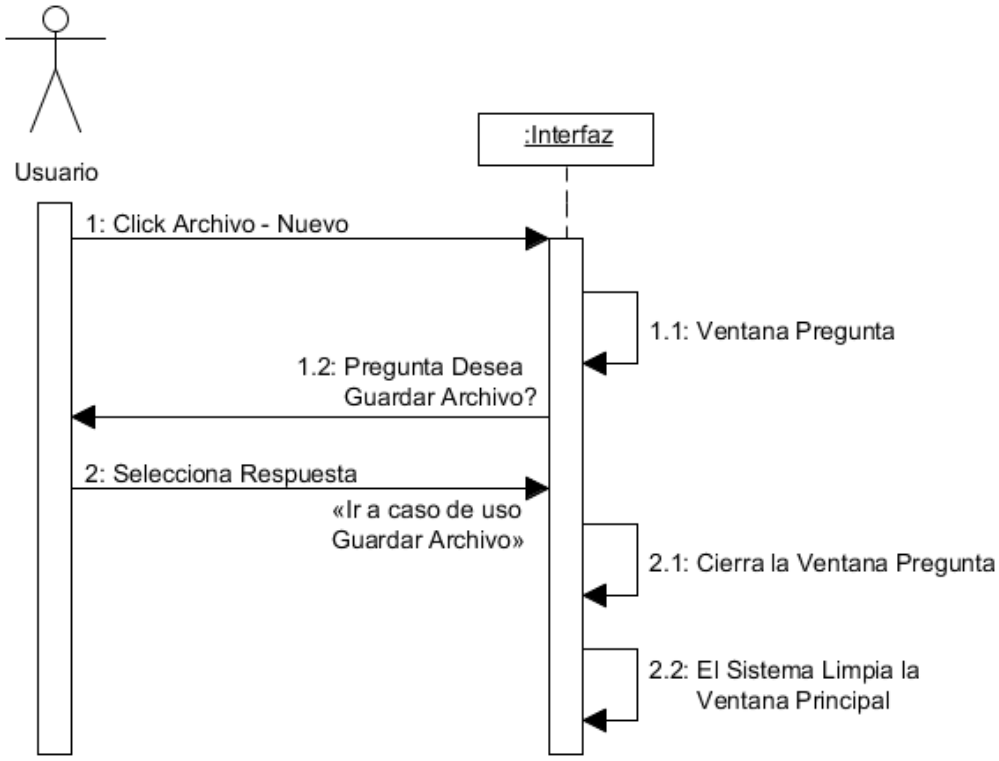


Imagen 3.7 Diagrama de Secuencia Guardar Archivo

3.4.7 NUEVO PROGRAMA



3.8 Diagrama de Secuencia Nuevo Programa

3.4.8 PROGRAMAR ROBOT

Para empezar el usuario escribe un programa en el editor de texto del sistema, luego en la consola de comandos hace el llamado a la función que desee y da clic en el botón ejecutar, el objeto Interfaz capta esta solicitud y llama al objeto interprete con la función ejecutar, el intérprete hace un llamado al objeto Lexer que realiza un análisis léxico al programa y el cual produce una secuencia de símbolos (números, identificadores, palabras reservadas, entre otros) que sirven para la posterior etapa. Luego el intérprete hace un llamado al objeto Parser que realiza un análisis sintáctico donde se comprueba las reglas de la gramática y devuelve una estructura de árbol que es útil para la siguiente etapa. El intérprete genera un código intermedio y se lo envía al objeto conexión Robot que se encarga de conectarse y enviar la solicitud de ejecución al servidor que ejecuta el código intermedio en su hardware.

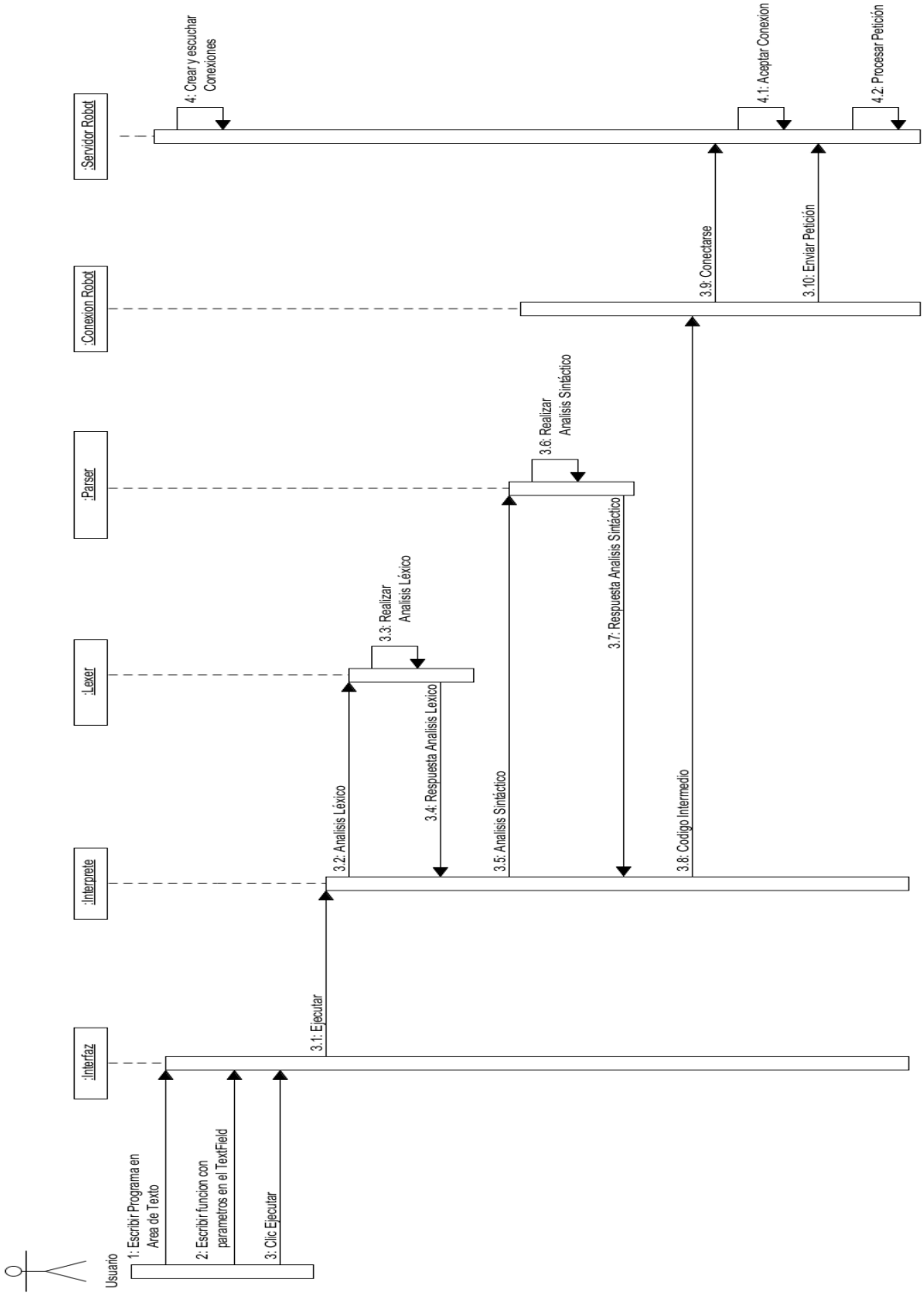


Imagen 3.9 Diagrama de Secuencia Programar Robot

3.5 MODELO FUNCIONAL

A continuación se ilustraran la serie de acciones que deben ser realizadas para cada caso de uso y la manera como puede reaccionar el sistema ante determinados eventos. Se utiliza diagramas de flujo de datos para mostrar las dependencias funcionales.

