Universidad de las Ciencias Informáticas

Facultad 5 “Entornos Virtuales”

Título: “Máquina Virtual sobre software libre para el intérprete del sistema Emedia”

Trabajo de Diploma para optar por el título de
Ingeniero en ciencias Informáticas

Autor(es): Yaidel Aguila González
Adrián Torres González
Tutor(es): Ing. Ana Silvia Tellería Martínez
Consultante(s): Daynel Marmol Lacal
Oliver Fernández Gil
Manuel Villanueva Betancourt

“Julio del 2008”

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Autores:

Yaidel Aguila González

_____

Adrián Torres González

_____

Tutor:

Ana Silvia Tellería Martínez

_____

Firma del Tutor

DATOS DE CONTACTO

Tutor:

Ing. Ana Silvia Tellería Martínez: (atelleria@uci.cu) Graduada de Ingeniaría en Ciencias Informáticas en el 2007 en la Universidad de las Ciencias Informáticas. Desde su graduación está trabajando en la UCI, desempeñándose como profesora.

Consultantes:

Lic. Daynel Marmol Lacal: (dmarmol@uci.cu) Graduado de Licenciatura en Ciencias de la Computación en la Universidad Central "Martha Abreu" de Las Villas en julio del 2005. Desde su graduación está trabajando en la UCI, desempeñándose como profesor de las asignaturas Programación II (Estructura de Datos) y Programación IV (Compiladores), desde su incorporación al centro se ha vinculado a la producción.

Lic. Oliver Fernández Gil: (oliver@uci.cu) Graduado de Licenciatura en Ciencias de la Computación. Desde su graduación está trabajando en la UCI, desempeñándose como profesor de Programación.

Manuel Villanueva Betancourt: (manuelvb@uci.cu)

AGRADECIMIENTOS

A nuestros familiares, compañeros, amigos, que de una forma u otra han estado junto a nosotros durante estos cinco años, a los que nos ayudaron en los momentos difíciles, a los que nos apoyaron en nuestro trabajo de diploma….

…A todos ellos muchas gracias por el apoyo y la comprensión.

DEDICATORIA

Yayo

…Dedico este trabajo, todos mis esfuerzos a mis padres, mis hermanos, a la Tuty por ser mi otra madre, a Ceci por apoyarme tanto en todo este tiempo y aguantar mis pesadeces, a Nancy, a mis amigos, compañeros y al resto de mi familia que siempre han estado ahí para mí….

Adrian

….Dedicarle en forma especial mi trabajo de diploma a toda mi familia, mis padres, hermanos, compañeros, amigos, todos los que me han apoyado todo este tiempo y que forman parte de mí.

RESUMEN

En la actualidad los Sistemas de Gestión del Aprendizaje (SGA) han abarcado distintos sectores de la sociedad en las últimas décadas. Dicha tecnología, puede encontrarse en la educación, salud, entre otros, ofreciendo servicios de forma interactiva.

Este trabajo abarca el diseño básico de un intérprete para el Sistema Emedia y la implementación de la Máquina Virtual de dicho intérprete para la ejecución del código intermedio. Se estudiaron y se obtuvieron los requerimientos del sistema y un primer diseño básico correspondiente a la primera iteración sobre el problema. La implementación realizada se centró en el componente correspondiente a la máquina virtual. El producto resultante en su versión inicial permite realizar operaciones matemáticas lo cual servirá como modelo y desarrollo base para el intérprete a empotrar en el sistema Emedia.

PALABRAS CLAVE:

Intérprete, analizador léxico, analizador sintáctico, analizador semántico, generador de código intermedio, máquina virtual, gramática.

TABLA DE CONTENIDO

TABLA DE CONTENIDO

TABLA DE CONTENIDO

TABLA DE CONTENIDO

INTRODUCCIÓN

"Si quieres un año de prosperidad, planta arroz.
Si quieres 10 años de prosperidad, planta árboles.
Si quieres prosperidad para siempre, educa un pueblo"
(Proverbio Chino)

Educar a una sociedad constituye un trabajo arduo y un deber, fundamental para el hombre. Cada vez se necesita más, que los jóvenes estén preparados, sean conscientes y comprometidos con el desarrollo social, de fuertes ideales y valores arraigados, con la capacidad de afrontar los retos del presente y del futuro.

La educación es imprescindible para el desarrollo de la sociedad e impacta en todas las esferas de la vida.

En la actualidad se puede apreciar cómo se pone en práctica la utilización de los nuevos medios de enseñanza en la educación, tales como: la radio, la televisión y las computadoras. También es habitual el acceso a cursos a través de la red y específicamente, mediante utilización de los Entornos Virtuales de Aprendizaje (EVA).

Un EVA, también conocido como Sistema de Gestión del Aprendizaje (SGA) es un software que puede ser instalado en un servidor y se utiliza para crear, gestionar y distribuir cursos. Sirve de contenedor de cursos y objetos de aprendizaje. Incorpora además, otras herramientas para facilitar la comunicación y el trabajo colaborativo entre profesores y estudiantes, así como el seguimiento y evaluación del alumno. (2000)

Un objeto de aprendizaje es una entidad informativa digital desarrollada para la generación de conocimiento. "Un recurso digital que puede ser rehusado para ayudar en el aprendizaje." (Wiley, 2000)

La Universidad de las Ciencias Informáticas (UCI), desarrolla y emplea constantemente software para el apoyo del proceso de enseñanza-aprendizaje (PEA). Un ejemplo de herramienta que se utiliza para apoyar este proceso es la Plataforma Moodle.

En el desarrollo de software destinado para facilitar el PEA en ocasiones es preciso tener dominio de conocimientos de programación no triviales. Sino se emplean lenguajes de programación como PHP, JavaSript o HTML, se corre el riesgo de obtener objetos de aprendizaje con funcionalidades limitadas. Su utilización tiene además otros inconvenientes, uno específico de los países de habla hispana lo constituye que para dichos países estos lenguajes no corresponden con la lengua nativa del experto en contenidos. Algunas de las irregularidades que se han detectado en el Proceso de Desarrollo de Software (PDS) para el aprendizaje que atentan contra la calidad de las aplicaciones resultantes son:

- La utilización de herramientas de propósito general no orientadas al conjunto de especificidades que se requieren para lograr mayor calidad en el desarrollo de aplicaciones orientadas a la educación.
  Las Herramientas más utilizadas por los profesores o tutores se dividen en tres grupos:
  - Herramientas Multimedia: Power Point, Premier, Macromedia Director.
  - Herramientas para creación y edición de documentos de tipo WYSISWYG: Adobe.
  - Herramientas de procesado de textos: Word. (Castro, y otros, 1996)

    Los problemas principales de dichos sistemas van desde las pocas funcionalidades que brindan algunos como el Power Point, hasta la complejidad de uso de otros como el Macromedia Director.
- Los expertos en contenidos con frecuencia, no dominan el PDS, y los desarrolladores no suelen poseer conocimientos suficientes en los diversos contenidos. Dicha característica trae consigo una brecha en la comunicación entre ambas entidades, con afectación en el desarrollo de herramientas que faciliten el aprendizaje. Por lo que se hace necesario vincularlos, para que puedan interactuar en el proceso de desarrollo de los objetos de aprendizaje y obtener resultados de mayor calidad.

Surge así la necesidad de crear un SGA que les facilite a los profesores la forma en que interactúan con los nuevos medios informáticos y contribuya a prescindir de las dificultades existentes en la actualidad. "Dándole además la posibilidad de que desarrollen, implementen sus propias ideas, que se sientan cómodos utilizando los nuevos recursos y medios de enseñanza." (Fung Goizueta, 2006)

Los sistemas desarrollados deben permitir una mayor interacción con los usuarios finales.

Para suplir las necesidades planteadas anteriormente es que surge la idea de crear el sistema Emedia cuyo objetivo es desarrollar y gestionar herramientas para el aprendizaje.

Emedia se desarrolla sobre software libre: "Teniendo en cuenta que el código es abierto y podemos negociar con nuestros clientes cualquier tipo de condiciones de pago o de transferencia tecnológica, dado que somos dueños de todo lo que tenemos y hacemos, las condiciones se pueden discutir sin ningún tipo de restricciones, no hay restricciones para establecer una negociación con un potencial cliente en cuanto a qué puede adquirir, cómo lo va adquirir o cómo lo va a usar." (Fung Goizueta, 2006)

El sistema Emedia está compuesto por componentes independientes entre los que se destacan:

- El "Autor de Contenido": Posibilita la creación de los guiones de contenidos necesarios para la construcción de cursos.
- El "Autor de Objetos de Aprendizaje": Se centra en la creación de los objetos de aprendizaje que luego serán utilizados en los cursos.

- El “Proyector”: Se ocupa de la visualización de los Objetos de Aprendizaje.
- El “Repositorio”: Posibilita el almacenamiento, acceso y gestión a los objetos de aprendizaje desarrollados.
- El “SGA”: Es la herramienta encargada del despliegue del sistema Emedia.

Un intérprete es una tecnología que analiza y ejecuta programas directamente a partir del código fuente o de una forma intermedia que representa al programa original en un lenguaje de alto nivel. (2006)
Puede ser utilizado de forma independiente para que se ejecute directamente en el sistema operativo o asociado a aplicaciones. Está compuesto por diferentes componentes, tales como:

- Analizador Léxico: Recibe el código fuente y devuelve una secuencia de tokens.
- Analizador Sintáctico: Recibe la secuencia de tokens y devuelve el árbol de sintaxis abstracta.
- Analizador Semántico: Recibe el árbol de sintaxis abstracta y devuelve el árbol de sintaxis abstracta decorado.
- Generador de Código Intermedio: Recibe el árbol de sintaxis abstracta y devuelve el código intermedio.
- Máquina Virtual: Recibe el código intermedio y lo ejecuta.

En la actualidad son diversas las aplicaciones que utilizan intérpretes como parte de ellas con el objetivo de alcanzar mayor expresividad. Algunos ejemplos de aplicaciones que utilizan intérpretes empotrados lo constituyen:

- Asistente matemático XMath: Analizador matemático, con entorno script que ofrece un lenguaje de programación de alto nivel, orientado a objetos llamado MathScript que brinda una depuración interactiva y posee una extensa biblioteca de funciones predefinidas, ambas características permiten disminuir el tiempo de desarrollo y de esfuerzo. El lenguaje asociado al asistente amplía sus funcionalidades. (2008)
- Gimp: Herramienta multiplataforma de manipulación fotográfica. “GIMP es un acrónimo de GNU Image Manipulation Program. En el GIMP se pueden realizar todo tipo de tareas de manipulación de imágenes, incluyendo retoque fotográfico, composición de imágenes y creación de imágenes.” (Neumann, y otros)

Utiliza un lenguaje de guionado (scripting) que permite automatizar tareas de manipulación de imágenes, el lenguaje se nombra Script-Fu y se instala en el sistema. (Neumann, y otros)

- Blender: Programa que integra un conjunto de herramientas para la creación de contenidos 3D, visualizaciones, "tanto imágenes estáticas como videos de alta calidad." (2007)

  Posee una característica muy poderosa y es que contiene asociado un intérprete de Python, lo cual "le permite a cualquier usuario añadir funcionalidades a Blender escribiendo un simple script de Python." (2007)

  Python está diseñado con el objetivo de ser usado como extensión para aplicaciones que necesiten una interfaz programable, por tal motivo, es que Blender lo utiliza. (2007)

Teniendo la situación expuesta anteriormente, se plantea el siguiente problema científico:

¿Cómo contribuir a la flexibilidad del sistema Emedia?

El objeto de estudio del presente trabajo científico es el diseño e implementación de intérpretes.

El campo de acción es la máquina virtual para el intérprete del sistema Emedia.

Por lo que se propone como objetivo general: Diseñar un intérprete para el sistema Emedia e implementar sobre software libre el módulo correspondiente a la máquina virtual

Tareas científicas

1. Fundamentar los principales elementos teóricos asociados al desarrollo de intérpretes, máquinas virtuales y a la concepción del sistema Emedia.
2. Identificar los requerimientos funcionales y no funcionales que guiarán el diseño del intérprete.
3. Elaborar el diseño del intérprete teniendo en cuenta los requerimientos identificados.
4. Elaborar una versión básica del lenguaje fuente.
5. Implementar sobre software libre la primera versión de la máquina virtual del intérprete.
6. Emitir valoración sobre los resultados finales.

