REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
PROGRAMA DE ESTUDIOS DE POSTGRADO EN COMPUTACIÓN

Un Sistema de Escritura de Traductores Vía Web

Autor: Jose Daniel Texier Ramírez

Tutor: Dr. Manuel Bermúdez

Mérida, Noviembre 2007

REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
PROGRAMA DE ESTUDIOS DE POSTGRADO EN COMPUTACIÓN

Un Sistema de Escritura de Traductores Vía Web

Autor: Texier Ramírez,Jose Daniel
C.I.: V-13.207.410
Tutor: Bermúdez, Manuel

Mérida, Noviembre 2007

Trabajo de grado presentado como requisito para optar al
título de Magister Scientiae en Computación

Reconocimientos

Mi más profundo agradeciemiento al Dr. Manuel Bermúdez por su gran apoyo en mi formación profesional y personal. Siempre estaré muy agradecido de haberlo conocido y de haber tomando su curso de Lenguajes de Programación, que cambió mi vida para siempre.

A mi Universidad Nacional Experimental del Táchira, la mejor Universidad del Mundo.

Al profesor Jhon Amaya por prestarme una ayuda incondicional siempre que lo necesitaba.

Dedicatoria

A Dios Todopoderoso, por terneme siempre saludable, al igual que todos mis familiares.

A mi papa y a mi mama, mis grandes ejemplos a seguir.

A mi hermana.

Un Sistema de Escritura de Traductores Vía Web

Texier Ramírez, Jose Daniel

dantexier@gmail.com

RESUMEN

Un compilador es un programa desarrollado en un lenguaje de programación que lee un archivo llamado programa fuente. Luego lo traduce y lo convierte en otro programa llamado objeto, o bien en su defecto genera una salida. Una buena manera de entender un lenguaje de programación es analizando su proceso de compilación, el cual es muy similar entre todos los paradigmas o enfoques de programación existentes. Se desea generar una herramienta que permita el aprendizaje en un curso universitario, de todo el proceso de compilación que ocurre en cualquier lenguaje de programación, sin importar la plataforma en la cual se aplique, logrando una mejor comprensión y una aplicación más inmediata. Para ello se ha desarrollado una aplicación Web, la cual al unirla con el compilador conforman un Sistema de Escritura de Traductores. Este sistema completo, permite extender y modificar el compilador en situaciones de educación a distancia. El sistema propuesto se implementó a través de un módulo en Moodle que permite interactuar con el compilador escogido por el coordinador del curso. Moodle es un sistema de gestión de curso libre (Course Management System, CMS) que ayuda a los educadores a crear comunidades de aprendizaje en línea. El software será distribuido bajo la licencia de software libre GPL.

Palabras claves: compilador, Moodle, educación a distancia, software libre.

Introducción

Actualmente en el mundo de la informática se cuenta con aproximadamente 300 sistemas operativos y más de 1500 lenguajes de programación, aunque los más usados son aproximadamente 6 sistemas operativos y 14 lenguajes de programación [14].

Estos lenguajes en general son implementados por un compilador. Un compilador es un programa desarrollado en un lenguaje de programación que lee un archivo llamado programa fuente. Luego lo traduce y lo convierte en otro programa llamado objeto, o bien en su defecto genera una salida [3]. Una manera apropiada de entender cualquier lenguaje de programación es analizando su proceso de compilación.

Actualmente existen cuatro paradigmas en el mundo de los lenguajes de programación: imperativo, orientado a objetos, lógico y funcional [11]. Cada paradigma de programación tiene muy bien definida su metodología de trabajo, a fin de lograr sus objetivos específicos [11]. El paradigma imperativo es el más utilizado y mejor desarrollado actualmente [24, 12, 14]. Emergió junto con las primeras computadoras, sus programas en los años cuarenta, y sus elementos reflejan directamente las características de la arquitectura de las computadoras modernas. El paradigma orientado a objetos [24] es un estilo de programación que involucra metodologías y técnicas para el modelado y desarrollo de software basado en la construcción y conexión de objetos. Una visión orientada a objetos incluye entre otros aspectos el análisis, diseño y codificación, basado en cuatro principios fundamentales: la abstracción de datos, el encapsulamiento de los datos en clases, el ocultamiento de información, y el polimorfismo de los objetos. El paradigma lógico [24] (también es conocido como programación declarativa), apareció como un paradigma independiente

en los años setenta. Se distingue de otros paradigmas porque obliga al programador declar los objetivos o hechos, en lugar de dar un algoritmo con el que se consiguen dichos objetivos. Los hechos se expresan como un grupo de reglas para obtener lo propuesto. Por esta razón a este paradigma se le conoce como la programación basada en reglas. Finalmente, el paradigma funcional [24] apareció a principios de los años sesenta. Su característica esencial es que los cálculos se ven como una función matemática. A diferencia de la programación imperativa, no hay un concepto explícito del estado de la memoria y por tanto no hay necesidad de una instrucción de asignación.

El presente trabajo tiene como objetivo, desarrollar e implementar un sistema que facilite al usuario, el aprendizaje del proceso de compilación de un lenguaje de programación. El mismo se realiza vía Web, donde el coordinador del curso puede desde cualquier parte del mundo supervisar el trabajo de cada uno de los usuarios del mismo. Además, esta alternativa representa una ventaja hoy día, ya que las universidades nacionales deben tener independencia de la plataforma [14].

El desarrollo que se expone está compuesto por cinco (5) capítulos, en los cuales se describe el estudio, diseño e implementación de un sistema de escritura de traductores vía Web. Estos capítulos están constituidos de la siguiente manera:

- En el Capítulo I se presenta al lector una visión general del problema, los objetivos, justificación y alcance asociado al problema.
- En el Capítulo II se presentan los antecedentes y las bases teóricas empleadas para el desarrollo de este sistema.

- En el Capítulo III se expone la metodología utilizada para el desarrollo y la validación del sistema.
- En el Capítulo IV se describe la aplicación Web desarrollada.
- En el Capítulo V se describen las iteraciones del desarrollo del sistema de escritura de traductores vía Web.
- Por último, se presentan las conclusiones, recomendaciones y los apéndices.

Contenido

Lista de Figuras

Capítulo 1

El Problema de Mantener o Extender un Compilador

1.1. Planteamiento del Problema

Con el presente proyecto, se desea generar una herramienta que facilite el proceso de aprendizaje del proceso de compilación, sin importar la plataforma en la cual se aplique. Para ello se plantea desarrollar una aplicación Web que permita extender y modificar un compilador de una forma interactiva y totalmente independiente de la plataforma que se use. Esta aplicación mostrará cada uno de los subprocesos de compilación en forma secuencial, donde cada cambio que se realice debe influir inmediatamente en los subprocesos siguientes y el usuario puede percibirlos, permitiéndole un mejor aprendizaje.

A continuación se exponen de forma resumida los subprocesos que conforman el proceso de compilación:

- Análisis léxico: el proceso de analizar caracteres del programa fuente, formando

unidades léxicas llamadas *token*.

- Análisis sintáctico: en este subproceso se identifica las estructuras sintácticas de la secuencia de tokens.
- Análisis de semántica estática: una vez establecida la estructura sintáctica producida por el analizador léxico del programa fuente, se analiza el significado de ese código, incluyendo análisis de compatibilidad de tipos de datos, declaración de variables, etc.
- Generación de código intermedio: en este subproceso se genera una versión intermedia de código semánticamente equivalente al código fuente.
- Optimización de código: en esta etapa se intenta mejorar el código intermedio, con el objetivo de producir un código de máquina más rápido de ejecutar, o bien con menor tamaño.
- Generación de código: en esta etapa se genera el código objeto, que por lo general consiste de código de máquina relocalizable o lenguaje ensamblador.

Bermúdez sugiere que a partir del trabajo realizado por él [12], se desarrolle lo propuesto en esta tesis en un entorno Web soportado por herramientas de última generación, para poder mejorar muchos aspectos que limitan la versión del año 2003 [13], tales como:

- El control de la aplicación no depende totalmente del instructor.
- No permite el acceso a varios usuarios, es decir, la aplicación es monousuario.
- La aplicación se debe instalar en el sitio donde cada uno realice su sesión práctica.
- No es interactivo ni atractivo para el usuario.

- No se presta para dar soporte a cursos a distancia.

El nuevo sistema amplía y fortalece los cursos o seminarios de compiladores e intérpretes y también tiene la ventaja de usar esta herramienta para educar a distancia.

1.2. Objetivo General

- Desarrollar e implementar un sistema de escritura de traductores que funcione como una aplicación Web.

1.3. Objetivos Específicos

- Diseñar un módelo lógico del sistema de escritura de traductores.
- Ejecutar y visualizar todos los subprocesos de compilación bajo el paradigma imperativo.
- Desarrollar una herramienta bajo la fundamentación del software libre.
- Elaborar los manuales del sistema y del usuario.
- Implementar el sistema de escritura de traductores.

1.4. Justificación e Importancia

El curso de compiladores a nivel mundial tuvo un gran auge en la década de los 80 y 90. A partir de año 2001 el curso de compiladores desaparece (al menos como curso obli-

gatorio) del curriculum recomendado por la ACM (Association for Computing Machinery) [1]. Además, Bermúdez en el 2003 [12] menciona las siguientes razones:

- Falta de madurez en el área. Muchos problemas en el área de compilación se consideran *resueltos*.
- Proliferación de diversos lenguajes y paradigmas de programación.
- El paradigma establecido del libro referente (Dragón Rojo) del área de Aho, Sethi y Ullman [3], el cual pedagógicamente es dificil aplicarlo en un curso.
- Todos los demas libros son la versión resumida favorita del autor, del material en el Dragón Rojo.
- Muy pocos graduados en el área, diseñan o mantienen compiladores.
- Cada día hay menos profesores en el área.
- El curso se considera obsoleto. El material ya no se considera indispensable.
- El proyecto no compagina con el material del curso.
- El objetivo de construir un compilador *de verdad* se considera obsoleto.

Las razones expuestas, permiten que surja la propuesta de desarrollar un software llamado Sistema de Escritura de Traductores, buscando una solución contraria a no dar el curso, sino mejorar su pedagogía. Por ello, se propone mejorar el software existente (el cual corre solamente bajo cónsola GNU/Linux) por uno que supere las deficiencias existentes a través de: permitir la interacción entre los usuarios para lograr que el alumno sea el

protagonista de su proceso de aprendizaje, ofrecer un entorno estable y seguro, en el cual el profesor puede supervisar el trabajo realizado por cada usuario sin importar la distancia.