3.5.1 ABRIR PROGRAMA

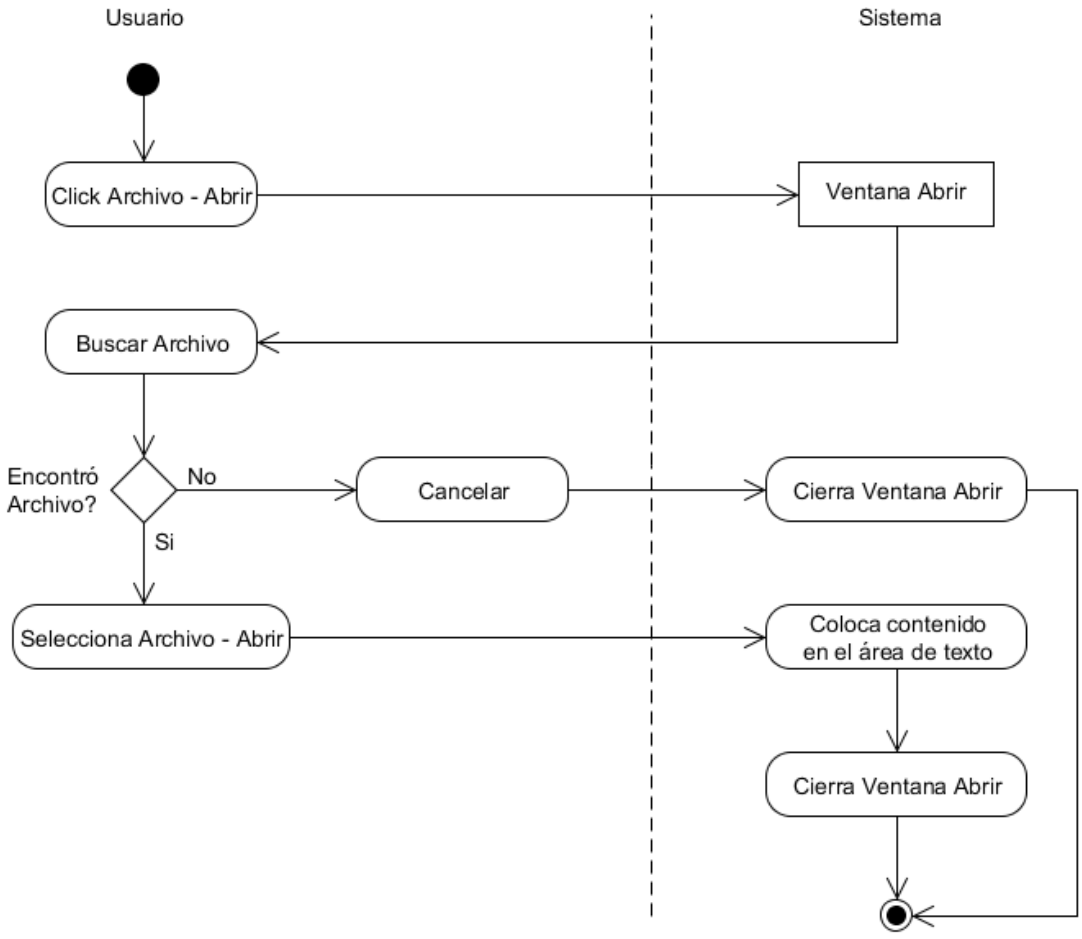


Imagen 3.10 Diagrama de Flujo Abrir Programa

3.5.2 CONFIGURAR ENTORNO

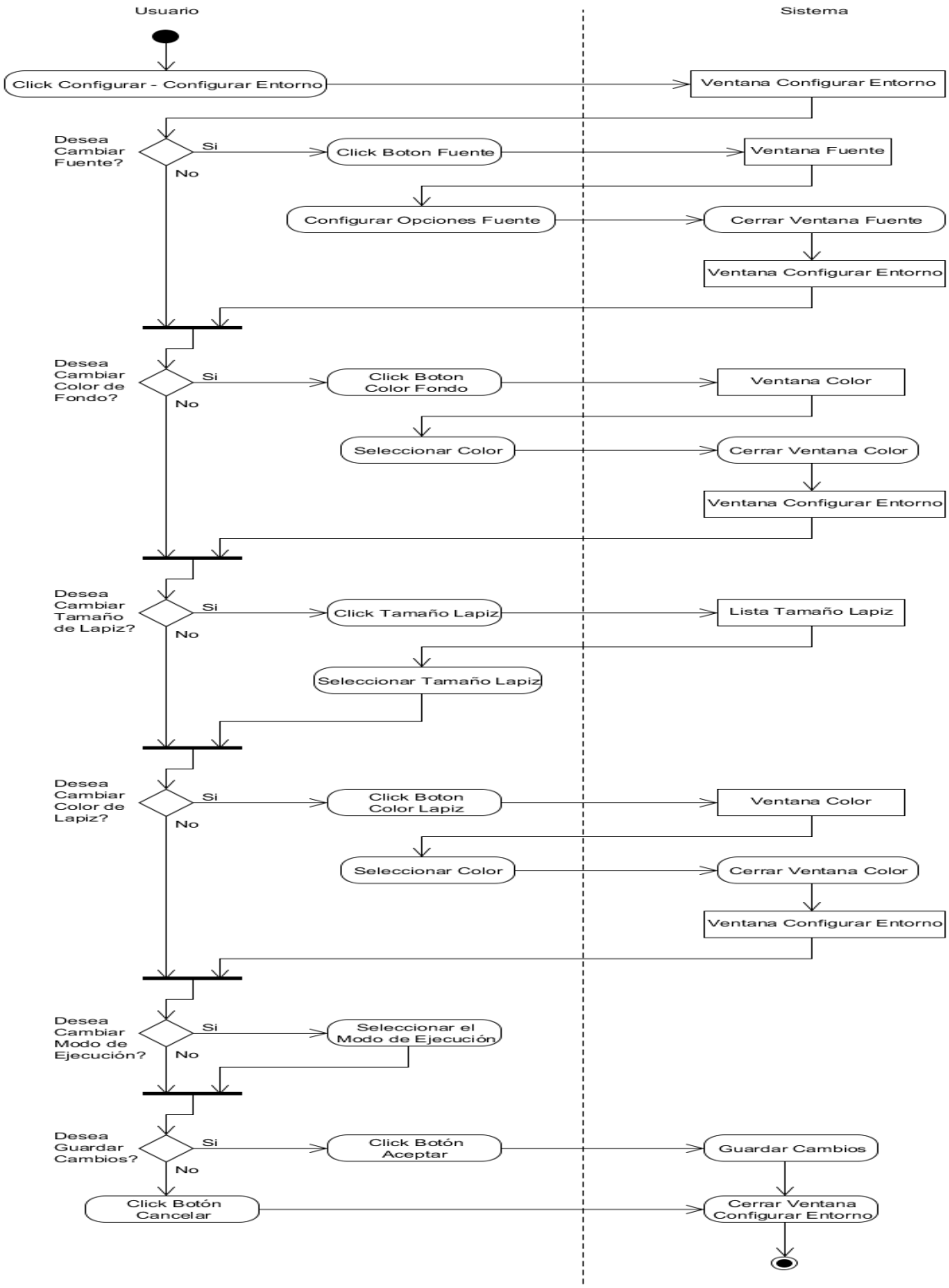


Imagen 3.11 Diagrama de Flujo Configurar Entorno

3.5.3 CONSULTAR AYUDA

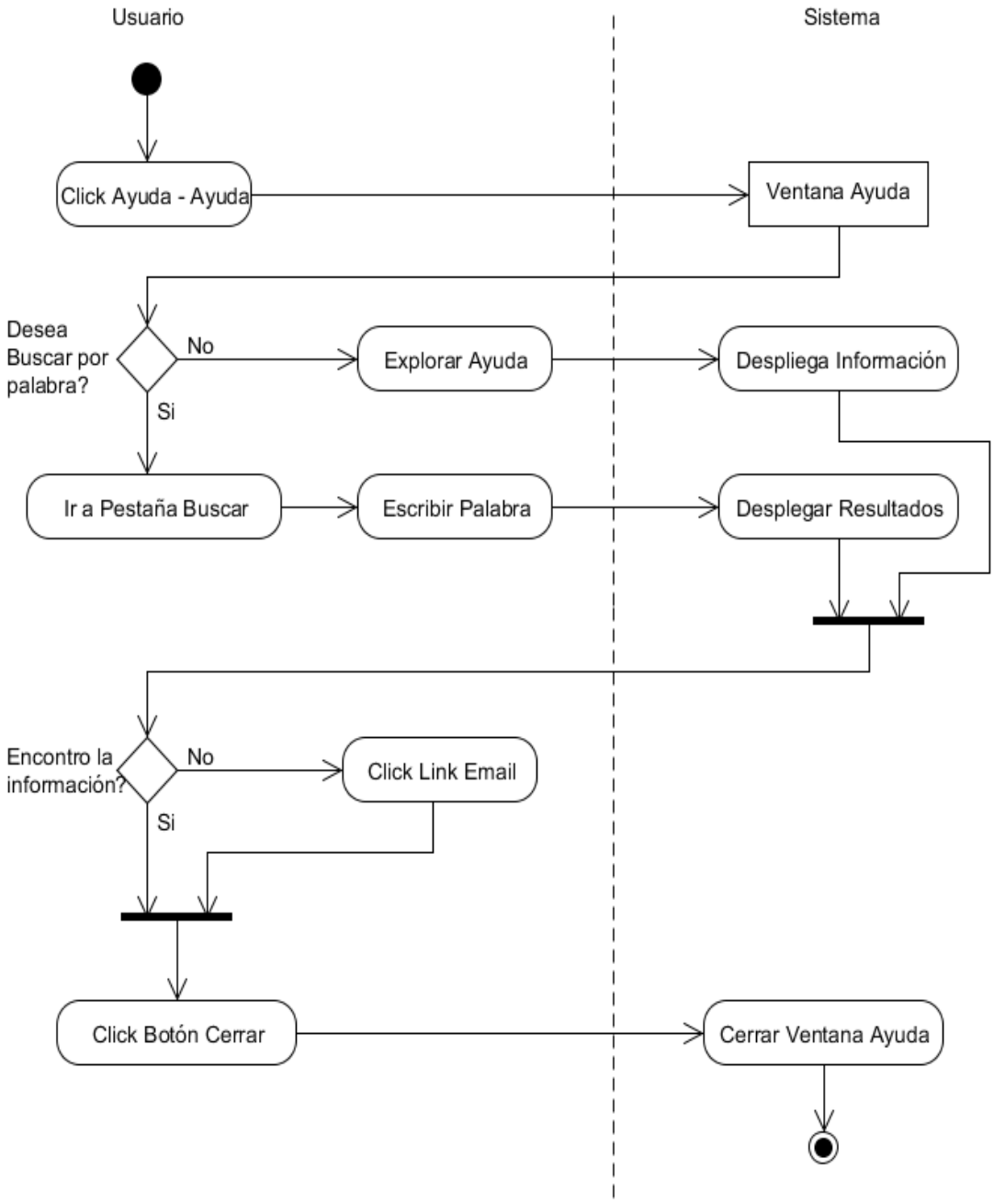


Imagen 3.12 Diagrama de Flujo Consultar Ayuda

3.5.4 CONSULTAR TUTORIAL

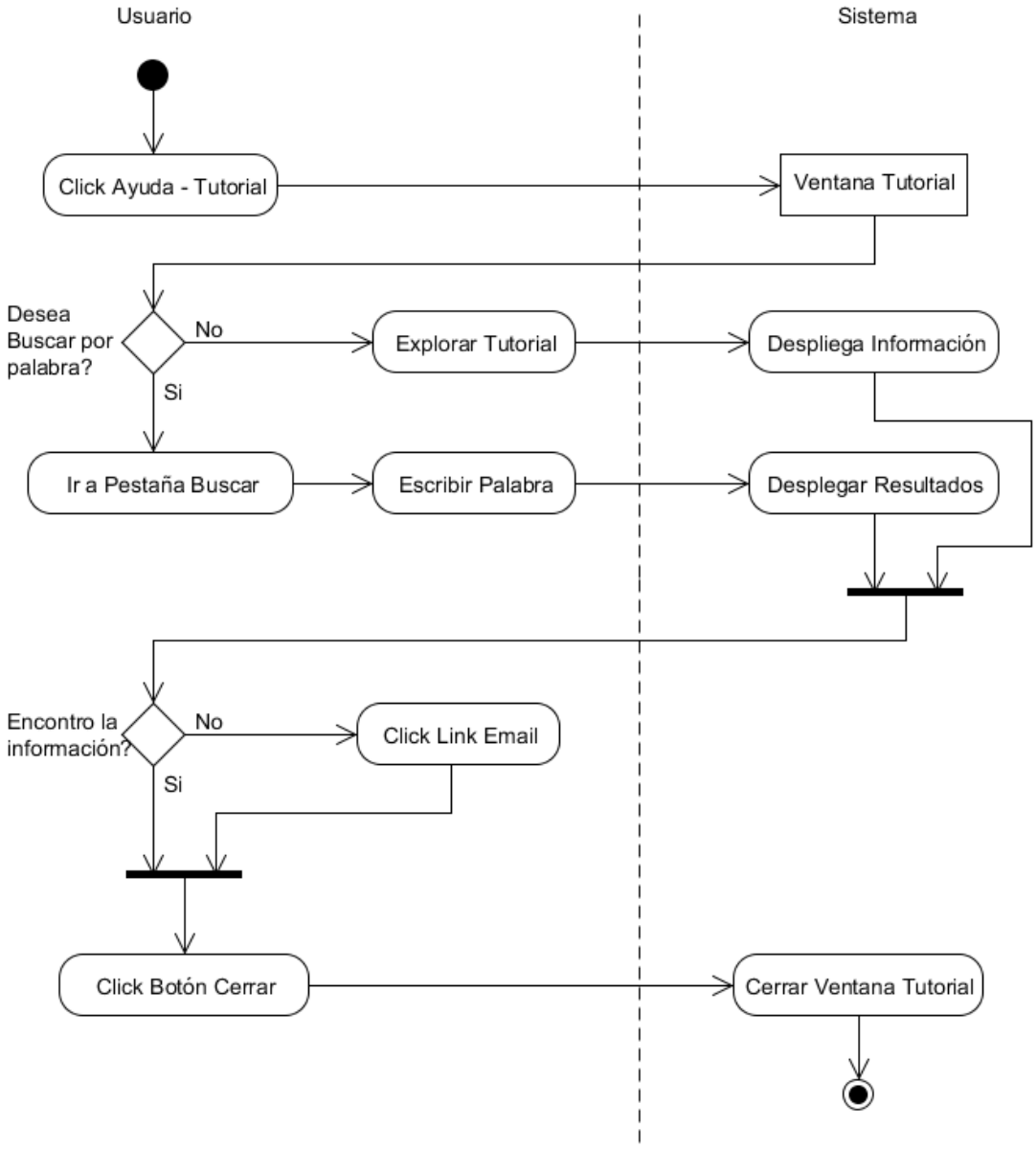


Imagen 3.13 Diagrama de Flujo Consultar Tutorial

3.5.5 EJECUTAR SIMULACIÓN

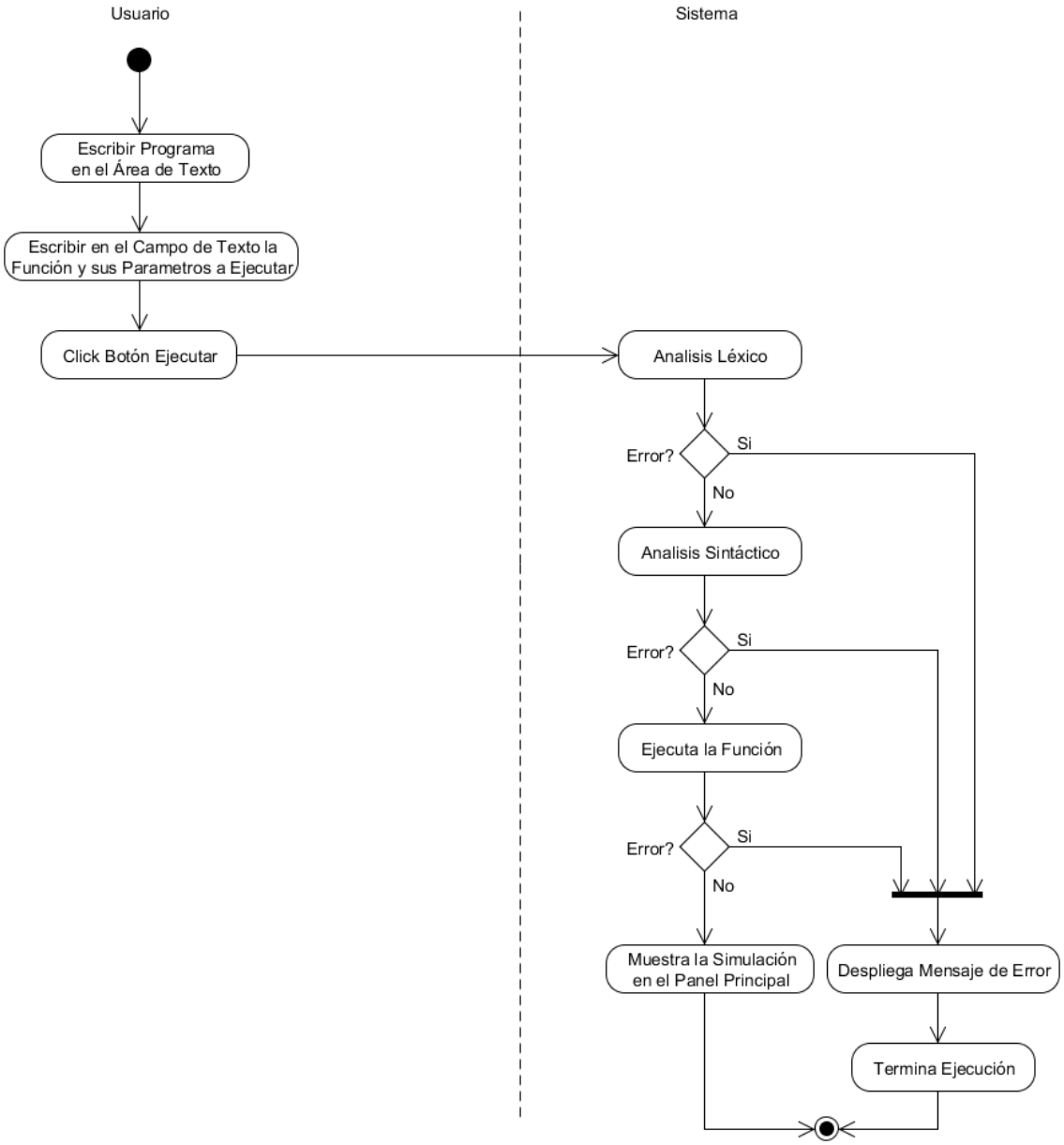


Imagen 3.14 Diagrama de Flujo Ejecutar Simulación

3.5.6 GUARDAR PROGRAMA

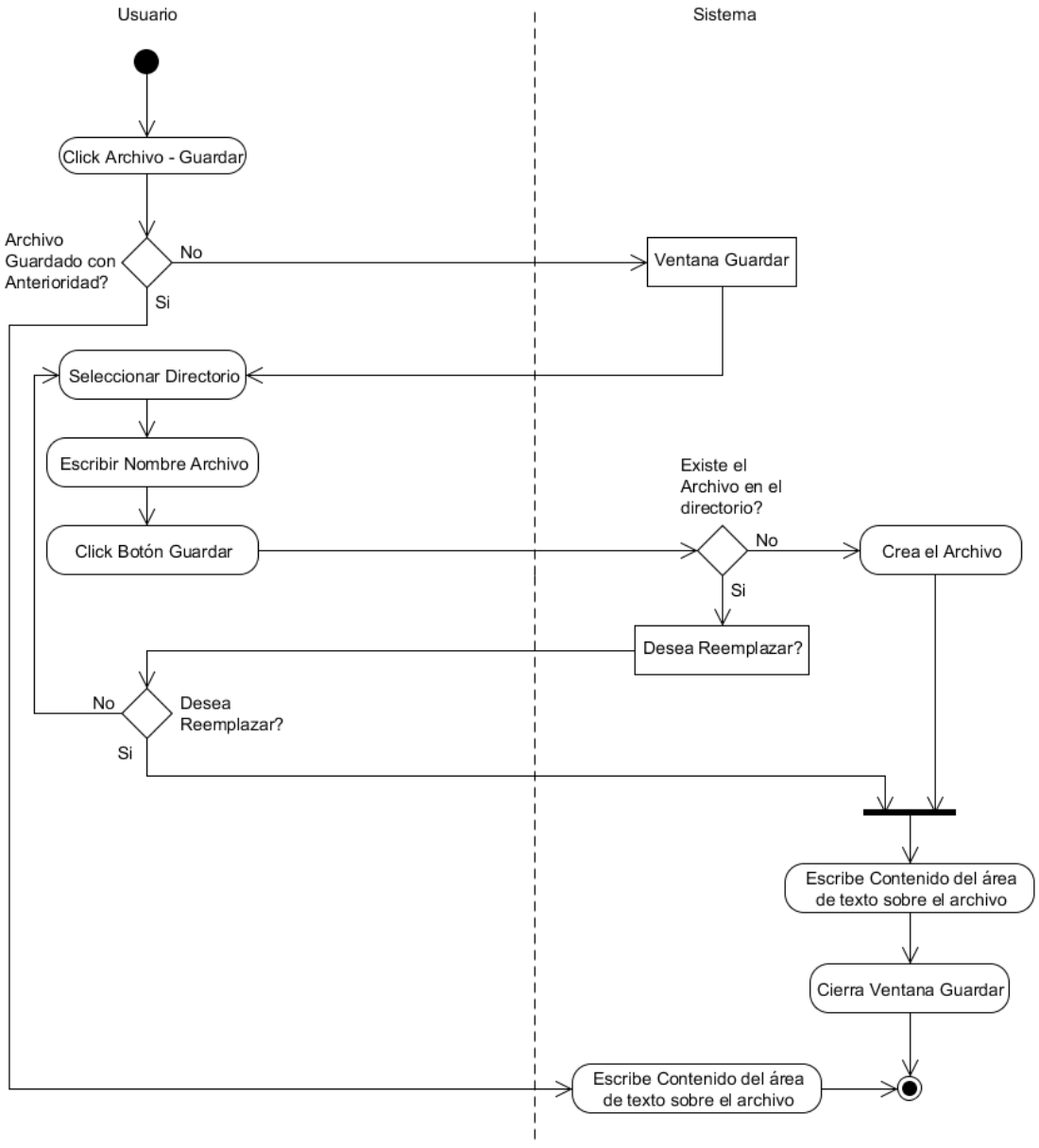


Imagen 3.15 Diagrama de Flujo Guardar Archivo

3.5.7 NUEVO PROGRAMA

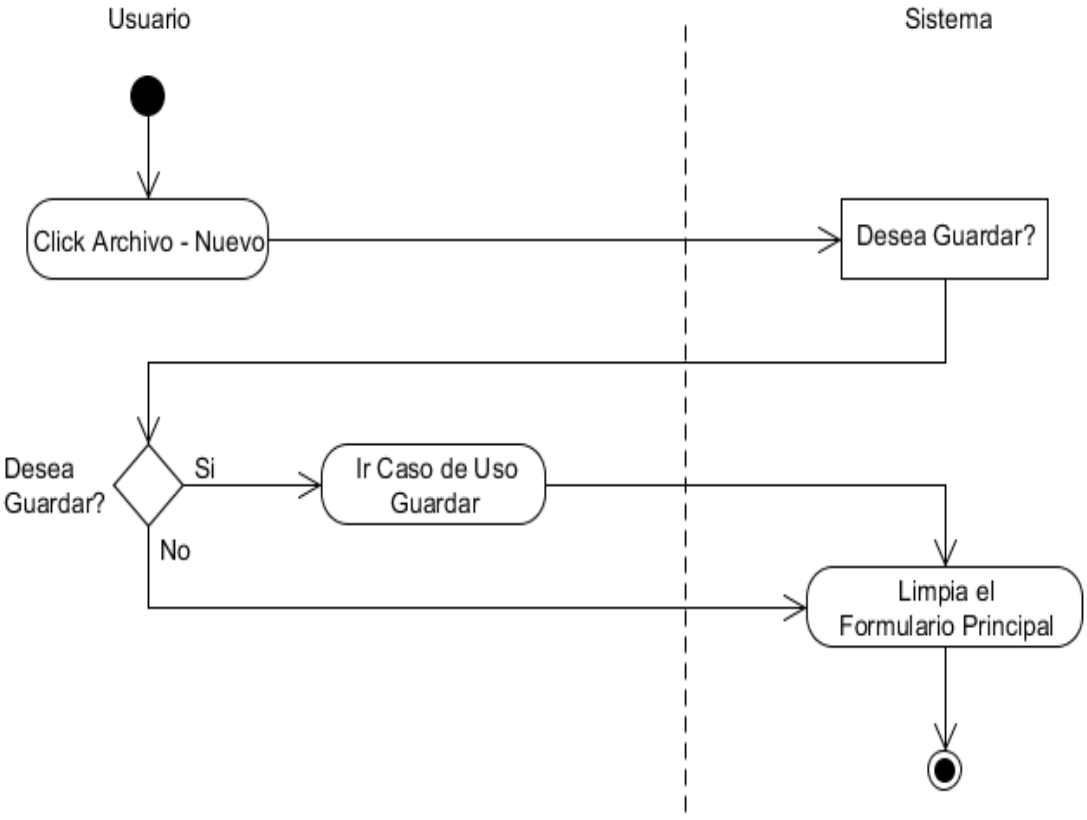


Imagen 3.16 Diagrama de Flujo Nuevo Programa

3.5.8 PROGRAMAR ROBOT

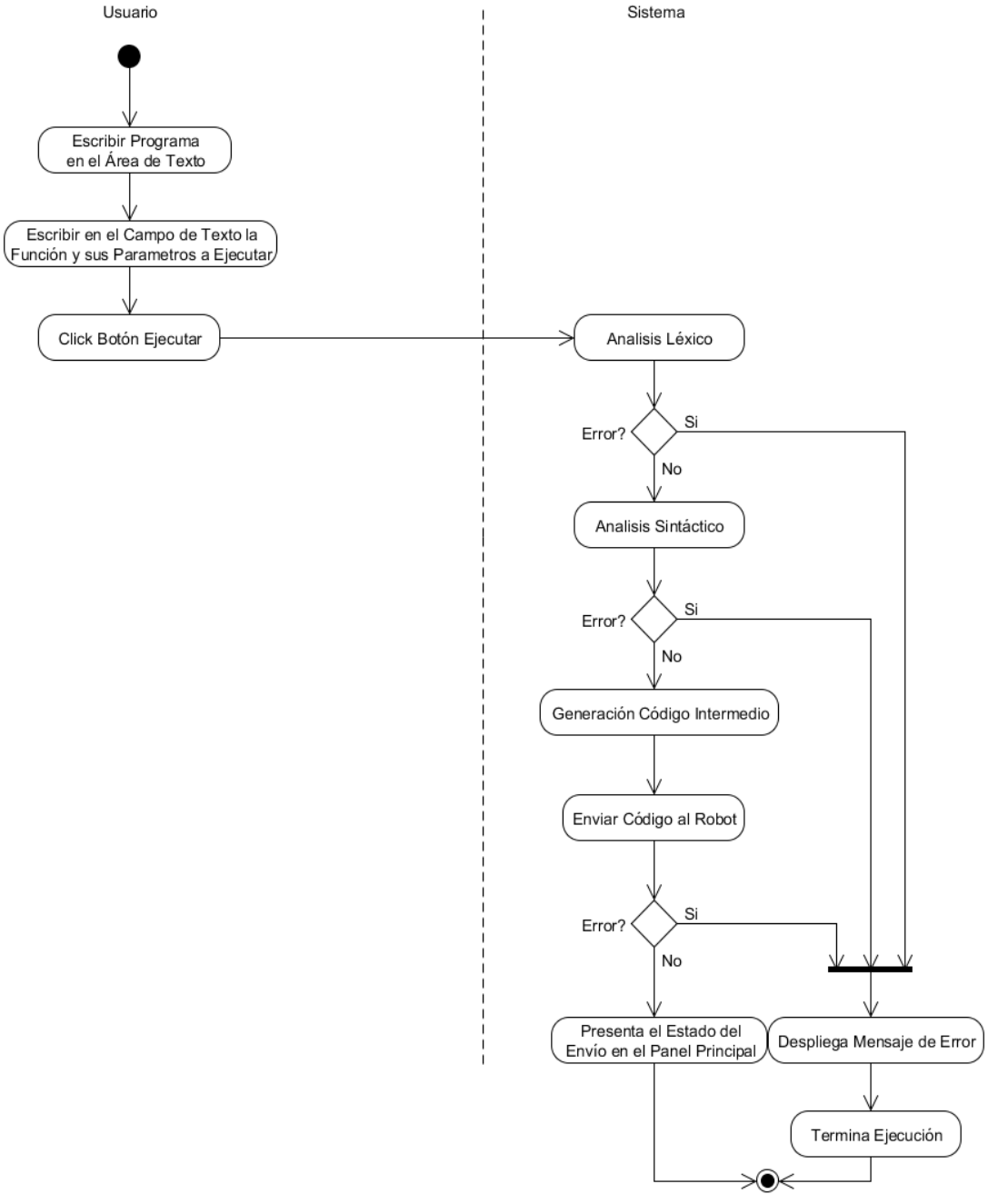


Imagen 3.17 Diagrama de Flujo Programar Robot

3.6 MODELO FINAL DE OBJETOS

DIAGRAMA DE CLASES

Las imágenes 3.18 y 3.19 presentan el diagrama de clases del sistema.

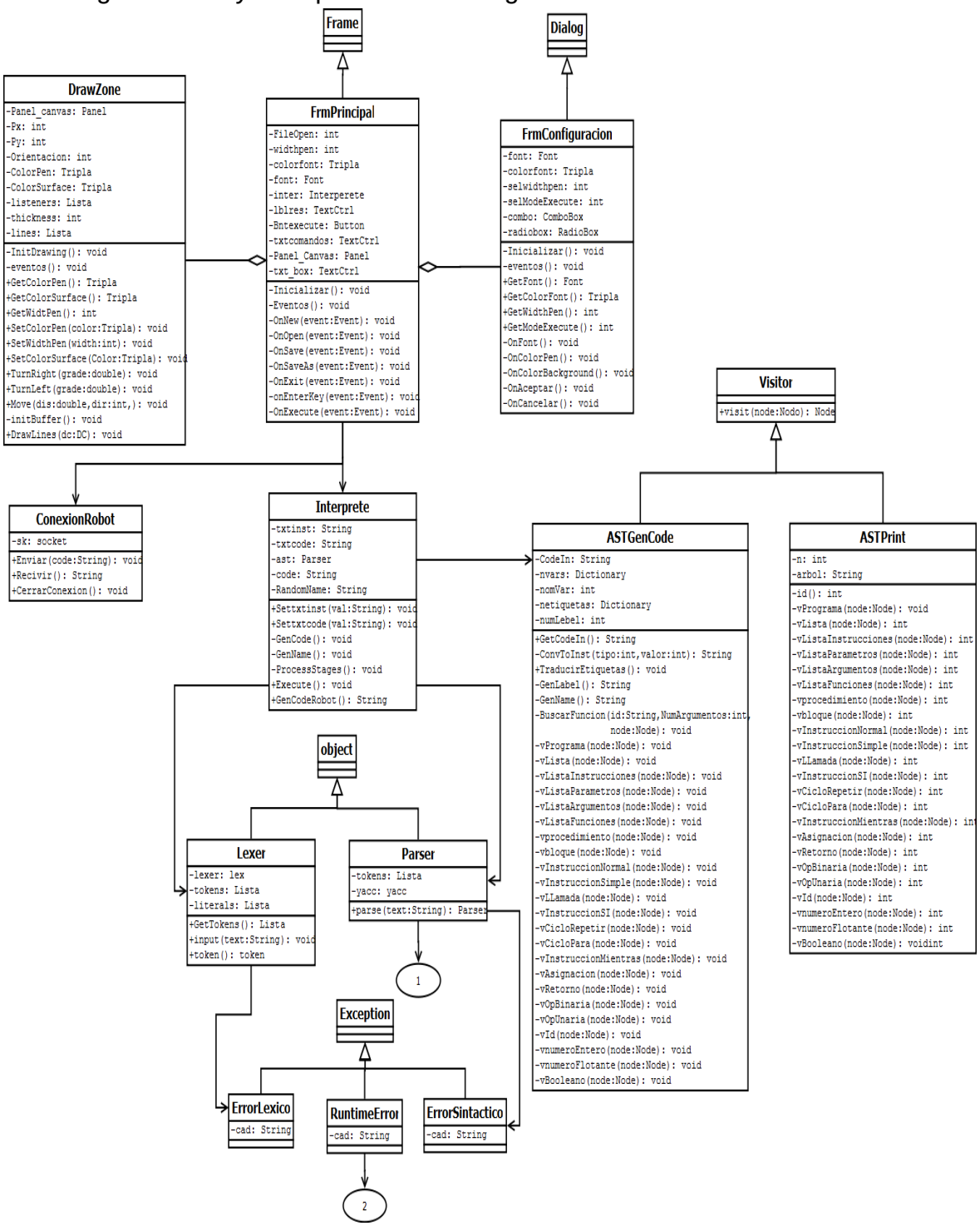


Imagen 3.18 Diagrama de Clases Parte 1

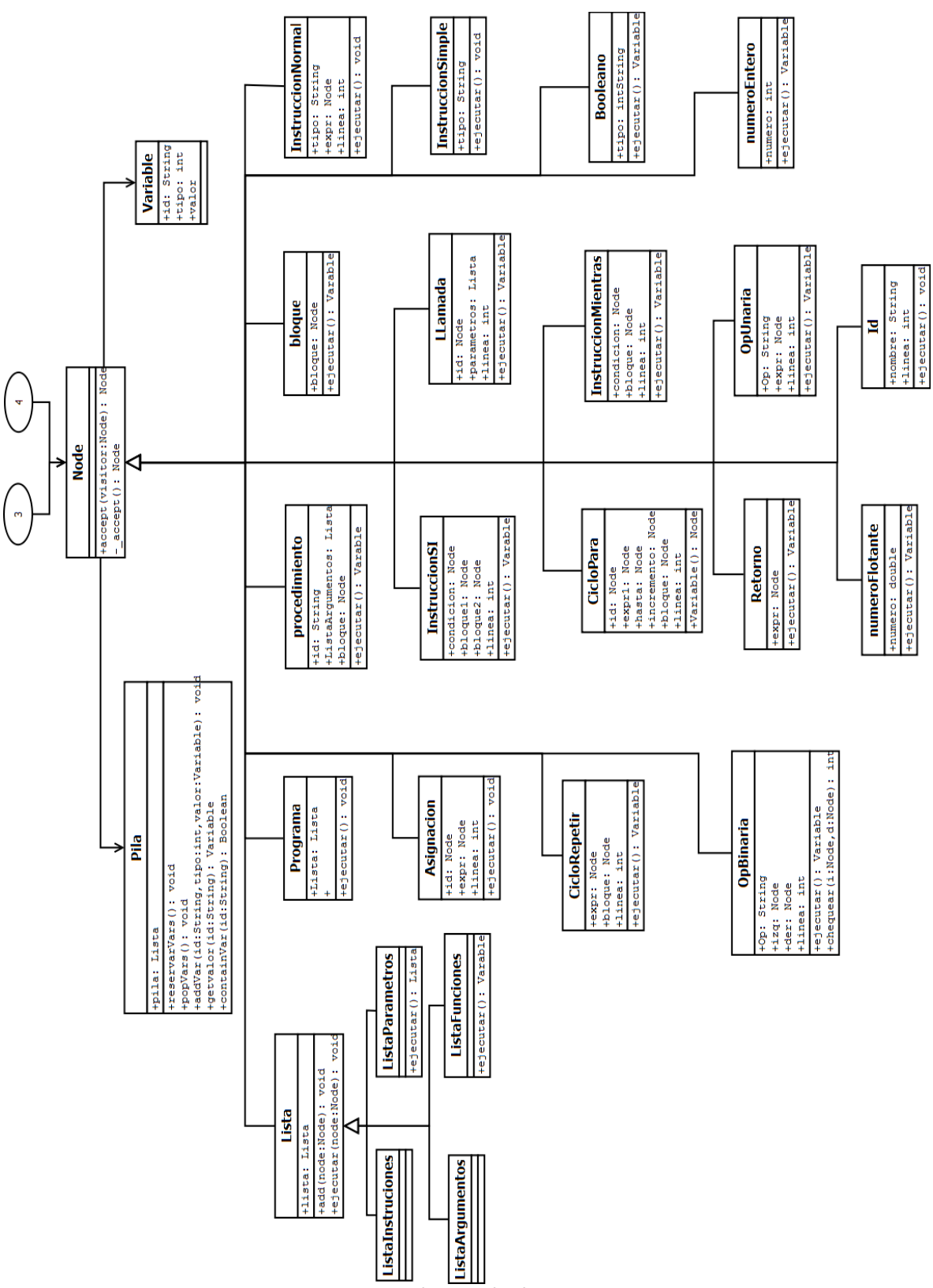


Imagen 3.19 Diagrama de Clases Parte 2

4. DISEÑO

4.1 ARQUITECTURA DEL SISTEMA

El Sistema computacional se desarrolló sobre la arquitectura cliente-servidor, en la cual se modela como un conjunto de servicios proporcionados por los servidores y un conjunto de clientes que usa estos servicios.

Para este proyecto se usó un solo servidor que se encuentra en la fpga del robot, y este atenderá las peticiones de un cliente pesado (fat-client). Se escogió un cliente rico debido a que es importante aprovechar los recursos de la fpga, y utilizando un cliente pesado se le quita responsabilidad de procesamiento al servidor.

4.2 SUBSISTEMAS

La tabla 4.1 muestra la descripción de los subsistemas.

Subsistema	Descripción
Interprete	Software que permite construir programas para ser ejecutados y vistos en 2D y para enviarle al robot el código traducido.
Analizador Léxico	Hace análisis léxico al programa.
Analizador Sintáctico	Hace análisis sintáctico al programa y crea el árbol AST.
Ejecutador AST	Ejecuta cada nodo del árbol AST.
Simulador	Muestra en forma gráfica (simulación en 2D) la ejecución del código
Configuración	Permite establecer configuraciones básicas del entorno.
Generador Seudocódigo	Traduce el código fuente a instrucciones de la máquina virtual que se encuentra en el servidor de la FPGA.
Ayuda	Brinda un tutorial de ejercicios y un sistema de ayuda.
Interfaz	Interfaz intuitiva que se presenta al usuario y que le sirve como entorno de desarrollo que integra los subsistemas del sistema interprete.
Conexión Cliente	Crea una conexión Ethernet y envía el programa al Robot.
Robot	Estructura física compuesta por una FPGA, motores y

	circuitos que muestra de forma física la ejecución del código desarrollado.
Conexión Servidor	Crea una conexión Ethernet y escucha las solicitudes del cliente.
Máquina virtual del Robot	Programa en C que toma el conjunto de instrucciones enviadas por el cliente y las ejecuta.
Procesador	Ejecuta el Intérprete del Robot sobre la FPGA y escucha las interrupciones generadas por el driver de los motores.
Driver de los Motores	Envía señales a los motores de acuerdo a las instrucciones y genera interrupciones indicando al procesador la terminación de los motores.

Tabla 4.1 Subsistemas

Diagrama de Bloques de Subsistemas

La imagen 4.1 muestra el diagrama de bloques del subsistema.

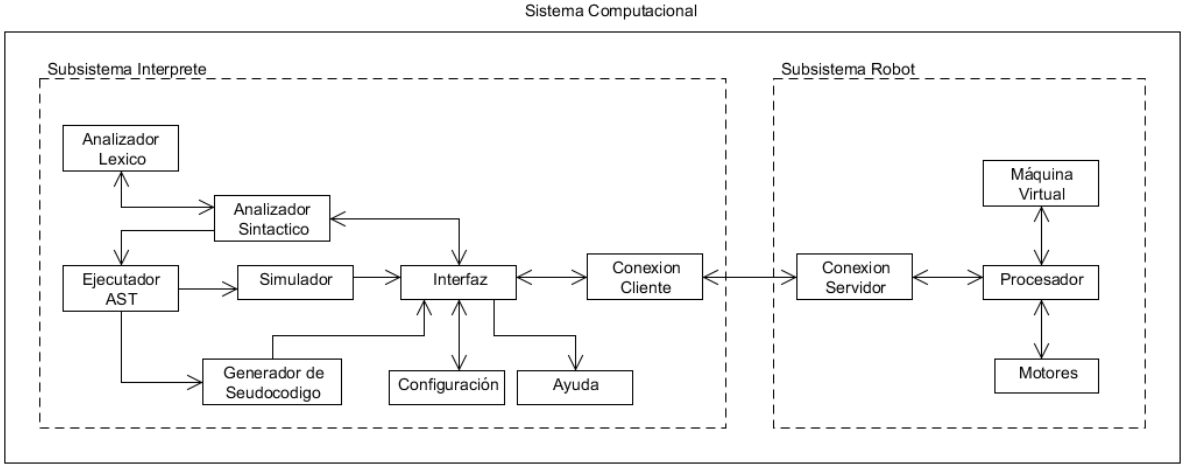


Imagen 4.1 Diagrama de bloques de Subsistemas

4.3 DISEÑO ESTÉTICO

4.3.1 DIAGRAMAS DE SECUENCIA DE VENTANAS

La imagen 4.2 presenta el diagrama de secuencia de ventanas del sistema.

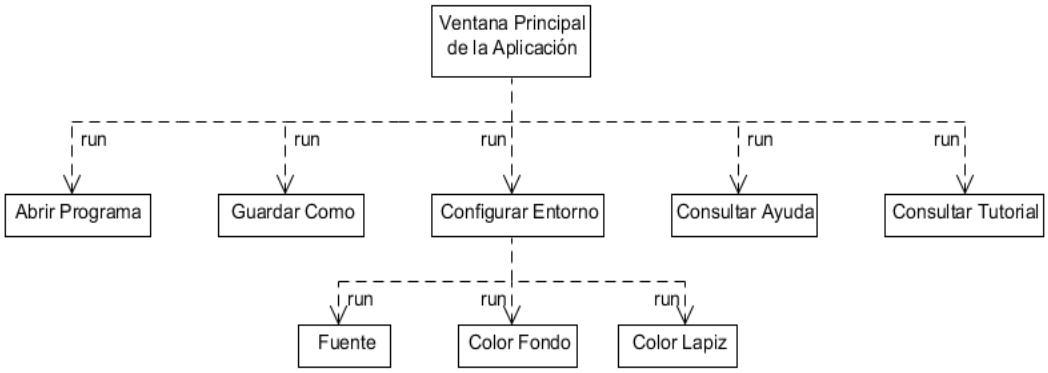


Imagen 4.2 Diagrama de Secuencia de Ventanas

4.3.2 VENTANA PRINCIPAL

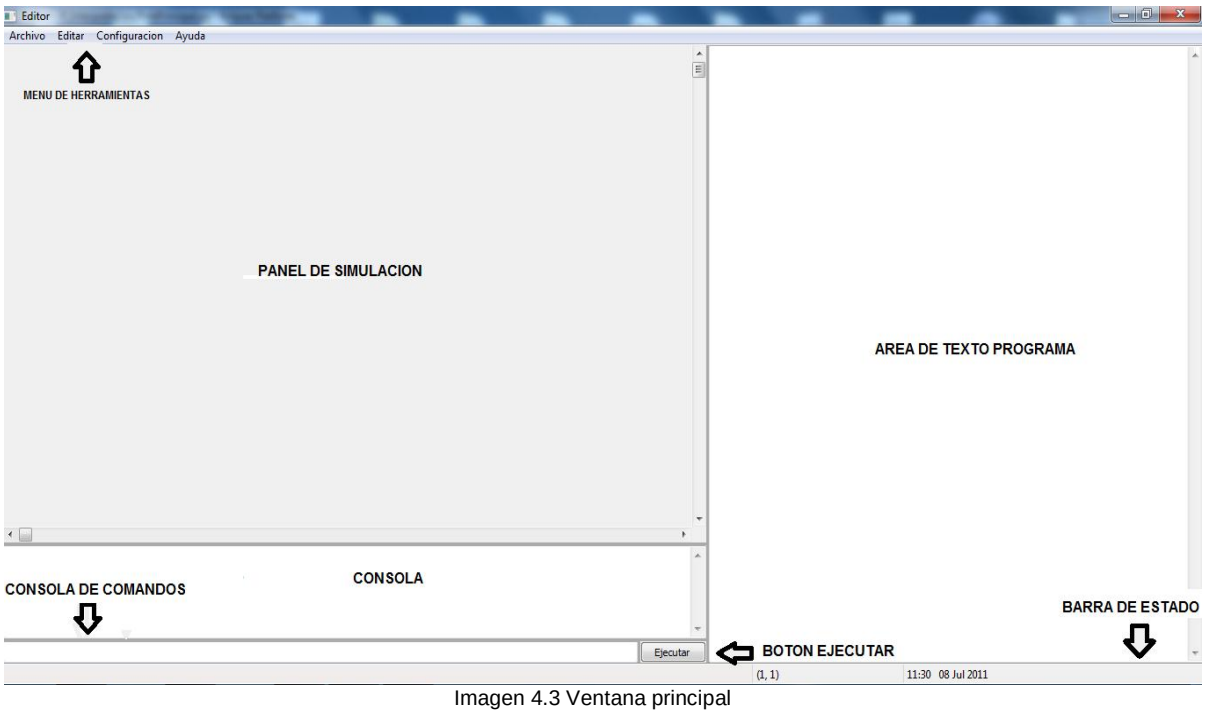


Imagen 4.3 Ventana principal

La ventana principal se observa en la imagen 4.3, esta aparece cuando el usuario ejecuta el programa.

La ventana principal está compuesta por:

- Menú de herramientas: Contiene las opciones de configuración, archivo y edición de la ventana principal. También cuenta con un sistema de ayuda y tutorial.
- Área de texto programa: Área en donde se puede escribir el programa a ejecutar.
- Consola de comandos: área en donde se puede hacer llamado a una función definida en el área de texto del programa, o de funciones simples del lenguaje.
- Botón ejecutar: botón que permite ejecutar lo que se encuentre en la consola de comandos.
- Panel de simulación: Área que me permite observar la simulación de lo que se ejecutó.
- Consola: Permite ver el histórico de la consola de comandos, los errores presentes en el código del programa y salida del comando escribe.
- Barra de estado: Muestra la fecha actual, la hora en que se abre la aplicación, la fila y columna donde está ubicado el cursor del área de texto del programa y la posición y dirección del robot en la simulación.