Métodos

Para dar cumplimiento a las tareas trazadas se combinan diferentes métodos y técnicas en la búsqueda y procesamiento de la información. Entre los fundamentales se encuentran:

Teóricos

Analítico – Sintético: Para la definición de las funcionalidades del intérprete.

Inductivo – deductivo: En la revisión y justificación del modelo y la tecnología para diseñar el intérprete.

Mediante el análisis de casos aislados se arribaron a proposiciones generales que luego se emplearon en la inferencia de la tecnología y modelo más adecuados a emplear en el diseño.

Análisis sistémico: Para el desarrollo del diseño mediante la elaboración de los diagramas de clases.

Método de modelación: En el desarrollo del diseño mediante la elaboración de los diagramas de colaboración.

Empíricos

Entrevista: Para la búsqueda de información en especialistas de alto nivel en el tema que pueden hacer un aporte sustancial a su desarrollo.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA. ESTADO DEL ARTE. FUNDAMENTACIÓN DE LAS HERRAMIENTAS A UTILIZAR

1.1Introducción

El intérprete a desarrollar para el sistema Emedia es una biblioteca que se insertaría en componentes del sistema para garantizar una serie de funcionalidades que amplíen la flexibilidad de dicho sistema.

En el presente capítulo se abordan principalmente los conceptos de Intérprete, SGA y Emedia como sistema, así como la definición de las herramientas a utilizar para el diseño básico del intérprete y el desarrollo de la máquina virtual.

1.2 Lenguajes de Programación

1.2.1 Historia

Los ordenadores son máquinas y como tales, necesitan constantemente interpretar las instrucciones que reciben, es por ello que surgen los lenguajes de programación, que hacen posible la comunicación con el microprocesador, utilizan términos y símbolos relacionados con el tipo de problema que se debe resolver, mediante el empleo de herramientas que brinda la informática.

Charles Babbage, profesor matemático de la universidad de Cambridge e inventor inglés, a principios del siglo XIX predijo muchas de las teorías en que se basan los actuales ordenadores. Creó la máquina analítica, que no pudo construirse hasta mediados del siglo XX. Con él colaboró Ada Lovelace, considerada como la primera programadora de la historia, quien realizó programas para la máquina de Babbage.

A continuación se muestra un cuadro evolutivo donde aparecen los lenguajes que por su uso y comercialización, han resultado ser los más populares durante el siglo XX:

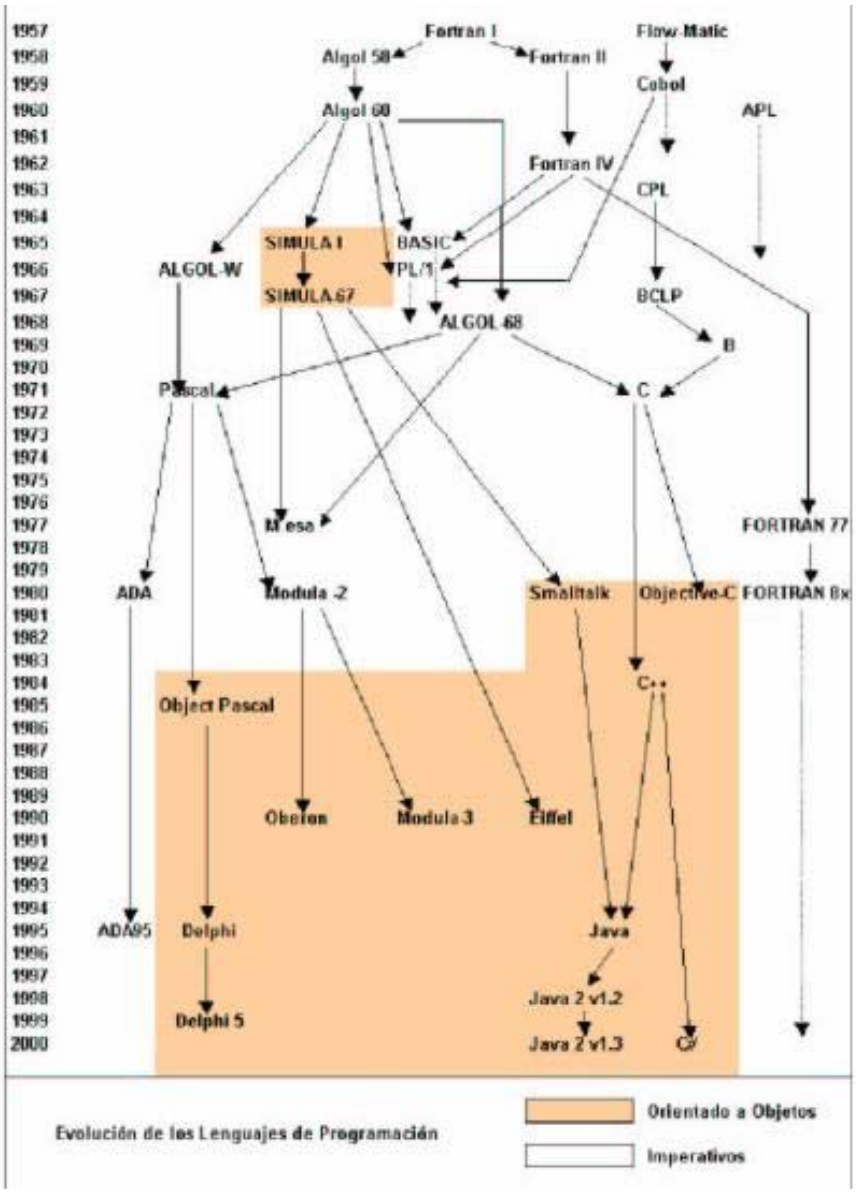

Fig 1. Evolución de los lenguajes (Romero Vargas)

1.2.2 Definición

“Un lenguaje de programación es un conjunto de normas lingüísticas que permiten codificar un programa para que sea entendido por el ordenador. “ (Mora Rioja, 2005)
Es una notación para escribir programas, a través de los cuales el desarrollador se puede comunicar con el hardware y dar así las órdenes adecuadas para la realización de un determinado proceso. (2005)
Se nombra lenguaje de programación al lenguaje que entienden las computadoras y con el cual el humano escribe un programa, es decir, un conjunto de palabras y símbolos que permiten escribir un programa que sea entendido por el ordenador.

1.2.3 Clasificación

Lenguaje de Bajo Nivel:

El lenguaje de bajo nivel es completamente dependiente de la máquina, un programa desarrollado con este lenguaje no se puede migrar o utilizar en otra máquina.
Este lenguaje aprovecha totalmente las características del hardware, dado que está diseñado a la medida de este. Existen dos clasificaciones fundamentales de lenguajes dentro de este grupo:

- Lenguaje máquina: El lenguaje máquina es el único que entiende directamente la computadora, utiliza el alfabeto binario. Las instrucciones escritas en cualquier lenguaje de máquina tienen por lo menos dos partes. La primera es la operación o el comando, que le muestra a la computadora la función que va a realizar. La segunda parte de la instrucción es el operando, que indica a la computadora donde hallar, almacenar los datos e instrucciones que se van a manipular. (2004)
- Lenguaje Ensamblador: Es el lenguaje que constituye la representación más directa del código máquina específico para cada arquitectura de computadoras. Actualmente se utiliza cuando se necesita manipular directamente el hardware o se pretenden rendimientos inusuales en los equipos.

Lenguaje de Alto Nivel:

Se encuentra más cercano al lenguaje natural que al lenguaje máquina. Dirigido principalmente a la solución de problemas mediante el uso de Estructuras Dinámicas de Datos (EDD).

Es el lenguaje más utilizado por los programadores. Está diseñado para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensamblador. Un programa escrito en lenguaje de alto nivel es independiente de la máquina (las instrucciones no dependen del diseño del hardware o de una computadora en particular). Además puede ser ejecutado con poca o ninguna modificación en diferentes tipos de computadoras. Es un lenguaje de programación en el que las instrucciones enviadas para que el ordenador ejecute ciertas órdenes son similares al lenguaje humano. Es necesario el uso de un intérprete que traduzca el lenguaje de alto nivel a un lenguaje de bajo nivel que el sistema pueda entender, ya que el ordenador no es capaz de reconocer estas órdenes.

1.3 Compilador

"Un compilador es un programa que lee un programa escrito en un lenguaje, el lenguaje fuente, y lo traduce a un programa equivalente en otro lenguaje, el lenguaje objeto" (Aho, y otros, 1998)

"Un compilador acepta programas escritos en un lenguaje de alto nivel y los traduce a otro lenguaje, generando un programa equivalente independiente, que puede ejecutarse tantas veces como se quiera. Este proceso de traducción se conoce como compilación". (2006)

"Los compiladores son programas o herramientas encargadas de compilar. Un compilador toma un texto (código fuente) escrito en un lenguaje de alto nivel y lo traduce a un lenguaje comprensible por las computadoras (código objeto)". (Definición de compilador, 2006)

Un compilador lee completamente un programa en un lenguaje de alto nivel y lo traduce en su integridad a un programa de código de máquina equivalente. El compilador le reporta al usuario la presencia de errores en el código del programa. (V. Aho, et al., 1986). Existen varios ejemplos de compiladores en GNU/Linux, entre ellos se encuentran g++, gcc.

1.4 Intérprete

1.4.1 Definición

"Un intérprete es un software que recibe un programa en lenguaje de alto nivel, lo analiza y lo ejecuta. Para analizar el programa completo, va traduciendo sentencias de código y ejecutándolas si están correctas, así hasta completar el programa origen". (Enciclopedia Libre Universal en Español, 2007)

Un intérprete lee un programa escrito en un lenguaje de alto nivel instrucción a instrucción y, para cada una de ellas, efectúa una traducción a las instrucciones de código de máquina equivalentes y las ejecuta inmediatamente. No hay un proceso de traducción separado por completo del de ejecución. Cada vez que ejecutamos el programa con un intérprete, se repite el proceso de traducción y ejecución, ya que ambos son simultáneos. (Marzal, y otros, 2003)

Un intérprete se caracteriza por traducir y ejecutar, de una en una, las instrucciones del código fuente de un programa, pero, sin generar como salida código objeto. El proceso que realiza un intérprete es el siguiente: lee la primera instrucción del código fuente y la ejecuta; a continuación de forma similar con la segunda instrucción y así sucesivamente, hasta llegar a la última instrucción del programa. Si se produce un error se detiene inmediatamente el proceso. Aunque existen estrategias de recuperación de errores que permiten proseguir analizando si las restantes líneas del código fuente son correctas.

1.4.2 Estructura

Para construir un intérprete es conveniente utilizar una Representación Interna (RI) del lenguaje fuente a analizar. De esta forma, la organización interna de la mayoría de los intérpretes se descompone en los módulos siguientes:

Traductor a Representación Interna: Toma como entrada el código del programa P en
Lenguaje Fuente, lo analiza y lo transforma a la representación interna correspondiente a dicho programa. Como parte de la traducción a representación interna ocurren varios procesos, por ejemplo:
Análisis Léxico, Sintáctico y Semántico.

Representación Interna (P/RI):
Código intermedio generado que posteriormente se ejecuta por la máquina virtual.

Tabla de símbolos: En el transcurso del proceso de traducción, resulta conveniente crear una tabla con información relativa a los símbolos que van apareciendo. La información a almacenar en la tabla depende de la complejidad del lenguaje fuente.

Se pueden almacenar información para instrucciones de salto, identificadores (nombre, tipo, línea en la que aparecen), etiquetas, o cualquier otro tipo de información que se necesite durante las distintas fases del intérprete.

Evaluador de Representación Interna: Partiendo de los datos de entrada y de la Representación Interna anterior, se llevan a cabo las acciones indicadas para obtener los resultados. Es necesario durante el proceso de evaluación ir contemplando la aparición de errores.

Tratamiento de errores: Mientras transcurre el proceso de evaluación pueden aparecer un conjunto de errores como desbordamiento de la pila, divisiones por cero, errores que el intérprete debe contemplar.