Por último, es importante destacar las ventajas que ofrece esta propuesta:

- Es un desarrollo que es distribuido bajo la licencia de software libre GPL.
- Promueve una pegadogía constructivista social [15].
- Es apropiada para el 100 % de los cursos en línea, así como también para complementar el aprendizaje presencial, debido a ser una aplicación Web e independiente de plataforma [11, 15].
- Fomenta la independencia de usuarios.

1.5. Alcance

El propósito específico de este proyecto es la implementación del Sistema de Escritura de Traductores vía Web, para ser usado en los cursos de Compiladores e Intérpretes de la Universidad Nacional Experimental del Táchira y de la Universidad de Florida, así como en cursos a distancia y seminarios que imparte el Dr. Bermúdez. El sistema requiere ser instalado en un servidor Web con soporte a PHP [19] y con disponibilidad del sistema Moodle [15].

El uso del mismo dependerá de la orientación del profesor responsable del curso, ya que el sistema cuenta con todas las opciones propias del sistema que desarrolló el Dr. Bermúdez en el año 2003.

Capítulo 2

Marco Teórico del Proceso de Compilación

2.1. Antecedentes de la Investigación

Las siguientes herramientas relacionadas con el área de compiladores se mencionan en el sitio Web: Compiler Tools [9]:

- Análisis léxico: Lex y Flex.
- Análisis sintáctico: Yacc, Bison, Eiffel, Llgen, Memphis.
- Análisis semántico: Beg e Iburg.
- Optimizador de código: Firm Optimizar, Optimix, Pag, entre otros.

Además, se pueden encontrar diversos compiladores para lenguajes muy específicos y netamente en modo consola como son: Antlr, Coco, Cup, Jay, Cocktail, Gentle, entre otros [9].

El tesis se basa en los trabajos previos realizados por Manuel Bermúdez en 2003 [13, 11], en el cual se desarrolló un Sistema de Escritura de Traductores en modo cónsola y limitado a la plataforma Linux; a partir de este trabajo se desarrollará el proyecto planteado, en un entorno Web soportado por herramientas de última generación.

2.2. Concepto de Traductor

Un traductor es un programa que traduce un programa escrito en un lenguaje fuente a un programa equivalente escrito en un lenguaje destino. Se generan, de ser necesario, mensajes de error. La figura 2.1, muestra el esquema básico de un traductor.

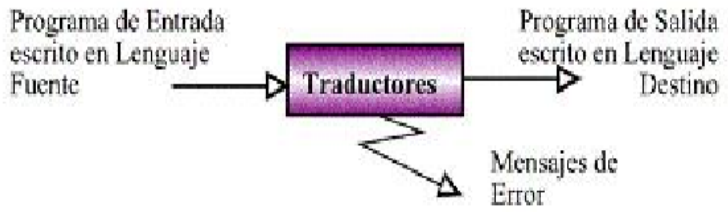

Figura 2.1: Esquema Básico de un Traductor

Es importante destacar la velocidad con la que hoy en día se puede construir un compilador. En la década de 1950, se consideró a los traductores como programas notablemente difíciles de escribir. El primer compilador de Fortran (Formula Translator) [24], por ejemplo, necesitó para su implementación el equivalente de 18 años-hombre. Sin embargo, hoy día un compilador básico puede ser el proyecto de fin de carrera de cualquier estudiante universitario de Informática.

2.3. Compilador

Un compilador es un programa que traduce de un lenguaje fuente a un lenguaje objetoo (véase Fig. 2.2). Se asume que el lenguaje objeto es de bajo nivel.

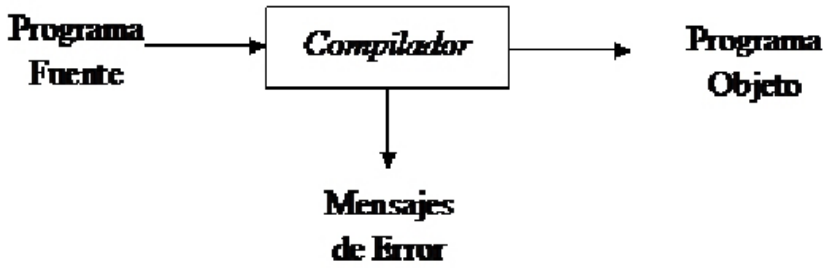

Figura 2.2: Proceso de un Compilador

A primera vista la diversidad de los compiladores puede parecer abrumadora. Hay docenas de lenguajes fuentes, desde los lenguajes de programación tradicionales, como FORTRAN o PASCAL, hasta los lenguajes especializados que han surgido virtualmente en todas las áreas de aplicación de la informática. Los lenguajes objetos son igualmente variados; un lenguaje objeto puede ser otro lenguaje de programación o el lenguaje de máquina de cualquier computadora entre un microprocesador y supercomputador. Los compiladores a menudo se clasifican como de una pasada, de múltiples pasadas, de carga y ejecución, de depuración o de optimación, dependiendo de cómo hayan sido construidos o de que función se suponen que realizan. A pesar de esta aparente complejidad, las tareas básicas que realiza cualquier compilador son esencialmente las mismas. Al comprender tales tar-

eas se pueden construir compiladores con una gran diversidad de lenguajes fuentes y objeto utilizando las mismas técnicas básicas.

2.3.1. El Contexto de un Compilador

Además de un compilador, se pueden necesitar otros programas para crear un programa objeto ejecutable. Un programa fuente se puede dividir en módulos almacenados en archivos distintos. La tarea de producir el programa fuente a menudo es realizado por un preprocesador. El preprocesador también puede incluir macros.

La figura 2.3 muestra los pasos del proceso de compilación. El programa objeto creado por el compilador puede requerir procesamiento adicional antes de poderlo ejecutar. El compilador de la figura 2.3 crea código en lenguaje ensamblador el cual es traducido por un ensamblador a código de maquina. Después se enlaza a algunas rutinas de biblioteca para finalmente producir el código que realmente se ejecuta en la máquina.

2.3.2. Las Fases de un Compilador

Conceptualmente, un compilador opera en fases, cada una de las cuales transforma el programa fuente de una representación en otra. En la figura 2.4 se muestra una descomposición típica de un compilador [3].

Análisis Léxico

Consiste de analizar caracter por caracter del archivo de entrada, produciendo unidades léxicas llamados tokens. Por ejemplo, en el análisis léxico los caracteres de la proposición

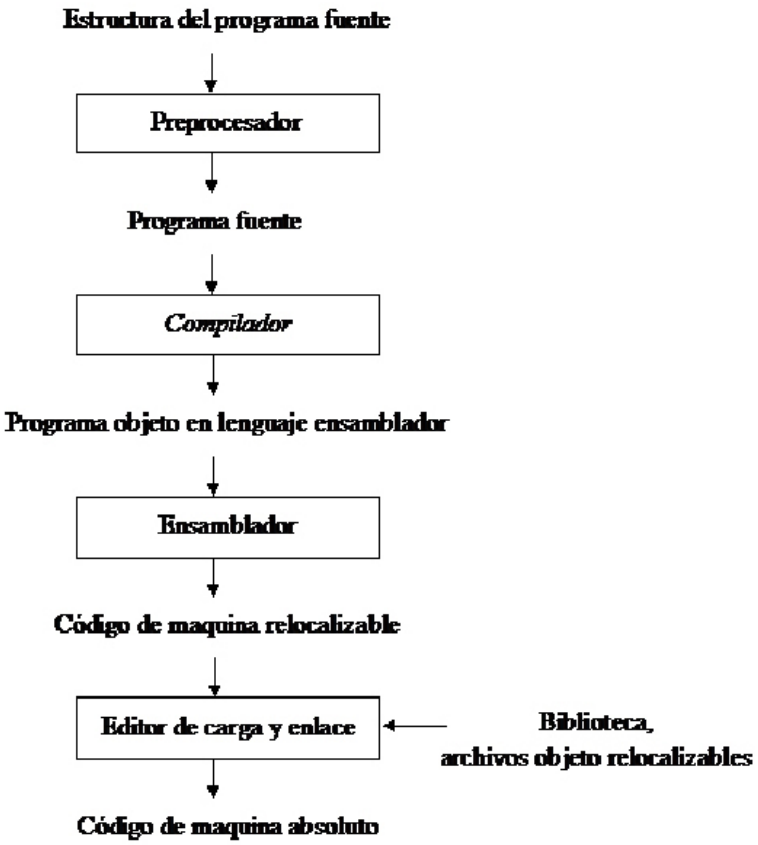

Figura 2.3: Sistema para Procesamiento de un Lenguaje

Figura 2.4: Fases de un Compilador

de asignación:

posición := inicial + velocidad * 60 ,

Se agruparían en los siguientes componentes léxicos::

1. El identificador: ***posición***.
2. El símbolo de asignación: **:=**.
3. El identificador: ***inicial***.
4. El signo de suma: **+**.
5. El identificador: ***velocidad***.
6. El signo de multiplicación: *****.
7. El número: ***60***.

Los espacios en blanco que separan los caracteres de estos componentes léxicos normalmente se eliminan durante el análisis léxico.

Desde los principios de los sesenta, se han introducido modelos formales para la definición de la sintaxis, los lenguajes utilizados para definir dichos modelos son llamados metalenguajes. Hay varios metalenguajes que han sido utilizados con éxito a la hora de definir formalmente la sintaxis de los lenguajes de programación. Hay uno en particular que está muy difundido: Backus-Naur Form (BNF) [3] .

Backus-Naur Form (BNF)

El formalismo BNF ha sido extraordinariamente útil para describir de forma precisa las cadenas que son programas legítimos y aislar todas las demás (las que tienen errores sintácticos de varios tipos). BNF es un metalenguaje basado en la teoría formal desarrollada por el lingüista Noam Chomsky [Chomsky 1957] [1]. Una gramática BNF es un conjunto de reglas de reescritura ρ, un conjunto de símbolos terminales Σ, un conjunto de símbolos no terminales N y un símbolo de comienzo S ϵ N. Cada regla de reescritura de ρ tiene el formato siguiente:

A $\rightarrow \omega$,

donde A ϵ N y ω ϵ (N U S)*. Es decir, ω es una cadena de símbolos terminales (se representan en negrita en adelante) y no terminales.