4.3.3 ABRIR PROGRAMA

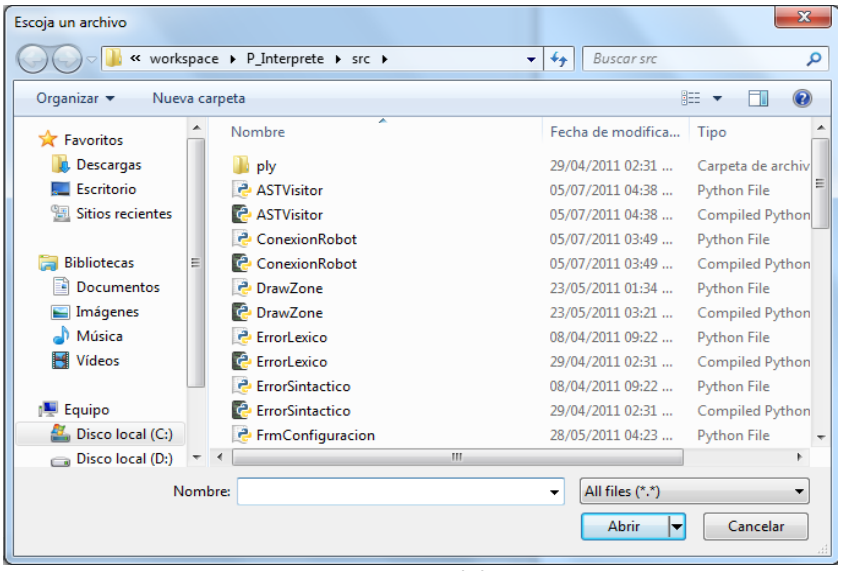


Imagen 4.4 Ventana Abrir Programa

La imagen 4.4 muestra la ventana de abrir programa que aparece cuando se da clic en la opción abrir del menú Archivo, permite desplazarse por el sistema de archivos para escoger el archivo a abrir con la aplicación.

4.3.4 GUARDAR COMO

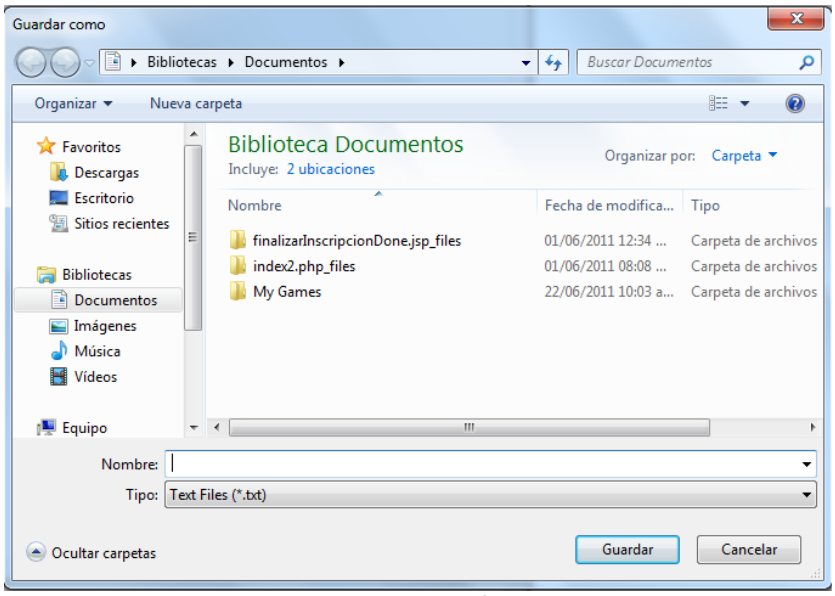


Imagen 4.5 Ventana Guardar Programa

La imagen 4.5 muestra la ventana guardar como que aparece cuando se da clic en la opción Guardar Como del menú Archivo, permite desplazarse por el sistema de archivos para escoger la ruta en donde quedara guardada la aplicación.

4.3.5 CONFIGURAR ENTORNO

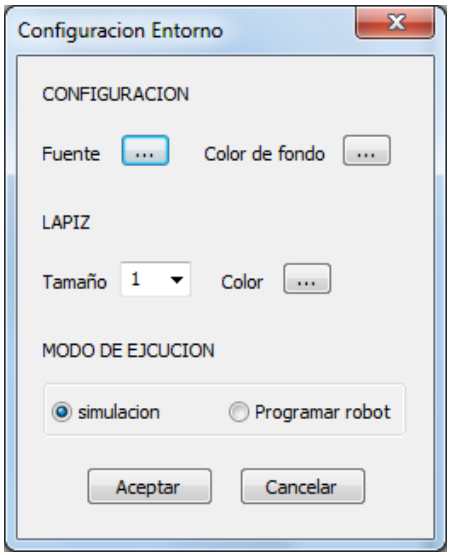


Imagen 4.6 Ventana Configurar Entorno

La imagen 4.6 muestra la ventana configurar entorno que aparece cuando se da clic en el menú configuración, permite establecer ciertas propiedades del panel de simulación, área de texto del programa, lápiz y el modo de ejecución.

4.3.6 CONSULTAR AYUDA

Imagen 4.7 Ventana Consultar Ayuda

La imagen 4.7 muestra la ventana de consultar ayuda, esta aparece cuando se da clic en la opción Ayuda del menú Ayuda. Permite navegar por un índice de temas relevantes del sistema.

4.3.7 CONSULTAR TUTORIAL

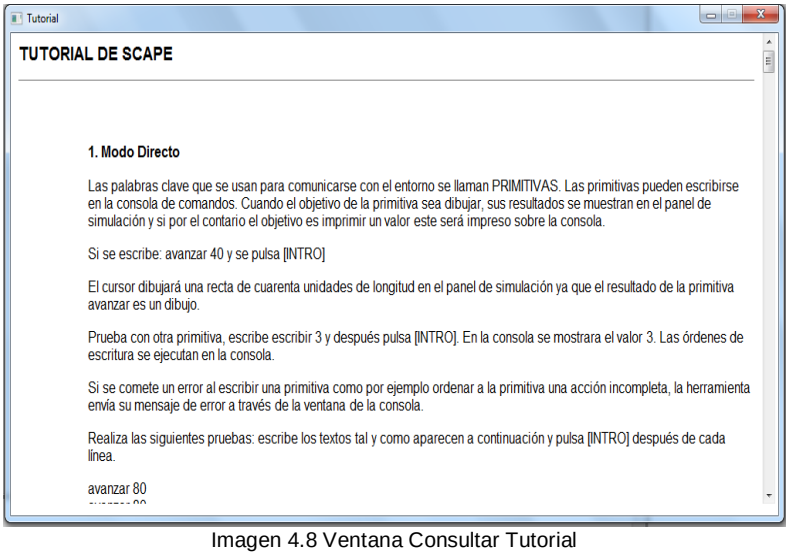


Imagen 4.8 Ventana Consultar Tutorial

La imagen 4.8 muestra la ventana consultar tutorial, aparece cuando se da clic en la opción Tutorial del menú Ayuda. Contiene una lista de ejercicios a realizar y temas que se deben conocer para el correcto funcionamiento de la herramienta.

4.3.8 FUENTE

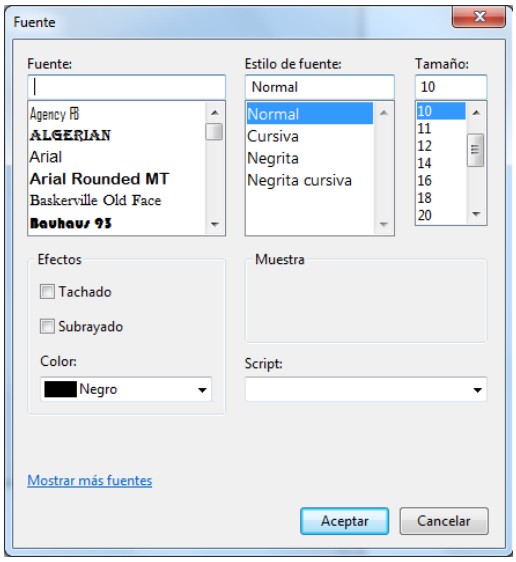


Imagen 4.9 Ventana Fuente

La imagen 4.9 muestra la ventana fuente, que aparece cuando se da clic en el botón fuente de la ventana de configuración. Permite establecer diferentes opciones de la fuente, como el tipo de letra, el estilo, tamaño, entre otros.

4.3.9 COLOR FONDO

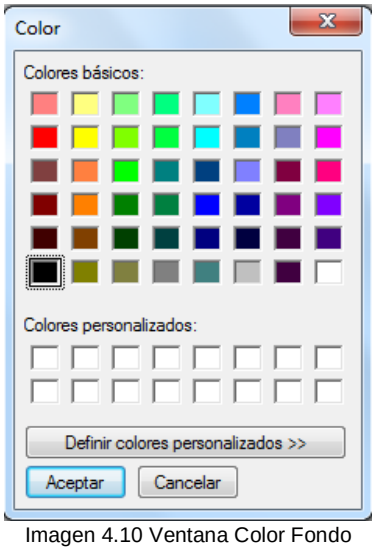


Imagen 4.10 Ventana Color Fondo

La imagen 4.10 muestra la ventana de color de fondo que aparece cuando se presiona el botón color fondo de la ventana de configuración.
Permite cambiar el color del panel de simulación.

4.3.10 COLOR LÁPIZ

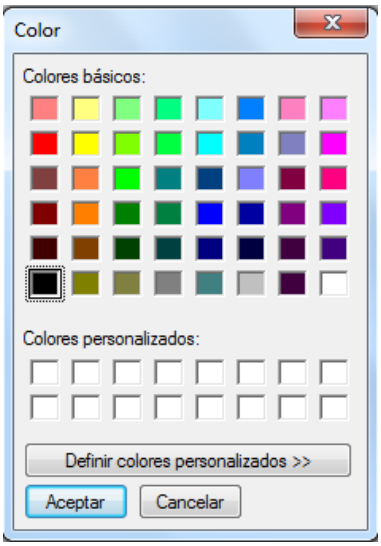


Imagen 4.11 Ventana Color Lápiz

La imagen 4.11 muestra la ventana color lápiz, que aparece cuando se presiona el botón color lápiz de la ventana de configuración. Permite cambiar el color del lápiz que actúa sobre el panel de simulación.

4.4 MODELO DE COMPONENTES

Describe los elementos físicos del sistema y sus relaciones. Los componentes incluyen archivos, bibliotecas compartidas, ejecutables, o paquetes.

La imagen 4.12 muestra el diagrama de componentes del subsistema intérprete.

DIAGRAMA DE COMPONENTES DEL SUBSISTEMA INTÉRPRETE

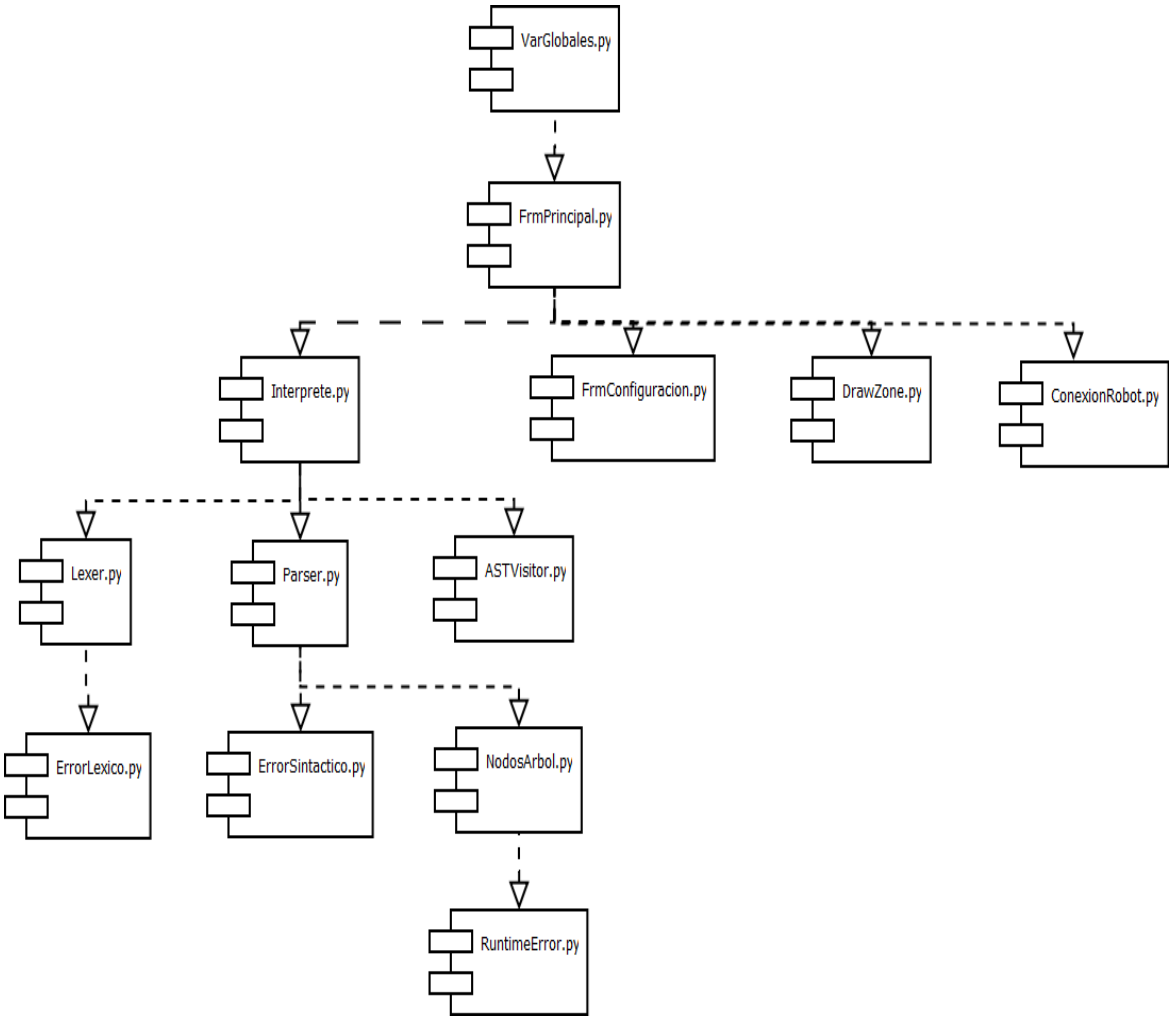


Imagen 4.12 Diagrama de componentes del subsistema interprete

4.5 MODELO DE DISTRIBUCIÓN

Muestra el despliegue de nodos locales y remotos en la organización del sistema, lo cual permite comprender mejor la topología del sistema distribuido.

Este diagrama consiste de nodos los cuales representan algún dispositivo de hardware como servidores, estaciones de trabajo, equipos computacionales, sensores, teléfonos, entre otros y muestran la configuración de los nodos, procesos, componentes y objetos que residen en ellos en tiempo de ejecución.

Las conexiones entre los nodos corresponden a las conexiones de red entre ellos.

DIAGRAMA DE DISTRIBUCIÓN

La imagen 4.13 muestra el diagrama de distribución del sistema.

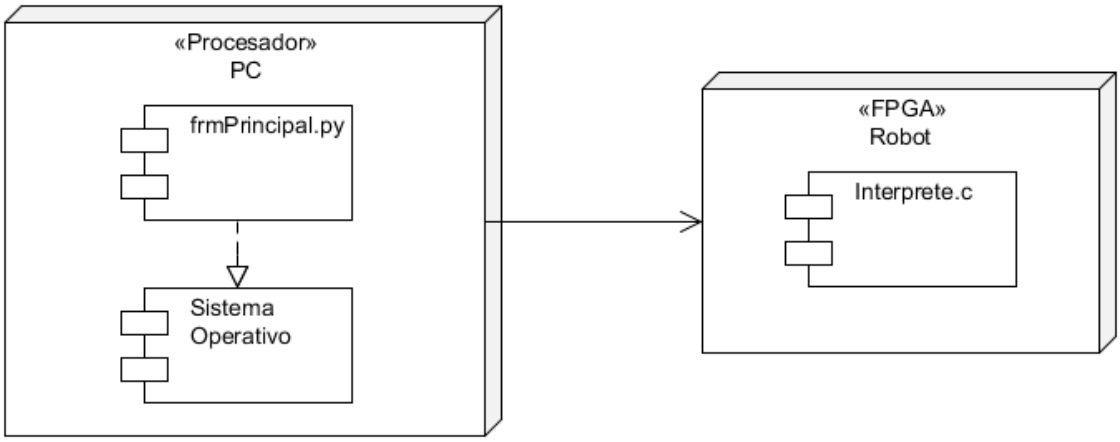


Imagen 4.13 Diagrama de Distribución

5. ESPECIFICACIÓN DEL LENGUAJE

5.1 INTRODUCCIÓN

5.1.1 CARACTERÍSTICAS

- Paradigmas: Imperativo, Funcional
- Técnica: Programación Estructurada
- Ámbito: Educativo
- Nivel: Alto
- Interpretado: Si
- Tipo de dato: Fuerte, Dinámico
- Influido por: Logo, Java, Python
- Plataforma: Multiplataforma
- Licencia: GNU GPL

5.1.2 NOTACIÓN

La descripción del análisis léxico y sintáctico usa una modificación de la notación de las gramáticas BNF. Usa el siguiente estilo de definición:

- Nombre ::= letra (letra | "_") * – *Un nombre es una letra seguida de una secuencia de cero o más letras y líneas al piso.*
- Letra ::= [a - z] | [A - Z] – *Una letra es cualquiera de las letras que hay en el alfabeto, puede ser mayúscula o minúscula (sin incluir "ñ" ni "Ñ").*
- Cada regla inicia con un nombre, el cual se define para identificar la regla y el símbolo *"::="*.
- La barra vertical *"|"* es usada para separar alternativas.
- El signo asterisco *"*"* significa cero o más repeticiones del ítem que lo precede.
- El signo más *"+"* significa una o más repeticiones del ítem que lo precede.
- Los corchetes *"[ ]"* son usados para expresar rangos de símbolos conocidos como las letras del inglés o los números del cero al 9 [0 - 9].
- Los paréntesis *"()"* son usados para agrupar.
- Los literales son encerrados en comillas simples *" ' ' "*.
- Los espacios en blanco son utilizados para separar tokens.

- Las reglas son normalmente contenidas en una sola línea, las reglas con muchas alternativas ocupan varias líneas, después de la primera línea inician con una barra vertical que expresa que son opciones de la regla.

5.2 ESTRUCTURA LÉXICA

5.2.1 SEPARADORES

Un programa no tiene líneas lógicas, debido a que el usuario podría escribir todo su programa en una sola línea, siendo los espacios en blanco, tabulaciones o caracteres de nueva línea los separadores entre cada uno de los tokens del programa, estos son ignorados tomándose como un único espacio en blanco. Esto permite añadir espacios para aumentar la claridad del programa.

5.2.1.1 ESPACIOS

Los espacios en blanco son considerados como separadores entre tokens, más de un espacio en blanco seguido no tiene sentido para el lenguaje el cual lo ignorará.

5.2.1.2 COMENTARIOS

Un comentario inicia con el carácter # y terminan en #. Los comentarios son ignorados.

5.2.1.3 DELIMITADORES

Los siguientes símbolos son considerados como delimitadores en un programa:

- “inicio” (Palabra reservada, al iniciar un bloque)
- “fin” (Palabra reservada, al terminar un bloque)
- “(“ (Paréntesis abierto, en cualquier caso)
- “)” (Paréntesis cerrado, en cualquier caso)
- “,” (Coma, al separar parámetros)
- “.” (Punto, en números con punto flotante)
- = (Igual, en la asignación)
- # (Numeral, en comentarios)

5.2.2 PALABRAS RESERVADAS

Los siguientes identificadores son utilizados como palabras reservadas o palabras claves y no pueden ser usadas como identificadores ordinarios.

- avanzar – av
- retroceder – rt
- alto – al
- bajarpluma – bp
- subirpluma – sp
- girarderecha – gd
- girarizquierda – gi
- borrapantalla – bo
- procedimiento – pr
- inicio
- fin
- si
- sino
- repetir
- mientras
- escribe
- falso
- verdadero
- retorno

5.2.3 LITERALES

Un identificador es descrito por la siguiente definición léxica:

- FACTOR:= NUMEROENTERO|NUMEROFLOTANTE
- FACTOR:= falso | verdadero

Los literales son notaciones para los valores constantes de alguno de los tipos de datos del lenguaje.

5.2.3.1 LITERALES NUMÉRICOS

Los literales numéricos solo incluyen el número absoluto y no incluyen signo, ejemplo: -1 es una expresión compuesta entre el operador unario “-“ y el literal 1. Existen dos tipos de literales numéricos, enteros y de punto flotante

Un literal entero es descrito por la siguiente definición léxica:

- numeroentero ::= nocerodigito digito * | “0”
- nocerodigito ::= [1 – 9]
- digito ::= [0 – 9]

El límite de los números enteros es -1048576 y 1048576.

Un literal flotante es descrito por la siguiente definición léxica:

- numeroflotante ::= ((nocerodigito digito *) + “.” digito +) | “0” “." digito +
- nocerodigito ::= [1 – 9]
- digito ::= [0 – 9]

La parte decimal es redondeada a 5 decimales.

5.2.3.2 LITERALES BOOLEANOS

Un literal booleano es descrito por la siguiente definición léxica:

- Booleanos ::= verdadero | falso

5.2.4 IDENTIFICADORES

Un identificador es descrito por la siguiente definición léxica:

- Identificador ::= letra (letra | digito | “_”) *
- letra ::= [a - z] | [A – Z]
- digito ::= [0 – 9]

Los nombres o identificadores no tienen limite en su longitud y son usados para referirse a entidades declaradas en un programa, las entidades del lenguaje son la creación de procedimientos, la declaración de variables o el llamado de procedimientos.

5.2.5 TIPOS

En el lenguaje solo existe una clase de tipo, los tipos primitivos, los cuales pueden ser almacenados en variables, pasados como argumentos en llamados a

procedimientos, retornados por procedimientos y operados por los operadores binarios y unarios del lenguaje. El lenguaje maneja un tipo especial None, el cual es retornado por procedimientos que en su flujo de programa no retornan ninguno de los tipos del lenguaje, el tipo None, no puede ser asignado a una variable o retornado por el usuario explícitamente, porque es considerado un error léxico o sintáctico, se maneja internamente para chequear en tiempo de ejecución si existen errores, también se pensó que para versiones futuras pueda ser incluido como un tipo e ir extendiendo las capacidades del lenguaje a otros paradigmas.

Los tipos primitivos del lenguaje son enteros, flotantes y booleanos. Una variable puede almacenar cualquiera de los tipos primitivos antes mencionados y su valor solo puede ser cambiado por una operación de asignación.

Los enteros y los flotantes son creados por un literal numérico o retornados de una operación aritmética, al igual que los booleanos que pueden ser creados por un literal booleano o retornados de una operación relacional o lógica.

5.2.5.1 TIPOS ENTEROS

Estos representan números en el rango de -1048576 y 1048575, cuando el resultado de una operación se sale del rango, se debe terminar la ejecución del programa.

5.2.5.2 TIPOS FLOTANTES

Estos representan números en el mismo rango de los enteros y manejan una precisión de 5 decimales, cualquier literal con más de 5 decimales es redondeado a 5 decimales.

5.2.5.3 TIPOS BOOLEANOS

Estos representan los valores verdadero y falso, internamente son representados con un uno y un cero respectivamente.

Las expresiones booleanas determinan el control de flujo de varios tipos de declaraciones.

- La declaración si
- La declaración mientras

5.2.6 OPERADORES

Los operadores se agrupan de la siguiente manera:

- Operadores aritméticos: *, /, +, - (binario), - (unario)
- Operadores relacionales: <, <=, >, >=, ==, !=
- Operadores booleanos: y, o, !

Para todos los casos los dos operandos deben ser de un tipo soportado por el operador y del mismo tipo, las operaciones entre números y booleanos son consideradas errores de tiempo de ejecución.

5.2.6.1 OPERADORES ARITMÉTICOS

Existen dos tipos de operadores multiplicativos: el operador multiplicación y el operador división. La operación multiplicación se hace con el caracter "*". La operación división se hace con el caracter "/", produciendo el cociente de sus operandos. También existen dos tipos de operadores de adición: el operador suma y el operador resta. El operador binario + hace la suma, cuando se aplica a dos operandos de tipo numérico produce la suma de los operandos, el operador binario - hace la resta, produciendo la diferencia de dos operandos numéricos.

El tipo resultante de hacer una operación es regido por el tipo de sus operandos y se muestra en la tabla 5.1.

Tipo del operando 1	Tipo del operando 2	Tipo resultante
entero	entero	entero
entero	flotante	flotante
flotante	entero	flotante
flotante	flotante	flotante

Tabla 5.1 Tipos resultantes entre operaciones

El signo resultante de los operadores multiplicativos es regido por el tipo de sus operandos y se muestra en la tabla 5.2.

Signo operando 1	Signo operando 2	Signo resultante
+	+	+
+	-	-

-	+	-
-	-	+

Tabla 5.2 Signo resultante de operadores multiplicativos

Todos los operadores aritméticos son binarios con excepción de uno especial de carácter unario, el cual es "-" que cambia de signo al operando y el operador ! (not). Las divisiones por cero terminan con la ejecución del programa.

5.2.6.2 OPERADORES RELACIONALES

Las operaciones relacionales se hacen con los caracteres:

- Menor que: <
- Menor o Igual que: <=
- Mayor que: >
- Mayor o igual que: >=
- Igual a: ==
- Diferente de: !=

Todos los operadores relacionales son binarios, los datos que se van a operar deben ser de tipo numérico, con la excepción de la igualdad y la diferencia que también admiten tipo booleano. El tipo resultante de una operación relacional es siempre booleano.

5.2.6.3 OPERADORES LÓGICOS

Las operaciones lógicas o booleanas se hacen con los caracteres:

- And: y
- Or: o
- Not: !

Todos los operadores booleanos son binarios con excepción de uno especial de carácter unario, el cual es "!" que niega el operando. Los datos que se van a operar deben ser de tipo booleano. El tipo resultante de una operación relacional es siempre booleano y se rigen según las tablas 5.3, 5.4, 5.5.

OPERADOR AND – Y

Operando 1	Operando 2	Resultado
VERDADERO	VERDADERO	VERDADERO
VERDADERO	FALSO	FALSO
FALSO	VERDADERO	FALSO
FALSO	FALSO	FALSO

Tabla 5.3 Operador and

OPERADOR OR – O

Operando 1	Operando 2	Resultado
VERDADERO	VERDADERO	VERDADERO
VERDADERO	FALSO	VERDADERO
FALSO	VERDADERO	VERDADERO
FALSO	FALSO	FALSO

Tabla 5.4 Operador or

OPERADOR NOT – !

Operando	Resultado
VERDADERO	FALSO
FALSO	VERDADERO

Tabla 5.5 Operador not

5.3 PROCEDIMIENTOS Y VARIABLES

5.3.1 DECLARACIONES

Una declaración introduce una entidad en un programa e incluye un identificador que se puede utilizar para referirse a esta entidad. En el lenguaje una declaración de entidad es una de las siguientes:

- Declaración de una variable en un bloque.
- Declaración de un procedimiento.

5.3.2 LLAMADOS

Es la forma de referirse a la declaración de un procedimiento para ejecutar el código que hay en esta, pasándole los parámetros que esta necesite. Cada llamado espera el retorno de un tipo primitivo o None.

5.3.3 REPRESENTACIÓN DE LOS DATOS

Todos los datos del lenguaje son representados por objetos internamente. Cada objeto tiene un tipo y un valor. Cada uno de estos objetos es mutable, es decir, cuando se modifican, no se crea otro objeto diferente, sino que solo se modifican los valores. La vida de cada objeto se extiende desde el punto de su declaración hasta el final de la ejecución del procedimiento en la cual el dato fue creado. Para futuras versiones se puede implementar un recolector de basura, para aumentar eficiencia en la ejecución.