"Dependiendo de la complejidad del código a analizar, el intérprete puede contener módulos similares a los de un compilador tradicional: Análisis Léxico, Sintáctico y Semántico. Durante la evaluación, el intérprete interactúa con los recursos del sistema, tales como la memoria, discos, etc." (Cabello Morales, y otros)

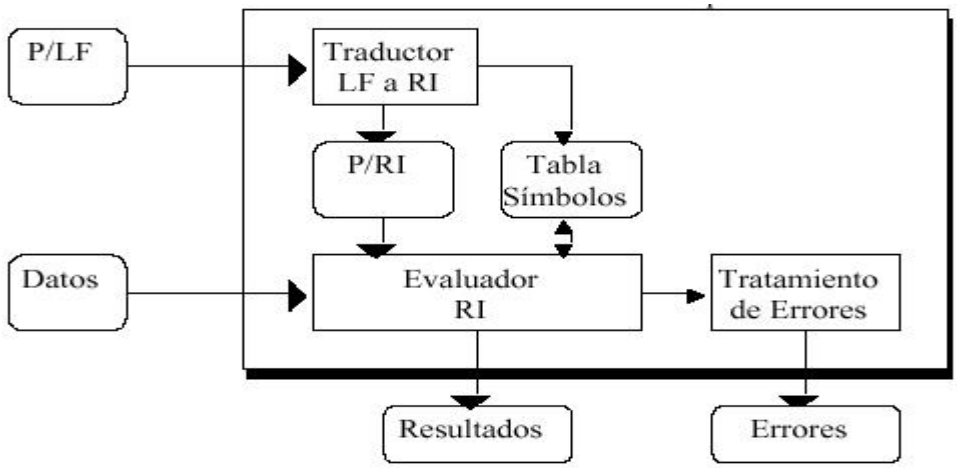

Fig 2. Organización interna de un intérprete

1.4.3 Intérprete vs Compilador

Los intérpretes informáticos, a diferencia de los compiladores, no generan un fichero ejecutable u otro programa equivalente en otro lenguaje.

Los intérpretes ejecutan los programas más lentamente, pues al tiempo de ejecución del código de máquina se suma el que consume la traducción simultánea. Además, un compilador puede examinar el programa de alto nivel abarcando más de una instrucción cada vez, por lo que es capaz de producir mejores traducciones. Un programa interpretado suele ser mucho más lento que otro equivalente que haya sido compilado (entre 2 y 100 veces más lento). (Marzal, y otros, 2003)

Mientras un intérprete toma las instrucciones del programa fuente, las traduce y ejecuta a lenguaje máquina una a una, un compilador realiza la traducción completa del programa fuente a código máquina, sin ejecutarlo, siendo posteriormente cuando se ejecute el programa una vez compilado.

Los intérpretes permiten una mayor flexibilidad que los compiladores y ciertos lenguajes de programación de alto nivel han sido diseñados para explotar esa mayor flexibilidad. Otros lenguajes de programación, por el contrario, sacrifican la flexibilidad en aras de una mayor velocidad de ejecución. La flexibilidad permite modificar y ampliar características del lenguaje fuente. Muchos lenguajes como Lisp, APL, Prolog, etc. surgieron en primer lugar como sistemas interpretados y posteriormente surgieron compiladores.
Los intérpretes, en general, son más sencillos de implementar, lo cual facilita el estudio de la corrección del intérprete y proporciona nuevas líneas de investigación como la generación automática de intérpretes a partir de las especificaciones semánticas del lenguaje. (Labra, y otros, 2004)
No es necesario contener en memoria todo el código fuente. Esto permite su utilización en
sistemas de poca memoria o en entornos de red, en los que se puede obtener el código fuente a medida que se necesita. (Plezbert, y otros, 1997)
Facilitan la meta-programación. Un programa puede manipular su propio código fuente a
medida que se ejecuta. Esto facilita la implementación de sistemas de aprendizaje automatizados. (Ait Kaci, 1991)
Aumentan la portabilidad del lenguaje: Para que el lenguaje interpretado funcione en otra
máquina sólo es necesario que su intérprete funcione en dicha máquina.
Puesto que no existen etapas intermedias de compilación, los sistemas interpretados facilitan el desarrollo rápido de prototipos, potencian la utilización de sistemas interactivos y facilitan las tareas de depuración. (Labra, y otros, 2004)

1.4.4 Lenguajes Interpretados.

Los lenguajes interpretados más usados son:

BASIC	(Código de instrucciones simbólico de propósito general para principiantes). El lenguaje BASIC fue diseñado por los profesores John G. Kemeny y Thomas E. Kurtz del Dartmouth College (Estados Unidos) en 1965, con el principal objetivo de proporcionar a los principiantes un lenguaje fácil de aprender. Es un lenguaje interactivo muy popular que tiene aceptación debido a la facilidad de su uso, útil para aplicaciones técnicas y de gestión. (Obregón)
JavaSript, VBScript	JavaSript es un lenguaje utilizado principalmente en páginas web. El lenguaje fue inventado por Brendan Eich en la empresa Netscape Communications, que fabricó los primeros navegadores web comerciales. Es un complemento del lenguaje HTML, le permite a la página realizar varias tareas por si misma, sin necesidad de estar sobrecargando el servidor del cual depende. (Obregón)
Perl, PHP	Perl es un lenguaje especializado en el procesamiento de textos, particularmente extraer y validar las respuestas a cuestionarios incluidos en páginas web., diseñado por Larry Wall, creado en 1987. (Obregón) PHP es un lenguaje para la programación de páginas web dinámicas, creación de aplicaciones gráficas independientes del navegador, programación en consola. Se acopla al HTML para determinar procedimientos que ha de realizar el servidor web. (Obregón) Una de sus fortalezas más significativas lo constituye la amplia comunidad que contribuye a su desarrollo.
Batch, Python	Batch (intérprete de comandos de Unix). Un fichero batch es un archivo de texto compuesto por comandos a ejecutar en un intérprete de comandos. Se utiliza para ejecutar series de comandos automáticamente. Generalmente el fichero batch presenta la extensión **.bat** Python es un lenguaje de programación interpretado e interactivo, capaz de ejecutarse en gran cantidad de plataformas. Fue creado por Guido Van

	Rossum en 1990. La sintaxis de Python es muy elegante, legible, además, ofrece un entorno interactivo que facilita la realización de pruebas. “El entorno de ejecución de Python detecta muchos de los errores de programación que escapan al control de los compiladores y proporciona información para detectarlos y corregirlos. Posee un juego de estructuras de datos que se pueden manipular de modo sencillo.” (Marzal, y otros, 2003)
Java	Java incorpora en el propio lenguaje muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads, ejecución remota, seguridad, componentes, acceso a bases de datos, etc.). “El principal objetivo del lenguaje Java es llegar a ser el “nexo universal” que conecte a los usuarios con la información, este está situado en el ordenador local, en un servidor de Web, en una base de datos o en cualquier otro lugar.” (Obregón) Algunos de los inconvenientes que presenta la Máquina Virtual de Java (MVJ) son: • Difícil de ampliar. Producto de la utilización de un byte para codificar el código de operación de las instrucciones del procesador virtual Java (bytecode), es difícil agregar nuevas instrucciones. • “No posee un árbol de análisis sintáctico. El código intermedio, usado en la MVJ, es simple y plano, es decir no incluye información acerca de la estructura del método original.” (Menchaca Méndez, y otros)

1.4.5 X-Talk

La utilización del paradigma de desarrollo X-Talk como lenguaje a interpretar, representa numerosas facilidades, sobre todo para profesores y tutores, quienes podrían usar un lenguaje de programación amigable, fundamentalmente en el desarrollo de herramientas para el apoyo al aprendizaje.

1.4.6 Funcionalidades Principales

1. Flexibilidad: permite realizar acciones complejas, imposibles o muy difíciles con un compilador, como las siguientes:

    1. Ejecución de cadenas de caracteres mediante operadores como "ejecute", "interprete" o "evalquote".
    2. Cambiar sobre la marcha el significado de los símbolos e incluso prescindir por completo de las declaraciones.
    3. Simplificar la gestión de memoria en los programas fuente.

2. Facilidad de depuración de programas: la interpretación puede interrumpirse en cualquier momento para modificar o examinar los valores de las variables en la ejecución. La tabla de símbolos está disponible. Se pueden corregir los errores y continuar.
3. Rapidez en el desarrollo.

1.5 Sistemas de Gestión del Aprendizaje

1.5.1 Definición

"Un SGA (plataforma de teleformación, entorno virtual de enseñanza-aprendizaje, sistema telemático de teleformación) es un software que se utiliza para la creación, gestión y distribución de cursos". (2000)
Es el lugar donde alumnos, profesores, tutores, interactúan con los diferentes contenidos.

1.5.2 Antecedentes y Actualidad de los SGA

El objetivo fundamental de los SGA es el de servir de contenedor de cursos, pero también incorporan otras herramientas para facilitar la comunicación y el trabajo colaborativo entre profesores y estudiantes, herramientas de seguimiento y evaluación del alumno, etc. Puede contar, entre otras, con herramientas de generación de contenidos y actividades, herramientas de comunicación, herramientas de gestión

administrativa, herramientas informativas. Algunos ejemplos de las herramientas más populares que integran, son agendas, glosarios, foros, chat, audioconferencias, videoconferencias, estadísticas, etc. (Gromaz Campos, y otros, 2008)

Un elemento fundamental para que un SGA cumpla con su objetivo principal es el empaquetamiento correcto de los objetos de aprendizaje (OA). Los OA son las porciones más pequeñas de instrucciones, informaciones que conforman un curso.

Se requiere entonces que un curso esté creado a partir de lo expuesto anteriormente, ya que los objetos pueden ser reutilizados en diversos sistemas y cursos, o sea, que sea posible migrar los contenidos a otros sistemas.

Un objeto de aprendizaje es un elemento esencial dentro de la construcción de un curso. Para construir un curso, se parte de los objetos de aprendizaje, los cuales pueden ordenarse dependiendo de las necesidades del estudiante. “Es necesario etiquetar (“poner datos”) a estos objetos de aprendizaje, con la finalidad de poder localizarlos fácilmente dentro de los archivos para que estén a disposición siempre que se requieran.” (Castro Solis)

1.6 Sistema Emedia

El sistema Emedia tiene como objetivo desarrollar y gestionar herramientas para el aprendizaje. Tiene dentro de sus prioridades prescindir de las dificultades existentes en los SGA actuales. Contribuyendo de esta forma, al incremento de la utilización de tecnologías informáticas por parte de los profesores, ya que la mayoría solo trabaja con el PowerPoint y no saben o no confían en la utilización de otros medios.

“Emedia, al facilitarle lo suficiente la forma en que pueden utilizar los nuevos medios, logra que se sientan libres (producto de la flexibilidad que le brindamos) de implementar lo que ellos crean.” (Fung Goizueta, 2006)

Con la preparación de los cursos, la obtención de información de los mismos y evaluación de los resultados, se amplía la participación de los expertos en contenidos, llevando de la mano además elementos tan importantes e imprescindibles como son, presupuestos de tipo pedagógico, de gestión académica y organizativa. (Fung Goizueta, 2006)

“El echo de que podemos dar soluciones a la medida es una gran ventaja. Podemos dar y producir cursos o elementos educativos para los objetivos específicos de cada cliente, a partir de la flexibilidad de la arquitectura y la concepción del proyecto. Nuestros productos y nuestra tecnología permite el ciclo de

investigación acción para que los profesores comprendan el fenómeno del e-learning y se puedan inmiscuir más en este asunto, otra razón son los precios, el hecho que estamos desarrollando con software libre." (Fung Goizueta, 2006)

El sistema comprende la creación, producción y gestión de cursos, tal y como lo desee el cliente. La tecnología usada contribuye a que los tutores, profesores se integren más al proceso de desarrollo de los elementos educativos.

El sistema Emedia está compuesto por un grupo de aplicaciones básicas:

- **Herramienta de Autor de Contenidos:** Es un elemento del sistema que automatiza las acciones de creación, organización y estandarización del proceso de producción de cursos. Le permite al especialista en contenidos convertir su experiencia y conocimientos en información fácil de usar por los desarrolladores, garantiza la coherencia entre lo que desea hacer el experto en contenido, lo que tiene que diseñar, y lo que ha de producirse e implementarse para la ejecución correcta en los sistemas informáticos. (Requisitos, generalidades de Emedia)
- **Herramienta de Autor de Objetos de Aprendizaje:** "La herramienta de producción de objetos automatiza el proceso de montaje del curso, se utiliza para chequear la implementación de todas las componentes del curso, la ejecución de cada OA para su validación, así como el análisis de un curso que esté en el repositorio." (Requisitos, generalidades de Emedia)
- **Repositorio:** Recibe paquetes de contenidos, guiones de contenidos. Además gestiona, almacena los paquetes y cambia la descripción de ellos. (Requisitos, generalidades de Emedia)
- **SGA:** Contiene un gestor de contenidos, gestor de usuarios, gestor de navegación, módulo de comunicaciones, módulo de estadísticas, así como un conjunto de funcionalidades adicionales. (Requisitos, generalidades de Emedia)

1.7 Intérprete para Emedia

Debido a la necesidad de brindar una variante para el desarrollo y manipulación de los OA en el sistema Emedia desde algún lenguaje interpretado, se hace necesario el desarrollo de un intérprete.