Los símbolos de N identifican categorías gramaticales (como Identifier, Integer, Expression, Loop y Program) y el símbolo de comienzo S identifica la categoría gramatical principal que estamos definiendo con la gramática. El símbolo S forma el alfabeto básico (por ejemplo, el conjunto de caracteres ASCII o Unicode) a partir de la se crean los programas. A continuación se presenta un par de reglas que definen la sintaxis de una categoría gramatical llamada binaryDigit.

binaryDigit $\rightarrow$ **0**

binaryDigit $\rightarrow$ **1**

[1] BNF fue adaptado a partir del módelo formal Chomsky por John Backus y Peter Naur en 1960 cuando lo utilizaron para desarrollar una definición sintáctica formal del lenguaje de programación Algol [24]. El término **BNF** significa Backus-Naur Form. La gramática BNF constituye una clase junto con las cuatro originalmente definidas por Chomsky. Estas clases se llaman gramática regular; de contexto libre, sensible al contexto y sin restricciones. BNF es un sinónimo de la gramática de contexto libre. La gramática regular es la clase más sencilla, y es análoga a las llamadas expresiones regulares y autómatas de estado finito en su poder de expresión. Las gramáticas sensible al contexto y sin restricciones son más potentes y, por tanto, más complejas que la gramática BNF, y no se utilizan generalmente en los tratamientos formales de la sintaxis de los lenguajes de programación

Este par de reglas expresa que el valor de un binaryDigit es o un 0 o un 1. La siguiente notación se utiliza normalmente para abreviar una serie de reglas que comparten el mismo símbolo no terminal en sus lados izquierdos.

binaryDigit → **0** | **1**

En esta notación, las alternativas están separadas por una barra vertical (|) , lo que significa o, así que la interpretación sigue siendo la misma que en el par de reglas original. El lado derecho de la regla BNF puede ser cualquier secuencia de símbolos terminales y no terminales, lo que permite definir de forma concisa una gran variedad de construcciones interesantes. La siguiente gramática BNF define la estructura de la categoría gramatical Integer como cualquier secuencia Digits.

Integer → Digit | Integer Digit

Digit → **0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**

La primera regla permite que un entero sea un Digit único o un Integer seguido inmediatamente por un Digit, mientras que la segunda define los dígitos decimales normales.Esta regla recursiva permite que un entero esté compuesto de una secuencia arbitraria de dígitos.

Análisis Sintáctico

La sintaxis de un lenguaje de programación utiliza el resultado del análisis léxico como base para definir la estructura de un, como ejemplo citamos las expresiones aritméticas:

x + 2 * y,

las asignaciones:

z = 2 * x + y;,

los ciclos:

for (i = 0: i < n: i++) a[i] = a[i] + 1;,

las definiciones de funciones, las declaraciones de variables (por ejemplo, **int n;**) e incluso los mismos programas completos. La sintaxis de un lenguaje utiliza BNF como herramienta principal para proporcionar una definición precisa y una guía estricta para que la fase de análisis sintáctico detecte errores sintácticos y produzca un árbol de sintaxis abstracta. Las categorías sintácticas Assignment y Expression, definen todas las secuencias de pasos de testigo que conforman una expresión aritméticas y asignaciones de resultados a una variable. A continuación se muestra un ejemplo de una sintaxis BNF sencilla de estas categorías.

Assigment → Identifier = Expression

Expression → Term | Expresión + Term | Expression - Term

Term → Factor | Term * Factor | Term / Factor

Factor → Identifier | Literal | (Expression)

A partir de esta gramática se puede crear fácilmente los árboles sintácticos y las derivaciones de expresiones con valores enteros, utilizando las mismas técnicas que se utilizan en el análisis de los pasos de testigo a nivel léxico. Por ejemplo, la Figura 2.5 muestra una derivación de la Expression x+2 *y. En este ejemplo, el árbol se desarrolla con Expression en su raíz, ya que es la clase gramática que interesa, y con la cadena que se está analizando como sus ramas, leyendo de izquierda a derecha. Las flechas de puntos del árbol indican

abreviaturas de clases gramaticales que se analizarían a nivel léxico, es decir, Identifier y Literal. Omitir los subárboles de estos elementos específicos simplifica la presentación general. Esta es una práctica común.

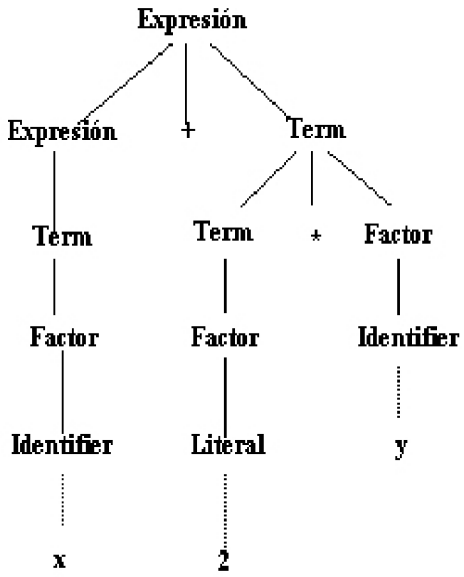

Figura 2.5: Árbol Sintáctico de la Expresión x+2*y

Análisis Semántico

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

Un componente importante del análisis semántico es la verificación de tipos. Aquí,

el compilador verifica si cada operador tiene operandos permitidos por la especificación del lenguaje fuente. Por ejemplo, las definiciones de muchos lenguajes de programación requieren que el compilador indique un error cada vez que se use un número real como índice de una matriz. Sin embargo, la especificación del lenguaje puede permitir ciertas coerciones a los operandos, por ejemplo, cuando un operador aritmético binario se aplica a un número entero y a un número real. En este caso, el compilador puede necesitar convertir el número entero a uno de punto flotante.

Administración de la Tabla de Símbolos

Una función esencial de un compilador es registrar los identificadores utilizados en el programa fuente y reunir información sobre los distintos atributos de cada identificador. Estos atributos pueden proporcionar información sobre la memoria asignada a un identificador, su tipo, su ámbito (la porción del programa donde el identicador es visible) y, en el caso de nombres de procedimientos, el número y tipos de sus argumentos, el método de transmisión de cada argumento (por ejemplo, por referencia) y el tipo que devuelve, si lo hay.

Para realizar estas actividades normalmente se requiere el uso de una tabla de símbolos, que es una estructura de datos que contiene un registro por cada identificador, con los campos para los atributos del identificador. La estructura de datos permite encontrar rápidamente el registro de cada identificador y almacenar o consultar rápidamente datos de ese registro.

Cuando el analizador léxico detecta un identificador en el programa fuente, el identificador se introduce en la tabla de símbolos. Sin embargo, normalmente los atributos de un identificador no se pueden determinar durante el análisis léxico. Por ejemplo, en una declaración en Pascal como el tipo real no se conoce cuando el analizador léxico encuentra posición, inicial, y velocidad (**var posición, inicial, velocidad : real;**)

Las fases restantes introducen información sobre los identificadores en la tabla de símbolos y después la utilizan de varias formas. Por ejemplo, cuando se está haciendo el análisis semántico y la generación de código intermedio, se necesita saber cuáles son los tipos de los identificadores, para poder comprobar si el programa fuente los usa de una forma válida y así poder generar las operaciones apropiadas con ellos. El generador de código, por lo general, introduce y utiliza información detallada sobre la memoria asignada a los identificadores.

Detección e Información de Errores

Cada fase del compilador puede encontrar errores. Sin embargo, después de detectar un error, cada fase debe tratar de alguna forma ese error, para poder continuar la compilación con consecuencias mínimas, permitiendo la detección de más errores en el programa fuente. Un compilador que se detiene cuando encuentra el primer error, no resulta tan útil como debiera.

Las fases de análisis sintáctico y semántico por lo general diagnostican una gran porción de los errores detectables por el compilador. La fase léxica puede detectar errores donde los

caracteres restantes de la entrada no forman ningún componente léxico del lenguaje. Los errores donde la cadena de componentes léxicos violan las reglas de estructura (sintaxis) del lenguaje son determinados por la fase de análisis sintáctico. Durante el análisis semántico el compilador intenta detectar construcciones que tengan la estructura sintáctica correcta, pero no tienen significado para la operación implicada. Por ejemplo, si se intenta sumar dos identificadores, uno de los cuales puede ser el nombre de una matriz, y el otro, el nombre de un procedimiento.

Generación de Código Intermedio

Después de los análisis sintáctico y semántico, algunos compiladores generan una representación intermedia explícita del programa fuente. Se puede considerar esta representación intermedia como un programa para una máquina abstracta. Esta representación intermedia debe tener dos propiedades importantes: debe ser fácil de producir y fácil de traducir al programa objeto.

Optimación de Código

La fase de optimación de código trata de mejorar el código intermedio, de modo que resulte un código de máquina mas rápido de ejecutar. Algunas optimaciones son triviales.

Hay mucha variación en la cantidad de optimización de código que ejecutan los distintos compiladores. En los que hacen mucha optimización, llamados ***compiladores optimizadores***, una parte significativa del tiempo del compilador se ocupa en esta fase. Sin

embargo, hay optimaciones sencillas que mejoran sensiblemente el tiempo de ejecución del programa objeto sin retardar demasiado la compilación.

Generación de Código

La fase final de un compilador es la generación de código objeto, que por lo general consiste en código de máquina relocalizable. Las posiciones de memoria se seleccionan para cada una de las variables usadas por el programa. Después, cada una de las instrucciones intermedias se traduce a una secuencia de instrucciones de máquina que ejecuta la misma tarea. Un aspecto decisivo es la asignación de variables a registros.

2.4. Paradigmas de Programación

Los lenguajes de programación están diseñados para comunicar ideas sobre algoritmos entre las personas y las computadoras. Con un campo de discusión tan limitado, las características expresivas de estos lenguajes son, necesariamente, limitadas, basándose en la pragmática computacional en lugar de en las efemérides o en el esoterismo. Los lenguajes de programación no son para escribir poesía. Aún así, dichos lenguajes permiten un sorprendentemente amplio margen de expresión algorítmica, que soporta una gran variedad de aplicaciones de computación a través de varios dominios de aplicación, como los sistemas de computación científica o de gestión de la información. Para conseguir semejante versatilidad, las distintas comunidades de programadores de estos dominios han desarrollado caminos especiales y diferentes, o paradigmas, para expresar algoritmos que se ajustan especialmente bien a sus propias áreas de aplicación. A continuación, se describen los cuatro paradigmas [11] de los lenguajes de programación.