5.3.4 ÁMBITOS

El ámbito de la declaración es la región de un programa dentro de la cual la entidad declarada puede ser referenciada usando un identificador, una declaración está en el ámbito en un particular punto de un programa si y solo si la declaración del ámbito incluye ese punto.

5.3.4.1 ÁMBITO PROCEDIMIENTOS

Cuando se declara una función el ámbito de esta es global, es decir, después de declarada puede ser llamada desde cualquier punto del programa.

5.3.4.2 ÁMBITO DE VARIABLES

El ámbito de la variables declaradas en una función es el bloque principal de esta función, no se extiende a otros procedimientos mediante el llamado de estas, ya que los argumentos del llamado a otra función son pasados por valor, con lo cual no se pasa la referencia a esos parámetros si no que se hace una copia de estos.

5.4 GRAMÁTICA DEL LENGUAJE

5.4.1 PROGRAMA

Un programa es una lista de procedimientos.

PROGRAMA:= LISTAPROCEDIMIENTOS
LISTAPROCEDIMIENTOS:= LISTAPROCEDIMIENTOS PROCEDIMIENTO|

5.4.2 PROCEDIMIENTOS

Un procedimiento se declara escribiendo el token procedimiento o pr seguido de un identificador, seguido de una lista de argumentos separados por coma dentro de paréntesis y luego un bloque.

PROCEDIMIENTO:= (procedimiento|pr) ID ‘(’LISTAARGUMENTOS ’)’ BLOQUE
PROCEDIMIENTO:= (procedimiento|pr) ID ‘(‘ ’)’ BLOQUE
LISTAARGUMENTOS:= LISTAARGUMENTOS ’,’ ID | ID

5.4.3 BLOQUE

Un bloque es una secuencia de instrucciones, dentro de los tokens inicio y fin.

BLOQUE:= inicio LISTAINSTRUCCIONES fin
LISTAINSTRUCCIONES:= LISTAINSTRUCCIONES INSTRUCCIÓN

Un bloque es ejecutado, ejecutando cada una de las instrucciones de la primera a la última. Algunas instrucciones pueden hacer que la ejecución del bloque termine abruptamente.

5.4.4 INSTRUCCIONES

Las instrucciones se pueden clasificar en instrucciones compuestas, simples y que actúan sobre el robot.

Las instrucciones compuestas son aquellas que contienen otras instrucciones, estas controlan la ejecución de las instrucciones que contienen de alguna manera, en general las instrucciones compuestas se escriben en varias líneas, las instrucciones compuestas del lenguaje son el si, el mientras, el para y el ciclo repetir.

- INSTRUCCION:= BIFURCACION
- INSTRUCCION:= CICLOREPETIR
- INSTRUCCION:= CICLOMIENTRAS
- INSTRUCCION:= CICLOPARA

Las instrucciones simples al contrario de las compuestas no contienen otras instrucciones.

- INSTRUCCION:= ASIGNACION
- INSTRUCCION:= LLAMADA

- INSTRUCCION:= retorno BOOLEANO1
- INSTRUCCION:= INSTRUCCIONNORMAL EXPRESION
- INSTRUCCIONNORMAL:= escribe
- INSTRUCCION:= INSTRUCCIONSIMPLE
- INSTRUCCIONSIMPLE:= borrarpantalla | bo

Las instrucciones que actúan sobre el robot se dividen a su vez en dos subgrupos: instrucciones de movimiento e Instrucciones de lápiz.

Las instrucciones de movimiento son instrucciones que le permiten controlar el desplazamiento y orientación del robot.

- INSTRUCCION:= INSTRUCCIONNORMAL EXPRESION
- INSTRUCCIONNORMAL:= avanza | av | retrocede | rt |girarizquierda| gi | girarderecha | gd
- INSTRUCCION:= INSTRUCCIONSIMPLE
- INSTRUCCIONSIMPLE:= alto | al

Las instrucciones de lápiz le permiten controlar lo que se desea dibujar o escribir sobre la superficie en la que se encuentra el robot.

- INSTRUCCION:= INSTRUCCIONSIMPLE
- INSTRUCCIONSIMPLE:= bajarpluma | bp | subirpluma | sp

A continuación se describen detalladamente cada una de estas instrucciones.

- Si

La declaración “si” se utiliza para la ejecución condicional:

- BIFURCACION:= si BOOLEANO1 BLOQUE ELSE
- ELSE:= sino BLOQUE |

Esta instrucción evalúa primero la expresión booleana, si la expresión booleana retorna verdadero se ejecuta el primer bloque o bloque de ejecución verdadero y luego sigue con la ejecución al final del segundo bloque o bloque de ejecución falso, si la expresión booleana retorna falso se ejecuta el segundo bloque.

La instrucción si también se puede declarar sin bloque de ejecución falso, en este caso si la expresión booleana retorna falso la ejecución continua al final del bloque de ejecución verdadero.

- Repetir

La declaración de “repetir” repite la ejecución de un bloque un número dado de veces:

- CICLOREPETIR:= (repetir|re) EXPRESION BLOQUE

El bloque de la instrucción se ejecuta el número de veces que este en la expresión, por lo cual la expresión debe ser una variable o literal numérico, en caso contrario es considerado un error.

- Mientras

La instrucción mientras repite la ejecución de un bloque tantas veces como la expresión sea verdadera.

- CICLOMIENTRAS:= mientras BOOLEANO1 BLOQUE

Las instrucción mientras ejecuta primero la expresión booleana, si la expresión booleana retorna verdadero se ejecuta el bloque, cuando el bloque termina la ejecución se vuelve a ejecutar la expresión booleana y esto continua hasta que la expresión booleana retorno falso, en tal caso continua con la ejecución al final del bloque.

- Para

La instrucción “para” inicializa una variable con una expresión que es ejecutada y chequea que el valor de inicialización sea igual al valor de la segunda expresión, en caso contrario ejecuta el bloque y aumenta la primera variable en el valor dado al incremento por la tercera expresión, continua repitiendo hasta que la primera variable sea igual a la segunda expresión.

- CICLOPARA:= para ID ‘=’ EXPRESION hasta EXPRESION ‘,’ incremento ‘=’ EXPRESION BLOQUE

- Asignación

La instrucción de asignación, conecta variables a valores.

- ASIGNACION:= ID ‘=’ BOOLEANO1

Esta instrucción evalúa la expresión del lado derecho la cual es ejecutada y su resultado lo asigna a la variable con identificador ID de la izquierda. Cuando es utilizada una variable no definida produce un error que termina con la ejecución del programa.

- Llamada

Ejecuta el código de un procedimiento declarado.

- LLAMADA:= ID ‘(’ LISTAPARAMETROS ‘)’
- LLAMADA:= ID ‘(’ ‘)’
- LISTAPARAMETROS:= LISTAPARAMETROS ‘,’ BOOLEANO1
- LISTAPARAMETROS:= BOOLEANO1

Esta instrucción ejecuta el código del procedimiento con identificador ID, el procedimiento llamado puede tener parámetros, estos parámetros deben ir encerrados con paréntesis, cada parámetro es una expresión que se ejecuta y es separado por el signo coma, si el procedimiento no se encuentra declarado el lenguaje presentar un error.

- Retorno

La instrucción retorno termina con la ejecución del procedimiento donde este declarado.

- INSTRUCCION:= retorno BOOLEANO1

Esta instrucción ejecuta la expresión e inmediatamente después termina con la ejecución del procedimiento en el cual este declarado, retornando el resultado de la expresión al procedimiento quien hizo el llamado.

- Escribe

Esta instrucción ejecuta la expresión y despliega su resultado en la consola de salida del sistema.

- INSTRUCCIONNORMAL:= escribe

- Borrar pantalla

La superficie de dibujo (panel de simulación del sistema) se limpia, el robot permanece en su posición actual.

- INSTRUCCIONSIMPLE:= borrarpantalla | bo

- Avanza

Esta instrucción ejecuta la expresión y su resultado si es numérico, lo representa en el robot avanzando esa cantidad de unidades en la orientación actual (en la simulación la orientación es dada por la punta del robot), el resultado es visto en el panel de simulación del sistema y en el robot físico, si el resultado de la expresión no es numérico, esto es interpretado como un error.

- INSTRUCCIONNORMAL:= avanza | av

- Retrocede

Esta instrucción ejecuta la expresión y su resultado, si es numérico, lo representa en el robot retrocediendo esa cantidad de unidades, el resultado es visto en el panel de simulación del sistema o en el robot físico, si el resultado de la expresión no es numérico, esto es interpretado como un error.

- INSTRUCCIONNORMAL:= retrocede | rt

- Girar izquierda

Esta instrucción ejecuta la expresión y su resultado si es numérico, lo representa en el robot girando hacia la izquierda esa cantidad de unidades, el resultado es visto en el panel de simulación del sistema y en el robot físico, si el resultado de la expresión no es numérico, esto es interpretado como un error.

- INSTRUCCIONNORMAL:= girarizquierda| gi

- Girar derecha

Esta instrucción ejecuta la expresión y su resultado si es numérico, lo representa en el robot girando hacia la derecha esa cantidad de unidades, el resultado es visto en el panel de simulación del sistema y en el robot físico, si el resultado de la expresión no es numérico, esto es interpretado como un error.

- INSTRUCCIONNORMAL:= girarderecha | gd

- Alto

Finaliza la ejecución del procedimiento en el cual esta aparece, similar al retorno pero sin retornar valores.

- INSTRUCCIONSIMPLE:= alto | al

- Bajar pluma

El robot baja el lápiz y lo pone sobre la superficie de dibujo, en la simulación el robot se dispone a dibujar en el panel de simulación del sistema y dibuja mientras avanza y retrocede.

- INSTRUCCIONSIMPLE:= bajarpluma | bp

- Subir pluma

El robot levanta el lápiz de la superficie de dibujo, en la simulación el robot no dibuja en el panel de simulación del sistema mientras avanza o retrocede.

- INSTRUCCIONSIMPLE:= subirpluma | sp

5.4.5 EXPRESIONES

Cuando una expresión en un programa es ejecutada el resultado puede ser una variable, un valor o None si la expresión es producto de llamar a un procedimiento que no retorna un valor.

Si una expresión denota una variable, y se requiere su valor para su uso en una evaluación, entonces el valor de esa variable se utiliza, en este contexto se puede hablar simplemente del valor de la expresión y esta expresión tiene un tipo conocido en tiempo de ejecución el cual es utilizado para hacer comprobación de tipos.

Los tipos de expresiones del lenguaje se pueden dividir entonces de acuerdo al tipo de operadores que tiene la expresión: Aritméticas, Relacionales y Lógicas.

En el lenguaje todas las expresiones son evaluadas en un orden específico de izquierda a derecha de acuerdo a la precedencia de cada operador del lenguaje y el valor que tengan los operandos en ese momento.

5.4.5.1 EXPRESIONES ARITMÉTICAS

Tienen la misma precedencia y son sintácticamente asociativos por la izquierda (agrupan de izquierda a derecha).

```
EXPRESION:= EXPRESION '+' TERMINO
          | EXPRESION '-' TERMINO
          | TERMINO
TERMINO:=  TERMINO '*' FACTOR
          | TERMINO '/' FACTOR
          | FACTOR
FACTOR:= '-' FACTOR
```

El tipo de cada uno de los operandos debe ser de tipo numérico, en caso contrario un error de tiempo de ejecución ocurre.

La tabla 5.6 se presenta información adicional sobre las expresiones aritméticas, la prioridad más alta es 1.

Operación	Operador	Aridad	Asociatividad	Prioridad
cambio de signo	-	unario	por la derecha	1
multiplicación	*	binario	por la izquierda	2
División	/	binario	por la izquierda	2
Suma	+	binario	por la izquierda	3
Resta	-	binario	por la izquierda	3

Tabla 5.6 Expresiones Aritméticas

5.4.5.2 EXPRESIONES RELACIONALES

Los operadores relacionales son sintácticamente asociativos por la izquierda (se agrupan de izquierda a derecha) aunque esto no es precisamente útil, por ejemplo a<b<c es siempre un error en tiempo de ejecución, porque el tipo de a<b es booleano y < no es un operador de booleanos. En el caso de los de igualdad,

a==b==c, el resultado de a==b es siempre booleano y c debe ser de tipo booleano o si no un error en tiempo de ejecución ocurre. Así entonces a==b==c no prueba que los tres parámetros sean iguales.

CONDICIONAL2:= CONDIONAL2 ‘<’ EXPRESION
CONDICIONAL2:= CONDIONAL2 <= EXPRESION
CONDICIONAL2:= CONDIONAL2 ‘>’ EXPRESION
CONDICIONAL2:= CONDIONAL2 >= EXPRESION
CONDICIONAL2:= EXPRESION

El tipo resultante de una operación relacional es siempre booleano y el tipo de cada operando de un operador de comparación numérica debe ser un tipo numérico o si no ocurre un error en tiempo de ejecución

CONDICIONAL1:= CONDIONAL1 == CONDICIONAL2
CONDICIONAL1:= CONDIONAL1 != CONDICIONAL2
CONDICIONAL1:= CONDICIONAL2

El == (igual que) y != (no es igual que) son operadores análogos a los operadores relacionales excepto por su menor precedencia, así a < b == c < d es verdadero cuando a<b y c<d tienen el mismo valor.

Los operadores de igualdad pueden ser usados para comparar dos operandos de tipo numérico o dos operandos de tipo booleano, en otro caso sucede un error en tiempo de ejecución.

En todos los casos a!=b produce el mismo resultado que !(a==b).

La tabla 5.7 presenta información adicional sobre las expresiones relacionales, la prioridad más alta es 1.

Operación	Operador	Aridad	Prioridad
igual que	==	binario	2
distinto que	!=	binario	2
menor que	<	binario	1
menor o igual que	<=	binario	1
mayor que	>	binario	1
mayor o igual que	>=	binario	1

Tabla 5.7 Expresiones Relacionales

5.4.5.3 EXPRESIONES LÓGICAS

Los operadores and y or toman operandos booleanos y su resultado es booleano, si alguno de los operandos no es booleano entonces ocurre un error en tiempo de ejecución, ambos tienen la misma precedencia.

FACTOR:= ‘!’ FACTOR
FACTOR:= ‘(’ BOOLEANO1 ‘)’
BOOLEANO1:= BOOLEANO1 o BOOLEANO2
BOOLEANO1:= BOOLEANO2
BOOLEANO2:= BOOLEANO2 y CONDICIONAL1
BOOLEANO2:= CONDIONAL1

Estos son sintácticamente asociativo por la derecha y es asociativo con respecto al resultado de evaluación, por ejemplo para la expresión (a y b) y c produce el mismo resultado de que a y (b y c).

El operador and evalúa el operando de la derecha solo si el valor de la izquierda es verdadero, y el operador or evalúa el operando de la derecha solo si el valor de la izquierda es falso.

El tipo de operando para el operador ! debe ser booleano, sino se produce un error de tiempo de ejecución y el resultado de aplicar el operador ! a una expresión es booleano.

La tabla 5.8 presenta información adicional sobre las expresiones lógicas.

Operación	Operador	Aridad	Prioridad
or	o	binario	2
and	y	binario	2
not	!	unario	1

Tabla 5.8 Expresiones Lógicas

5.4.6 GRAMÁTICA COMPLETA

PROGRAMA:= LISTAPROCEDIMIENTOS
LISTAPROCEDIMIENTOS:= LISTAPROCEDIMIENTOS PROCEDIMIENTO|
PROCEDIMIENTO:= (procedimiento|pr) ID ‘(’LISTAARGUMENTOS ’)’ BLOQUE
PROCEDIMIENTO:= (procedimiento|pr) ID ‘(‘ ’)’ BLOQUE
LISTAARGUMENTOS:= LISTAARGUMENTOS ’,’ ID | ID

BLOQUE:= inicio LISTAINSTRUCCIONES fin
LISTAINSTRUCCIONES:= LISTAINSTRUCCIONES INSTRUCCIÓN
INSTRUCCION:= INSTRUCCIONNORMAL EXPRESION
INSTRUCCION:= INSTRUCCIONSIMPLE
INSTRUCCION:= LLAMADA
INSTRUCCION:= BIFURCACION
INSTRUCCION:= CICLOREPETIR
INSTRUCCION:= CICLOPARA
INSTRUCCION:= CICLOMIENTRAS
INSTRUCCION:= ASIGNACION
INSTRUCCION:= retorno BOOLEANO1
ASIGNACION:= ID '=' BOOLEANO1
LLAMADA:= ID '(' LISTAPARAMETROS ')'
LLAMADA:= ID '(' ')'
LISTAPARAMETROS:= LISTAPARAMETROS ',' BOOLEANO1
LISTAPARAMETROS:= BOOLEANO1
CICLOREPETIR:= (repetir|re) EXPRESION BLOQUE
CICLOPARA:= para ID '=' EXPRESION hasta EXPRESION ',' incremento '=' EXPRESION BLOQUE
CICLOMIENTRAS:= mientras BOOLEANO1 BLOQUE
BIFURCACION:= si BOOLEANO1 BLOQUE ELSE
ELSE:= sino BLOQUE |
INSTRUCCIONNORMAL:= avanza | av | retrocede | rt |girarizquierda| gi | girarderecha | gd
INSTRUCCIONSIMPLE:= alto | al | bajarpluma | bp | subirpluma | sp | borrarpantalla | bo | escribe | es
BOOLEANO1:= BOOLEANO1 o BOOLEANO2
BOOLEANO1:= BOOLEANO2
BOOLEANO2:= BOOLEANO2 y CONDICIONAL1
BOOLEANO2:= CONDIONAL1
CONDICIONAL1:= CONDIONAL1 == CONDICIONAL2
CONDICIONAL1:= CONDIONAL1 ¡= CONDICIONAL2
CONDICIONAL1:= CONDICIONAL2
CONDICIONAL2:= CONDIONAL2 '<' EXPRESION
CONDICIONAL2:= CONDIONAL2 <= EXPRESION
CONDICIONAL2:= CONDIONAL2 '>' EXPRESION
CONDICIONAL2:= CONDIONAL2 >= EXPRESION
CONDICIONAL2:= EXPRESION
EXPRESION:= EXPRESION '+' TERMINO
| EXPRESION '-' TERMINO

| TERMINO
TERMINO:= TERMINO ‘*’ FACTOR
| TERMINO ‘/’ FACTOR
| FACTOR
FACTOR:= ‘(’ BOOLEANO1 ‘)’
FACTOR:= ‘-’ FACTOR
FACTOR:= NUMEROENTERO|NUMEROFLOTANTE
FACTOR:= ‘!’ FACTOR
FACTOR:= ID
FACTOR:= Falso | Vervadero
FACTOR:= LLAMADA

6. INTÉRPRETE

Un programa es leído por el Analizador Sintáctico, su entrada es un flujo de tokens generados por el Analizador Léxico para luego validar la sintaxis de estos tokens en la gramática especificada.

6.1 ANALIZADOR LÉXICO

El analizador léxico es la primera fase del intérprete. Su principal función consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis.

Esta etapa se implementó con la herramienta LEX de PLY, la cual es usada para dividir una cadena de entrada en tokens. Un token es una cadena de caracteres, que se clasifica de acuerdo a las reglas de una definición léxica, cada token posee un tipo para diferenciarlo de otros. En la herramienta LEX se definen cada uno de los tokens mediante una expresión regular y esta se encarga de separar el documento en cadenas que coinciden con la definición que se le da a cada tipo de token.

Para la construcción de la etapa léxica definimos los siguientes tokens con sus respectivas expresiones regulares:

numeroentero ::= [1 – 9] [0 – 9] * | “0”
numeroflotante ::= (([1 – 9] [0 – 9] *) + “.”[0 – 9] +) | “0” “."[0 – 9] +
Identificador ::= ([a - z] | [A – Z]) (([a - z] | [A – Z]) | [0 – 9] | “_”) *
Comentario ::= #([a - z] | [A - Z] | [0 - 9] | ”_” | .)*#

También se definieron tokens para cada una de las palabras reservadas y operadores binarios y unarios que se pueden ver en la especificación del lenguaje.

La primera definición indica que un número entero es una cadena que inicia con cualquier número del uno al nueve, seguido de cero o más repeticiones de los números del cero al nueve. O un número entero también puede ser un único cero.

La segunda definición indica que un número flotante es la ocurrencia de una o más repeticiones de un número del uno al nueve seguido de cero o más

ocurrencias de un número de cero a nueve. Seguido del carácter "." y una o más repeticiones de un número del cero al nueve. O un numero decimal es un cero seguido del carácter "." y una o más ocurrencias de un número del cero al nueve.

La herramienta LEX de PLY de acuerdo a la anterior definición, en el caso que la entrada tenga la cadena "3 10.5 54" la separa en los siguientes tokens: "3" "10.5" "54", y el tipo de cada uno seria respectivamente numeroentero, numeroflotante, numeroentero.

Cuando el elemento leído de la cadena de entrada no coincide con ninguna de las definiciones de los tokens al analizador léxico informa sobre este error. En la mayoría de los lenguajes de programación los usuarios tienden a cometer algunos errores muy comunes, por ejemplo intentar definir un identificador con un número como su primer carácter, por lo cual una forma de identificar estos errores es hacer definiciones de tokens que coincidan con estos errores y así presentar mensajes más específicos del error que pueda tener el usuario.

Se construyeron tokens para identificar errores en los números, identificadores y comentarios del lenguaje, las expresiones son las siguientes:

Numeromalformado ::= ((0+|[1-9]+)\.)|(\.([0-9]+))|0+[0-9]+

Identificadormalformado::=([a-zA-Z_]\w*(\.\w*)+|\d+\.\d+[a-zA-Z_]\w*|[a-zA-Z_]\w*\.|\.[a-zA-Z_]\w*|\d+[a-zA-Z_]\w*)|_\w*

Comentario malformado ::= \#[\w\W]*?(?<!\#)|\#

6.2 ANALIZADOR SINTÁCTICO

La etapa del análisis sintáctico es la encargada de revisar que los tokens proporcionados por el analizador léxico, estén ubicados y agrupados de acuerdo a la definición de la gramática del lenguaje. Dicho de otra manera, que los tokens pertenezcan a frases gramaticales validas, que el intérprete utiliza para sintetizar la salida.

La herramienta utilizada para la etapa sintáctica fue YACC, en esta herramienta la sintaxis es usualmente especificada en términos de una gramática de la forma BNF, en este tipo de gramáticas se distingue los símbolos terminales de los no terminales ya que los primeros son declarados en mayúscula y los segundos en minúscula, además los primeros son símbolos definidos en la etapa léxica. Cada

regla de producción de la gramática tiene asociado un comportamiento semántico y es a menudo especificado usando una técnica conocida como traducción de sintaxis dirigida. En la traducción de sintaxis dirigida los atributos son conectados a cada símbolo de la gramática con una acción. Siempre que una producción de la gramática es reconocida, la acción describe que hacer, por ejemplo dada la gramática de la tabla 6.1, es posible escribir la especificación de una calculadora simple.

Gramática	Acción
expresion: expresion + term	expresion.val = expresion.val + term.val
expresion: expresion – term	expresion.val = expresion.val – term.val
expresion: term	expresion.val = term.val
term = term1 * factor	term.val = term.val * factor.val
term = term1 / factor	term.val = term.val / factor.val
term = factor	term.val = factor.val
factor = NUMBER	factor.val = int(NUMBER.lexval)
factor = (expresion)	factor.val = expresion.val

Tabla 6.1: Especificación de una calculadora simple

Una buena manera de pensar sobre la traducción de sintaxis dirigida es ver cada símbolo en la gramática como un tipo de objeto. Asociado a cada símbolo hay un valor que representa su "estado" (por ejemplo, el atributo val de las acciones anteriormente descritas). Las acciones semánticas se expresan entonces como una colección de funciones o métodos que operan sobre los símbolos y valores asociados.

La herramienta yacc utiliza el análisis sintáctico LR o análisis por desplazamiento y reducción. El análisis sintáctico LR es una técnica ascendente (bottom up) que trata de reconocer el lado derecho de las reglas de producciones de una gramática. Siempre que en el lado derecho de la producción se encuentre una entrada valida, la acción apropiada para el análisis es disparada y los símbolos de la gramática del lado derecho son reemplazados por el símbolo no terminal de la gramática en el lado izquierdo.

6.3 ÁRBOL DE SINTAXIS ABSTRACTA (AST)

Una de las actividades que se realiza en la etapa sintáctica, además de verificar la sintaxis del programa es construir un AST (árbol de sintaxis abstracta) con cada una de las acciones que se activan cuando la herramienta YACC realiza el análisis del programa fuente. Un AST es una representación en forma de árbol de la

estructura sintáctica abstracta (simplificada) del código fuente escrito en el lenguaje de programación de alto nivel. Cada nodo del árbol denota una construcción que ocurre en el código fuente. La sintaxis es abstracta en el sentido que no representa cada detalle que aparezca en la sintaxis verdadera. Por ejemplo, el agrupamiento de los paréntesis está implícito en la estructura arborescente, y una construcción sintáctica tal como la instrucción (si condición) puede ser denotada por un solo nodo con dos ramas, cada procedimiento tiene su propio AST. Veamos un ejemplo:

```
Pr cuadro (lado)
Inicio
        Repetir 4
        Inicio
                Av lado
                Gi 90
        fin
fin
```

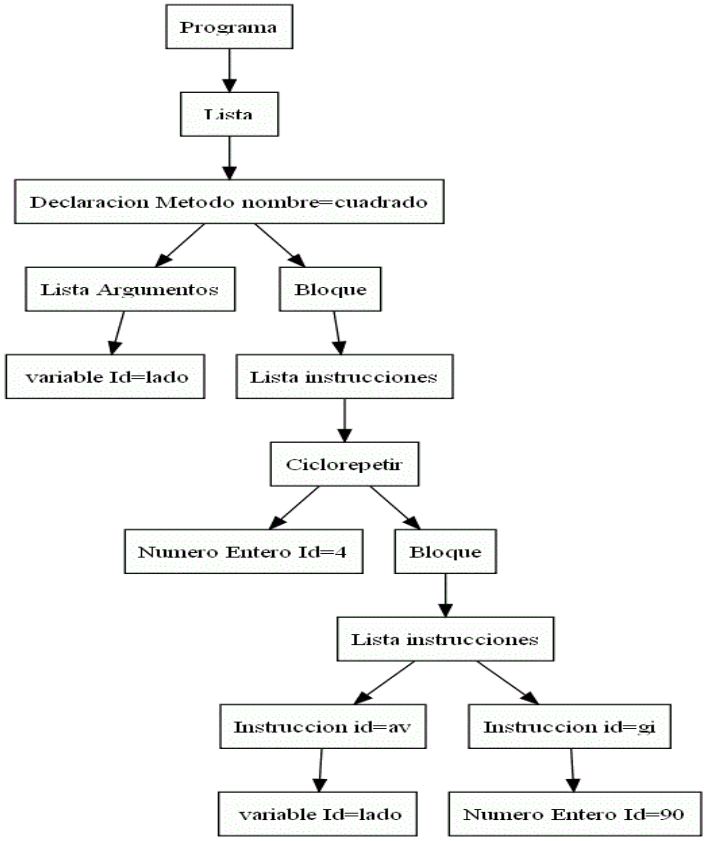


Imagen 6.1: Ejemplo AST

Como se puede ver en la imagen 6.1 cada nodo del árbol representa una construcción del código fuente, el procedimiento cuadro es representado por un nodo, del cual salen dos nodos más que representan los argumentos y el cuerpo del procedimiento o bloque. El cuerpo del procedimiento posee una lista de instrucciones, las cuales se leen de izquierda a derecha, la primera instrucción del procedimiento es el ciclo repetir, este nodo tiene a su vez otros dos nodos, los cuales representan la cantidad de repeticiones (nodo izquierdo) y el bloque del ciclo (lado derecho), el bloque del ciclo contiene primero la instrucción avanzar (av) que a su vez contiene la expresión lado, indicando que se avanzara lado unidades, la segunda instrucción del bloque es girar izquierda (gi) y esta contiene la expresión 90, que significa que gira a la izquierda 90 grados.