Se llega a esta conclusión luego de un estudio de los intérpretes existentes y más utilizados como el intérprete de Python y el de Java, que a pesar de tener muchas ventajas presentan un grupo de características que van en contra de los objetivos planteados, la fundamental es que estos intérpretes no pertenecen al lenguaje nativo del experto en contenido, el español.

El intérprete a desarrollar formaría parte del "Proyector".

El Proyector utilizará un intérprete al visualizar OA que hayan sido parcial o completamente desarrollados a través del lenguaje interpretado que se utilice. Se hace necesario también el uso de un intérprete en las herramientas autor, por ejemplo en el Autor de Contenidos se puede definir la navegación a través de un lenguaje interpretado.

Con la elaboración de un modelo flexible del intérprete se puede lograr por ejemplo, cambiar de idioma el lenguaje con suma facilidad, agregando al sistema nuevos módulos.

El intérprete del sistema Emedia debe brindar la posibilidad de interpretar código de alto nivel, utilizando varios lenguajes interpretados y garantizando que al menos uno sea cercano al experto en contenido, sea fácil de entender, amigable. La utilización de lenguajes X-Talk tiene como ventaja el acercamiento del experto en contenidos al proceso de desarrollo de los OA.

1.8 Software Libre

1.8.1 Definición

El software libre es un tipo particular de software que le permite al usuario el ejercicio de cuatro libertades básicas:

- Estudiar cómo funciona el programa, y adaptarlo a las necesidades individuales. El acceso al código fuente es una condición previa para esto.
- Usar el programa, con cualquier propósito.
- Distribuir copias, con lo que el individuo puede ayudar a su vecino.
- Mejorar el programa y hacer públicas las mejoras a los demás, con el objetivo de que toda la comunidad se beneficie. El acceso al código fuente es un requisito previo para esta libertad.

  (La definición de software libre)

1.8.2 Ventajas:

Ventajas del Software Libre:

1- Escrutinio Público: Al ser muchos las personas que tienen acceso al código fuente, eso lleva a un proceso de corrección de errores muy dinámico, no hace falta esperar que el proveedor del software saque una nueva versión.
2- Mayor seguridad y privacidad: Los sistemas de almacenamiento y recuperación de la información son públicos. Cualquier persona puede ver y entender como se almacenan los datos en un determinado formato o sistema. Existe una mayor dificultad para introducir código malicioso.
3- Garantía de continuidad: El software libre puede seguir siendo usado aun después de que haya desaparecido la persona que lo elaboró, dado que cualquier técnico informático puede continuar desarrollándolo, mejorándolo o adaptándolo.
4- Ahorro en costos: En cuanto a este tópico debemos distinguir cuatro grandes costos: de adquisición, de implantación (este a su vez se compone de costos de migración y de instalación), de soporte o mantenimiento, y de interoperabilidad. El software libre principalmente disminuye el costo de adquisición ya que al otorgar la libertad de distribuir copias la puedo ejercer con la compra de una sola licencia y no con tantas como computadoras posea (como sucede en la mayoría de los casos de software propietario). Cabe aclarar que también hay una disminución significativa en el costo de soporte, no ocurriendo lo mismo con los costos de implantación y de interoperabilidad.

1.8.3 Importancia

El desarrollo de software bajo códigos abiertos permite entender cómo funcionan estos programas y quién controla las máquinas en donde están instalados.

Existe un modelo de negocios para los que trabajan con software libre y con el concepto de propiedad GPL, que otorga el permiso de distribuir las copias pero bajo las mismas condiciones en que las recibió.

"Lo que hace el software libre es reducir el alcance del derecho de autor a su mínima expresión". (ABN, 2008)

Los desarrolladores no tienen que trabajar desde cero para hacer programas porque existen muchas aplicaciones que se pueden reutilizar.

El modelo de negocios con software libre está dirigido a prestar servicios como el desarrollo de programas, el soporte y entrenamiento.

Para el desarrollo del software libre es necesario el licenciamiento abierto que permita la reaplicación de los programas.

Al estar libremente disponibles los códigos fuente de los sistemas operativos, lenguajes de programación, bibliotecas, interfaces de usuario y demás aplicaciones básicas, las empresas pueden desarrollar utilidades y programas basados en ellos compitiendo por ofrecer una mejor calidad.

1.9 **Herramientas a utilizar**

1.9.1 Herramienta de modelado Rational Software Arquitect

Para la modelación del intérprete se utilizó la herramienta Rational Software Arquitect (RSA).

"Rational Software Architect proporciona un soporte de desarrollo y diseño integrado para el desarrollo dirigido por el modelo con UML". (Rational Software Architect)

Es una herramienta de diseño y desarrollo integrados que potencia el desarrollo orientado al modelado con UML para la creación de aplicaciones y servicios con buena arquitectura.

RSA es una herramienta de modelado y diseño visual, basada en el lenguaje UML (Unified Modeling Languaje).

Con RSA se pueden unificar todos los aspectos del desarrollo y el diseño de software.

1.9.2 Eclipse

Para realizar el prototipo de interfaz del intérprete se investigó sobre la herramienta Eclipse.

Eclipse es un Entorno Integrado de Desarrollo (IDE) multiplataforma libre para crear aplicaciones clientes de cualquier tipo. Fue creado originalmente por International Business Machines Corporation (IBM), ahora lo desarrolla la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse emplea módulos para proporcionar toda su funcionalidad. El mecanismo de módulos permite que el entorno de desarrollo soporte otros lenguajes además de Java. Por ejemplo, existe un módulo para dar soporte a C/C++ +.

1.9.3 Bison

Bison es un generador de analizadores sintácticos de propósito general que convierte una descripción gramatical para una gramática independiente del contexto (LALR) en un programa en C/C++ que analice esa gramática.

El proceso real de diseño de lenguajes utilizando Bison, desde la especificación de la gramática hasta llegar a un compilador o intérprete funcional, se compone de estas etapas:

1. Especificar formalmente la gramática en un formato que reconozca Bison. Para cada regla gramatical en el lenguaje, describir la acción que se va a tomar cuando una instancia de esa regla sea reconocida. La acción se describe por una secuencia de sentencias en C/C++.
2. Escribir un analizador léxico para procesar la entrada y pasar tokens al analizador sintáctico. El analizador léxico podría escribirse a mano en C/C++. Este puede también generarse utilizando otras herramientas.
3. Escribir una función de control que llame al analizador producido por Bison.
4. Escribir las rutinas de informe de errores.

Para hacer que este código fuente escrito se convierta en un programa ejecutable, es necesario seguir estos pasos:

1. Ejecutar Bison sobre la gramática para producir el analizador.
2. Compilar el código de salida de Bison, al igual que cualquier otro fichero fuente.
3. Enlazar los ficheros objeto para producir el producto final.

1.9.4 Lenguaje C++

El lenguaje C++ se desarrolló a partir de 1980 y su autor fue B. Stroustrup. El C++ era inicialmente era una extensión del lenguaje C que fue llamada "C with classes". (Lenguaje C++)

Para la implementación de la máquina virtual del intérprete se utilizará como lenguaje de programación C++.

Fue elegido C++ porque posee características superiores a otros lenguajes. Las más importantes son: programación orientada a objetos, portabilidad, brevedad, programación modular y velocidad.

Además, se trata de un lenguaje de programación estandarizado (ISO/IEC 14882:1998), ampliamente difundido, y con una biblioteca estándar C++ que lo ha convertido en un lenguaje universal, de propósito

general, y ampliamente utilizado tanto en el ámbito profesional como en el educativo. Válido para plataformas Windows y Unix/Linux.

CAPÍTULO 2: PROPUESTA DE DISEÑO DEL INTÉRPRETE

2.1 Introducción

En el presente capítulo se comienza a tener una visión del intérprete a diseñar y se dan los primeros pasos para su concepción práctica. Se capturan los requerimientos, se modelan los casos de uso del sistema y se define el diseño básico del intérprete.

2.2 Captura de Requerimientos

Los Requerimientos son condiciones o capacidades que deben estar presentes en un sistema o componentes de sistema.

2.2.1 Requerimientos Funcionales:

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir.

Los requerimientos funcionales del intérprete son:

RF1. Configurar la comunicación con la aplicación anfitriona.

RF2. Gestionar Código Fuente

- RF2.1. Recibir código fuente
- RF2.2. Ejecutar código fuente
  - RF2.2.1 Realizar el Análisis Léxico
    - RF2.2.1.1 Detectar errores léxicos
  - RF2.2.2 Realizar el Análisis Sintáctico
    - RF2.2.2.1 Detectar errores sintácticos
  - RF2.2.3 Realizar el Análisis Semántico
    - RF2.2.3.1 Detectar errores semánticos
  - RF2.2.4 Efectuar la Generación de Código intermedio
  - RF2.2.5 Ejecutar el código intermedio

RF2.3 Informar Errores.

2.2.2 Requerimientos No Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Son características que hacen al producto atractivo, usable, rápido o confiable.

Los requerimientos no funcionales que debe cumplir el sistema son:
RNF1. Fácil de usar

Tipo: Usabilidad

Cualquier persona con conocimientos básicos sobre la informática o el uso de las computadoras y con un conocimiento avanzado sobre la materia en curso debe poder darle uso al sistema.

RNF2. Montar el sistema sobre el sistema operativo GNU/Linux
Tipo: Requerimiento de Software
Una PC instalada con sistema operativo GNU/Linux.

RNF3 Lenguaje de desarrollo
Tipo: Restricciones en la Implementación
El intérprete será desarrollado en lenguaje C++

2.3 Modelo Casos de Uso del Sistema

2.3.1 Actores del Sistema

Los actores son los roles que un usuario o usuarios del sistema llevan a cabo en algún momento. También pueden ser otros sistemas con los que el sistema en proceso de modelado tenga interacción.

Tabla1

Actores	Justificación
Aplicación Anfitriona	Es quien va a interactuar directamente con el intérprete, módulo en el cual se montará el intérprete.

2.3.2 Descripción de los Casos de Uso del Sistema

Tabla2

CU -1	Configurar la comunicación con la aplicación anfitriona.
Actor.	Aplicación Anfitriona
Descripción.	Mediante este caso de uso se permitirá la comunicación del intérprete con la aplicación anfitriona.

Tabla3

CU -2	Gestionar Código Fuente.
Actor.	Aplicación Anfitriona
Descripción.	Mediante este caso de uso el intérprete analizará, ejecutará e informara los errores

	que presenta el código fuente.