2.4.1. Paradigma Imperativo

La programación imperativa es el paradigma más utilizado y mejor desarrollado actualmente [14]. Es el paradigma que emergió junto con las primeras computadoras y sus programas en los años cuarenta y sus elementos reflejan directamente las características de arquitectura de las computadoras más modernas.

A finales de los años treinta, Alan Turing, John Von Neumann y otras personas reconocieron que tanto un programa como sus datos podían residir en la memoria principal de la computadora. En los años cuarenta, las primeras computadoras almacena sus programas fuera de su memoria, normalmente utilizando un panel de conexiones. La idea de almacenar un programa en la memoria de la computadora condujo a un enorme incremento de la potencia de las computadoras. Hoy día un lenguaje de programación es de recorrido completo si contiene variables enteras, valores y operaciones y tiene instrucciones de asignación y las construcciones de control de la secuencia de instrucciones, condicionales e instrucciones de ramificación. Todos los demás formatos de instrucciones (bucles while y for, selecciones case, declaraciones de procedimientos y llamadas, etc.) y tipos de datos (cadenas, valores de coma flotante, etc.) están en los lenguajes modernos sólo para mejorar la facilidad de programación de distintas aplicaciones complejas. Un lenguaje de programación imperativo es un lenguaje de recorrido completo que además soporta un cierto número de características fundamentales que han nacido con la evolución del paradigma de programación imperativa desde los años cuarenta [24]. Estas características incluyen:

- Tipos de datos para números reales, caracteres, cadenas, lógico y sus operadores;
- Estructuras de control, bucles for y while, instrucciones case (switch);

- Asignación de elementos y arreglos;
- Asignación de elementos y estructuras de grabación;
- Comandos de entrada y salida;
- Punteros;
- Procedimientos y funciones.

Algunos de los lenguajes de programación imperativos mas conocidos actualmente son: Ada, C, Basic, Clipper, Fortran, Pascal, etc.

2.4.2. Paradigma Orientado a Objetos

El paradigma orientado a objetos es un estilo de programación que involucra una serie de metodologías y técnicas para el modelaje y desarrollo de software basado en la construcción y conexión de componentes.

El soporte fundamental de la orientación a objetos es el modelo objeto. Un objeto es una entidad cuyo comportamiento se caracteriza por las acciones que realiza. El módelo objeto posee una serie de propiedades [11], entre las cuales destacan:

- Abstracción: representación de las características que describen un objeto, en forma independiente de su implementación.
- Ocultamiento de información: el contenido (información) de un objeto es invisible al mundo exterior.

- Encapsulamiento: una aplicación de software se divide en partes o módulos, cada uno de los cuales exhibe un alto grado de independencia de la aplicación y los otros módulos.

- Polimorfismo: un objeto puede ser de más de un tipo (o pertenecer a más de una clase) a la vez.

En un lenguaje orientado a objetos, el tipo de datos va unido a las inicializaciones y otras operaciones de ese tipo. Se refiere al tipo como una clase, las variables internas se llaman variables de instancias y sus inicializaciones se llevan a cabo mediante métodos especiales llamados constructores y otras operaciones las implementan los métodos. Algunos ejemplos de lenguajes de programación que siguen este paradigma son: Smalltalk, Java y C++.

2.4.3. Paradigma Funcional

La programación funcional apareció como un paradigma independiente a principios de los años sesenta. Su creación se debió a las necesidades de los investigadores en el campo de la inteligencia artificial y en sus campos secundarios del cálculo simbólico, pruebas de teoremas, sistemas basados en reglas y procesamiento de lenguaje natural. Estas necesidades no estaban bien cubiertas por los lenguajes imperativos de la época. El lenguaje funcional original fue Lisp, desarrollado por John McCarthy (en 1960) y descrito en Lisp 1.5 programmer's manual [24]. La descripción es notable, tanto por su claridad como por su brevedad (el manual sólo tiene 106 páginas). Contiene no sólo una descripción del sistema Lisp, sino también una definición formal.

La característica esencial de la programación funcional es que los cálculos se ven como una función matemática que hace corresponder entradas y salidas. A diferencia de la programación imperativa, no hay notaciones implícitas de estado y, por tanto, no hay necesidad de una instrucción de asignación. Así, el efecto se modela en forma recursiva, ya que no hay manera de incrementar o disminuir el valor de una variable en el estado. Sin embargo, como aspecto práctico, la mayoría de los lenguajes funcionales soportan las nociones de variable, asignación y bucle. Lo importante es que estos elementos no son parte del módelo de programación funcional puro. Algunos ejemplos de lenguajes funcionales: Ml, Caml, Haskell, Lisp, Cálculo Lambda y RPAL [11, 24].

2.4.4. Paradigma Lógico

También es conocido como programación declarativa y apareció como un paradigma independiente en los años setenta. Se distingue de otros paradigmas porque necesita que el programador declare los objetivos del cálculo, en lugar del algoritmo detallado con el que se consiguen dichos objetivos [24]. Las aplicaciones de la programación declarativa se encuadran en dos dominios principales: diseño de bases de datos e inteligencia artificial. En el área de las bases de datos el lenguaje declarativo dominante ha sido SQL [16], mientras que en el área de la inteligencia artificial, Prolog ha tenido mucha influencia.

Uno de los aspectos más importantes de la programación declarativa respecto a la inteligencia artificial ha sido expresar las especificaciones para las soluciones de los problemas en forma de expresiones en lógica matemática. Este estilo, también llamado programación lógica, estuvo motivado por la necesidad de los investigadores de un procesamiento de lenguaje natural y una demostración automática de teoremas. Los lenguajes de progra-

mación convencionales no han cumplido bien estas necesidades de los investigadores. Sin embargo, la escritura de la especificación de un teorema o de una gramática como una expresión lógica formal proporcionó un vehículo efectivo para estudiar el proceso de la demostración de un teorema y el análisis de lenguaje natural en un ambiente de laboratorio experimental.

Prolog es el lenguaje principal utilizado en la programación lógica. El desarrollo de Prolog está basado en dos principios potentes descubiertos por Robinson, llamados resolución y unificación [24]. Prolog apareció en 1970, a partir del trabajo de Colmerauer, Rousseau y Kowalski [24], y ha sido el lenguaje de programación lógica principal hasta el día de hoy. Las aplicaciones de la programación lógica están especialmente generalizadas en las áreas del procesamiento de lenguaje natural, razonamiento automático y demostración de teoremas, investigación de bases de datos y sistemas expertos.

Dos características de los programas lógicos que explotan estos dominios de aplicación son el no determinismo y la búsqueda de retroceso. Un programa lógico no determinista puede tener varias soluciones para un problema y no una sola, como sería la norma en otros dominios de programación. Es más, el mecanismo de búsqueda de retroceso que permite el no determinismo es interno dentro del intérprete de Prolog y, por tanto, está disponible para todos los programas de Prolog. En contraste, la utilización de otros lenguajes para escribir programas de búsqueda de retroceso necesita que el programador defina explícitamente el mecanismo de búsqueda de retroceso.

2.5. El Software

La definición más formal de software es la atribuida a la IEEE (Instituto de Ingenieros Eléctricos y Electrónicos, 1993), en su estándar 729, *la suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo* [22].

Bajo esta definición el concepto de software va más allá de los programas de cómputo en sus distintas formas: código fuente, binario o código ejecutable, además de su documentación. Es decir, el software es todo lo intangible [5].

2.5.1. Software Libre

El software libre es aquel que puede ser distribuido, modificado, copiado y usado; por lo tanto, debe venir acompañado del código fuente para hacer efectivas las libertades que lo caracterizan. Para Stallman [17] el software libre es una cuestión de libertad, no de precio. Para comprender este concepto, se debe pensar en la acepción de libre como en ***libertad de expresión*** y no es ***gratuito***. En términos del citado autor el software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Y se refiere especialmente a cuatro clases de libertad para los usuarios de software:

1. Libertad 0: la libertad para ejecutar el programa sea cual sea el propósito.
2. Libertad 1: la libertad para estudiar el funcionamiento del programa y adaptarlo a las necesidades propias.

3. Libertad 2: la libertad para redistribuir copias.

4. Libertad 3: la libertad para mejorar el programa y luego distribuirlo para el bien de toda la comunidad. Software libre es cualquier programa cuyos usuarios gocen de estas libertades.

2.5.2. Software Privativo

El software no libre o mejor conocido como software privativo limita su uso y prohíbe la redistribución o modificación. El software privativo puede negar el derecho a usar el programa con cualquier objetivo, estudiar su funcionamiento y adaptarlo a propias necesidades [17].

2.5.3. Licencias para el Software Libre

Se acostumbra publicar software libre con una licencia de software libre. Generalmente se utiliza la Licencia Pública General de GNU (GNU GPL) [20], pero también se utilizan otras licencias de software libre. Para el software GNU únicamente se usan otras licencias si son compatibles con la GNU GPL. La documentación del software libre debería ser documentación libre, para que se pueda redistribuir y mejorar al igual que el software al cual describe. Para que sea documentación libre se acostumbra ser publicada con una licencia de documentación libre. Generalmente se utiliza la Licencia de Documentación Libre de GNU (GNU FDL), aunque en ocasiones también se usan otras licencias de documentación libre.

2.5.4. Copyleft

Copyleft es la forma general de hacer un programa software libre y requiere que todas las modificaciones y versiones extendidas del programa sean también software libre. El modo más simple de hacer un programa libre es ponerlo en el dominio público, o sea, sin copyright. Esto permite que la gente comparta el programa y sus mejoras, si así lo desean. Pero también permite a quien no quiera cooperar convertir el programa en software privativo. Pueden hacer cambios y distribuir el resultado como un producto privativo. Las personas que reciban el programa en su forma modificada no poseen la libertad que el autor original les dió debido a que el intermediario se la ha retirado.

2.6. World Wide Web

También conocida como la telaraña de información mundial (Web), es la forma de presentar la información más utilizada, actualmente, en Internet. Esta forma de presentar la información, en lo que se suele llamar páginas, sitios y portales Web, es la principal responsable del aumento espectacular en el uso de Internet de estos últimos años [23]. Las páginas Web reúnen dos características importantes:

- Son páginas multimedia: contienen texto, imágenes, sonidos, vídeo, etc.
- Son páginas hipertexto: contienen enlaces (palabras, imágenes o botones) donde se puede hacer un clic con el ratón de forma sencilla y sin necesidad de conocer la dirección (URL) del lugar a donde se quiere ir, cambiar a otra página donde se ofrece más información sobre el tema que de esta mirando. Este proceso se conoce como navegar.