De una manera similar para cada código fuente en esta etapa se construye su AST, el cual es de vital importancia para las etapas posteriores de ejecución y generación de código intermedio. En la construcción de compiladores se incluye una etapa adicional de análisis semántico en la cual se hace chequeo de tipos sobre las expresiones de acuerdo al tipo de cada variable en su declaración, en lenguajes interpretados o de tipado dinámico solo es posible conocer el tipo de cada variable con total certeza solo durante la etapa de ejecución.

6.4 ÁMBITO Y MANEJO DE VARIABLES

Cuando se ejecuta un procedimiento se crea una estructura llamada ámbito que almacena las variables como pares clave – valor, donde la clave hace referencia al nombre de la variable y el valor hace referencia a otra estructura que contiene el tipo y valor de la variable. Cuando un ámbito es creado es introducido a una estructura FIFO, esta estructura contendrá en su cima el ámbito que pertenece al procedimiento actual en ejecución y cuando hay una llamada a un procedimiento se crea un nuevo ámbito y se introduce a la pila, cuando termina la ejecución de ese procedimiento se saca ese ámbito de la pila y se continua con el ámbito que se encontraba antes en la estructura.

A continuación se presentara un ejemplo de la ejecución de una expresión. Consideremos el siguiente código del lenguaje de alto nivel y su respectivo AST que se observa en la imagen 6.2.

```
pr expresion (lado)
inicio
        a = lado +20*10
fin
```

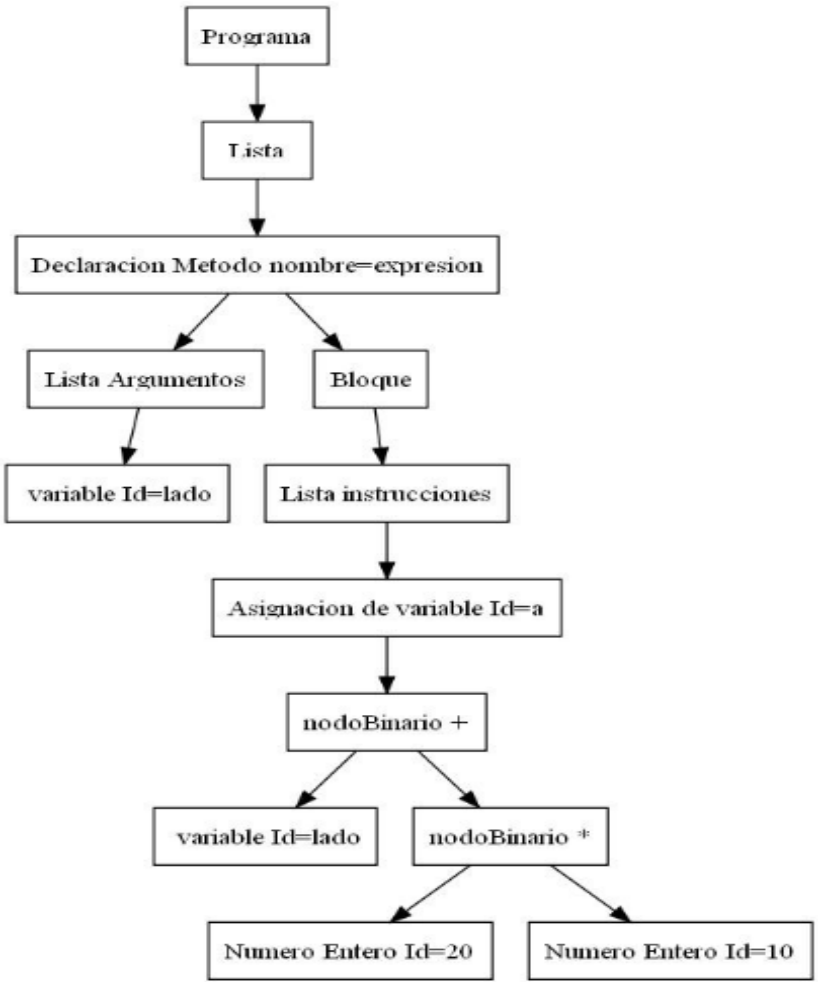


Imagen 6.2: Ejemplo ejecución AST

Cuando se ejecuta un procedimiento, si el procedimiento posee parámetros, sus valores de llamada son asignados a cada variable en la lista de argumentos del nodo declaración de método expresión, después se continua con la ejecución del nodo que representa el bloque del procedimiento, este bloque es una lista de instrucciones del programa fuente, este nodo contiene la instrucción de asignación a = lado + 20*10, el nodo de asignación tiene el nodo + como hijo, los hijos del nodo + son la variable lado y el nodo *, los hijos del nodo * son la constante 20 y la constante 10, el AST empieza la ejecución en los nodos hojas por lo tanto primero se realiza la ejecución del nodo izquierdo del nodo binario +, este nodo se ejecuta simplemente retornando el valor de la variable lado, luego se ejecuta el lado derecho que corresponde al nodo * ,este nodo ejecuta su parte izquierda y luego su parte derecha, el retorno de cada lado lo multiplica y lo retorna al nodo padre + que a su vez realiza la operación + entre el retorno del lado izquierdo y el derecho y retorna el valor al nodo de asignación donde finalmente se crea la variable a en el ámbito actual y se le asigna el valor retornado por su hijo.

6.5 ANÁLISIS EN TIEMPO DE EJECUCIÓN

Entre las características del lenguaje se encuentra la de no poseer declaración de tipos y por el contario tener asignación dinámica de los tipos del lenguaje, por lo cual solo puede hacerse chequeo de tipos durante la ejecución.

El código fuente está representado en el AST generado en la etapa sintáctica y puede decirse que el AST es una representación intermedia del código fuente, la ejecución del programa fuente se hace a través del AST, con el uso del patrón visitante se hace un recorrido en profundidad del AST, tomando ciertas consideraciones y decisiones en algunos nodos visitados, en cada nodo visitado se ejecuta una función que hace lo que se encuentra en la especificación del lenguaje.

En las instrucciones que poseen expresiones se verifica la compatibilidad de los tipos de los operandos según sus operadores. Por ejemplo como se mencionó anteriormente en la especificación del lenguaje no se puede aplicar una operación aritmética a un número y un booleano, si tuviéramos el siguiente código en la herramienta:

```
pr expresion (lado)
inicio
        a = lado +20*verdadero
fin
```

Se Obtendría el AST de la imagen 6.3.

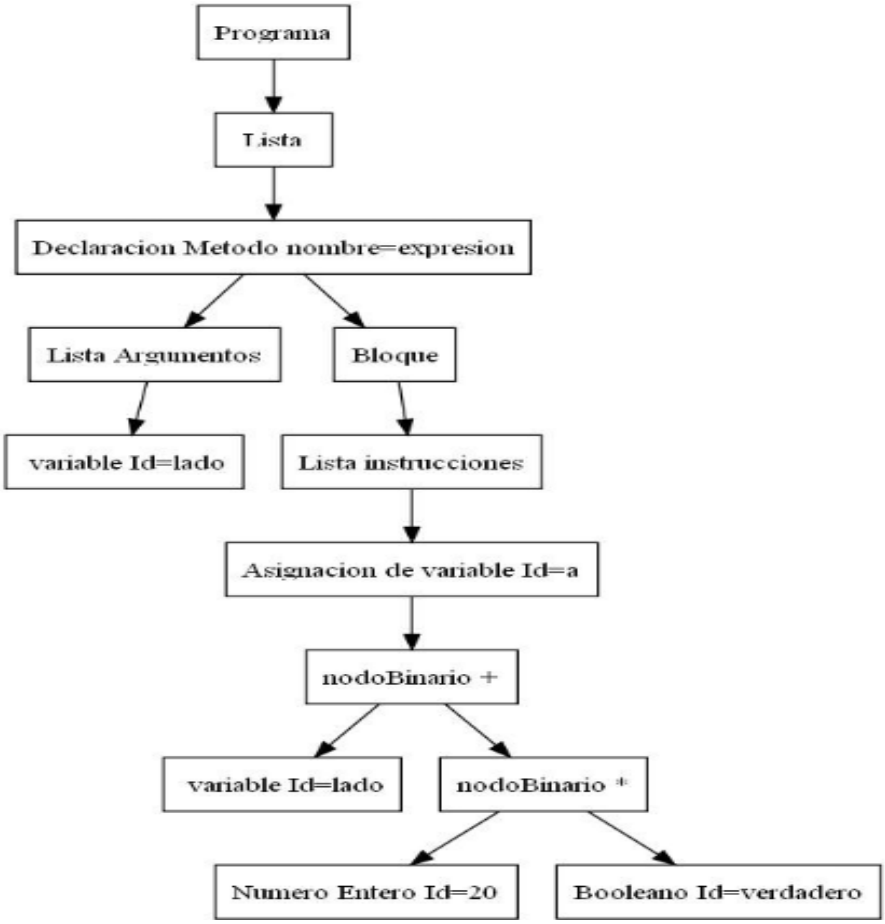


Imagen 6.3: Ejemplo AST en el análisis en tiempo de ejecución

La ejecución seguiría un proceso parecido al anterior ejemplo, pero cuando llegue a la ejecución del nodo binario * observara que los tipos de sus nodos hijos son incompatibles, finalizara la ejecución y mostrara el error Incompatibilidad de tipos entre la constante de tipo entera y el booleano. También se sigue un proceso similar para evaluar si se presentan en el código fuente los demás errores descritos en la especificación del lenguaje.

6.6 EJECUCIÓN DE ESTRUCTURAS DE CONTROL

6.6.1 CICLOS

El proceso se explicara con el siguiente código que tiene la estructura de control mientras:

```
pr Ejecucion_mientras (lado)
inicio
```

```
        i = 0
        mientras (i < lado)
        inicio
                av i
                i = i + 1
        fin
fin
```

El AST del código se muestra en la imagen 6.4.

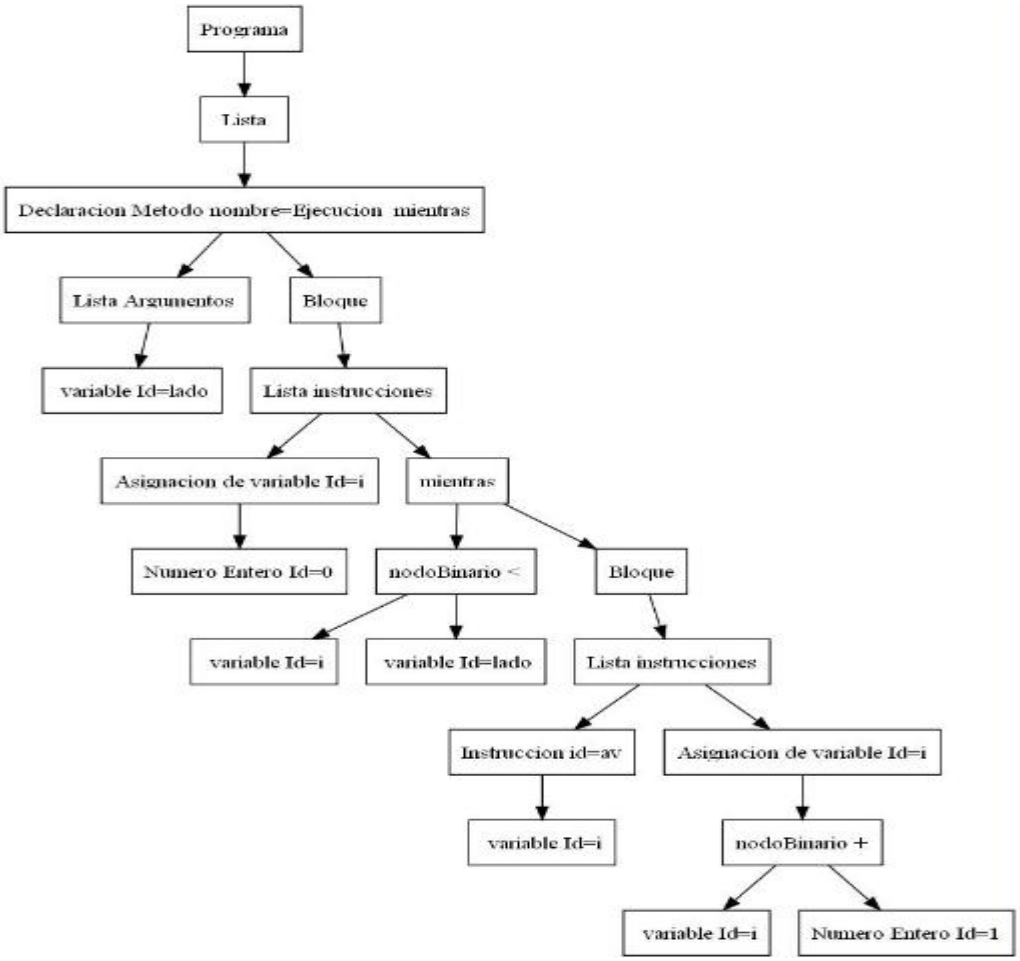


Imagen 6.4 Ejemplo de AST con estructura de control mientras

La ejecución del árbol es similar a lo descrito en ejemplos anteriores hasta el nodo mientras, en este, primero se ejecuta la expresión asociada a la condición, y después se ejecuta el nodo bloque mientras la expresión retorne un valor verdadero.

6.6.2 DECISIÓN

El proceso se explicara con el siguiente código que tiene la estructura de control si:

```
pr Ejecucion_si (lado)
inicio
        si lado < 50
        inicio
                av 10
        fin
        sino
        inicio
                av 20
        fin
fin
```

El AST del código se muestra en la imagen 6.5.

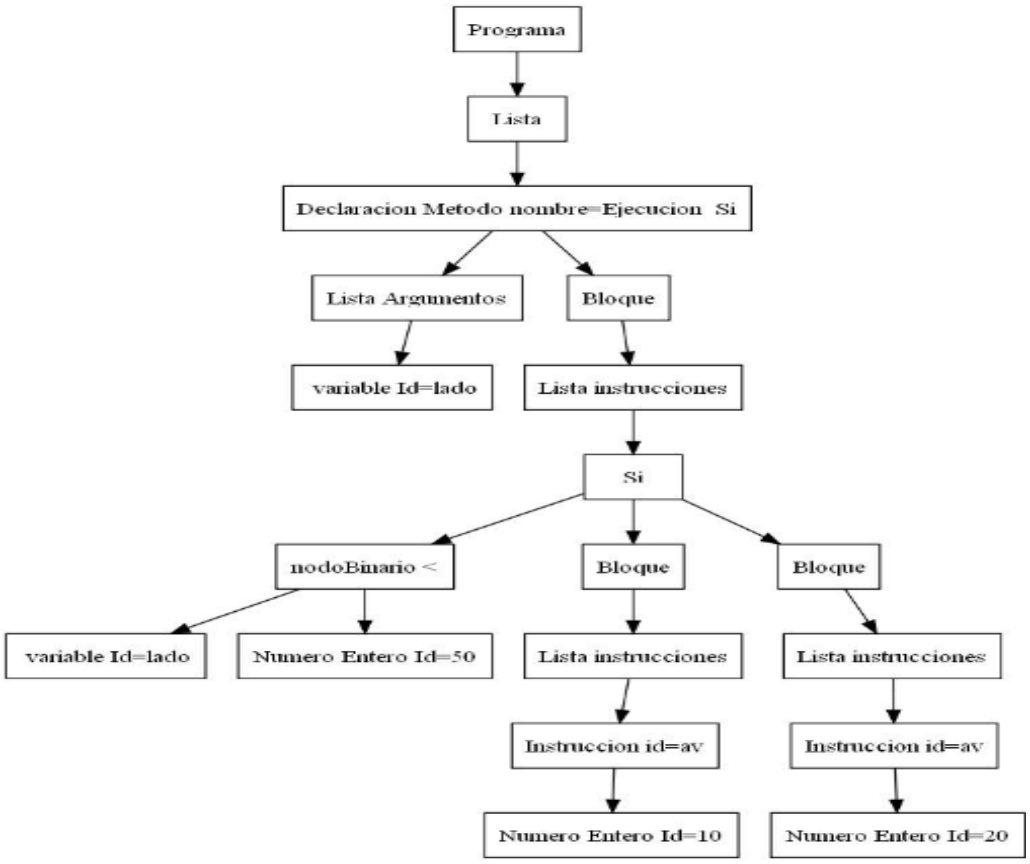


Imagen 6.5 Ejemplo de AST con estructura de control si

La ejecución del árbol es similar a lo descrito en ejemplos anteriores hasta el nodo si, en este, primero se ejecuta la expresión asociada a la condición, si la expresión retorna verdadero se ejecuta el nodo del bloque verdadero, de lo contrario se ejecuta el nodo del bloque falso.

6.7 GENERADOR DE CÓDIGO PARA LA MÁQUINA VIRTUAL

En esta sección se describirá como generar código para la máquina virtual que corre sobre la FPGA, las especificaciones de la máquina virtual se pueden observar en el capítulo de descripción del robot. Para realizar la generación del seudocódigo se parte de la representación intermedia generada en la etapa sintáctica, el AST.

La máquina virtual es un intérprete orientado a una máquina de pila, la cual contiene una pila para computar las expresiones, una pila de datos para guardar las variables y una pila de retornos para almacenar direcciones de instrucciones.

6.7.1 GENERACIÓN DE CÓDIGO DE EXPRESIONES

Para explicar la generación de las expresiones utilizaremos el siguiente código fuente:

```
pr expresion ()
inicio
        i = 12
        a = 12 * 323 + i
fin
```

El AST del código descrito anteriormente se muestra en la imagen 6.6:

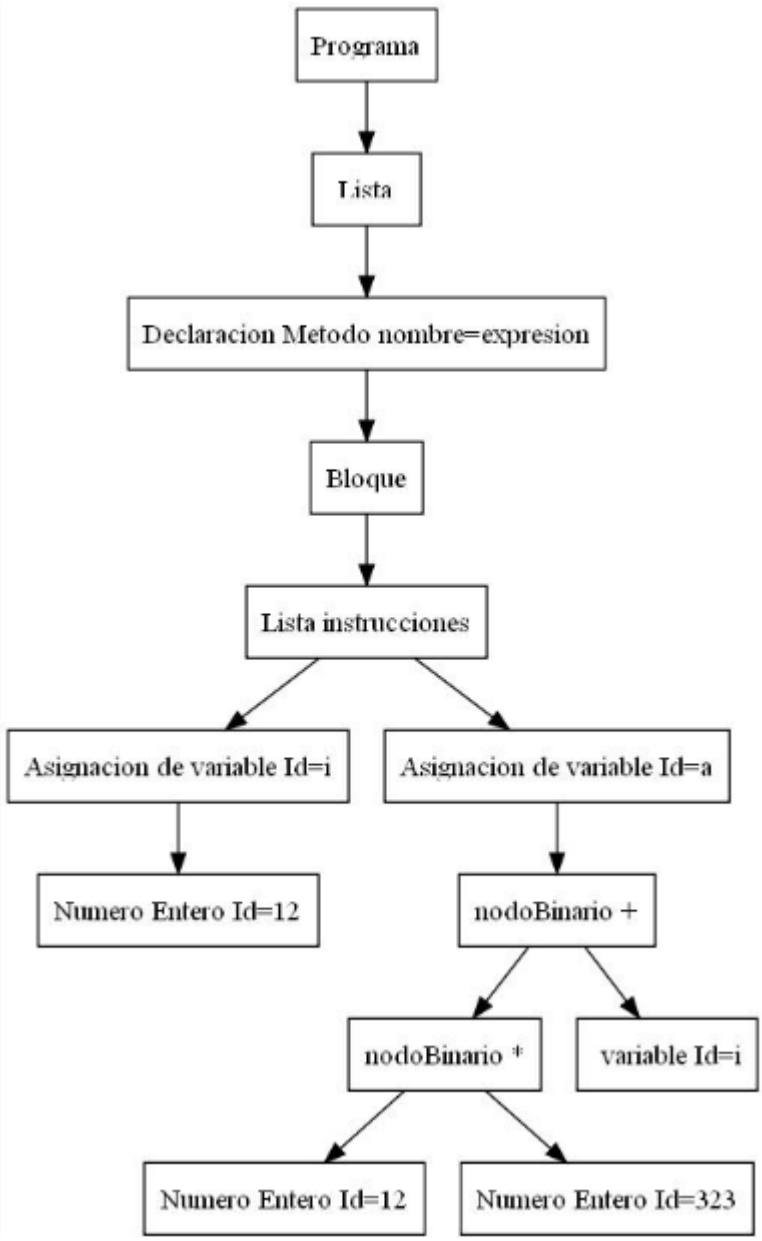


Imagen 6.6 Ejemplo de AST

Se realiza un recorrido en profundidad, cuando se llega a un nodo de declaración de procedimiento se genera su código agregando una etiqueta con el nombre del método (expresion), después sigue la generación de código para el bloque del método, las instrucciones del bloque están dispuestas de izquierda a derecha siendo la instrucción que se encuentra más a la izquierda la primera instrucción. Cuando llega al nodo de asignación de variable primero se genera código para la expresión de la asignación, todos los nodos de operadores binarios y unarios generan código para cada uno de sus operandos y luego para ellos mismos, en nuestro caso cuando se visita el nodo binario + primero se visita sus dos hijos: el

nodo binario * y nodo variable i, para el nodo binario * se genera código para cada uno de sus hijos, sus hijos son los nodos numero entero 12 y numero entero 323, la generación de código para estos nodos es pushe elemento (pushe 12), luego se genera código para el nodo binario * añadiendo la instrucción operador (*). Con lo anterior se generaría código para el nodo izquierdo del nodo binario +, para generar código de su otro hijo que es un nodo variable nombre_variable (variable i) se debe colocar la instrucción pushv nombre_variable (pushv i), luego se generaría código para el nodo binario + añadiendo la etiqueta operador (+). Cuando se termina de visitar todos los nodos que corresponden a la expresión se genera código para la asignación así: = nombre variable (= a), finalmente se agrega la etiqueta finfuncion para indicar la terminación del código. Según lo anterior la generación de código del AST anterior queda:

```
1       expresion:
2       pushe 12
3       pushe 323
4       *
5       pushe 212
6       +
7       = a
8       finfuncion
```

6.7.2 GENERACIÓN DE CÓDIGO DE ESTRUCTURAS DE CONTROL

6.7.2.1 CICLOS

Para explicar la generación de código de la estructura mientras se utilizara el código fuente y AST de la imagen 6.4

Se sigue un proceso similar al explicado anteriormente hasta llegar al nodo del mientras, pero como el procedimiento contiene el parámetro lado según las especificaciones de la máquina virtual se debe agregar una etiqueta = parámetro por cada parámetro que reciba la función. En el nodo mientras se genera código agregando la etiqueta mientras1, luego se genera código para la expresión de condición como se vio anteriormente y a continuación se agrega la instrucción jumpfalse finminetras1, después se genera código para el bloque de instrucciones del mientras y al final del bloque se agrega la instrucción goto mientras1, después se agrega la etiqueta finmientras1, la cual indica el fin del bloque que pertenece al mientras1 y finalmente se agrega la etiqueta finfuncion. Por lo anterior la generación del código queda:

```
1       Ejecucion_mientras
2        = lado
3       pushe 0
4       = i
5       mientras1
6       pushv i
7       pushv lado
8       <
9       jumpfalse finmientras1
10      pushv i
11      av
12      pushv i
13      pushe 1
14      +
15      goto mientras1
16      finmientras1
17      finfuncion
```

6.7.2.2 DECISIÓN

Para explicar la generación de código de la estructura si se utilizara el código fuente y AST de la imagen 6.5

Se sigue un proceso similar al explicado anteriormente hasta llegar al nodo del si. En el nodo si primero se genera código para la expresión de la condición como se hizo en ejemplos pasados, luego se agrega una etiqueta jumpfalse sino1, después se genera código para el bloque de código verdadero, después se agrega la instrucción goto finsi1, se agrega la etiqueta sino1, después se genera código para el bloque falso, después se agrega la etiqueta finsi1 y finfuncion. Por lo anterior la generación del código queda:

```
1       Ejecucion_si
2        = lado
3       pushv lado
4       pushe 50
5       <
6       jumpfalse sino1
7       pushe 10
8       av
9       goto finsi1
```

```
10      sino1
11      pushe 20
12      av
13      finsi1
14      finfuncion
```

6.7.3 GENERACIÓN DE CÓDIGO PARA LLAMADOS DE FUNCIONES

Para generar código para llamados se debe primero generar código para los parámetros si existen, cada parámetro es una expresión y se tratan como se vio anteriormente, luego se agrega la instrucción call funcion (call expresion), donde función es el nombre de la función.

6.8 COMUNICACIÓN

Como se mencionó anteriormente en el subsistema intérprete se debe implementar un componente que se encargue de enviar el seudocódigo equivalente al código programado al servidor que se encuentra en la FPGA, este componente se llama conexión cliente y es un programa cliente que utiliza sockets.

En la imagen 6.7 se presenta el diagrama de flujo de datos entre el cliente y servidor.

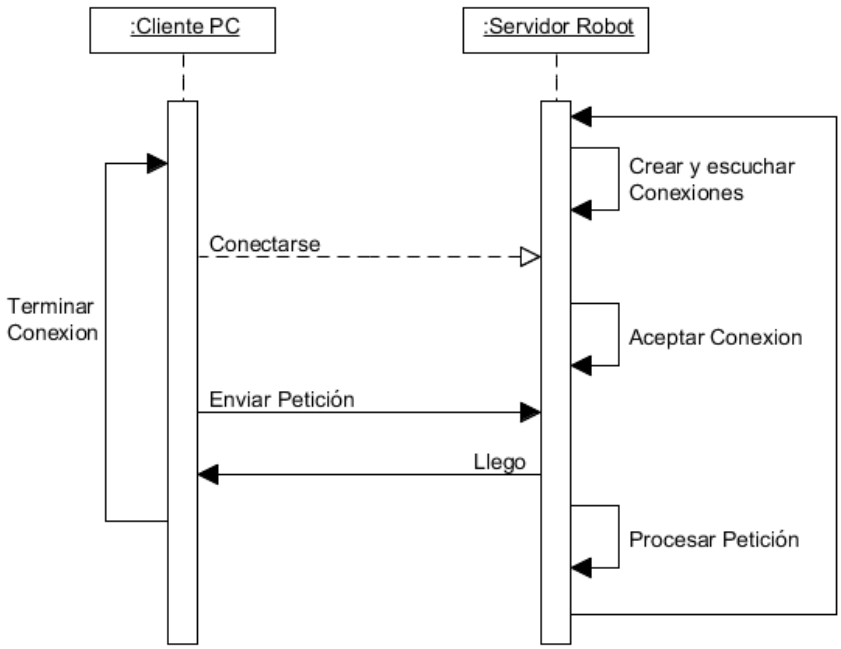


Imagen 6.7 Comunicación PC – Robot

7. SIMULACIÓN 2D

El objetivo de la simulación 2D es representar gráficamente los movimientos del robot y presentarlos en pantalla para que el usuario si no tiene el prototipo físico, pueda observar el comportamiento del robot de acuerdo al programa realizado.