2.3.3 Diagrama de Casos de Uso del Sistema

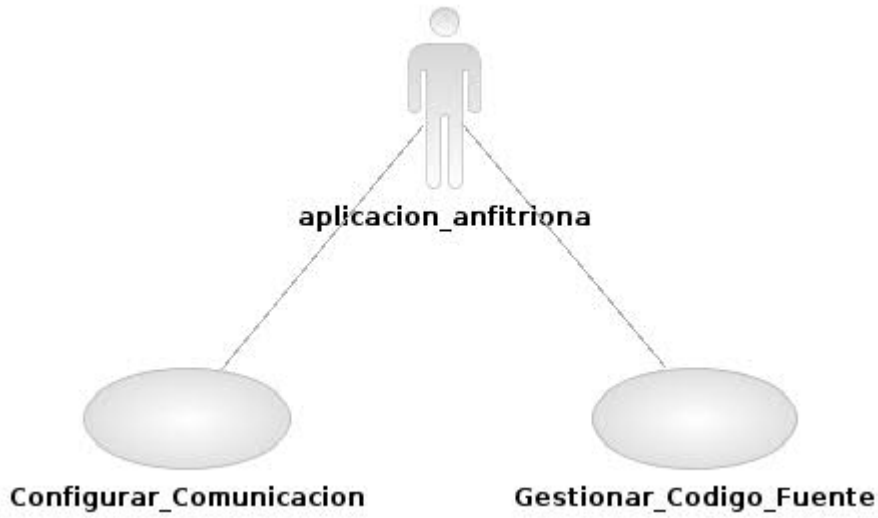

Fig 2. Diagrama de Casos de Uso del Sistema

2.3.4 Especificación de los Casos de Uso en formato expandido

Tabla4. Configurar la comunicación con la aplicación anfitriona

Caso de Uso:	Configurar la comunicación con la aplicación anfitriona
Actores:	Aplicación Anfitriona
Propósito	Permitir que el intérprete conozca y tenga acceso a las clases de la aplicación anfitriona.
Resumen:	El caso de uso es iniciado cuando la aplicación anfitriona empotra o

	monta el intérprete. Durante el curso del empotramiento el sistema le da a conocer al intérprete las clases y objetos a los cuales puede tener acceso.
Referencia	R1 Configurar la comunicación con la aplicación anfitriona.
Precondiciones	
Postcondiciones	

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. La aplicación anfitriona carga la biblioteca que contiene al intérprete e invoca al mecanismo que permitirá la inicialización del intérprete.	2. El intérprete comienza su proceso de inicialización.
3. El sistema le "entrega" a la entidad encargada en el intérprete referencias o punteros a las entidades de la aplicación anfitriona a las que el intérprete podrá acceder.	4. El intérprete recibe los punteros o referencias a las entidades de la aplicación anfitriona a las que podrá acceder, las almacena y prosigue con las inicializaciones correspondientes.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
Prioridad	**Media**
Prototipo de Interfaz	

Tabla5. Gestionar Código Fuente

Caso de Uso:	Gestionar Código Fuente.
Actores:	Aplicación Anfitriona
Propósito	Que el intérprete analice, ejecute e informe los errores del código fuente.
Resumen:	El caso de uso se inicia cuando la aplicación anfitriona le envía el código fuente al intérprete, posteriormente este lo analiza, lo ejecuta e informa los errores que tenga.
Referencia	R1. Configurar la comunicación con la aplicación anfitriona. R2. Gestionar Código Fuente
Precondiciones	La configuración del intérprete con la aplicación anfitriona debe estar establecida.
Postcondiciones	

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Envía el código fuente.	2. Recibe el código fuente.
	3. Ejecuta código fuente.
	4. Informa a la aplicación anfitriona si el código fuente pudo ser ejecutado con éxito.
5. Recibe respuesta si fue exitosa o no la ejecución del código fuente por parte del	6. Informa Errores.

intérprete.	
Sección Ejecutar código fuente	
Respuesta del sistema	
	3.1 Realiza el análisis Léxico del código fuente
	3.2 Detecta errores léxicos
	3.3 Realiza análisis Sintáctico
	3.4 Detecta errores sintácticos
	3.5 Realiza análisis Semántico.
	3.6 Detecta errores semánticos.
	3.7 Genera código intermedio.
	3.8 Ejecuta código intermedio.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Prioridad	Crítico
Prototipo de Interfaz	

2.4 Modelo de Dominio

2.4.1 Diagrama

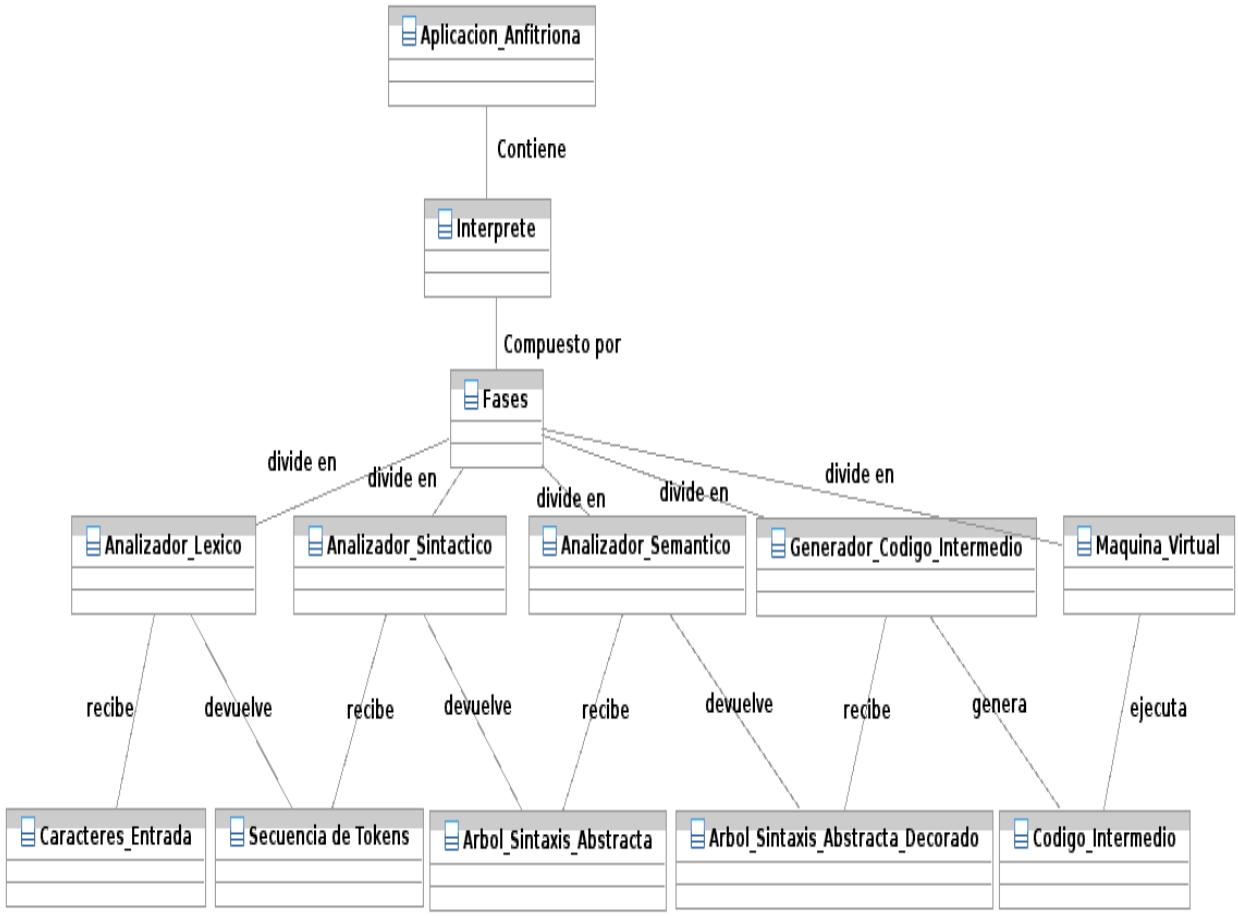

Fig 3. Modelo de dominio

2.4.2 Especificaciones del dominio

A continuación se presentan un conjunto de conceptos que conforman el glosario de términos del modelo del dominio, con el objetivo de facilitar el entendimiento de los términos manejados en el diagrama.

Aplicación Anfitriona: Sistema donde se va a empotrar el intérprete.

Intérprete: Software que recibe el programa en lenguaje de alto nivel, lo analiza y ejecuta.

Fases: Etapas que conforman el desarrollo del intérprete.

Analizador Léxico: Es la fase que lee el programa fuente carácter por carácter.

Analizador Sintáctico: Fase que toma los tokens que le envía el analizador léxico y comprueba si con ellos se puede formar alguna sentencia válida del lenguaje.

Analizador Semántico: Fase en la que se comprueba la corrección semántica de las instrucciones y se detectan errores semánticos.

Generador Código Intermedio: Traduce el programa fuente y genera el código intermedio.

Máquina Virtual: Fase final del intérprete que ejecuta el código intermedio.

Caracteres Entrada: Conjunto de símbolos, letras, dígitos que se entran por teclado.

Secuencia de Tokens: Secuencia de componentes Léxicos, unidades mínimas, con significados propios.

Árbol Sintaxis Abstracta: Estructura de datos que se utiliza para manejar la información semántica de un código.

Árbol Sintaxis Abstracta Decorado: Árbol de Sintaxis Abstracta ampliado con atributos.

Código Intermedio: Es una lista de tipos de valores enteros, generado por el generador de código intermedio, y que es ejecutado por la máquina virtual.