2.6.1. Aplicaciones Web

Una aplicación Web es un sistema informático que los usuarios utilizan accediendo a un servidor web a través de Internet o de una intranet. Las aplicaciones Web son populares debido a la practicidad del navegador Web como cliente ligero. La habilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software en miles de potenciales clientes es otra razón de su popularidad. Aplicaciones como los webmails, wikis, weblogs, tiendas en línea y la Wikipedia misma son ejemplos bien conocidos de aplicaciones Web.

Las interfaces Web tienen ciertas limitantes en la funcionalidad del cliente. Métodos comunes en las aplicaciones de escritorio como dibujar en la pantalla o arrastrar-y-soltar no están soportadas por las tecnologías Web estándar. Los desarrolladores web comúnmente utilizan lenguajes interpretados del lado del cliente para añadir más funcionalidad, especialmente para crear una experiencia interactiva que no requiera recargar la página con demasiada frecuencia. Recientemente se han desarrollado tecnologías para coordinar estos lenguajes con tecnologías del lado del servidor, como por ejemplo PHP [19]. Además, se incrementado el uso de AJAX [19], la cual es una técnica de desarrollo Web que usa una combinación de varias tecnologías.

Aunque muchas variaciones son posibles, una aplicación Web está comúnmente estructurada como una aplicación de tres capas. En su forma más común, el navegador Web es la primera capa, un motor usando alguna tecnología web dinámica (ejemplo: CGI, PHP, Java Servlets o ASP) es la capa de en medio, y una base de datos como última capa. El navegador Web manda peticiones a la capa media, que la entrega valiéndose de consultas

y actualizaciones a la base de datos generando una interfaz de usuario. Esto se ilustra en la figura 2.6.

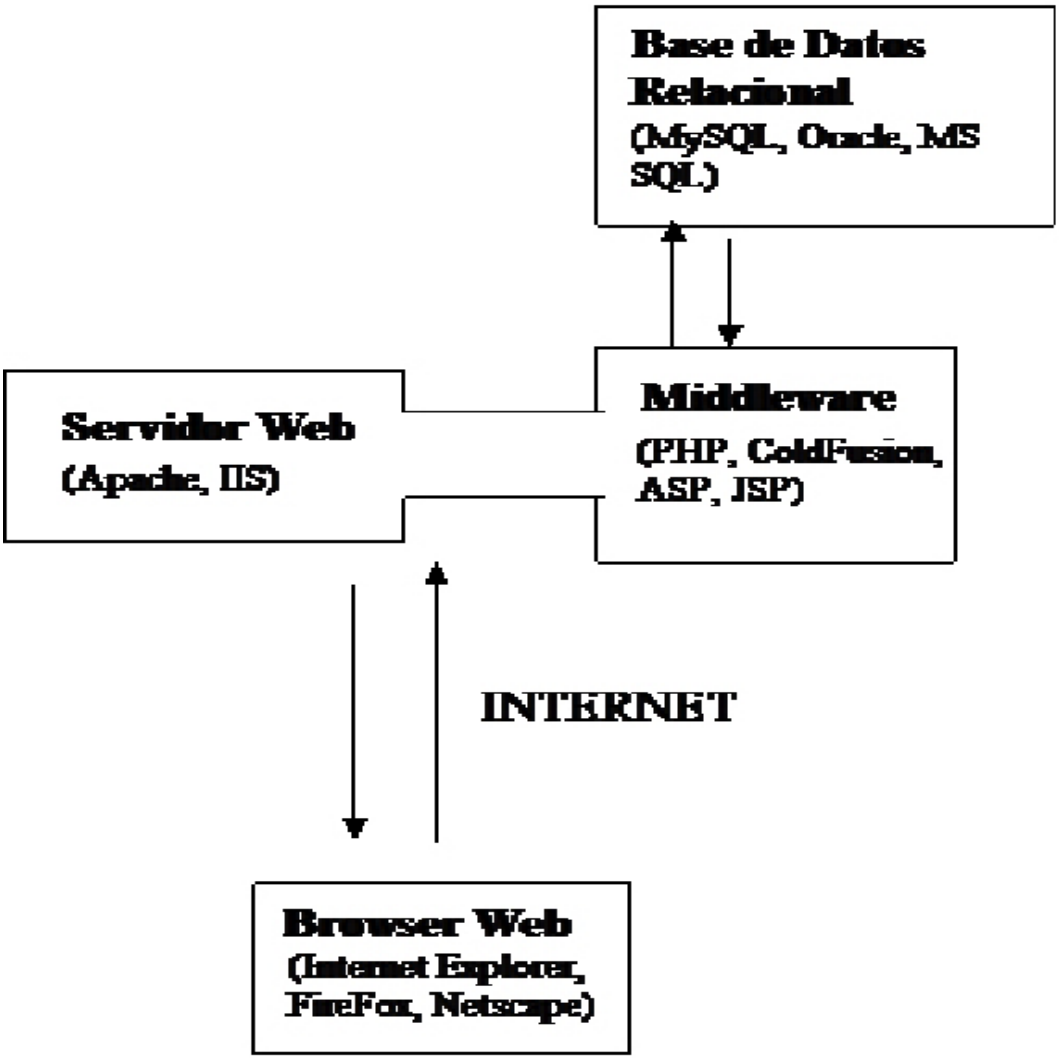

Figura 2.6: Arquitectura de Aplicaciones Web

2.6.2. PHP

Existe el famoso acrónimo recursivo de PHP significa Hypertext Pre-Processor. PHP es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor. En otros palabras, dentro de una página HTML se puede encontrar código PHP el cual va a ser ejecutado cuando la página sea visitada. Luego su código PHP es interpretado por un servidor Web y genera HTML u otra salida que el visitante desee ver.

2.7. Moodle

Moodle es un sistema de gestión de cursos de libre distribución (Course Management System CMS) que ayuda a los educadores a crear comunidades de aprendizaje en línea [15]. Moodle fue creado por Martin Dougiamas, quien fue administrador de WebCT en la Universidad Tecnológica de Curtin. Basó su diseño en las ideas del constructivismo que afirman que el conocimiento se construye en la mente del estudiante en lugar de ser transmitido sin cambios a partir de libros o enseñanzas. Un profesor que opera desde este punto de vista crea un ambiente centrado en que el estudiante le ayuda a construir ese conocimiento con base a sus habilidades y conocimientos propios en lugar de simplemente publicar y transmitir la información que se considera que los estudiantes deben conocer.

La primera versión de la herramienta apareció el 20 de agosto de 2002 y, a partir de ahí han aparecido nuevas versiones de forma regular. Hasta diciembre de 2006, la base de usuarios registrados incluye más de 19.000 sitios en todo el mundo y está traducido a más de 60 idiomas [15]. El sitio más grande dice tener más de 170.000 estudiantes. La palabra Moodle era al principio un acrónimo de Modular Object-Oriented Dynamic Learning

Environment (Ambiente de Aprendizaje Dinámico Modular Orientado a Objetos), lo que tiene algún significado para los programadores y teóricos de la educación, pero también se refiere al verbo anglosajón ***moodle***, que describe el proceso de deambular perezosamente a través de algo, y hacer las cosas cuando se desee. Las dos acepciones se aplican a la manera en que se desarrolló Moodle y a la manera en que un estudiante o profesor podría aproximarse al estudio o enseñanza de un curso en línea.

En términos de arquitectura, se trata de una aplicación Web que puede funcionar en cualquier computador en el que se pueda ejecutar PHP. Opera con diversas bases de datos SQL como por ejemplo MySQL y PostgreSQL. La licencia que utiliza Moodle es la GPL.

2.7.1. Características de Moodle

Moodle es un producto activo y en evolución. Se enumeran algunas de sus muchas características:

- **Diseño general**: promueve una pegadogía constructivista social (colaboración, actividades, reflexión crítica, etc.). Apropiada para cursos en línea, así como también para complementar el aprendizaje presencial. Tiene una interfaz de navegador de tecnología sencilla, ligera, eficiente, y compatible. Es fácil de instalar en casi cualquier plataforma que soporte PHP. Sólo requiere que exista una base de datos (misma que puede compartir). Con su completa abstracción de bases de datos, soporta las principales marcas de bases de datos (excepto en la definición inicial de las tablas). La lista de cursos muestra descripciones de cada uno de los cursos que hay en el servidor, incluyendo la posibilidad de acceder como invitado. Se ha puesto énfasis en una seguridad sólida en toda la plataforma. Todos los formularios son revisados, las cookies

encriptadas, etc. La mayoría de las áreas de introducción de texto (materiales, mensajes de los foros, entradas de los diarios, etc.) pueden ser editadas usando el editor HTML, tan sencillo como cualquier editor de texto de Windows.

- **Administración del sitio**: el sitio es administrado por un usuario administrador, definido durante la instalación. Los temas permiten al administrador personalizar los colores del sitio, la tipografía, presentación, etc., para ajustarse a sus necesidades. Pueden añadirse nuevos módulos de actividades a los ya instalados en Moodle. Los paquetes de idiomas permiten una localización completa de cualquier idioma. Estos paquetes pueden editarse usando un editor integrado. Actualmente hay paquetes de idiomas para 34 idiomas. El código está escrito de forma clara en PHP bajo la licencia GPL, fácil de modificar para satisfacer las necesidades específicas del usuario.

- **Administración de los usuarios**: los objetivos son reducir al mínimo el trabajo del administrador, manteniendo una alta seguridad. Soporta un rango de mecanismos de autenticación a través de módulos de autenticación, que permiten una integración sencilla con los sistemas existentes. Método estándar de alta por correo electrónico: los estudiantes pueden crear sus propias cuentas de acceso. La dirección de correo electrónico se verifica mediante confirmación.

- **Método LDAP**: las cuentas de acceso pueden verificarse en un servidor LDAP [26]. El administrador puede especificar qué campos usar.