El área sobre la cual se mueve el robot en la simulación es representado a través de un plano cartesiano que tiene origen (0,0) en su esquina superior izquierda y aumenta positivamente en el eje x hacia la derecha y en el eje y hacia abajo, su unidad de medida es en pixeles. El robot se encuentra inicialmente en el plano en la posición central del eje x y el eje y con orientación hacia el este (0 grados). Las instrucciones avanzar y retroceder mueven el robot hacia adelante y hacia atrás respectivamente según su orientación, para establecer la posición final en el movimiento del robot se calculan sus coordenadas finales teniendo en cuenta el triángulo rectángulo que se forma entre la posición inicial y final del robot. En este triángulo la hipotenusa es la magnitud de la distancia a recorrer y el ángulo formado entre el eje x y la hipotenusa es el ángulo de orientación, utilizando geometría básica se calculan los lados del triángulo y se le suman a las coordenadas iniciales. Ver imagen 7.1

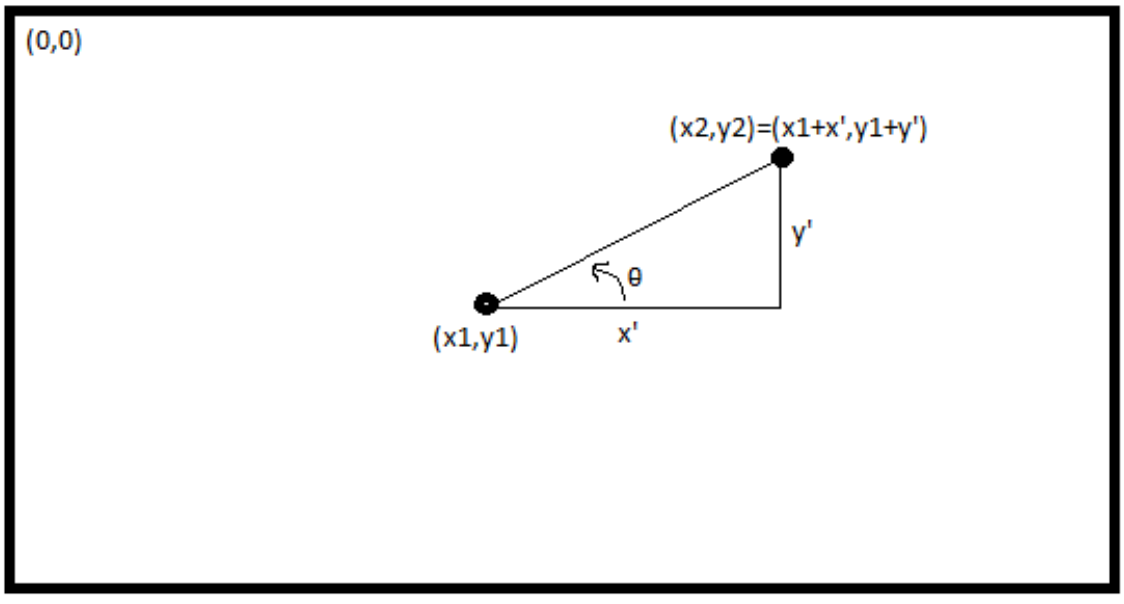


Imagen 7.1 Calculo de posición final del movimiento del robot

Sobre la simulación se pueden hacer configuraciones del tamaño y color del lápiz, además del color de fondo.

Para la implementación se utilizó la librería wxPython, el cual es un kit de herramientas multiplataforma para la creación de aplicaciones gráficas. El autor principal de wxPython es Robin Dunn. Con los desarrolladores de wxPython se pueden crear aplicaciones wxPython en Windows, Mac y en varios sistemas Unix.

8. ROBOT

8.1. INTRODUCCIÓN

Uno de los objetivos del proyecto se refiere a la construcción de un prototipo de robot que pueda ejecutar un código programado por un usuario. Para la construcción del prototipo se usa una FPGA Spartan 3E en donde se describe un procesador llamado microblaze, este procesador esta descrito en VHDL y controla los periféricos, entre estos periféricos están los motores que hacen que el robot se desplace y gire, también posee un conjunto de dispositivos de entrada-salida adheridos al procesador como lo es una interfaz de puerto Ethernet para recibir la información enviada del computador al robot.

8.2. ESTRUCTURA FÍSICA

Imagen 8.1 Vista superior de la estructura final.

Imagen 8.2 Vista frontal de la estructura final

Como se observa en la imagen 8.2 hay tres leds de colores verde, amarillo y rojo que representan respectivamente: robot encendido, robot programado y pluma. El led de pluma representa el dispositivo mecánico que bajara el elemento que pintara la superficie, no se implementó físicamente debido al alcance del proyecto, pero se deja planteado para que otro grupo de estudiantes lo realice como parte de otro trabajo de grado.

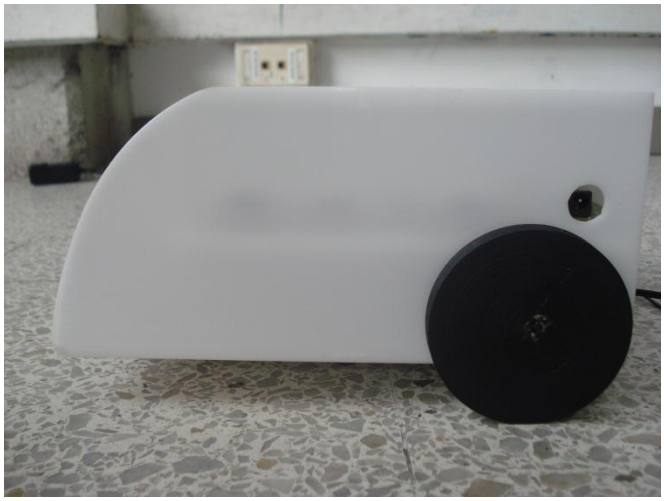


Imagen 8.3 Vista lateral de la estructura final

Como se observa en la imagen 8.3 la vista cuenta con una rueda de 4 centímetros de radio, también tiene un orificio que permite alimentar la FPGA.

Imagen 8.4 Vista trasera de la estructura final

Como se observa en la imagen 8.4 en la parte superior de la vista hay dos orificios utilizados para transmitir datos a la FPGA a través del puerto Ethernet y un puerto USB. En la parte inferior hay otro orificio para la alimentación del circuito de los motores.

Imagen 8.5 Vista superior sin la tapa

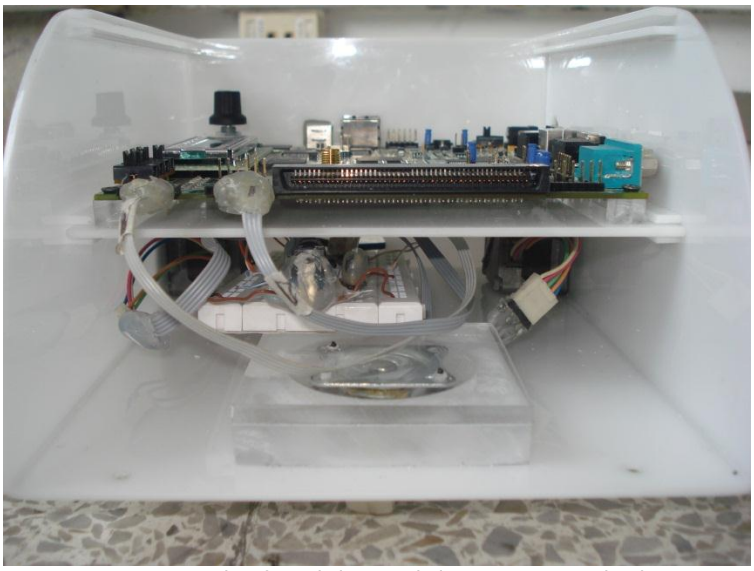


Imagen 8.6 Vista frontal sin tapa de la estructura terminada

Como se observa en la imagen 8.6 la estructura contiene una plataforma en la parte superior para soportar la FPGA, y en la parte inferior están los motores y su respectivo circuito.

8.3. CIRCUITO ELÉCTRICO

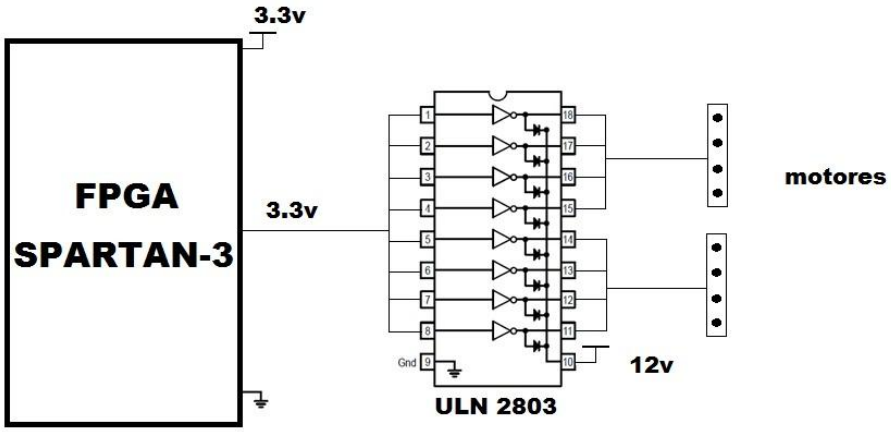


Imagen 8.7 Circuito eléctrico ULN 2803

Como se observa en la imagen 8.7 la FPGA tiene una salida de 8 bits que se conecta a los pines uno a nueve del ULN 2803, esta señal de 8 bits es generada por el driver implementado en vhdl en la FPGA encargado de especificar las señales necesarias para alimentar las bobinas de los motores paso a paso para su correcto movimiento. Los pines once a diecinueve de este integrado representan las salidas de las respectivas entradas, las cuales alimentan los dos motores.

Debido a que la señal eléctrica generada por la FPGA es de baja intensidad y no alcanza los umbrales necesarios para que los motores se muevan se debe utilizar el ULN 2803, este integrado es una interface que acondiciona las señales de entrada, aumentando su tensión y corriente.

8.4. FPGA

Para el desarrollo del proyecto se trabajó con una placa de desarrollo FPGA Spartan 3E. Tiene como núcleo el dispositivo XC3S500E que incluye 20 multiplicadores dedicados, 4 DCM, 20 BRAM y quinientas mil puertas equivalentes, lo que permite diseñar y testear aplicaciones con un alto grado de procesado.

La placa dispone de un gran número de interfaces, tanto análogas (convertidor ADC 14 bits y DAC de 10 bits) como digitales y recursos de almacenamiento.
Características

- Xilinx Spartan 3E (dispositivo de puerta de 500 K),
- Programador JTAG integrado
- 50 MHz a bordo de reloj,
- Flash de 128Mbit,
- 16Mbit SPI flash,
- 64Mbyte DDR SDRAM,
- 4 canales de CAD de 12 bits
- ADC de 14 bits de canal dual.
- 20 por 2 LCD
- Codificador rotatorio
- 8 conmutadores
- 8 LEDs
- Conector Ethernet
- VGA conector
- dos conectores de RS232

8.5. PROCESADOR

El procesador MicroBlaze es un procesador RISC (Set de Componentes de Instrucciones Reducidas) de 32 bits, desarrollado por Xilinx. El mismo es compatible con las familias Spartan y Virtex. MicroBlaze cuenta con una arquitectura Harvard con buses de 32 bits separados para acceso a datos e

instrucciones y permite el acceso a estos recursos a través de memoria cache. Actualmente cuenta con interfaces para los buses OPB, LMB y PLB. También permite ejecutar cualquier programa escrito en ANSI C y manipular los periféricos desde este.[1]

8.5.1 REPERTORIO DE INSTRUCCIONES

La restricción de espacio, fuerza a un diseño sencillo, que encaja perfectamente con la idea de las arquitecturas tipo RISC (Reduced Instruction Set Computer), donde el limitado número de instrucciones permite simplificar la unidad de decodificación. En el caso de MicroBlaze, el número total de instrucciones que soporta son 87, si se consideran diferentes las instrucciones que operan con valores inmediatos de aquellas que realizan la misma operación con registros. Adicionalmente, cada instrucción se ha elegido para que el tamaño de la ALU también sea reducido. Las instrucciones que requieran un procesamiento complejo habrán de realizarse en un hardware específico, diseñado utilizando los restantes recursos de la FPGA.

Instrucciones de 32 Bits:

- Tipo A: 2 Registros Fuente, 1 Registro Destino.
- Tipo B: 1 Registro Fuente, 1 Operando inmediato de 16 Bits extensible a 32, 1 Registro Destino.

Categorías de las Instrucciones:

- Aritmética
- Lógica
- Salto
- Carga-Almacenamiento
- Instrucciones especiales.

8.5.2 PIPELINE

Una arquitectura RISC aumenta fácilmente su rendimiento por medio de la segmentación (Pipelining). En el caso de MicroBlaze, el número de etapas de pipeline es 3 (Fetch, Decode y Execute), ejecutando una instrucción por ciclo de reloj. Como contrapartida, la segmentación debe incorporar mecanismos para evitar los problemas relacionados con los saltos de programa. En un salto, la pila está llena de instrucciones que no corresponden con el flujo de ejecución. Estos

[1] http://www.microcontroladorespic.com/tutoriales/FPGAs/Procesador-MicroBlaze.html

riesgos están tratados por hardware en MicroBlaze: cada vez que se produce un salto, el pipeline se vacía cuando el salto se hace efectivo.

8.5.3 REGISTROS INTERNOS Y CACHE.

Las FPGAs actuales disponen de memoria distribuida con un tiempo de acceso corto, cuando son utilizadas por la lógica cercana. Este tipo de memoria es utilizada por MicroBlaze para materializar sus 32 registros internos, el contador de programa y el registro de estado. Este último sólo contiene el bit de acarreo, habilitación de cachés, indicador de estado de parada (break), error en el FSL y bit de excepción producida por división por cero. Además de estos registros internos, MicroBlaze utiliza un buffer de 16 instrucciones mapeado en los registros de desplazamiento SRL de los slices. Este recurso es fundamental para un buen rendimiento del procesador.

La utilización de cachés es una práctica habitual en arquitecturas modernas; el diseño de MicroBlaze no es una excepción. Pero en este caso, la utilización y configuración de los tamaños de caché pueden ser fijados por el usuario. Para ello, existe el siguiente conjunto de opciones a la hora de configurar el procesador: habilitación de caché de instrucciones y/o datos, tamaño, rango de direcciones y tamaño de palabra, aunque esta última opción solo es válida para la caché de datos. Opciones no configurables por el usuario son tamaño de los bloques de caché, la política de sustitución de bloques o el grado de asociatividad de la caché. Por último se debe destacar que las caches son de tipo asociativo.

8.5.4 BUSES DEL SISTEMA

MicroBlaze sigue el modelo de arquitectura Harvard, donde datos e instrucciones son almacenados en memorias diferentes. De nuevo, gracias a la capacidad de reconfiguración de las FPGAs, es posible configurar el sistema con diferentes opciones sobre los buses, pudiéndose reducir de este modo el tamaño final del sistema. MicroBlaze utiliza el estándar CoreConnect creado por IBM, para conectar diferentes elementos en un circuito integrado. Un aspecto interesante es que CoreConnect permite reducir la carga capacitiva del bus, repartiéndola entre varios buses. Así, se consiguen mayores rendimientos, dado que los retardos de pistas globales son muy importantes en FPGAs. El estándar define tres tipos de buses, cada uno con unas características de velocidad y conectividad:

LMB: (Local Memory Bus): Bus síncrono de alta velocidad, utilizado para conectar periféricos y los bloques de memoria interna de la FPGA. Solo admite un maestro

en su implementación para MicroBlaze y puede ser utilizado tanto para instrucciones como para datos. Este bus es compatible con el PLB (Processor Local Bus) incluido en el estándar CoreConnect, pero a diferencia de éste, el LMB no admite varios maestros ni tamaños de palabra diferentes a 32 bits.
OPB: (On-chip Peripheral Bus): Bus síncrono utilizado para conectar periféricos con tiempos de acceso variables. Soporta varios maestros y la conectividad de muchos periféricos es sencilla gracias a su identificación por multiplexación distribuida. Soporta transferencias de tamaño de palabra dinámico.
DCR: (Device Control Register): Bus síncrono diseñado para una conexión tipo anillo de periféricos con ancho de banda muy bajo. Solo soporta un maestro y está diseñado para minimizar el uso de lógica interna.

8.5.5 INTERRUPCIONES Y EXCEPCIONES

El procesador MicroBlaze contiene una línea de interrupciones, la cual al ser activada hace que el procesador ejecute una rutina de manejo de interrupciones que ha de ser especificada al compilador. Las excepciones se tratan de forma similar. Cuando una de ellas ocurre, se paraliza el procesamiento de instrucciones y se ejecuta una rutina de manejo de excepciones. En el caso de que el sistema necesite manejar más de una interrupción, será necesaria la utilización de un periférico específico (OPB Interrupt Controller), que se encarga de multiplexar e identificar las diferentes fuentes de interrupción.

8.6 MOTORES

Un motor paso a paso es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa es que es capaz de avanzar una serie de grados (paso) dependiendo de sus entradas de control. El motor paso a paso se comporta de la misma manera que un conversor digital-analógico y puede ser gobernado por impulsos procedentes de sistemas lógicos.

Este motor presenta las ventajas de tener alta precisión y repetitividad en cuanto al posicionamiento, lo cual los hacen ideales para los requerimientos del robot ya que se necesita mover el robot distancias muy precisas.

La aplicación maneja números con cinco decimales de precisión, su parte entera representan los cm. Debido a la naturaleza de los motores solo se puede trabajar con número de pasos enteros, entonces se convierte la parte decimal a números con un decimal de precisión y se hace la siguiente conversión:

0.1 = 1 paso
0.2 = 2 pasos
0.3 = 2 pasos
0.4 = 3 pasos
0.5 = 4 pasos
0.6 = 5 pasos
0.7 = 6 pasos
0.8 = 6 pasos
0.9 = 7 pasos

Cuando en la aplicación se avanza una unidad de distancia representa un centímetro en el prototipo físico y para hacer esto se debe mandar a ejecutar 8 pasos en cada motor.

8.6.1 FUNCIONAMIENTO

Básicamente los motores paso a paso están constituidos normalmente por un rotor que se observa en la imagen 8.8, sobre el que van aplicados distintos imanes permanentes y por un cierto número de bobinas excitadoras bobinadas en su estator.

La imagen 8.9 muestra las bobinas, que son parte del estator y el rotor es un imán permanente. Toda la conmutación (o excitación de las bobinas) deber ser externamente manejada por un controlador.

Imagen 8.8 Rotor

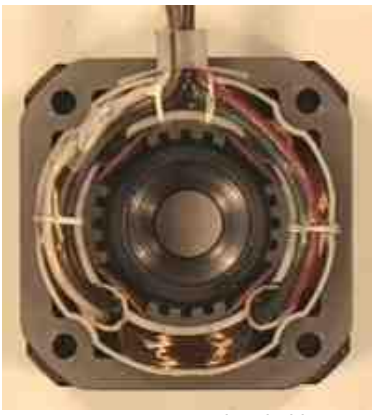


Imagen 8.9 Estator de 4 bobinas

8.6.2 SECUENCIA

Las secuencias más usadas generalmente recomendadas por el fabricante son la secuencia de paso completo que se observa en la tabla 8.1 y la de medio paso que se observa en la tabla 8.2. Con estas secuencias el motor avanza y se obtiene un alto torque de paso y de retención, la diferencia entre estos dos tipos de secuencia radica fundamentalmente en la precisión del paso. Para revertir el sentido de giro, simplemente se deben ejecutar las secuencias en modo inverso.

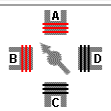
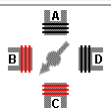
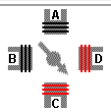
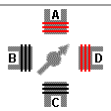
PASO	Bobina A	Bobina B	Bobina C	Bobina D	Imagen
1	ON	ON	OFF	OFF	
2	OFF	ON	ON	OFF	
3	OFF	OFF	ON	ON	
4	ON	OFF	OFF	ON	

Tabla 8.1: Paso completo

PASO	Bobina A	Bobina B	Bobina C	Bobina D	Imagen
1	ON	OFF	OFF	OFF	

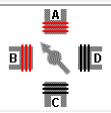
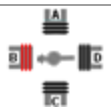
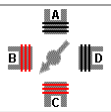
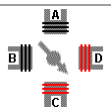
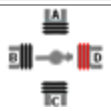
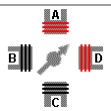
2	ON	ON	OFF	OFF	
3	OFF	ON	OFF	OFF	
4	OFF	ON	ON	OFF	
5	OFF	OFF	ON	OFF	
6	OFF	OFF	ON	ON	
7	OFF	OFF	OFF	ON	
8	ON	OFF	OFF	ON	

Tabla 8.2: Medio paso

8.6.3 MOTORES UNIPOLARES

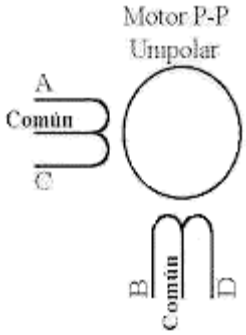


Imagen 8.10 Motor unipolar

La imagen 8.10 muestra la configuración interna de un motor unipolar.

Estos motores suelen tener 5 o 6 cables de salida, dependiendo de su conexión interna. Este tipo se caracteriza por ser más simple de controlar. En la figura 8.11 se aprecia un ejemplo de conexionado para controlar un motor paso a paso unipolar mediante el uso de un ULN2803, el cual es un arreglo de 8 transistores

tipo Darlington capaces de manejar cargas de hasta 500mA. Las entradas de activación (Activa A, B, C y D) pueden ser directamente activadas por una FPGA.

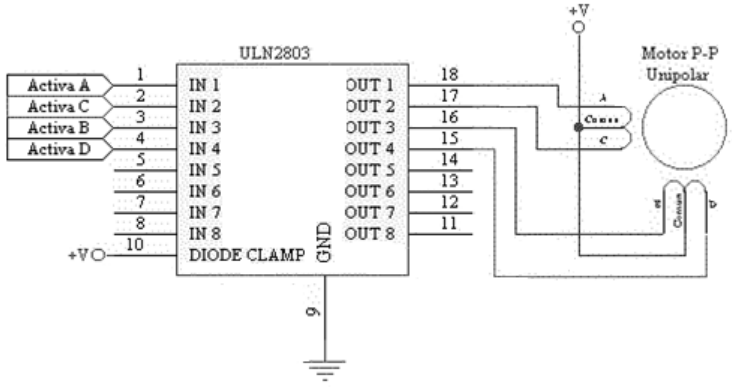


Imagen 8.11 Circuito motor unipolar

8.6.4 CONTROLADOR

Los motores paso a paso necesitan que en sus cables de entrada se pase una determinada secuencia de pulsos para energizar sus bobinas internas, entre cada secuencia de pasos debe existir un determinado retardo ya que las bobinas son dispositivos mecánicos que no pueden responder tan rápido como un dispositivo digital, es esta entonces una de las funciones principales del controlador, emitir la secuencia de pasos con un determinado retardo. El controlador debe ejecutar un determinado número de pasos que corresponden a una distancia específica que el robot debe moverse, por lo cual se especifica como una entrada del controlador. Los movimientos del robot pueden ser hacia adelante o en reversa, por lo cual debe existir una entrada que indique la dirección del movimiento. El controlador debe controlar un dispositivo de ejecución más lenta que la del procesador, por lo cual el controlador debe generar una interrupción cuando el dispositivo de entrada-salida (motores) termine su tarea y con esto el procesador pueda hacer otros cálculos mientras el controlador está trabajando.

Entradas del controlador:

- Numero_pasos: señal de 30 bit la cual indica el número de pasos que el controlador va a ejecutar
- Direccion: señal de un bit que indica la dirección del movimiento del motor

Salidas del controlador:

- Salida_cables: señal de 4 bits para las bobinas del motor paso a paso

- Interrupcion: señal de un bit para informar al procesador que terminó de ejecutar los pasos

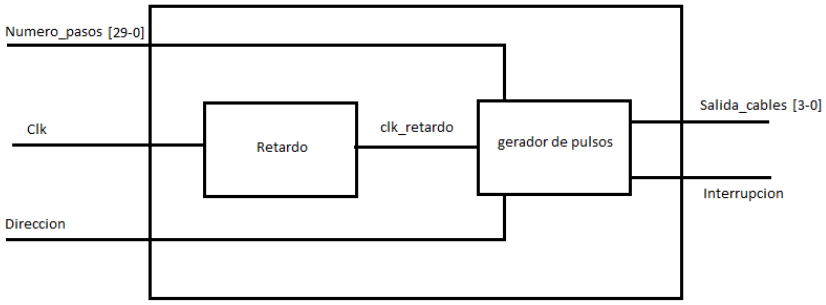


Imagen 8.12 Controlador

La imagen 8.12 muestra el diagrama de bloques del controlador.

El módulo “retardo” de acuerdo al reloj que viene del procesador genera un tiempo de retardo, el retardo específico para los motores del proyecto es de 16.25 milisegundos, para poder tener en un segundo 61 pasos. Cada 16.25 milisegundos el modulo cambia la señal de clk_retardo en 1, esta funciona como señal de activación y es entrada del módulo generador de pulsos.

El modulo generador de pulsos recibe como entrada la señal numero_pasos, clk_retardo y la Direccion, este módulo genera la secuencia de pulsos, envía cada pulso a la salida del módulo salida_cables cada vez que la señal clk_retardo se pone en 1, la dirección (adelante o atras) está dada por la señal Direccion si esta en 1 se generan los pulsos en orden y si es 0 se generan los pulsos en sentido inverso, cada vez que se produce la secuencia se produce un paso, el modulo repite esto el número de veces indicado en la señal Numero_pasos (la señal numero_pasos tiene como bit menos significativo el cero) y cuando termina pone la señal de salida interrupción en 1 para generar la interrupción de aviso de fin de tarea.

8.7. COMUNICACIÓN

Como se mencionó anteriormente, el intérprete que trabaja en el computador genera un seudocódigo de instrucciones equivalente al código programado por el usuario en el lenguaje de alto nivel, este seudocódigo es transferido al robot.

Para comunicar dos programas (situados en dispositivos distintos) y permitir que puedan intercambiar cualquier flujo de datos lo más común y utilizado es un socket de computación.
Sobre el procesador en la fpga se ejecuta un programa servidor que atiende peticiones del cliente que se ejecuta sobre un PC, el programa servidor cuando llega alguna petición del cliente, ejecuta un programa llamado máquina virtual del robot que se detalla más adelante, el cual se encarga de ejecutar el código que fue enviado por el cliente. La imagen 8.13 presenta el flujo de los datos del PC al programa servidor en la FPGA.