2.5 Arquitectura y Diseño de la aplicación

2.5.1 Clases del Análisis

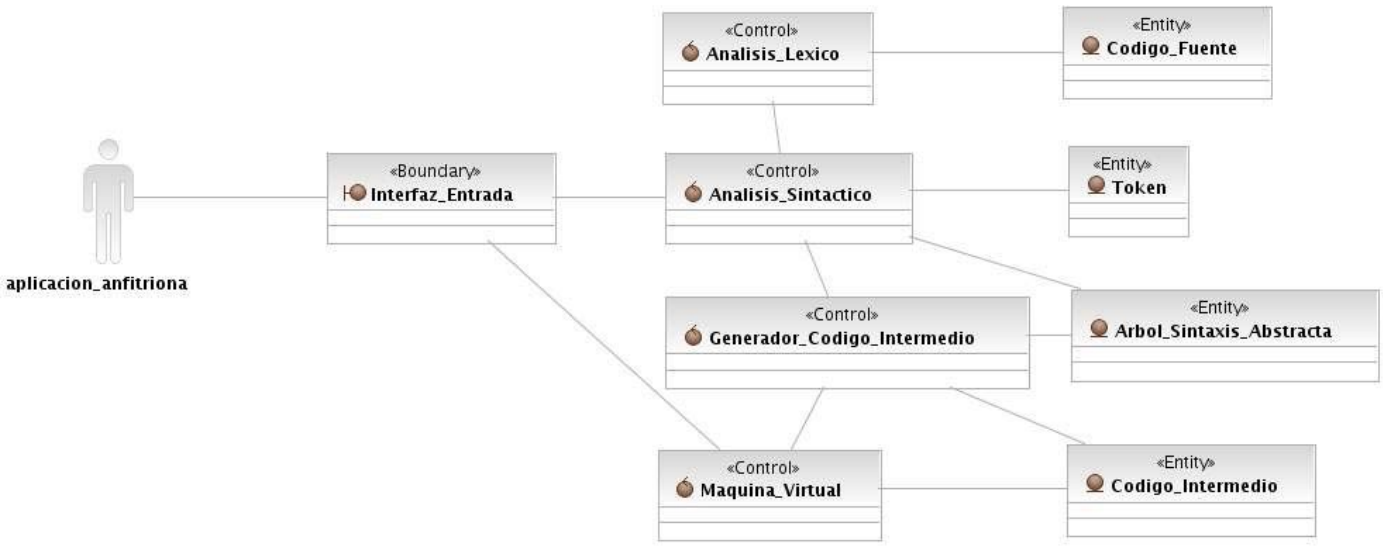

Fig 4. Diagrama de Clase Gestionar_Código_Fuente

2.5.2 Diagramas de Colaboración

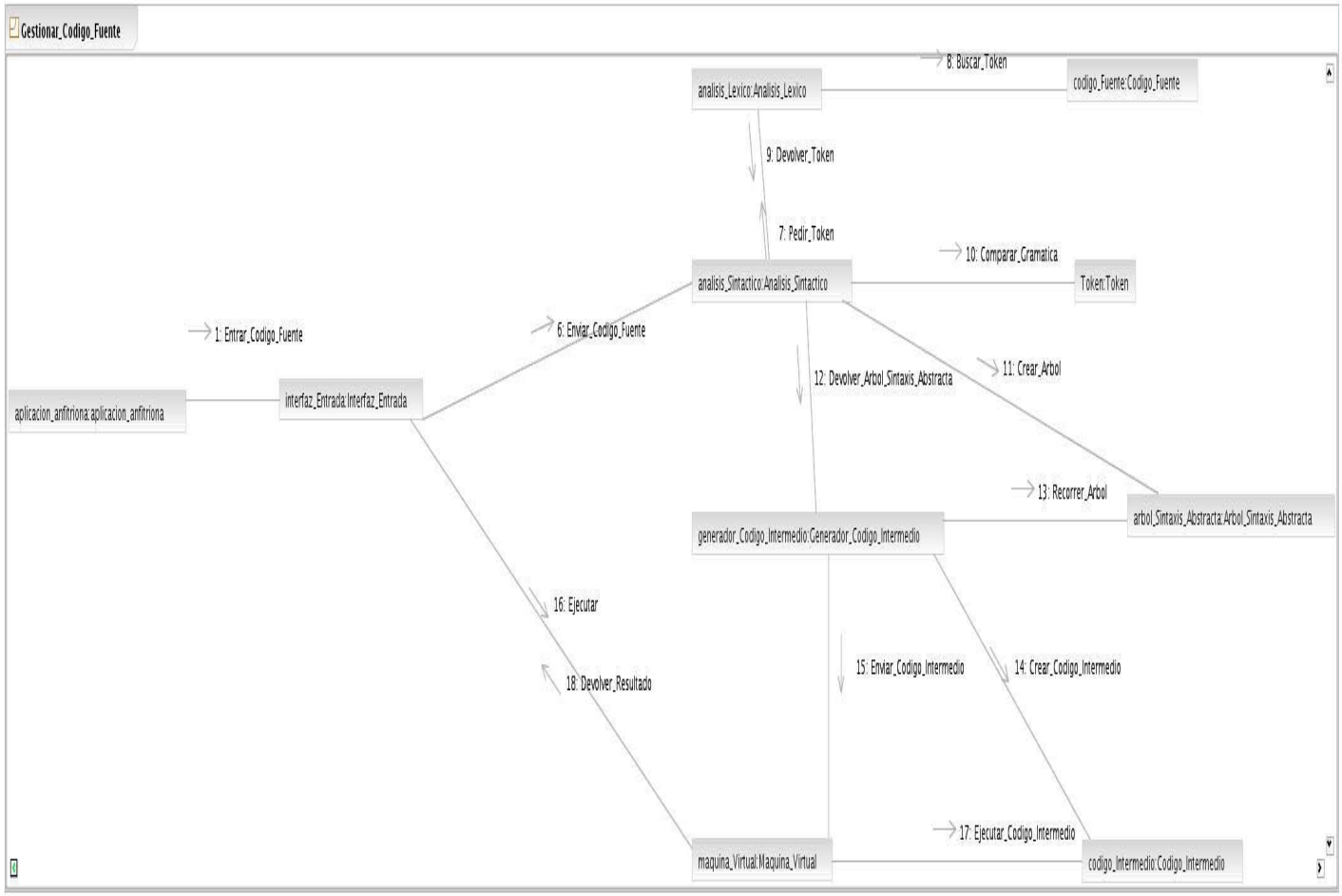

Fig 5. Diagrama de Colaboración de clase Gestionar_Código_Fuente

2.5.3 Diagrama de Clases del intérprete

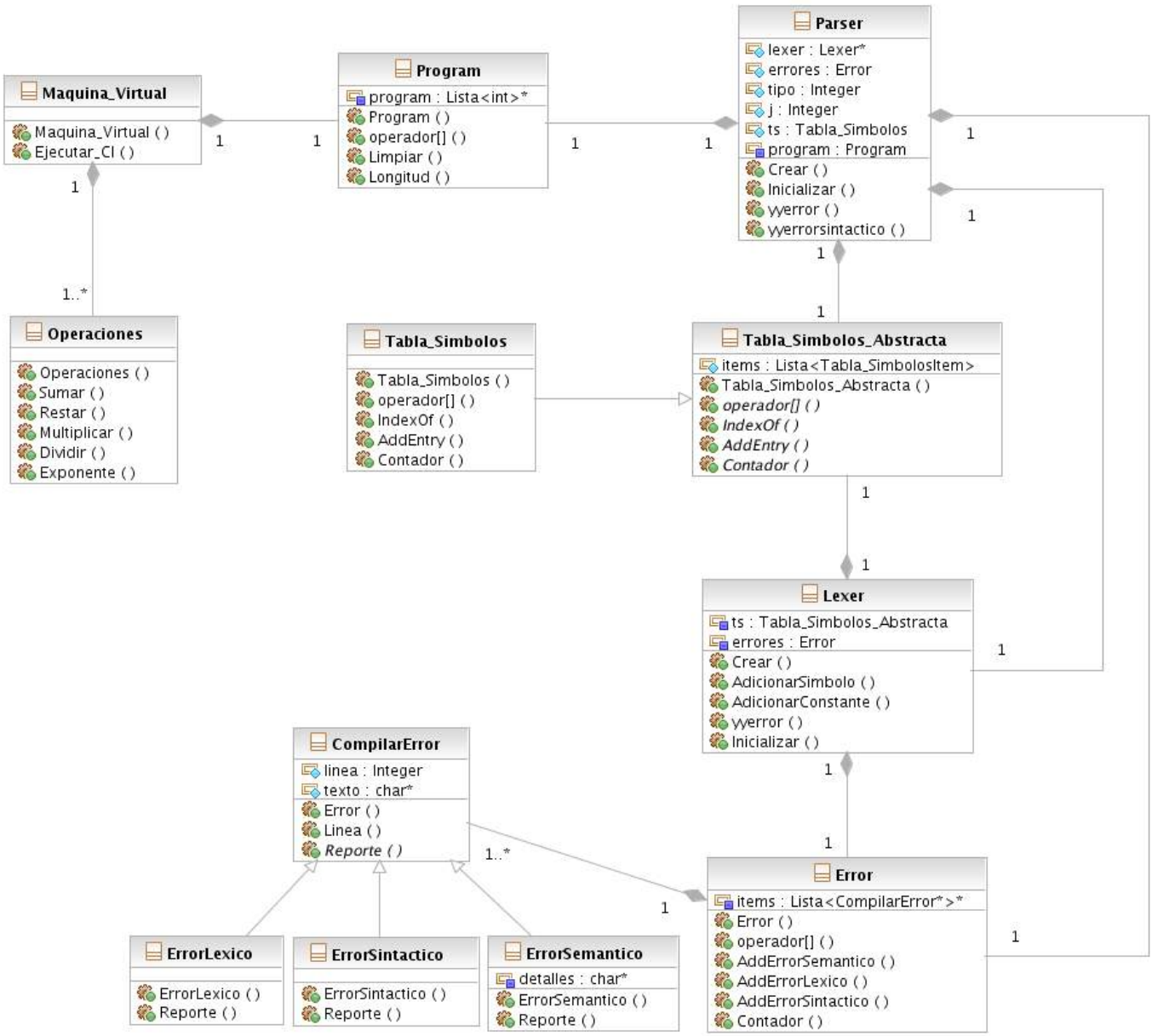

Fig 6. Clases del Intérprete

2.5.4 Diagrama de Despliegue

Fig 7. Diagrama de Despliegue

CAPITULO 3: ESPECIFICACIÓN DE LAS FASES DEL INTÉRPRETE E IMPLEMENTACIÓN DE LA MÁQUINA VIRTUAL.

3.1 Introducción

En este capítulo se definen el alfabeto y la primera versión de la gramática del lenguaje así como cada una de las etapas del intérprete, centrándose en la implementación de la Máquina Virtual para la ejecución del código intermedio. Además se describen las clases del intérprete y se analizan los estándares de codificación.

3.2 Definición del lenguaje a interpretar

3.2.1 Alfabeto.

Alfabeto (.): Se entiende por alfabeto un conjunto arbitrario, finito y no vacío, de símbolos. Entre los alfabetos más comunes podemos mencionar el alfabeto binario {0, 1}, el alfabeto latino {a, b, c, d, e, f, g, …}, etc.
Cadena: Una cadena es una secuencia de símbolos de cierto alfabeto colocados uno a continuación del otro.

Lenguaje: Un lenguaje sobre un alfabeto es un conjunto de cadenas sobre dicho alfabeto que obedece una cierta regla de formación.

3.2.2 Gramática

La gramática es un ente o modelo matemático que permite especificar un lenguaje, es decir, es el conjunto de reglas capaces de generar todas las posibilidades combinatorias de ese lenguaje, y sólo las de dicho lenguaje, ya sea éste un lenguaje formal o un lenguaje natural.

Los objetivos de una gramática son:

- Definir si una sentencia pertenece o no al un lenguaje.
- Describir estructuralmente las sentencias del lenguaje.

Definición: Una gramática es un cuádruplo G = {N, Q., P, S} donde:

N: Conjunto de símbolos no terminales.

Q: Conjunto de símbolos terminales.

P: Conjunto de Reglas de Producción.

S: Axioma o símbolo distinguido (S .N)

La primera versión de la gramática definida para el intérprete consta de reglas para realizar operaciones matemáticas básicas: suma, resta, multiplicación, división, potenciación.

A continuación se muestra la gramática definida utilizando la herramienta Bison:

```
input:    /* cadena vacía */
        | input linea
;
linea:   '\n'
      | exp '\n'   { printf ("\t%.10g\n", $1); }
      | error '\n' { yyerrok;                  }
;
exp:      NUM                {
                                  program[j++] = CARGA;
                                  program[j++] = ts[$1.entry].dir;
                                  $$ = ts[$1.entry];

                              }

        | exp  '+' exp       {
                                  program[j++] = SUM;
                                  $$ = $1 + $3;
                             }

       | exp  '-' exp        {
                                   program[j++] = SUB;
                                  $$ = $1 - $3;
                            }
```

```
    | exp  '*' exp       {
                              program[j++] = MULT;
                              $$ = $1 * $3;
                          }

    | exp  '/' exp       {
                             program[j++] = DIV;
                             $$ = $1 / $3;
                           }

    | '-'  exp  %prec NEG {
                              program[j++] = CARGA;
                              program[j++] = ts[$2.entry].dir;
                              $$ = ts[-$2.entry];
                           }

    | exp  '^' exp         {
                              program[j++] =  POW;
                              $$ = pow ($1, $3);
                           }

    | '('  exp ')'           {
                              program[j++] = CARGA;
                              program[j++] = ts[$2.entry].dir;
                              $$ = ts[$2.entry];
                           }
;
```

3.3 Especificación de cada fase.

3.3.1 Análisis Léxico.

“La fase de análisis léxico lee los caracteres en el programa fuente y los agrupa en una cadena de componentes léxicos en los que cada componente representa una secuencia lógicamente coherente de caracteres” (Aho, y otros, 1998)

El analizador léxico es la primera fase de un compilador. Su función principal es leer los caracteres de entrada y producir como salida una secuencia de tokens que el parser luego utiliza en

su análisis sintáctico. La interacción entre ambos componentes (analizador léxico y analizador sintáctico), representada en la Fig. 8, básicamente es de la siguiente forma: Ante un pedido de un token, el analizador léxico lee una secuencia de caracteres del código fuente del programa hasta identificar un token, el cual es retornado como respuesta al pedido del analizador sintáctico.

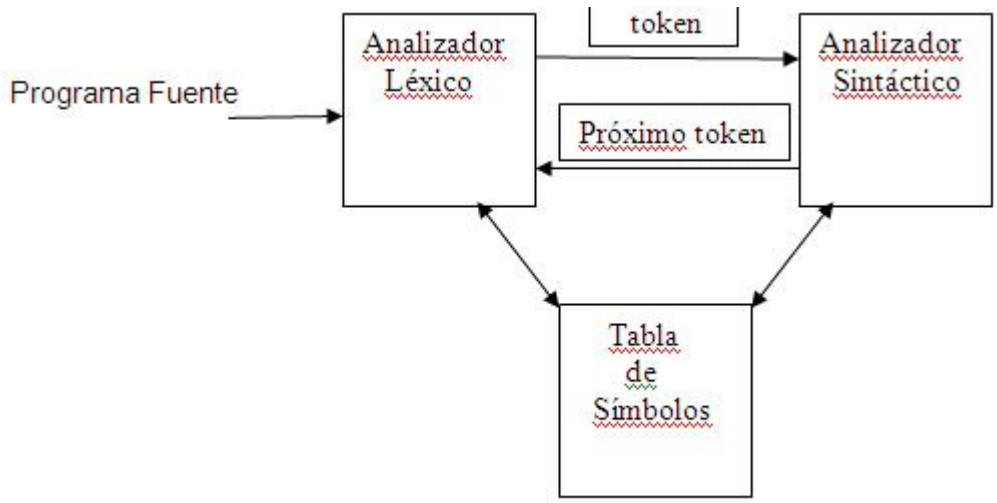

Fig 8. Interacción entre el A. Léxico y Sintáctico

Antes de implementar un Analizador Léxico para un lenguaje determinado, es necesario diseñar su especificación. Para ello hay que identificar primeramente la colección de tokens que compone el lenguaje. Cada token debe estar especificado mediante algún esquema generador o un esquema reconocedor, preferiblemente expresiones regulares. Luego es necesario Especificar el Diagrama de

Transición del Analizador Léxico.

Expresión regular del token NUM:

NUM→ ((1|...|9)+ (0)*)+ (.) (1|...|9)+ | (0) (.) ((0)* (1|...|9)+)+| ((1|....|9)* (0))*

Suma→ +
Resta→ -
Multiplicación→ *
División→/
Potenciación → ^
A continuación se representan los Diagramas de Transición del Analizador Léxico.