- **Administración de cursos**: el profesor tiene control total sobre todas las opciones de un curso. Se puede elegir entre varios formatos de curso tales como semanal, por temas o el formato social, basado en debates. Ofrece una serie flexible de activi-

dades para los cursos: foros, diarios, cuestionarios, materiales, consultas, encuestas y tareas. En la página principal del curso se pueden presentar los cambios ocurridos desde la última vez que el usuario entró en el curso, lo que ayuda a crear una sensación de comunidad. La mayoría de las áreas para introducir texto (materiales, envío de mensajes a un foro, entradas en el diario, etc.) pueden editarse usando un editor HTML WYSIWYG (What You See Is What You Get - Lo que se ve es lo que se obtiene) integrado. Todas las calificaciones para los foros, diarios, cuestionarios y tareas pueden verse en una única página (y descargarse como un archivo con formato de hoja de cálculo). Además permite el registro y seguimiento completo de los accesos del usuario y dispone de informes de actividad de cada estudiante, con gráficos y detalles sobre su paso por cada módulo (último acceso, número de veces que lo ha leído) así como también de una detallada historia de la participación de cada estudiante, incluyendo mensajes enviados, entradas en el diario, etc., en una sola página. Por último la herramienta permite enviar por correo electrónico copias de los mensajes enviados a un foro, los comentarios de los profesores, etc. en formato HTML o de texto.

Capítulo 3

Metodología

En el presente capítulo, se abordará el tipo de investigación utilizada en el proyecto, haciendo hincapié en la justificación respectiva y el diseño metodológico, así como las fases correspondientes.

3.1. Tipo de Investigación

El presente trabajo corresponde a la modalidad de Proyectos Especiales [25], porque el mismo conllevó a una creación tangible y apropiada para ser utilizada en cursos para demostrar soluciones a problemas presentados a la hora de diseñar algoritmos. Cuenta con una directa vinculación con una competencia profesional directa del postgrado al cual se opta, donde existen cursos como Lenguajes de Programación, Análisis y Diseño de Algoritmos, Desarrollo de Software, Teoría de Autómatas, entre otras. Cuenta con objetivos y enfoques metodológicos no previstos en el Manual de la Universidad Pedagogíca Experimental Libertador (UPEL)[25], que por su carácter innovador produjo un aporte significativo al área

de conocimiento del trabajo seleccionado.

3.2. Diseño Metodológico

3.2.1. Modelo de Ingeniería Web

El desarrollo de la investigación se efectuó de acuerdo al modelo de Ingeniería Web (IWeb) propuesto por Pressman en el 2002 [18], donde indica que la IWeb demanda un proceso de software incremental y evolutivo. Pressman también señala que el modelo en las primeras versiones puede ser un modelo en papel o un prototipo, y durante las últimas iteraciones se producen versiones cada vez más completas del sistema diseñado. La IWeb se divide en un número de actividades estructurales, también llamadas regiones de tareas (ver figura 3.1). Generalmente, existen entre tres y seis regiones de tareas, las cuales no necesariamente se deben aplicar todas por cada iteración.

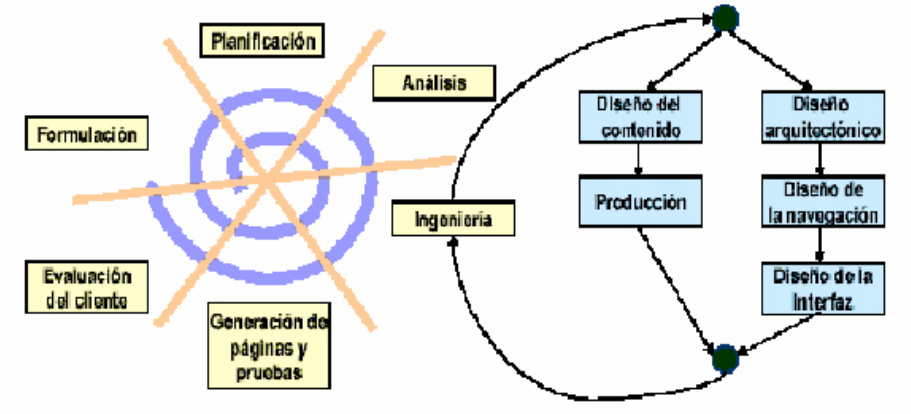

Figura 3.1: Modelo de Proceso IWeb

El proceso IWeb comienza con la ***formulación***, que es la actividad que identifica las metas y los objetivos de la Aplicación Web (WebApp) a desarrollar, y establece el ámbito del incremento, es decir, incremento del software por cada etapa.

La ***planificación*** estima el costo global del proyecto, evalúa los riesgos asociados con el esfuerzo del desarrollo y define una planificación del desarrollo bien granulada para el incremento final de la WebApp.

El ***análisis*** establece los requisitos técnicos para la WebApp e identifica los elementos del contenido que se van a incorporar. También se definen los requisitos del diseño gráfico.

La actividad de ***ingeniería*** incorpora dos diseños principales. El primero se refiere al diseño del contenido y la producción, y que de ser necesario, será desarrollado por personas no pertenecientes al desarrollo del sistema, es decir, diseñadores gráficos. Se lleva a cabo la integracion de las tareas referentes al diseño arquitectonico (estructura global de hipermedia), diseño de navegación (permiten al usuario acceder al contenido y a los servicios) y diseño de la interfaz (impactar con la primera impresión).

La ***generación de páginas*** es una actividad de construcción que hace mucho uso de las herramientas automatizadas para la creación de la WebApp. El contenido definido en la actividad de ingeniería se fusiona con los diseños arquitectónicos, de navegación y de la interfaz para elaborar páginas Web ejecutables en HTML, XML y otros lenguajes orientados al proceso. Durante esta actividad también se lleva a cabo la integración con el software

intermedio, conocido como middleware. Las **pruebas** ejercitan la navegación, intentan describir los errores de applets, guiones y formularios, y ayuda a asegurar que la WebApp funcionará correctamente en diferentes entornos (navegadores).

Cada incremento producido como parte del proceso IWeb se revisa durante la actividad de ***evaluación del cliente***. Es en este punto en donde se solicitan cambios. Estos cambios se integran en la siguiente ruta mediante el flujo incremental del proceso.

3.2.2. Implementación del Modelo IWeb

En esta investigación sucedieron cuatro iteraciones. En cada una se deben aplicar las seis actividades, pero las mismas estarán sujetas al ámbito definido en la formulación del problema.

Se describen las diferentes actividades del modelo IWeb en consonancia a los objetivos definidos. Las cuatro (4) iteraciones fueron las siguientes:

- Primera iteración, definido en un trabajo en papel.
- Segunda iteración, en la cual se realizaron prototipos del sistema.
- Tercera iteración, se realizó un sistema con todas los subprocesos de un compilador.
- Cuarta iteración, se incluyó el sistema realizado en la iteración anterior en Moodle, obteniendo una aplicacion robusta.

Las actividades de acuerdo al modelo IWeb, que permitieron desarrollar el sistema que será usado en el curso de Compiladores el próximo año en la Universidad Nacional Experimental del Táchira y en la Universidad de Florida, son:

- Formulación: se identificaron las metas y objetivos del sistema para establecer el ámbito de las cuatro iteraciones aplicadas.
- La planificación: se evaluaron los riesgos asociados con el esfuerzo del desarrollo, y el tiempo de ejecución de la misma.
- El análisis: permitió establecer los requisitos técnicos del sistema y se identificaron los elementos del contenido tomando como base el software de consola desarrollado anteriormente. También se tomaron en cuenta los requerimientos del diseño gráfico.
- Ingeniería: se logro la integración del diseño arquitectónico, de navegación y de interfaz.
- Generación de páginas y pruebas: se fusionó el análisis y diseño del sistema con la escogencia del lenguajes de programación Web PHP. Además se realizaron pruebas basadas en la metodologia IWeb.
- Evaluación del cliente: permitió corregir errores por cada iteración, obteniendo como resultado una evolución en comparación con la iteración anterior.

En el próximo capítulo se describirán al detalle las diferentes actividades vinculadas a cada una de las iteraciones de acuerdo a la metodología IWeb.

Capítulo 4

Descripción de la Aplicación Web

En el presente capítulo, se explica cada una de las fases de la aplicación Web. Estas fases y subfases las clasificamos de la siguiente manera:

- Fase 1 - Compiler. Se especifica el compilador, en este módulo se definen las reglas de juego, es decir, como funcionará el analizador léxico (a través del comando LEX en GNU/Linux) y el analizador sintáctico (a través del comando YACC en GNU/Linux). También permite definir como será el proceso de análisis de semántica estática y la generación de código.

- Fase 2 - Run. Se corre el compilador, a partir de un programa fuente, se obtienen sus tokens, un árbol sintáctico, su semántica y el código de máquina.

- Fase 3 - Interpreter. Se corre el programa compilado, este módulo permite ejecutar el código de máquina generado y mostrar la secuencia de pasos de la ejecución del compilador.

Distribución general de la aplicacióon Web:

- Fase 1: Compiler.
    - Scanner.
    - Parser.
    - Contrainer.
    - Generator.
- Fase 2: Run.
    - Source.
    - Scanning.
    - Parsing.
    - Constrain.
    - GenCode.
- Fase 3: Interpreter.
    - Code.

Antes de empezar a explicar detalladamente la descripción de la aplicación, se tiene que destacar que el compilador se encuentra en modo consola, disponible en el servidor, es decir, el único sitio donde se puede correr es desde esa maquina. Es una de las razones por las cuales se desarrolla este proyecto. En el gráfico que se presenta a continuación (ver figura 4.1, se observa el diseño del compilador donde se destacan las tres fases principales del diseño de la aplicación Web desarrollada. Pero antes se destaca, que el sistema cuenta con los subproceso llamados: *flex, gcc, pgen y yacc*. El *flex* y *yacc*, son herramientas que

permiten generar salidas léxicas y sintácticas de acuerdo a las reglas definidas para estos procesos [17]. El *pgen* es un subproceso definido por Bermúdez [13] que permite adaptar la salida de las reglas sintácticas al *yacc*. Y el *gcc* es el compilador de C para GNU/Linux [6].

La primera fase es conocida como COMPILER, se encuentra de color azul claro (cada una representa una subfase de COMPILER), la segunda fase llamada RUN esta de color amarillo (son cinco nombres de archivos que representan cada subfase de RUN) y la última fase que tiene como nombre INTERPRET es de color verde claro (esa es la única subfase de INTERPRET). También se observa que los rectángulos representan entidades de software, es decir, programa que se van a ejecutar tales como: lex o flex, yacc, gcc, entre otros. Pero también hay programa objetos generados por el compilador y que llevan como nombre: pgen, parser, Constrain, CodeGen, Interpretet.