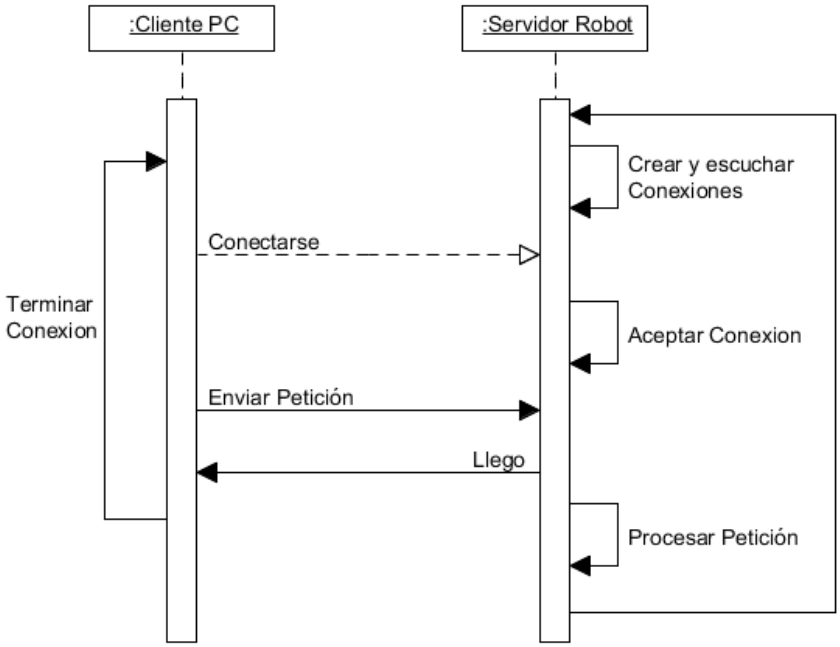


Imagen 8.13 Comunicación PC - Robot

PROTOCOLO

Para realizar la comunicación entre el cliente y el servidor se utilizara el protocolo tcp ya que es importante que todos los datos sean entregados en su destino sin errores y en el mismo orden trasmitidos, debido a que la perdida de cualquier dato podría producir un resultado inesperado.

El formato del mensaje será simplemente un conjunto de cadenas de números separadas por fin de línea (\n) y cada cadena representa una instrucción a ejecutar.

8.8 MÁQUINA VIRTUAL

Cuando el intérprete del lenguaje en el computador está en modo “programar robot” crea un seudocódigo del programa escrito por el usuario, este seudocódigo es enviado al robot donde se encuentra un programa llamado máquina virtual el cual se encarga de ejecutar el seudocódigo del programa del usuario sobre el prototipo robot, esta soporta los mismos tipos de datos, ciclos, bifurcaciones y chequeo de tipos descritos en la especificación del lenguaje, además define varias áreas de datos durante la ejecución de un programa, todas las áreas son creadas cuando la máquina virtual inicia la ejecución y algunas son destruidas solo cuando termina la ejecución, la máquina virtual no soporta varios hilos de ejecución, por lo tanto solo existe un PC de ejecución.

8.8.1 PILAS

La máquina virtual para la ejecución de cualquier programa del lenguaje crea 3 pilas, que son equivalentes a lo que en otras máquinas virtuales son los frames, cada pila almacena un tamaño fijo de datos, para el funcionamiento en dispositivos que tienen pocos recursos de memoria.

8.8.1.1 PILA DE VARIABLES

La máquina virtual utiliza esta pila para almacenar el identificador, el tipo y valor que puede tener una variable en un determinado ámbito de la ejecución de un programa, las operaciones sobre esta pila son:

- Agregar variable: agrega una variable en el ámbito de la cima de la pila
- Agregar ámbito: agrega un nuevo ámbito a la cima de la pila
- Eliminar ámbito: elimina el ámbito de la cima de la pila

Un ámbito es una estructura que almacena estructuras con los atributos: identificador, tipo y valor, los tipos que se pueden almacenar son los descritos en la especificación del lenguaje, todas las variables locales son almacenadas en un ámbito, un ámbito es creado cada vez que hay una invocación de un procedimiento y el ámbito es destruido cuando su procedimiento de invocación completa su ejecución.

8.8.1.2 PILA DE OPERANDOS

La máquina virtual utiliza esta pila para almacenar resultados de operaciones parciales que se puedan realizar entre los datos del lenguaje, esta pila solo guarda un par *tipo:valor*, la máquina virtual posee instrucciones para cargar constantes o valores de variables locales en la pila de operandos, otras instrucciones toman valores(operandos) de la pila de operandos y operan sobre estos y su resultado es introducido en la pila, es decir, con esta pila se ejecutan todas las expresiones del lenguaje, las operaciones de esta pila se describirán con el set de instrucciones de la máquina virtual.

8.8.1.3 PILA DE RETORNOS

La máquina virtual utiliza esta pila para almacenar los índices del registro PC, cuando se hace una invocación de un procedimiento, se guarda el valor de PC+1 para retornar cuando termine la ejecución del procedimiento, las operaciones de esta pila son descritas con el set de instrucciones.

El tamaño de cada una de las pilas está dado por las limitaciones en memoria del dispositivo sobre el cual corra la máquina virtual.

La ejecución de un método puede terminar abruptamente si existe un error en tiempo de ejecución descrito en las especificaciones del lenguaje.

8.8.2 INSTRUCCIONES

Una instrucción de la máquina virtual consiste de 6 bits de código de operación, especificando la operación que se va a realizar, seguido por cero o un operando que será usado para la operación, muchas instrucciones no tienen operandos y consisten solo de un código de operación.

Algunas instrucciones de la máquina virtual codifican información del tipo sobre las operaciones que realizan. Por ejemplo la instrucción *pushe* carga el contenido de la constante de tipo entero a la pila de operandos, la instrucción *pushr* hace lo mismo pero con una constante real, las dos instrucciones tienen idéntica implementación pero distinto código de operación.

Las instrucciones que utilizan tipos booleanos como *pushb* representan este como un entero, en el caso de instrucciones de salto como el *goto* utilizan un entero como valor de salto y el identificador de variable es un entero, algunas instrucciones pueden utilizar valores enteros o reales.

Todas las instrucciones tienen un tamaño de 41 bits, el código de operación usa los 6 bits más significativos de la izquierda, teniendo como más significativo el bit de más a la izquierda, para las instrucciones que utilizan operandos numéricos el bit 34 de derecha a izquierda es el signo de la constante, para el caso de los enteros los primeros 20 bits de derecha a izquierda son el valor del entero, para el caso de los reales los primeros 14 bits de derecha a izquierda son la parte decimal del número y del 14 al 33 representan la parte entera, el bit más significativo es el 33, en el caso de instrucciones como *pushv*, *jumpfalse*, *goto* y *call* se utilizan los 20 bits menos significativos a la derecha como operador.

La tabla 8.3 muestra todos los tipos de instrucciones de la máquina virtual.

código	código de operación	entero	real
0	pushv	constante	
1	pushe	constante	
2	pushr		constante
3	pushb	constante	
4	=	constante	
5	+		
6	-		
7	*		
8	/		
9	uminus		
10	<		
11	>		
12	<=		
13	>=		
14	==		
15	!=		
16	Y		
17	O		
18	!		
19	jumpfalse	constante	
20	goto	constante	
21	call	constante	
22	retorno		
23	finfuncion		
24	etiqueta		
25	avanzar	constante	constante

26	retroceder	constante	constante
27	girarderecha	constante	constante
28	girarizquierda	constante	constante
29	borrarpantalla	constante	constante
30	bajarpluma	constante	constante
31	subirpluma	constante	constante
32	escribe	Constante	constante
33	alto		

Tabla 8.3 Codigos de instrucciones

A continuación se presenta la descripción de cada instrucción de la máquina virtual:

- pushv

Pushv		Constante
40 - 35	34 - 40	19 - 0

Pide el valor y el tipo de la constante en el ámbito de la cima de la pila de variables y lo inserta en la cima de la pila de operandos.

- pushe

Pushe			Constante
40 - 35	34	33 - 20	19 - 0

Inserta la constante entera en la pila de operandos, el signo de la constante es el bit 34, 1 negativo, 0 positivo.

- pushr

Pushe		Constante entera	Constante decimal
40 – 35	34	33 - 14	13 - 0

Inserta la constante real en la pila de operandos, el signo de la constante es el bit 34, 1 negativo, 0 positivo. El valor de la constante real se obtiene de concatenar la constante entera con constante decimal.

- pushb

Pushb		Constante
40 – 35	34 - 20	19 - 0

Inserta la constante en la pila de operandos, los valores son 1 si es verdadero y 0 si es falso.

- =

=		Constante
40 - 35	34 - 20	19 - 0

Saca un elemento de la cima de la pila de operandos y lo inserta en el ámbito de la cima de la pila de variables con identificador constante.

- +

+	
40 - 35	34 - 0

Saca dos elementos de la cima de la pila de operandos y los suma, el resultado lo introduce en la cima de la pila de operandos, si los dos elementos no son de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- -

-	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y hace la resta del primer elemento con el segundo, el resultado lo introduce en la cima de la pila de operandos, si los dos elementos no son de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- *

*	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y los multiplica, el resultado lo introduce en la cima de la pila de operandos, si los dos elementos no son de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- /

/	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y hace la división, el primer elemento sacado es el numerador y el segundo es el denominador, el resultado lo introduce en la cima de la pila de operandos, si los dos elementos no son de tipo numérico o si hay división por cero, se termina con la ejecución del programa.

- uminus

uninus	
40 - 35	34 – 0

Saca un elemento de la cima de la pila de operandos y realiza el operador unario, - , el resultado lo introduce en la cima de la pila de operandos, si el elemento no es de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- <

<	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y realiza el operador relacional menor: ¿el primer elemento sacado es menor que el segundo?, el resultado lo introduce en la cima de la pila de operandos y su tipo es booleano, si

los dos elementos no son de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- >

>	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y realiza el operador relacional mayor: ¿el primer elemento sacado es mayor que el segundo?, el resultado lo introduce en la cima de la pila de operandos y su tipo es booleano, si los dos elementos no son de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- <=

<=	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y realiza el operador relacional menor o igual que: ¿el primer elemento sacado es menor o igual que el segundo?, el resultado lo introduce en la cima de la pila de operandos y su tipo es booleano, si los dos elementos no son de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- >=

>=	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y realiza el operador relacional mayor o igual que: ¿el primer elemento sacado es mayor o igual que el segundo?, el resultado lo introduce en la cima de la pila de operandos y su tipo es booleano, si los dos elementos no son de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- ==

==	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y realiza el operador de igualdad, igual: ¿el primer elemento sacado es igual que el segundo? el resultado lo introduce en la cima de la pila de operandos y su tipo es booleano, si los dos elementos no son del mismo tipo se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- !=

!=	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y realiza el operador de igualdad, diferente que: ¿el primer elemento sacado es diferente que el segundo?, el resultado lo introduce en la cima de la pila de operandos y su tipo es booleano, si los dos elementos no son del mismo tipo se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- Y

Y	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y realiza el operador booleano and, el resultado lo introduce en la cima de la pila de operandos y su tipo es booleano, si los dos elementos no son del tipo booleano se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- O

O	
40 - 35	34 – 0

Saca dos elementos de la cima de la pila de operandos y realiza el operador booleano or, el resultado lo introduce en la cima de la pila de operandos y su tipo es booleano, si los dos elementos no son del tipo booleano se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- !

!	
40 - 35	34 – 0

Saca un elemento de la cima de la pila de operandos y realiza el operador booleano not, el resultado lo introduce en la cima de la pila de operandos y su tipo es booleano, si el elemento no es de tipo booleano se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- jumpfalse

Jumpfalse		Constante
40 - 35	34 -20	19 - 0

Saca un elemento de la cima de la pila de operandos, si el valor sacado es falso el registro PC cambia al valor de la constante, si es verdadero sigue con la instrucción siguiente, si el elemento sacado no es booleano se termina con la ejecución del programa ya que sucedió un error.

- goto

goto		Constante
40 - 35	34 - 20	19 - 0

Cambia el valor del registro PC al valor de la constante.

- call

call		Constante
40 - 35	34 - 20	19 - 0

Inserta un nuevo ámbito a la cima de la pila de variables, inserta la dirección PC + 1 a la cima de la pila de retornos y cambia el valor del registro PC al valor de la constante.

- retorno

Retorno	
40 – 35	34 - 0

Elimina el ámbito de la cima de la pila de variables, saca un elemento de la cima de la pila retornos y cambia el valor del registro PC por el valor sacado de la pila de retornos.

- finfuncion

Finfuncion	
40 - 35	34 - 0

Elimina el ámbito de la cima de la pila de variables, introduce un nulo a la cima de la pila de operandos y saca un elemento de la cima de la pila retornos y cambia el valor del registro PC por el valor sacado de la pila de retornos.

- etiqueta

Etiqueta	
40 - 35	34 - 0

Sigue con la próxima instrucción.

- avanzar

avanzar	
40 - 35	34 - 0

Saca un elemento de la cima de la pila de operandos y envía al driver de los motores la cantidad pasos que equivalen al valor sacado de la pila de operandos en cm, con el valor de la dirección de los dos motores en 0, si el elemento no es de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- retroceder

Retroceder	
40 - 35	34 - 0

Saca un elemento de la cima de la pila de operandos y envía al driver de los motores la cantidad pasos que equivalen al valor sacado de la pila de operandos en cm, con el valor de la dirección de los dos motores 1, si el elemento no es de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- girarderecha

girarderecha	
40 - 35	34 - 0

Saca un elemento de la cima de la pila de operandos y envía al driver de los motores la cantidad pasos que equivalen al valor sacado de la pila de operandos en cm para realizar el giro, con el valor de la dirección del motor derecho en 1 y el motor izquierdo en 0, si el elemento no es de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- girarizquierda

girarizquierda	
40 - 35	34 - 0

Saca un elemento de la cima de la pila de operandos y envía al driver de los motores la cantidad pasos que equivalen al valor sacado de la pila de operandos en cm para realizar el giro, con el valor de la dirección del motor derecho en 0 y el motor izquierdo en 1, si el elemento no es de tipo numérico se termina con la ejecución del programa porque se considera un error de tiempo de ejecución.

- borrarpantalla

borrarpantalla	
40 - 35	34 - 0

Sigue con la próxima instrucción.

- bajarpluma

Bajarpluma	
40 - 35	34 - 0

Enciende el led correspondiente a bajarpluma.

- subirpluma

Subirpluma	
40 - 35	34 - 0

Apaga el led correspondiente a bajarpluma.

- escribe

Escribe	
40 - 35	34 - 0

Saca un elemento de la cima de la pila de operandos y sigue con la próxima instrucción.

- alto

alto	
40 - 35	34 - 0

Elimina el ámbito de la cima de la pila de variables, saca un elemento de la cima de la pila retornos y cambia el valor del registro PC por el valor sacado de la pila de retornos.

8.8.3 TRADUCIENDO EL CÓDIGO

La máquina virtual del intérprete fue diseñada para soportar el lenguaje de programación especificado, pero se debe entender que la máquina virtual es útil para la prospectiva de escribir un intérprete, a través de esta sección nos concentraremos en explicar cómo traducir un código del lenguaje de alto nivel al seudocódigo que ejecuta la máquina virtual.

Sintaxis: *<índice> <código de operación> [operando] [comentario]*

El <índice> es el índice del código de operación de la instrucción en el arreglo que contiene las instrucciones del seudocódigo del usuario, el <código de operación> es el nemotécnico para el código de operación, luego pueden ir cero o un

operando de la instrucción y el <comentario> que es una descripción opcional de la instrucción.

Consideremos el siguiente procedimiento del lenguaje.

```
pr cuadro()
inicio
        I = 0
        mientras(i<4)
        inicio
                .
                .

                .
                I = i + 1
        fin
fin
```

El intérprete traducirá el procedimiento cuadro a:

```
0       cuadro:         //etiqueta
1       pushe 0         //hace un push de la constante 0 en la pila de operandos
2       = i             //hace un pop almacena este valor en la variable local i
3       mientras 1:
4       pushv i         //hace un push con valor de la variable en la pila de operandos
5       pushe  4
6       <               // saca dos elementos de la pila de operandos y los compara
7       jumpfalse 16
        .
        .
        .
12      pushv  i
13      pushe 1
14      +
15      = i
16      goto 3
17      finmientras 1:
18      finFuncion:
```

La máquina virtual es orientada a pila y muchas operaciones toman uno o dos operandos de la pila de operandos del ámbito actual de la pila de variables o de la pila de retorno o insertan valores en estas. Un nuevo ámbito es creado cada vez que un procedimiento es invocado, y con este es creada una nueva cima para la pila de operandos y pila de retornos, en el proceso de ejecución muchos ámbitos, cimas de operandos y retornos pueden ser creados, pero solo el que corresponda a la invocación del método actual estará activo.

Para introducir constantes de números enteros en la cima de la pila de operandos se utiliza la instrucción pushe, para cada tipo de constante del lenguaje existe una instrucción de push, para introducir el valor de una variable a la pila de operandos se utiliza la instrucción pushv y para asignar valores a variables se utiliza la instrucción "=".

Para simular las iteraciones de un ciclo, como por ejemplo el ciclo mientras que está dividido en la condición y el bloque del ciclo, se utiliza la instrucción jumpfalse, la cual saca un elemento de la cima de la pila de operandos y si este valor es falso salta a la instrucción que tiene como índice su operando (el final del bloque) y continua con la ejecución, si su condición se cumple ejecuta el bloque, en donde al final siempre debe haber una instrucción goto la cual sirve para saltar a ejecutar otro punto del programa, muy similar al jumpfalse pero sin condición. Con la instrucción goto vamos al inicio del ciclo mientras y volvemos a evaluar la condición y así sucesivamente.

La máquina virtual opera toda la aritmética sobre la pila de operandos. Miremos el siguiente procedimiento que hace avanzar el robot una determina cantidad de distancia

```
pr distancia()
Inicio
        a = 10+12/23-3
        av a
fin
```

Los operandos para las operaciones aritméticas son sacados de pila de operandos y el resultado de la operación es introducido en la pila de operandos, así estos resultados introducidos en la pila de operandos se convierten en operandos de otras operaciones, con lo cual una expresión es ejecutada, miremos la instrucción de código anterior a = 10+12/23-3, esta instrucción es traducida de la siguiente forma.

```
Pushe 10
Pushe 12
Pushe 23
/
+
Pushe 3
-
= a
```

Primero se introduce la constante 10 a la pila de operandos y a continuación se introducen los operandos 12 y 23, según las especificaciones del lenguaje la división tiene mayor precedencia que la suma o la resta por lo tanto a continuación se realiza la operación división sacando los dos últimos operandos y realizándola, el resultado es introducido nuevamente a la pila de operandos, a continuación se realiza la operación suma con el último dato introducido y el operando 10 y su resultado es introducido en la pila de operandos, después se introduce la constante 3 y a continuación se hace la operación resta y su resultado es introducido en la pila de operandos, en este momento el resultado de la expresión está completo con lo cual solo queda asignarlo a la variable a con la instrucción = la cual saca este último elemento de la pila de operandos y lo guarda en la pila de variables con identificador a.

Ahora examinemos como se traduce un ciclo, para este caso utilizaremos el ciclo repetir, consideremos el siguiente procedimiento.

```
pr CicloRepetir()
Inicio
        repetir 4
        inicio
                .
                .
                .
        fin
fin
```

El intérprete traducirá el procedimiento anterior de la siguiente manera:

```
1       CicloRepetir :
2       pushe 0
```

```
3       = xx
4       repetir 1:
5       pushv  xx
6       pushe  4
7       <
8       jumpfalse 17
9       .
10      .
11      .
12      pushv xx
13      pushe  1
14      +
15      = xx
16      goto 4
17      finrepetir 1
18      finfuncion
```

El ciclo repetir en el lenguaje de alto nivel repite n veces un bloque de código, pero como podemos observar no hay una variable que se incremente a medida que va iterando, un truco para solucionar este problema es introducir una variable en el lenguaje de bajo nivel e inicializarla en cero como se hace en las instrucciones 2 y 3, y luego se procede similar al ciclo mientras, con la variable introducida se crea la expresión de condición del ciclo, en este caso se introduce a la pila el valor de la variable xx, que es la variable introducida y a continuación la expresión de repetición del ciclo que en este caso es una constante de valor 4 y con el operador relacional < hacemos la comparación y a continuación evaluamos la condición con la instrucción jumpfalse que saca un elemento de la cima de la pila de operandos y si es falso salta al final del bloque y si es verdadero continua con la instrucción siguiente, en el bloque del ciclo se debe incrementar la variable introducida en 1 y volver al principio del ciclo para repetir la iteración.

En el caso de las estructuras de decisión como la instrucción si, consideremos el siguiente fragmento de código:

```
Si i<10
inicio
.
.
.
fin
```

```
sino
inicio
.
.
.
fin
```

Este fragmento de código es traducido como sigue:

```
1       pushv i
2       pushe 10
3       <
4       jumpfalse 9
5       .
6       .
7       .
8       goto 13
9       sino 1:
10      .
11      .
12      .
13      finsi
```

Primero se traduce la condición, que se traduce como cualquier expresión, luego con el resultado que arroja la expresión en la línea 4 con la instrucción jumpfalse se evalúa, si la expresión arrojo como resultado falso se salta a la instrucción 9 para seguir con las instrucciones del bloque falso de la declaración si y si la expresión arrojo verdadero se sigue con la próxima instrucción, es decir, se ejecutan las instrucciones del bloque verdadero de la declaración y al final del bloque, instrucción 8 con la instrucción goto se salta al final del bloque falso de la declaración.

Una invocación o un llamado de un procedimiento es implementada con la instrucción call, la cual toma como argumento el índice de la primera instrucción del método invocado, la etiqueta inicial del procedimiento, cada una de las expresiones que corresponden a los argumentos de la función son ejecutados antes del llamado e introducidos en la pila de operandos para su posterior recepción en el procedimiento de invocación. Veamos un ejemplo:

```
pr proced()
```

```
inicio
        Otroproced(2,4)
fin
```

El anterior procedimiento es traducido así:

```
1       proced
2       pushe  2
3       pushe 4
4       call 27
5       finfuncion
```

El llamado al procedimiento otroproced tiene dos parámetros, estos parámetros son expresiones que su resultado se tiene que computar en la pila de operandos y estos son pasados al procedimiento llamado en la pila de operandos, la instrucción call antes de saltar a la instrucción con índice 27 crea un nuevo ámbito en la pila de variables y en la pila de retornos guarda el índice de la próxima instrucción para regresar cuando termine la ejecución del procedimiento llamado.

Los argumentos del procedimiento son pasados al procedimiento llamado por medio de la pila de operandos y aquí son sacados en un orden específico para ser asignados a los parámetros del procedimiento llamado. Continuemos con el ejemplo:

```
pr otroproced(a,b)
inicio
        av a+b
fin
```

El procedimiento se traduce

```
27      otroproced
28      = b
29      = a
30      pushv a
31      pushv b
32      +
33      av
34      finfuncion
```

Cuando un procedimiento es ejecutado los parámetros con que se tiene que ejecutar deben estar en la pila de operandos, por lo que lo primero que se debe hacer en el procedimiento es sacar estos parámetros y asignarlos a las variables que toman estos valores en el procedimiento, en ejemplo esto se hace en las instrucciones 28 y 29. Luego se procede a traducir de forma normal las siguientes instrucciones.

9. MANUAL DE USUARIO

9.1 ACERCA DE SCAPE

Scape es un software educativo para la enseñanza de programación estructurada que permite aprender conceptos como, definición de procedimientos y variables, estructuras de control, recursión, entre otros. Además provee al usuario una interfaz sencilla y amigable.

La herramienta está dirigida a personas novatas en programación.

9.2 REQUERIMIENTOS MÍNIMOS

9.2.1 REQUERIMIENTOS EN HARDWARE

- 512 MB de RAM
- Procesador de 1 GHZ
- Puerto Ethernet de 100 Mbits
- 50 MB de espacio libre en disco duro
- Cable RJ-45 directo

9.2.2 REQUERIMIENTOS DE SOFTWARE

- Tener instalado Python a partir de la versión 2.7
- Tener instalado WXPython compatible con la versión del programa mencionado anteriormente
- Configurar el puerto Ethernet del equipo con la dirección IP 192.168.1.2 con mascara de subred 255.255.255.0

9.3 INTERFAZ

- Ventana principal

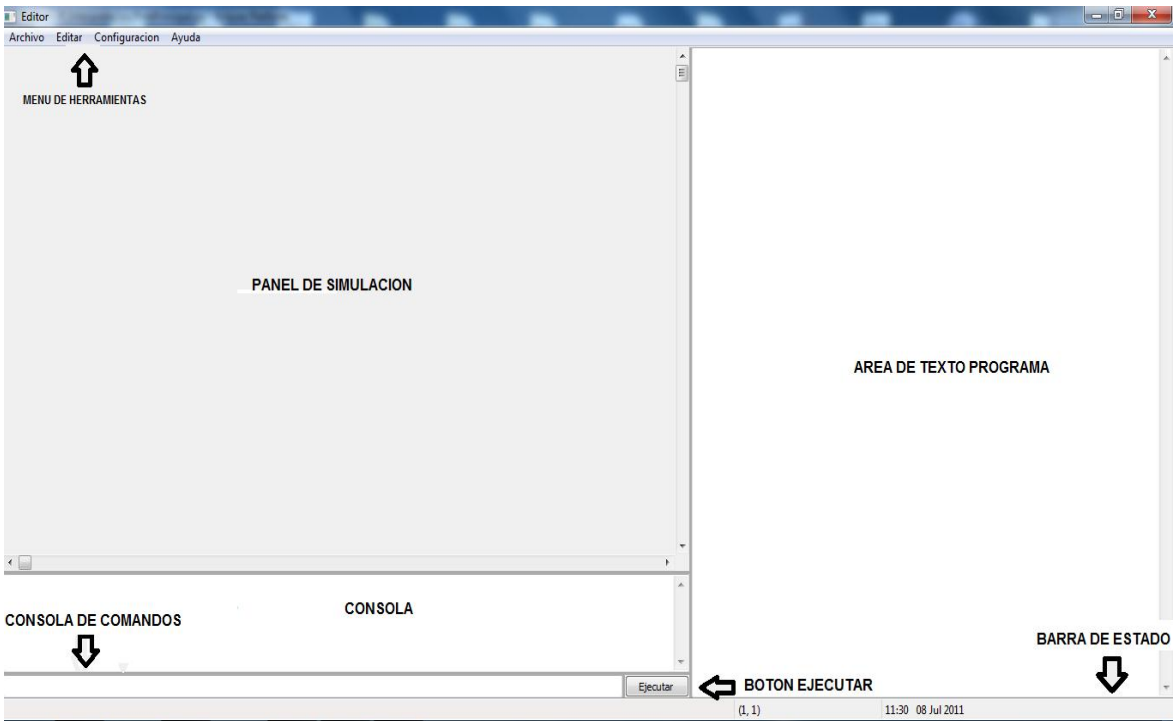


Imagen 9.1 Ventana principal

La imagen 9.1 presenta la ventana principal, que aparece cuando el usuario ejecuta el programa.

- Barra de Menú de herramientas

Contiene las opciones de configuración, archivo y edición de la ventana principal. También cuenta con un sistema de ayuda y tutorial.

- Menú Archivo

Cuando se da clic en el menú Archivo aparecen las opciones Abrir Programa, Guardar, Guardar Como y Salir.

Abrir Programa permite cargar un código fuente en el área de texto del programa.