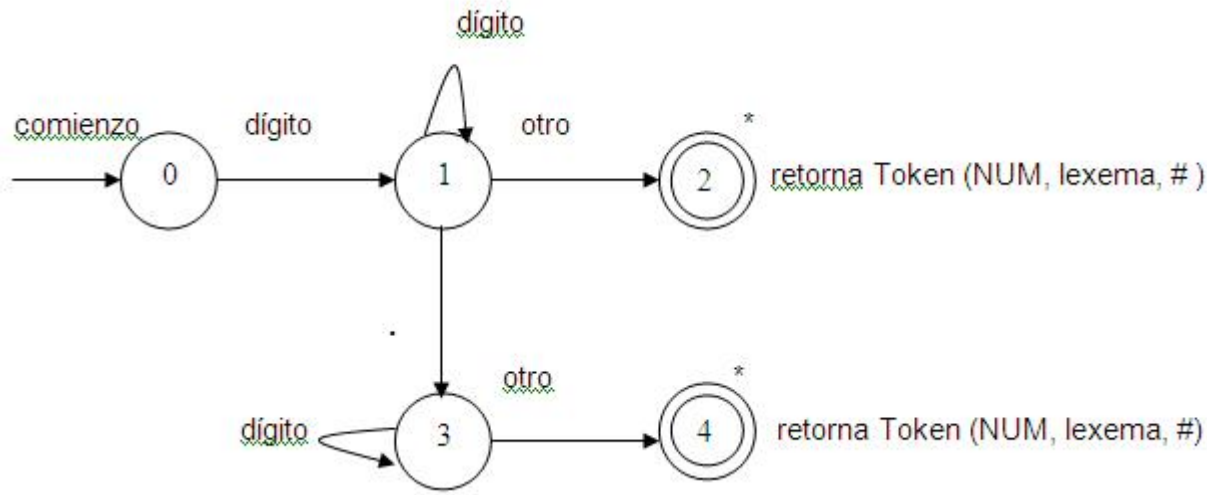

Fig 9. Diagrama de transición para el token NUM

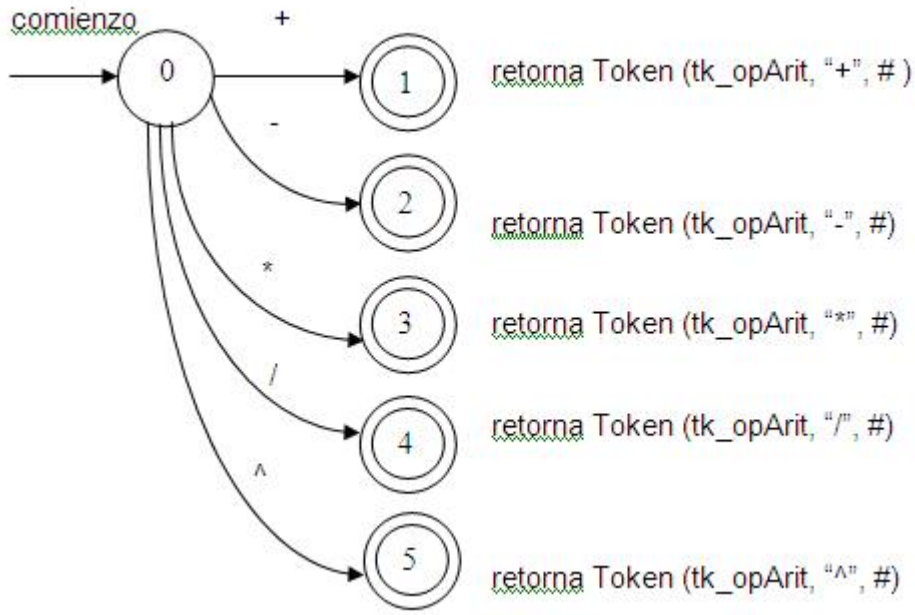

Fig 10. Diagrama de transición para los operadores aritméticos.

3.3.2 Análisis Sintáctico.

Comprueba si la secuencia de tokens que representa al texto de entrada, en base a una gramática dada es correcta. A medida que va validando que la secuencia de tokens cumpla con las reglas gramaticales correspondientes va generando el árbol de sintaxis abstracta que será utilizado en las posteriores fases del proceso de ejecución del intérprete.

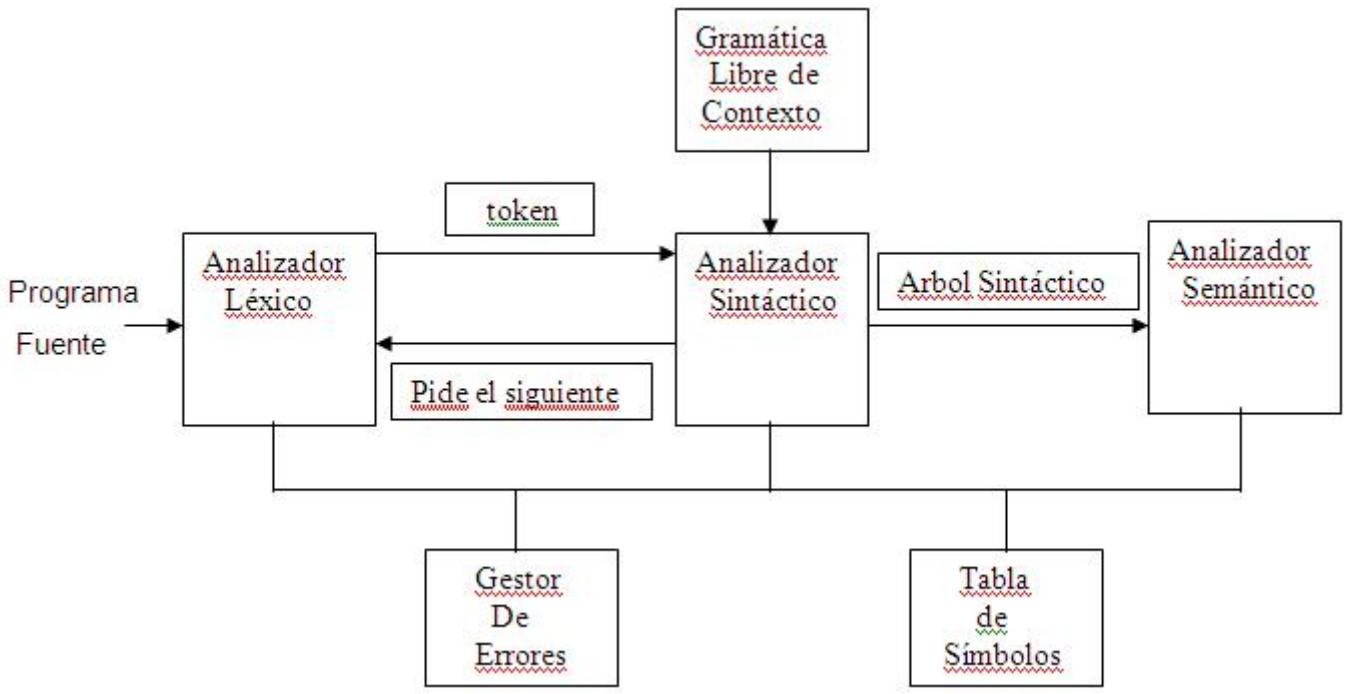

Fig 9. Interacciones del A. Sintáctico

En la práctica, el analizador sintáctico también realiza otro conjunto de acciones, las principales son:

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

3.3.3 Análisis Semántico.

El análisis semántico en un procesador de lenguaje es la fase encargada de detectar la validez semántica de las sentencias aceptadas por el analizador sintáctico. La sintaxis de un lenguaje de programación es aquella parte del lenguaje que puede ser descrita mediante una gramática libre de

contexto, mientras que el análisis semántico es la parte de su especificación que no puede ser descrita por la sintaxis.

"El análisis semántico dota de un significado coherente a lo que hemos hecho en el análisis sintáctico. El chequeo semántico se encarga de que los tipos que intervienen en las expresiones sean compatibles o que los parámetros reales de una función sean coherentes con los parámetros formales." (Gálvez Rojas, y otros, 2005)

3.3.4 Generación de Código Intermedio.

Esta fase recibe el árbol de sintaxis abstracta decorado y devuelve como salida el código intermedio. Existen 2 formas principales de representación del código intermedio:

- Representación Arbórea: árboles de sintaxis abstracta
- Representación Lineal: polaca inversa, código 3 direcciones. (Ribadas Pena, 2006)

Ejemplos de Representaciones Intermedias:

- Diana: representación arbórea usada en compiladores de ADA
- RTL (Register Transfer Language): usada en familia de compiladores GCC
- Código-P: compiladores Pascal (también común en intérpretes)
- WAM (Warren Abstract Machine): intérpretes Prolog
- Byte codes Java (Ribadas Pena, 2006)

En el sistema se emplea la representación lineal, la más factible para la gramática definida. Específicamente se trabaja con la Notación Polaca Inversa (RPN):

Notación postfija con los operadores situados a continuación de los operandos. Muy usado como código intermedio en intérpretes.

"En el momento de la evaluación de la notación polaca se utiliza una pila en la que se guardan los operandos en orden de aparición de izquierda a derecha por lo que el operador actual opera sobre los elementos (operandos) que se encuentran en el tope de la pila." (2007)

3.3.5 Ejecución de Código Intermedio.

Es la fase encargada de la ejecución del código generado por la fase anterior, en este caso de la cadena polaca generada, el código intermedio.

Explicando mejor el proceso: el código fuente se compila, detectando los errores sintácticos, y se genera el código intermedio. Para poder ejecutar el código, se construye un intérprete. Este intérprete es capaz de leer cada una de las instrucciones de código intermedio y ejecutarlas. Se le denomina a este intérprete "Máquina Virtual".

3.4 Descripción de las Clases

Tabla 5. Descripción clase Máquina Virtual

Nombre: Maquina_Virtual	
Tipo de clase: Controladora	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre	**Maquina_Virtual()**
Descripción	Constructor de la clase.
Nombre	**Operador()**
Descripción	Este método booleano recibe un carácter y va comparándolo con el arreglo de operadores declarado si coincide con uno de ellos retorna true, o sea, que el carácter es un operador, sino retorna false, que el carácter es un operando
Nombre	**Ejecutar_CI**(ListaSE<int>*p)
Descripción	Se analiza la expresión de izquierda a derecha. Si se encuentra un operando se coloca en la pila (se apila). En cambio, si se encuentra un operador se le aplicarán los dos operandos superiores de la pila y

	se elimina de la pila los dos operandos y se apila el resultado de la última operación. El proceso se repite hasta que se haya tratado a toda la expresión. El último valor que habrá en la pila será el resultado de analizar a la expresión.

Tabla 6. Descripción clase Programa

Nombre: Programa	
Tipo de clase: Entidad	
Atributo	**Tipo**
programa	Lista<int>*
Para cada responsabilidad	
Nombre	**Programa()**
Descripción	Constructor de la clase.
Nombre	**Operador[] ()**
Descripción	Se le pasa una posición en la lista de enteros, y devuelve el valor que se encuentra en esa posición.
Nombre	**Limpiar()**
Descripción	Limpia la lista de enteros, en otras palabras, la vacía.
Nombre	**Longitud()**
Descripción	Devuelve la cantidad de elementos de la lista

Tabla 7. Descripción clase Error

Nombre: Error

Tipo de clase: Entidad	
Atributo	**Tipo**
items	Lista<Compilar_Error*>
Para cada responsabilidad	
Nombre	**Error()**
Descripción	Constructor
Nombre	**Operador[]()**
Descripción	Se le pasa una posición en la lista de errores y devuelve el error que se encuentra en dicha posición
Nombre	**Add_Error_Semantico()**
Descripción	Adiciona a la lista el error semántico y devuelve la posición donde lo adicionó.
Nombre	**Add_Error_Lexico()**
Descripción	Adiciona a la lista el error léxico y devuelve la posición donde lo adicionó.
Nombre	**Add_Error_Sintactico()**
Descripción	Adiciona a la lista el error sintáctico y devuelve la posición donde lo adicionó.