Después de entender el funcionamiento interno de la aplicación Web, se explicará los componentes de la misma basadas en un diseño acorde a cumplir los objetivos planteados para la enseñanza de un curso de Compiladores e Intérpretes [12]. Las tres modalidades son:

- Modalidad de edición del compilador (Compiler).
- Modalidad de compilación del compilador (Run).
- Modalidad de ejecución del compilador (Interpret).

En la anterior figura (ver 4.2), se observa la distribución del entorno grafico, donde se pueden observar las tres modalidades representadas en las tres primeras pestañas. Luego, al hacer clic en cualquiera de ellas, se mostraran en el nivel de pestañas debajo de ellas, las

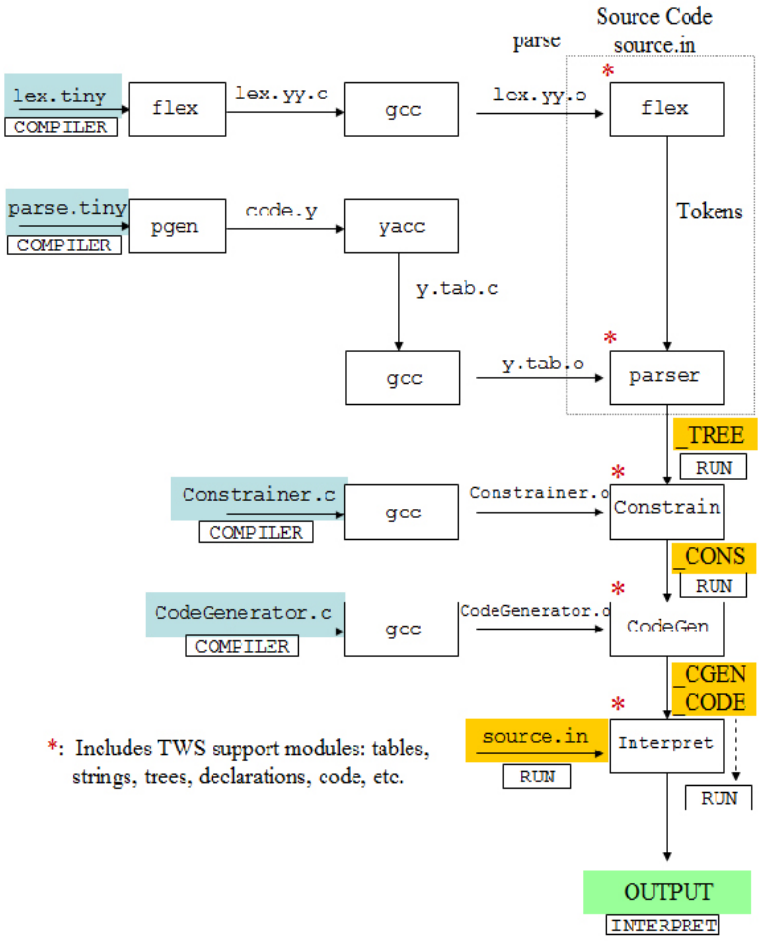

Figura 4.1: Estructura Interna del Compilador - Modo Cónsola.

subfases correspondientes a la misma. Además se observan seis puntos claves a destacar:

- 1. Indica el nombre del usuario.
- 2. Corresponde a la sección de fases y subfases de la aplicación Web.
- 3. Indica la fase y subfase seleccionadas.
- 4. Representa el frame para la edición de los archivos.

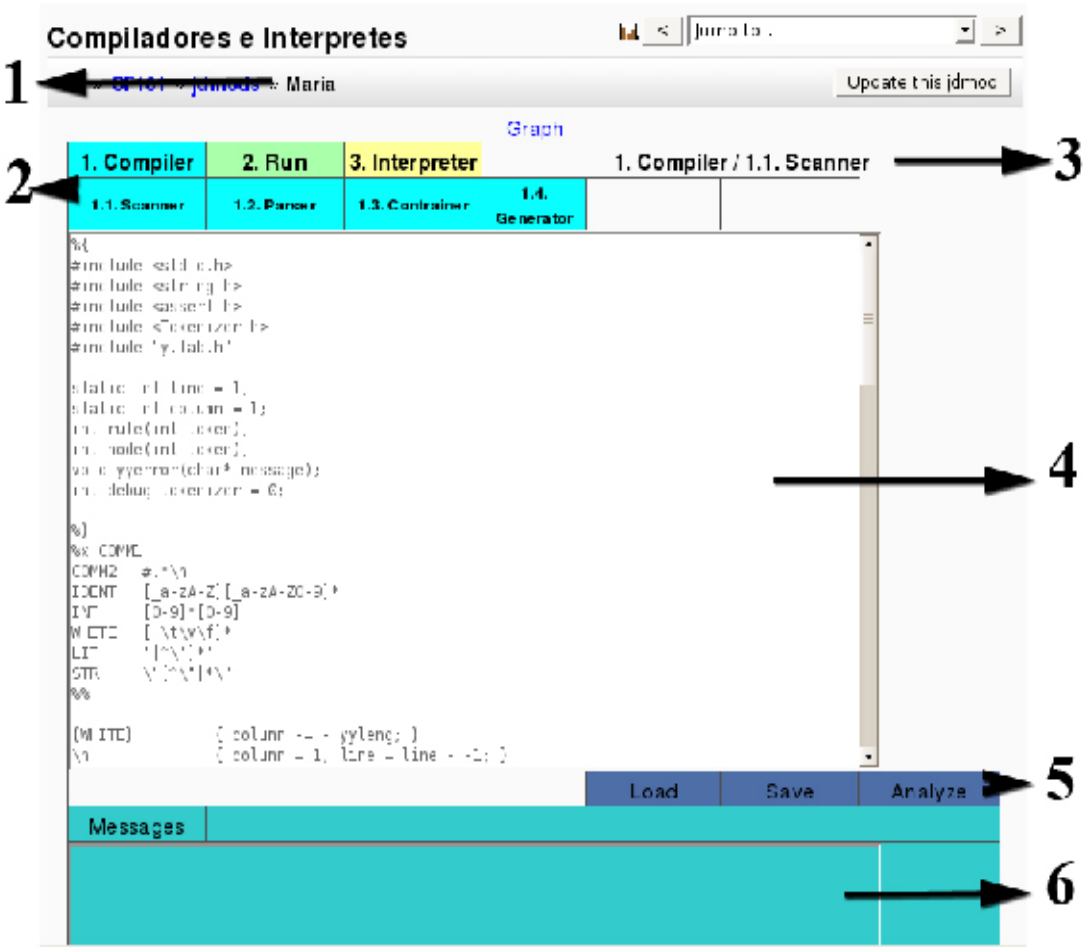

Figura 4.2: Estructura General de la Aplicación Web.

- 5. Representan los botones para ejecutar las acciones que tiene la herramienta.
- 6. Representa el frame de los mensajes de ejecución correspondientes a cada fase.

A continuación se explicará a explicar cada una de las modalidades y paralelamente se explica un ejemplo sencillo (de aplicación) de agregar el operador matemático de multiplicación (*) al lenguaje ***Tiny*** [1], porque no se encuentra definido inicialmente.

[1] Es el nombre del lenguaje definido en la implementación de este compilador.

4.1. Modalidad de Edición del Compilador

En esta fase, se puede cambiar todas las condiciones del compilador, es decir, se puede configurar al gusto del instructor o del estudiante, pasando por las condiciones léxicas, sintácticas, semánticas y de generación de código. Para una mejor comprensión de cada una de ellas, se explicará con se agrega un operador matemático (multiplicación) para que sea reconocido por el compilador.

4.1.1. Especificación Léxica

Esta subfase consiste en el proceso de ir palabra por palabra, para que el compilador pueda conformar el token correspondiente a las condiciones establecidas en esta fase. Si se quiere agregar un operador nuevo, en este caso, el de multiplicación. Se debe decirle al compilador que lo reconozca primero como un token. Por ello se agrega la línea referencia por el * en la figura 4.3.

4.1.2. Especificación Sintáctica

En esta fase se definen las especificaciones sintácticas. Luego de que el compilador reconozca el operador *, se debe indicar la regla sintáctica que permita construir el árbol sintáctico único para el programa fuente. Esta regla esta definida en la figura 4.4.

4.1.3. Especificación Semántica

Una vez establecidas las reglas léxicas y sintácticas, se deben definir las reglas semánticas. Esto se realiza en esta subfase, donde estas reglas se van aplicando a medida que se

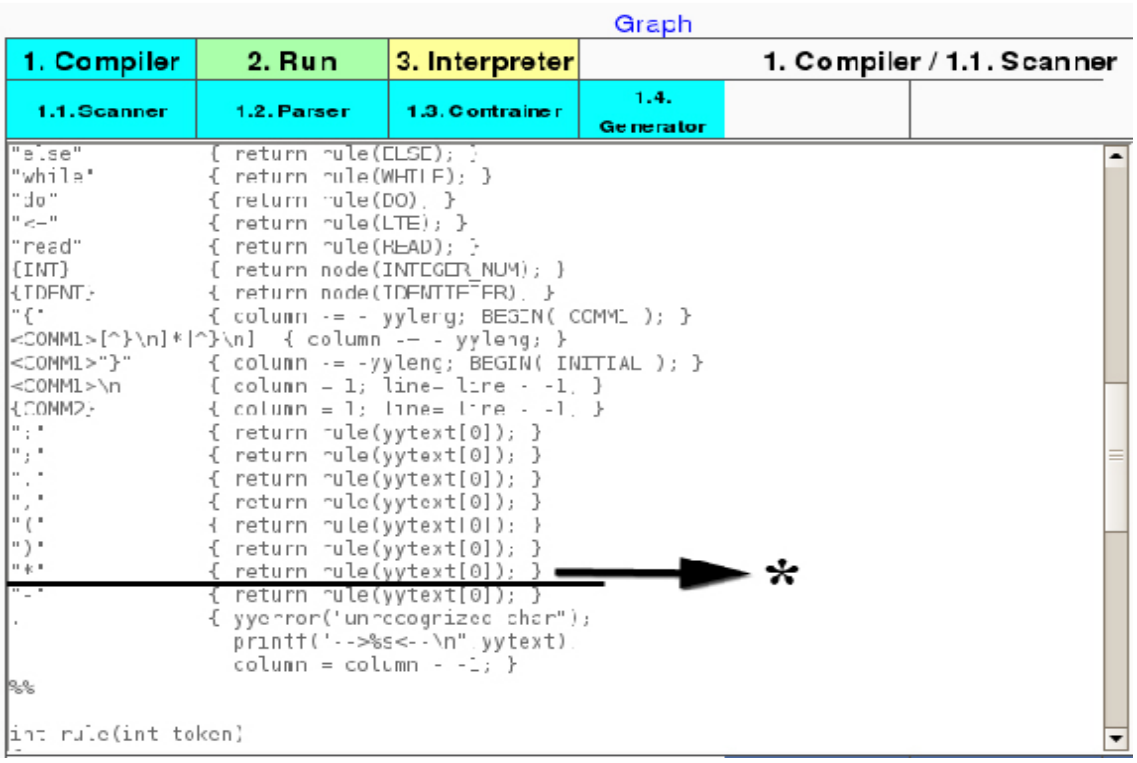

Figura 4.3: Fase 1/1 - Scanner - Análisis Léxico.