La ventana que aparece se presenta en la imagen 9.2:

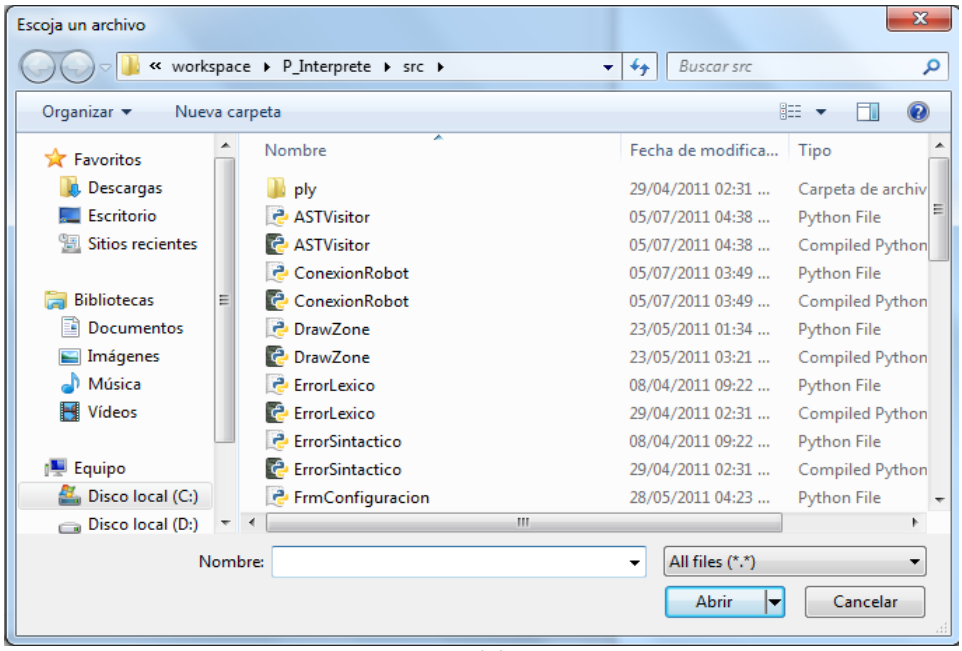


Imagen 9.2 Abrir Programa

La imagen 9.2 presenta la ventana de abrir programa, que permite desplazarse por el sistema de archivos para escoger el archivo a abrir con la aplicación. El botón Abrir le permite confirmar la operación y el botón cancelar permite salir de la ventana sin realizar ninguna operación.

Guardar permite guardar los cambios realizados en el archivo en el cual estamos trabajando y no despliega ventana. Guardar Como permite guardar el trabajo en un nuevo archivo. La ventana que aparece se observa en la imagen 9.3:

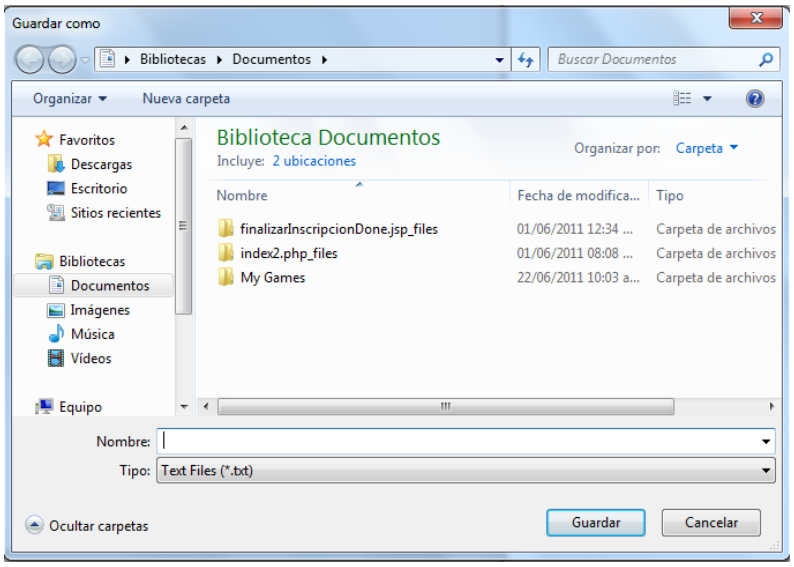


Imagen 9.3 Guardar Como

Esta ventana también permite desplazarse por el sistema de archivos, pero para escoger la ruta en donde quedara guardada la aplicación. El botón guardar permite confirmar la operación y el botón cancelar permite salir de la ventana sin realizar ningún cambio.

Salir permite cerrar la ventana principal y finalizar las operaciones. Esta opción despliega un cuadro de mensaje que se observa en la imagen 9.4 que nos da la opción de guardar los cambios antes de salir.

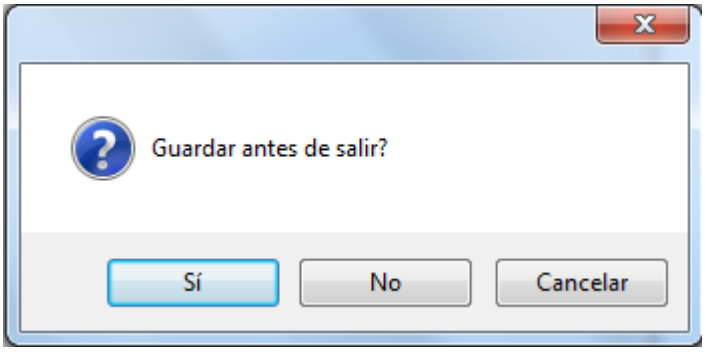


Imagen 9.4 Cuadro de mensaje

El botón Sí guarda el progreso si el archivo ya estaba creado, sino despliega la ventana de guardar como. El botón No cierra la ventana principal y no guarda nada. Finalmente si damos en el botón Cancelar volveremos a la ventana principal.

- Menú Editar

Contiene las opciones que se pueden aplicar a un subconjunto de código fuente en el área de texto del programa. Contiene opciones como Cortar, Copiar, Pegar, Seleccionar todo, Borrar.

Cortar permite substraer un subconjunto seleccionado del código fuente. Copiar permite realizar una copia de un subconjunto seleccionado del código fuente sin extraerlo. Pegar permite introducir texto en la posición del cursor en el área de texto del programa. Borrar permite eliminar un subconjunto del código fuente seleccionado. Seleccionar todo permite escoger todo el código fuente para realizar alguna de las anteriores operaciones.

- Menú configuración

Contiene la única opción Configurar entorno, la cual permite establecer ciertas propiedades del panel de simulación, área de texto del programa, lápiz y modo de ejecución.

La ventana que despliega se observa en la imagen 9.5.

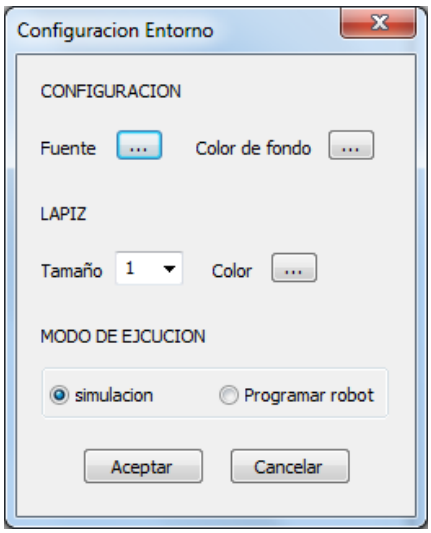


Imagen 9.5 Configuración de entorno

La ventana se divide en tres áreas: fuente y fondo, lápiz y modo de ejecución. Si damos clic en el botón fuente de la primera área de la ventana configuración de entorno se desplegara la siguiente ventana:

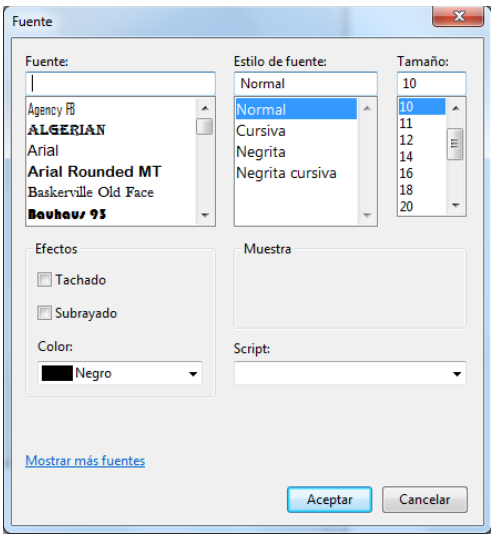


Imagen 9.6 Fuente

La ventana que se observa en la imagen 9.6 cuenta con opciones para modificar el tipo de letra, estilo, tamaño y color. También tiene opciones para subrayar el texto. El botón aceptar permite confirmar la operación y el botón cancelar permite salir de la ventana sin realizar ningún cambio. Si damos clic en el botón color fondo de la primera área de la ventana configuración de entorno se desplegara la ventana observada en la imagen 9.7.

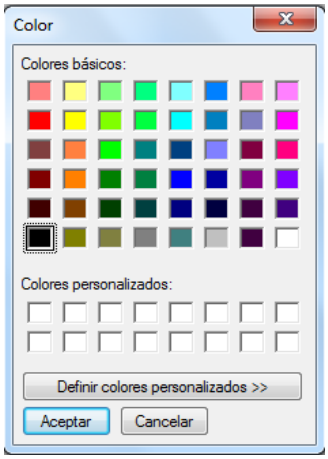


Imagen 9.7 Color Fondo

Esta ventana permite seleccionar el color del panel de simulación mediante una serie de cajas con colores. El botón aceptar permite confirmar la operación y el botón cancelar permite salir de la ventana sin realizar ningún cambio.

En la segunda área de la ventana configuración de entorno hay una lista con los tamaños permitidos del lápiz, el lápiz es el tamaño de la línea que aparece en el panel de simulación cuando escribimos el comando bajar pluma. En esta área también hay un botón color que despliega la ventana que hay en la imagen 9.8.

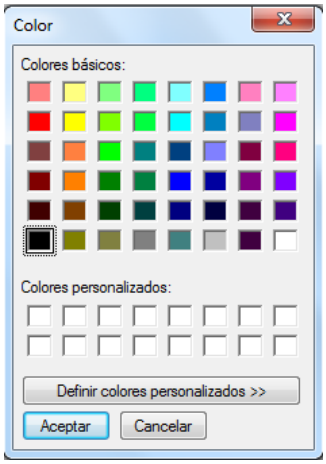


Imagen 9.8 Color lápiz

Esta ventana permite cambiar el color del lápiz que actúa sobre el panel de simulación mediante una serie de cajas con colores. El botón aceptar permite confirmar la operación y el botón cancelar permite salir de la ventana sin realizar ningún cambio.

En la tercera área de la ventana configuración de entorno hay dos radio button que permiten escoger el modo de ejecución, si se escoge la opción simulación el resultado de ejecutar el código fuente será presentado en el panel de simulación. Si por el contrario se escoge la opción programar robot, el código fuente será ejecutado por el prototipo físico.

La ventana configuración de entorno también tiene los botones aceptar y cancelar, el botón aceptar permite confirmar la operación y el botón cancelar permite salir de la ventana sin realizar ningún cambio.

- Menú Ayuda

Contiene la opción tutorial y la opción ayuda.

Tutorial contiene una lista de ejercicios a realizar y temas que se deben conocer para el correcto funcionamiento de la herramienta. La ventana que se despliega si da clic sobre esta opción se muestra en la imagen 9.9:

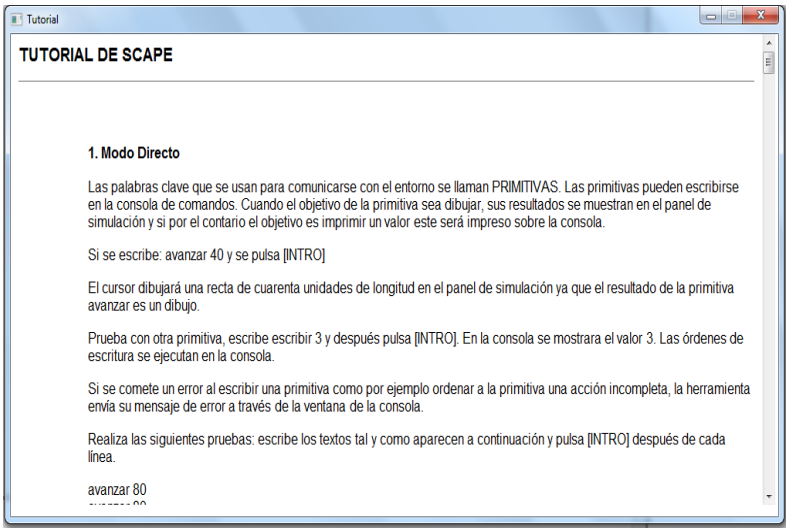


Imagen 9.9 Tutorial

La ventana Tutorial contiene un índice que permite tener una vista preliminar de los temas del tutorial, además tiene un botón find que permite buscar las coincidencias del texto ingresado en el cuadro de texto en el documento.

- Área de texto programa

Es un área en donde se puede escribir el programa a ejecutar.

- Consola de comandos

Área en donde se puede hacer el llamado a una función definida en el área de texto del programa, o de funciones simples del lenguaje.

- Botón ejecutar

Es un botón que permite ejecutar lo que se encuentre en la consola de comandos.

- Panel de simulación

Área que me permite observar la simulación de lo que se ejecutó.

- Consola

Permite ver el histórico de la consola de comandos, los errores presentes en el código del programa y salida del comando escribe.

- Barra de estado

Muestra la fecha actual, la hora en que se abre la aplicación, la fila y columna donde está ubicado el cursor del área de texto del programa y la posición y dirección del robot en la simulación.

9.4 LENGUAJE

9.4.1 COMENTARIOS

Los comentarios se inician con el símbolo # y se extienden hasta encontrar otro símbolo #. El intérprete no tiene en cuenta los comentarios, lo cual es útil si deseamos poner información adicional en nuestro código, como por ejemplo una explicación sobre el comportamiento de una sección del programa.

9.4.2 VARIABLES

Las variables se definen de forma dinámica, lo que significa que no se tiene que especificar cuál es su tipo de antemano y puede tomar distintos valores en la ejecución.

X = 21
X = 32.32
X = verdadero

9.4.3 TIPOS DE DATOS

La tabla 9.1 muestra los tipos de datos del lenguaje.

Tipo	Clase	Notas	Ejemplo
Entero	Números enteros	Precisión fija	20
Flotante	Numero decimal	Coma flotante	100.32
Booleano	Booleanos	Valor booleano. Verdadero, falso	verdadero

Tabla 9.1 Tipos de Datos

9.4.4 OPERADORES

OPERADORES ARITMÉTICOS

Operación	operador	Aridad	Prioridad
cambio de signo	-	Unario	1
multiplicación	*	Binario	2
división	/	binario	2
suma	+	binario	3
resta	-	binario	3

Tabla 9.2 Operadores Aritméticos

En la tabla 9.2 muestra los operadores aritméticos, la prioridad más alta es la 1.

OPERADORES RELACIONALES

Operación	operador	aridad	Prioridad
es igual que	==	binario	1
es distinto que	!=	binario	1
es menor que	<	binario	1
es menor o igual que	<=	binario	1
es mayor que	>	binario	1
es mayor o igual que	>=	binario	1

Tabla 9.3 Operadores Relacionales

En la tabla 9.3 muestra los operadores relacionales, la prioridad más alta es la 1.

OPERADORES LÓGICOS

Operación	operador	aridad	Prioridad
or	O	binario	2
and	Y	binario	2
Not	No	unario	1

Tabla 9.4 Operadores lógicos

En la tabla 9.4 muestra los operadores lógicos, la prioridad más alta es la 1.

9.4.5 CONDICIONALES

Una sentencia condicional (*si*) ejecuta su bloque de código interno sólo si se cumple cierta condición. Se define usando la palabra clave si seguida de la condición, y el bloque de código. Opcionalmente, puede haber un bloque final (la palabra clave sino seguida de un bloque de código) que se ejecuta cuando la condición es falsa.

```
si x<21
inicio
        …
fin
```

9.4.6 PROCEDIMIENTOS

Las procedimientos se definen con la palabra clave procedimiento o pr, seguida del nombre de la función, sus parámetros encerrados entre paréntesis y el bloque. El valor devuelto en las procedimientos será el dado con la instrucción retorno.

9.4.7 CICLOS

MIENTRAS

El bucle mientras evalúa una condición y, si es verdadera, ejecuta el bloque de código interno. Continúa evaluando y ejecutando mientras la condición sea verdadera. Se define con la palabra clave mientras seguida de la condición, y a continuación el bloque de código interno.

```
mientras(condicion)
inicio
        …
fin
```

REPETIR

El bucle repetir ejecuta el bloque de código interno n veces, se define con la palabra clave repetir seguida de un número (el número de repeticiones) y a continuación el bloque de código interno.

```
repetir 4
inicio
        …
fin
```

PARA

El ciclo para ejecuta el código interno de código de acuerdo a la evaluación de una variable de control y su tope.

El ciclo para contiene tres elementos: la inicialización de la variable de control con su tope, el incremento, que define la manera en que la variable de control de ciclo debe cambiar cada vez que el computador repite un ciclo. Y el bloque de instrucciones a ejecutar.

Para definir utilizar correctamente esta estructura con la herramienta se coloca la palabra clave para seguido de la inicialización de la variable de control con una expresión, luego se coloca la palabra clave hasta seguido de una expresión que representa el tope, luego se coloca una coma, la palabra clave incremento, el carácter = y una expresión que contiene el tope de la variable de control. Por último se coloca el bloque a ejecutar entre inicio y fin. Ejemplo:

```
para i = 2 + 3 hasta 10, incremento = 2
inicio
        …
fin
```

9.4.8 INSTRUCCIONES

- avanzar n - av n

El robot avanza hacia adelante n unidades, se define con la palabra clave avanzar o av seguido de un numero entero o flotante.

- girarizquierda n - gi n

El robot gira hacia la izquierda n grados, se define con la palabra clave girarizquierda o gi seguido de un número entero o flotante.

- girarderecha n - gd n

El robot gira hacia la derecha n grados, se define con la palabra clave girarderecha o gd seguido de un número entero o flotante.

- retroceder n - rt n

El robot retrocede n unidades, se define con la palabra clave retroceder o rt seguido de un numero entero o flotante.

- escribe n

Se escribe en la consola el dato n, se define con la palabra clave escribe seguido de cualquier tipo de dato del lenguaje.

- alto – al

Se termina con la ejecución del procedimiento actual.

- bajarpluma – bp

El robot baja el lápiz para hacer gráficos, se define con la palabra clave bajarpluma o bp.

- subirpluma – sp

El robot sube el lápiz para hacer gráficos, se define con la palabra clave subirpluma o sp.

- borrarpantalla – bo

Se limpia el área de gráficos del robot, se define con la palabra reservada borrarpantalla o bp.

- retorno n

Se termina con la ejecución del procedimiento actual y retorna n.

CONCLUSIONES

- Un sistema computacional como herramienta educativa didáctica mejora los procesos de enseñanza y de aprendizaje a través del logro de objetivos dando prioridad al significado del aprendizaje.

- Este sistema Computacional para la enseñanza de programación en el cual el usuario se involucra de forma activa es una alternativa didáctica al método tradicional de enseñanza debido a que el núcleo del aprendizaje radica en la generación y resolución de problemas.

- En el método tradicional de enseñanza el alumno es un receptor pasivo del conocimiento, en un sistema computacional fundamentado en el aprendizaje basado en problemas el aprendizaje está centrado en el alumno.

- En un sistema en el cual el usuario pueda ver materializado el resultado de la solución propuesta para resolver un problema determinado que se plantee inicialmente, generalmente incentiva al usuario a construir su propio conocimiento y a relacionarlo con el conocimiento previo.

- Un sistema computacional por sí solo no es suficiente para facilitar el aprendizaje si no está acompañado de una metodología de enseñanza adecuada.

- La mejor metodología para enseñar con este sistema computacional es el aprendizaje basado en problemas.

- El éxito que este proyecto pueda tener, depende de la motivación de los usuarios con respecto al uso y tiempo que deben dedicar a interactuar con el sistema.

RECOMENDACIONES

- Para un primer acercamiento al sistema se recomienda revisar el tutorial que incluye la aplicación.

- Para obtener la máxima precisión sobre el sistema implementado, el tamaño de la rueda debe ser de 4 cm de radio.

- Para una mejor interacción hombre maquina se recomienda la implementación de la transferencia del programa del usuario al robot mediante tecnología inalámbrica.

- Se recomienda extender el sistema computacional para que se pueda programar multiparadigma.

- Se recomienda agregar un módulo de baterías para la alimentación y así facilitar el desplazamiento autónomo del robot.

- Para desarrollar comercialmente el proyecto se pueden buscar alternativas menos costosas que las FPGA's como lo son los microcontroladores.

- Consultar la adecuación de la metodología MOSCA mencionada en este trabajo o una a fin, para la evaluación del proyecto mediante cuestionarios de valoración cualitativos.

- Para una posterior etapa del proyecto en la cual se evalúe el sistema en un aula de clase, se debe implementar el mecanismo de subir y bajar la pluma física para marcar la trayectoria del robot en una superficie.

- Para facilitar el proceso de instalación del sistema se recomienda implementar un instalador para cada una de las principales plataformas.

- Para mejorar la accesibilidad del sistema con respecto al idioma, se recomienda implementar en el lenguaje el equivalente de las instrucciones en inglés.

- Para extender la vida útil del sistema se recomienda incluir sensores que le permitan evitar colisiones contra objetos externos al sistema.

- Se recomienda incluir multiprocesamiento en el robot.

BIBLIOGRAFÍA

LIBROS CONSULTADOS

- Compiladores: Principios, Técnicas y Herramientas. – Aho Sethi, Ullman Pearson.
- Diseño de compiladores modernos - Grune Dick, Bal Henri E, Jacobs Ceril J.H, Langendoen Koen G.
- Ingeniería de software - I. SOMMERVILLE. - 6A ED.
- Sistemas Operativos / WILLIAM STALLINGS. - 2A ED.
- Sistemas operativos modernos / ANDREW S. TANENBAUM. - 3a Ed.
- Sistemas distribuidos: Principios y paradigmas - Tanenbaum Andrew S, Van Steen Maarten.
- Sistemas operativos: Diseño e implementación - ANDREW S. TANENBAUM. - 2A ED.
- Wxpython in action - NOEL RAPPIN, ROBIN DUNN.
- Python para todos – Gonzáles Duque Raúl.
- Communication of the acm – education from interest to values – DISALVO BETSY, BRUCKMAN AMY.
- Digital fundamentals with vhdl - FLOYD THOMAS.
- Actos de significado: Mas alla de la revolución cognitiva – BRUNER JEROME S.
- El lenguaje de programación c – KERNIGHAN BRIAN W., DENNIS RITCHIE.
- Revista tecnura - Compilador y traductor de pseudocódigo para la lógica de programación (CompiPrograмación) - VANEGAS CARLOS ALBERTO.
- The java language specification - GOSLING JAMES, JR. STEELES GUY L., JOY BILL.

DOCUMENTACIÓN WEB

- http://www.tecnun.es/departamentos/doi/personal/nserrano/proyectos-final-de-carrera/analisis-de-lenguajes-visuales-para-la-ensenanza-de-programacion-en-entorno-universitario.html Consultado en Marzo de 2010
- http://www.mcgraw-hill.es/bcv/guide/capitulo/8448146433.pdf
- http://www.colombiestad.gov.co/index.php?option=com_content&task=view&id=353&Itemid=90 Consultado en Marzo de 2010

- Plan Decenal Nacional De Educación. Consultado en Abril de 2010
- http://www.eduteka.org/modulos.php?catx=9&idSubX=272&ida=78&art=1 Consultado en Marzo de 2010
- http://el.media.mit.edu/Logo-foundation/logo/index.html Consultado en Abril de 2010
- http://www.rmm.cl/index_sub.php?id_contenido=5349&id_seccion=3437&id_portal= 520 Consultado en Abril de 2010
- http://www.eduteka.org/modulos.php?catx=9&idSubX=284&ida=908&art=1
- Consultado en Mayo de 2010
- http://www.uoc.edu/rusc/5/2/dt/esp/hernandez.pdf Consultado en Mayo de 2010
- http://www.eduteka.org/modulos/9/286 Consultado en Mayo de 2010
- http://www.mastermagazine.info/termino/4368.php Consultado en Abril de 2010
- http://www.investigacion.frc.utn.edu.ar/labsis/Publicaciones/InvesDes/Compiladores/rxc.htm Consultado en Mayo de 2010
- http://www.xilinx.com/company/gettingstarted/index.htm Consultado en Junio de 2010
- http://www.miky.com.ar/fpga_2004.pdf Consultado en Agosto de 2010
- http://www.xilinx.com/tools/microblaze.htm Consultado en Agosto de 2010
- http://en.wikipedia.org/wiki/MicroBlaze Consultado en Agosto de 2010
- http://redalyc.uaemex.mx/src/inicio/ArtPdfRed.jsp?iCve=61580304 Consultado en Septiembre de 2010
- http://www.academia-interactiva.com/evaluacion.pdf Consultado en Agosto de 2010
- http://www.enlaces.cl/doc/evaluacion_recursos_educativos.doc. Consultado en Octubre de 2010
- Teorías del aprendizaje y métodos de enseñanza con ordenadores. Manuel Area Moreira. Consultado en Junio de 2010
- http://ausubel.idoneos.com/index.php/368873 Consultado en Abril de 2010
- http://www.omerique.net/twiki/pub/Main/TrabajoSegundoPaloma/Ausubel.pdf Consultado en Mayo de 2010
- Compiladores Principios, técnicas y herramientas. Alfred V. Aho – Ravi Sethi Consultado en Mayo de 2010
- http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5492549&queryText%3Dmeaningful+learning%26openedRefinements%3D*%26searchField%3DSearch+All Consultado en Marzo de 2011
- http://www.utemvirtual.cl/plataforma/aulavirtual/assets/asigid_745/contenidos_ara/39247_david_ausubel.pdf Consultado en Marzo de 2011

- http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5236308&queryText%3Dmeaningful+learning%26openedRefinements%3D*%26searchField%3DSearch+All Consultado en Marzo de 2011
- AUSUBEL-NOVAK-HANESIAN (1983) Psicología Educativa: Un punto de vista cognoscitivo. 2° Ed. TRILLAS México Consultado en Marzo de 2011
- http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=4755792&queryText%3Ddiscovery+learning%26openedRefinements%3D*%26searchField%3DSearch+All Consultado en Marzo de 2011
- http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5593719&queryText%3Ddiscovery+learning%26openedRefinements%3D*%26searchField%3DSearch+All Consultado en Abril de 2011
- http://www.csi-csif.es/andalucia/modules/mod_ense/revista/pdf/Numero_18/OLGA_ZARZA_CORTES01.pdf Consultado en Abril de 2011
- http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5529601&queryText%3Dproblem+based+learning%26openedRefinements%3D*%26pageNupage%3D2%26searchField%3DSearch+All Consultado en Abril de 2011
- http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=5460038&queryText%3Dproblem+based+learning%26openedRefinements%3D*%26searchFsear%3DSearch+All Consultado en Abril de 2011
- https://tspace.library.utoronto.ca/bitstream/1807/8986/1/rc01037.pdf Consultado en Abril de 2011
- http://www.rieoei.org/deloslectores/1460Santillan.pdf Consultado en Abril de 2011
- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4959073 Consultado en Abril de 2011
- http://www.microcontroladorespic.com/tutoriales/FPGAs/Procesador-MicroBlaze.html Consultado en Abril de 2011
- http://java.sun.com/docs/books/jls/first_edition/html/index.html Consultado en Junio de 2011
- http://docs.python.org/reference/ Consultado en Junio de 2011