Tabla 8. Descripción clase Compilar_Error

Nombre: Compilar_Error	
Tipo de clase: Entidad	
Atributo	**Tipo**
linea	int
texto	Char*

Para cada responsabilidad	
Nombre	**Compilar_Error()**
Descripción	Constructor
Nombre	**Linea()**
Descripción	Retorna un número de línea
Nombre	**Reporte()**
Descripción	Método virtual puro

Tabla 9. Descripción clase Error_Lexico

Nombre: Error_Lexico	
Tipo de clase: Entidad	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre	**Error_Lexico()**
Descripción	Constructor
Nombre	**Reporte()**
Descripción	Método que devuelve el mensaje de error, con el formato Error Lexico en línea ‘x’, ‘error’

Tabla 10. Descripción clase Error_Sintactico

Nombre: Error_Sintactico	
Tipo de clase: Entidad	
Atributo	**Tipo**

Para cada responsabilidad	
Nombre	**Error_Sintactico()**
Descripción	Constructor
Nombre	**Reporte()**
Descripción	Método que devuelve el mensaje de error, con el formato Error Sintáctico en línea ‘x’, ‘error’

Tabla 11. Descripción clase Error_Semantico

Nombre: Error_Semantico	
Tipo de clase: Entidad	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre	**Error_Semantico()**
Descripción	Constructor
Nombre	**Reporte()**
Descripción	Método que devuelve el mensaje de error, con el formato Error Semántico en línea ‘x’, ‘error’

3.5 Estándares de Codificación

Está diseñado en español debido a que es nuestro idioma nativo. El conocimiento de los estándares seguidos para el desarrollo del intérprete permitirán un mayor entendimiento del código, y es una exigencia de los autores del mismo que cualquier módulo que se añada debe estar codificado siguiendo estos estándares.

Nombre de los ficheros:

Se nombrarán los ficheros .h y .cpp de la siguiente manera:

NombreFichero.h

NombreFichero.cpp

Clases: class Nombre_Clase;

Declaración de variables:

Los nombres de las variables comenzarán con minúscula, en caso de tener más de una palabra se separan por un '_'

Métodos constructor y destructor:

NombreClase ();

~ NombreClase ();

Funciones:

bool Method1 (…);

int Method2 (…);

NombreClase* Method3 (…);

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases se nombrarán "Devuelve" y "Cambia", como los demás métodos:

Obtención del valor:

int Devuelve_Var ();

```
{ return var;}
```

CONCLUSIONES

Se han cumplido los objetivos del trabajo, con la implementación de una primera versión de la máquina virtual del intérprete, que será utilizado como una biblioteca dentro de componentes del sistema Emedia, dando cabida a futuras implementaciones que amplíen y eleven las funcionalidades y el nivel de la gramática del sistema.

RECOMENDACIONES

Como principales recomendaciones tenemos:

- Continuar ampliando la gramática definida, para aumentar las funcionalidades del intérprete.
- Llevar a cabo la implementación completa del intérprete.

BIBLIOGRAFÍA

2007. ¿Qué es Blender? *Blender.* [En línea] 2007. [Citado el: 20 de junio de 2008.] http://wiki.blender.org/index.php/Manual.es/PartI/Introduction.

ABN. 2008. SomosLibres.org. *Destacan la importancia del software libre para la independencia tecnológica.* [Online] febrero 7, 2008. http://www.somoslibres.org/modules.php?name=News&file=article&sid=1743.

Aho, Alfred V, y otros. 1998. *Compiladores:principios, técnicas y herramientas.* s.l. : Pearson Education, 1998.

Ait Kaci, H. 1991. *Warren's Abstract Machine, a tutorial reconstruction.* s.l. : MT Press, 1991.

Cabello Morales, Luis Angel y Vazquez Rivera, Damir Enrique. Traductor y Su Estructura. *Programación de sistemas.* [En línea] [Citado el: 20 de mayo de 2008.] http://usuarios.lycos.es/psistemas/index.php?tema=43.

Castro Solis, Elizabeth. Estándares en los Sistemas de Gestión de Aprendizaje. [En línea] [Citado el: 8 de enero de 2008.] http://bibliopress.wordpress.com/2007/06/29/estandares-en-los-sistemas-de-gestion-de-aprendizaje/.

Castro, M, Losada, P y Peire, J. 1996. Comparación de Técnicas y Herramientas de Autor para la Generación de Aplicaciones Educativas. *II Congreso de Tecnologías Aplicada a la Enseñanza de la Electrónica.* Sevilla : s.n., 1996.

2006. Compilador. *Compilador.* [En línea] 2006. [Citado el: 8 de enero de 2008.] http://www.linux10.com.ar/Glosario/terminos/compilador.htm.

2007. Conferencia Programación IV, UCI. *Otras fases del proceso de compilación. Construcción del árbol de sintaxis abstracta. Chequeo de tipos. Generación de código intermedio. Interpretación.* 2007.

David Kon, Marcos. El software libre. [En línea] [Citado el: 7 de enero de 2008.] http://www.monografias.com/trabajos12/elsoflib/elsoflib.shtml#VENTAJ.

2006. Definición de compilador. [En línea] 2006. [Citado el: 8 de enero de 2008.] http://www.alegsa.com.ar/Dic/compilador.php.

2007. Enciclopedia Libre Universal en Español. *Intérprete informático.* [En línea] 3 de diciembre de 2007. http://enciclopedia.us.es/index.php/Int%C3%A9rprete_inform%C3%A1tico.

Fung Goizueta, Juan Antonio. 2006. *Entrevista sobre Emedia.* octubre de 2006.

Gálvez Rojas, Sergio y Mora Mata, Miguel Ángel. 2005. *Compiladores.* 2005.

BIBLIOGRAFIA

García, Jorge Luis. Entornos Virtuales de enseñanza.Un sistema didáctico? [En línea] [Citado el: 26 de noviembre de 2007.] http://contexto-educativo.com.ar/2003/4/nota-06.htm.

Gromaz Campos, Manuel, Arribí Vilela, Jesús y Rodríguez Malmierca, María José. 2008. E-learning: metodologías, tecnologías y tendencias. [En línea] 7 de enero de 2008. http://www.cibersociedad.net/congres2004/grups/fitxacom_publica2.php?grup=18&=es&id=229.

2005. Historia y Evolución de los Lenguajes de Programación. [En línea] 2005. [Citado el: 10 de enero de 2008.] http://html.rincondelvago.com/lenguajes-de-programacion_historia-y-evolucion.html.

Intérprete. [En línea] [Citado el: 7 de enero de 2008.] http://www.mallorcaweb.net/mostel/glosario.htm.

2006. Intérprete Informático. [En línea] 2006. [Citado el: 11 de noviembre de 2007.] http://enciclopedia.us.es/index.php/Int%C3%A9rprete_inform%C3%A1tico.

La definición de software libre. [En línea] [Citado el: 8 de febrero de 2008.] http://www.gnu.org/philosophy/free-sw.es.html.

Labra, J, y otros. 2004. *Intérpretes y Diseño de Lenguajes de Programación.* 2004.

Lenguaje C++. [En línea] [Citado el: 7 de enero de 2008.] http://tecni.com/es/c.htm.

2004. Lenguaje de Máquina. *Lenguaje de Programación.* [En línea] 2004. [Citado el: 10 de noviembre de 2007.] http://entren.dgsca.unam.mx/introduccion/leng_maq.html.

Lindholm, Tim y Yellin, Frank. *The JavaTM Virtual Machine Specification.* second edition. s.l. : Sun Microsystems, Inc.

Marzal, Andrés y Gracia, Isabel. 2003. *Introducción a la programación con Python.* 2003.

Menchaca Méndez, Rolando y García Carballeira, Félix. Arquitectura de la Máquina Virtual Java. [En línea] [Citado el: 26 de octubre de 2007.] http://www.revista.unam.mx/vol.1/num2/art4/.

Mora Rioja, Arturo. 2005. Descripción de Programas. *Desarrollo de Aplicaciones Informáticas.* 2005.

Neumann, Sven y Natterer, Mitch. Gimp. *Comenzando con el GIMP.* [En línea] http://docs.gimp.org/es/introduction.html.

—. Uso de guiones Script-Fu. *Comenzando con el GIMP.* [En línea] [Citado el: 20 de junio de 2008.] http://docs.gimp.org/es/gimp-concepts-script-fu.html.

Obregón, R. Lenguajes interpretados vs Lenguajes Compilados: Desidia, capricho o tendencia. [En línea] [Citado el: 15 de noviembre de 2007.] http://robregonm.blogspot.com/2006/02/lenguajes-interpretados-vs-lenguajes.html.

Plezbert, Michael y K. Cytron, Ron. 1997. 24th Annual SIGPLAN-SIGACT Symposium of Principies of Programming Languages. *Does "Just in Time"="Better Late than Never"?* París : s.n., 1997.

Rational Software Architect. [En línea] [Citado el: 10 de enero de 2008.] http://www.rational.com.ar/herramientas/softwarearchitect.html.

Requisitos, generalidades de Emedia.

Ribadas Pena, Francisco José. 2006. Generación de Representaciones Intermedias. *PROCESADORES DE LENGUAJES.* 7 de mayo de 2006.

Rodríguez, Daniel. 98-99. Lenguajes de script. [En línea] 98-99. [Citado el: 27 de noviembre de 2007.] http://www.publispain.com/supertutoriales/diseno/html/cursos/7/script.html.

Romero Vargas, Francisco. Historia de los lenguajes de programación. *FUNDAMENTOS DE PROGRAMACIÓN.*

2007. Scripts en Python. *Blender.* [En línea] 11 de octubre de 2007. [Citado el: 20 de junio de 2008.] http://wiki.blender.org/index.php/Manual.es/PartXV/Python_Scripting.

2000. Sistema de Gestión del Aprendizaje. [En línea] 2000. [Citado el: 27 de noviembre de 2007.] http://tecnologias.gio.etsit.upm.es/elearning/lms--learning-management-system--sistema-de-gestion-de-aprendizaje--25.asp.

V. Aho, Alfred, Sethi, Rabi and D. Ullman, Jeffrey. 1986. *Compilers Techniques and Tools.* s.l. : Addison Wesley, Pearson Education, Inc., 1986.

Wiley, David A. 2000. Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. *The Instructional Use of Learning Objects.* [En línea] 2000, 2000. [Citado el: 26 de noviembre de 2007.] http://reusability.org/read/chapters/wiley.doc.

2008. Xmath. *MATRIXx Design and Development Tools.* [En línea] 2008. [Citado el: 20 de mayo de 2008.] http://www.ni.com/matrixx/xmath.htm.

Zapata, Miguel. 2003. Sistemas de gestión del aprendizaje – Plataformas de teleformación. 2003.

ANEXOS

La forma general de una gramática de Bison es la siguiente:

%{
declaraciones en C/C++
%}
Declaraciones de Bison
%%
Reglas gramaticales
%%
Código C adicional

Aquí está la forma de compilar y ejecutar el archivo del analizador:
Lista los archivos en el directorio actual.
% ls
fichero.tab.c fichero.y
Compila el analizador de Bison.
`-lm' le dice al compilador que busque la librería math para pow.
% cc fichero.tab.c -lm -o salida
# Lista de nuevo los archivos.
% ls
salida fichero.tab.c fichero.y
El archivo salida contiene el código ejecutable.

GLOSARIO DE TÉRMINOS

Código Fuente: Un conjunto de líneas que conforman un bloque de texto, escrito según las reglas sintácticas de algún lenguaje de programación destinado a ser legible por humanos. Un Programa en su forma original, tal y como fue escrito por el programador, no es ejecutable directamente por el computador, debe convertirse en lenguaje de máquina mediante compiladores, ensambladores o intérpretes.

Código Objeto: Se llama código objeto en programación al código resultante de la compilación del código fuente.

Máquina Virtual (definición particular): Fase final del intérprete, es una clase que se encarga de ejecutar el código intermedio generado.

Framework: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Alfabeto Binario: Consta de dos únicos símbolos 0 y 1, denominados bits (abreviatura inglesa de dígitos binarios). Sus instrucciones son cadenas binarias (cadenas o series de caracteres de dígitos 0 y 1) que especifican una operación y, las posiciones (dirección) de memoria implicadas en la operación se denominan código máquina o instrucciones de máquina.

Estructura de datos: Forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación.

GLOSARIO DE ABREVIATURAS

SGA (Sistema de Gestión del Aprendizaje)
EVA (Entorno Virtual de Aprendizaje)
UCI (Universidad de las Ciencias Informáticas)
PEA (Proceso de Enseñanza-Aprendizaje)
PDS (Proceso de Desarrollo de Software)
AVM (Máquina Virtual de ActionScript)
MVJ (Máquina Virtual de Java)
EDD (Estructura Dinámica de Datos)
P/RI (Representación Interna)
OA (Objeto de Aprendizaje)
RSA (Rational Software Arquitect)
UML (Unified Modeling Language)
IDE (Entorno Integrado de Desarrollo)
IBM (International Business Machines)
RUP (Rational Unified Process)