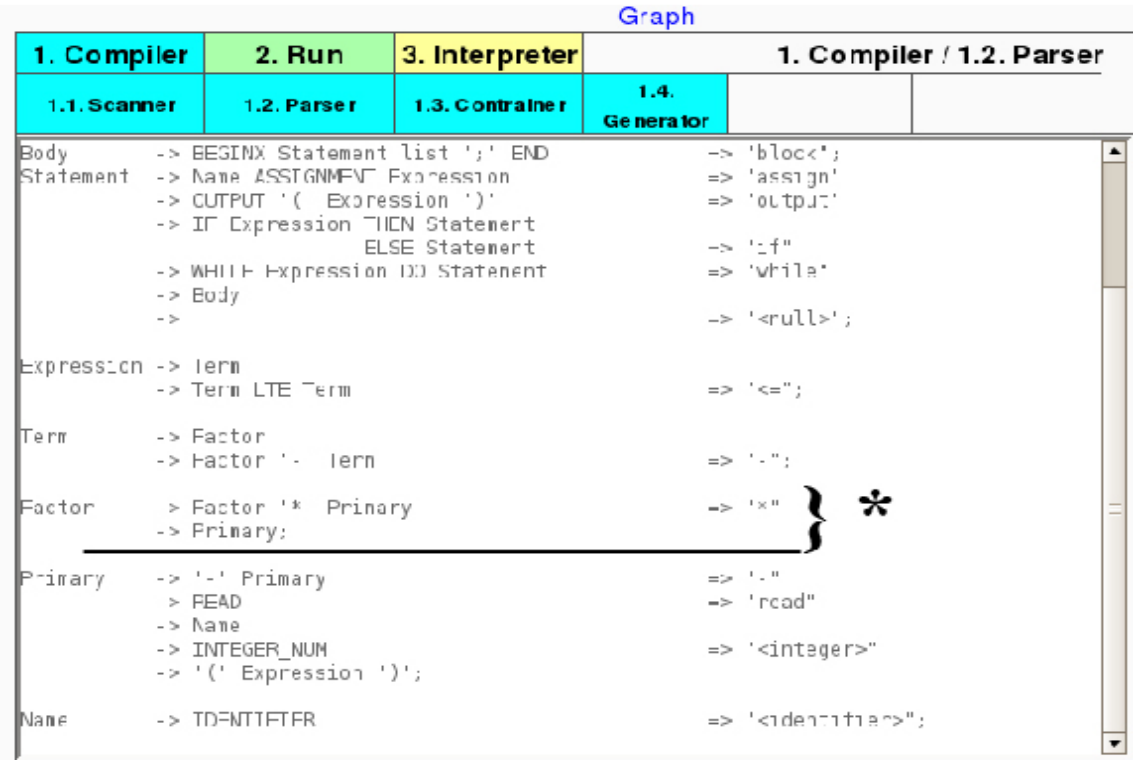

Figura 4.4: Fase 1/2 - Parser - Análisis Sintáctico.

realiza el recorrido del árbol sintáctico, de acuerdo a como se definió en el compilador. Para el ejemplo del operador de multiplicación, se deben realizar tres pasos:

- Primero, se debe indicar al análisis semántico que existe un nuevo operador (ver figura 4.5).
- Segundo, se debe indicar el caso de tener que aplicar la operación correspondiente (ver figura 4.6), se destaca que en este caso en particular, proceso del nodo ***** (operador de multiplicación) se hace en forma idéntica a la del proceso del nodo existente **-** (operador menos). Por eso se agrega ***case: MultiplyNode:***.
- Tercero, como se debe realizar la operación (ver figura 4.6).

4.1.4. Especificación de Generación de Código

En esta subfase, se genera el código correspondiente al árbol obtenido. Por ello, en el ejemplo, se deben realizar dos pasos, primero el generador de código debe reconocer el operador (ver figura 4.7) y luego, se debe generar las instrucciones de código. Nuevamente, el proceso de generar instrucciones para el nodo *, es similar al proceso del nodo existente ***case LENode:*** (ver figura 4.8).

4.1.5. Resultados de la Modalidad

Finalmente, después de los cambios realizados en las subfases, se tiene que compilar el compilador para reconocer cualquier error sintáctico en el mismo. Si la compilación fue exitosa, se muestra el mensaje correspondiente (ver figura 4.9).

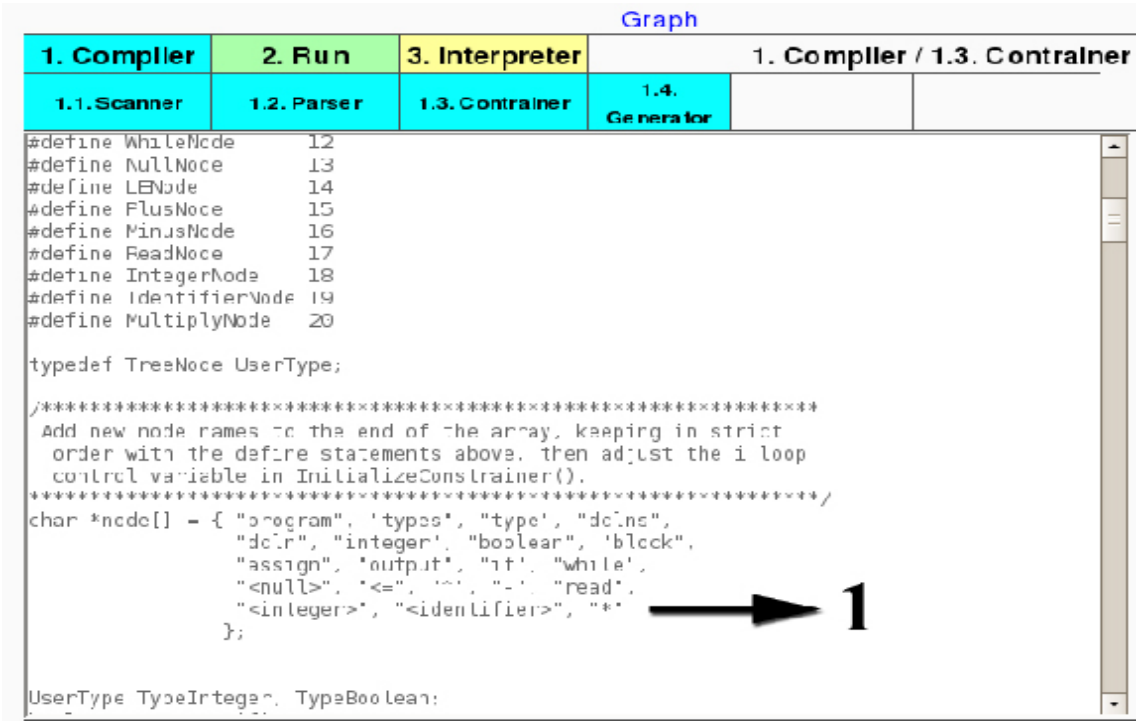

Figura 4.5: Fase 1/3 - Constrainer - Análisis Semántico - Parte I.

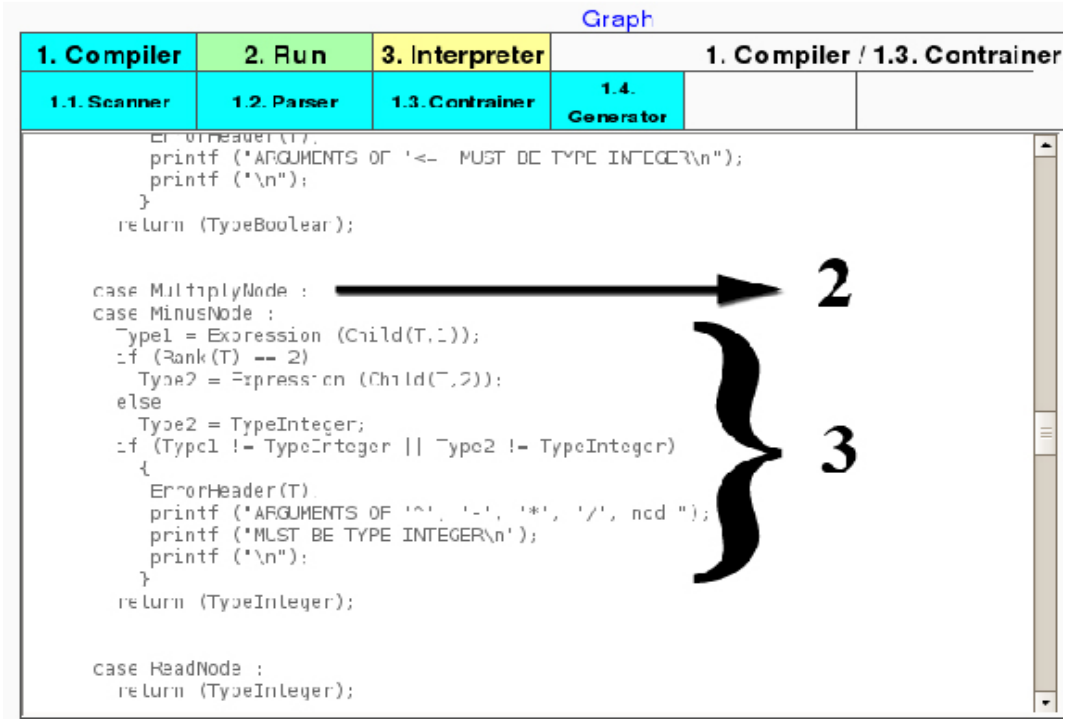

Figura 4.6: Fase 1/3 - Constrainer - Análisis Semántico - Parte II.

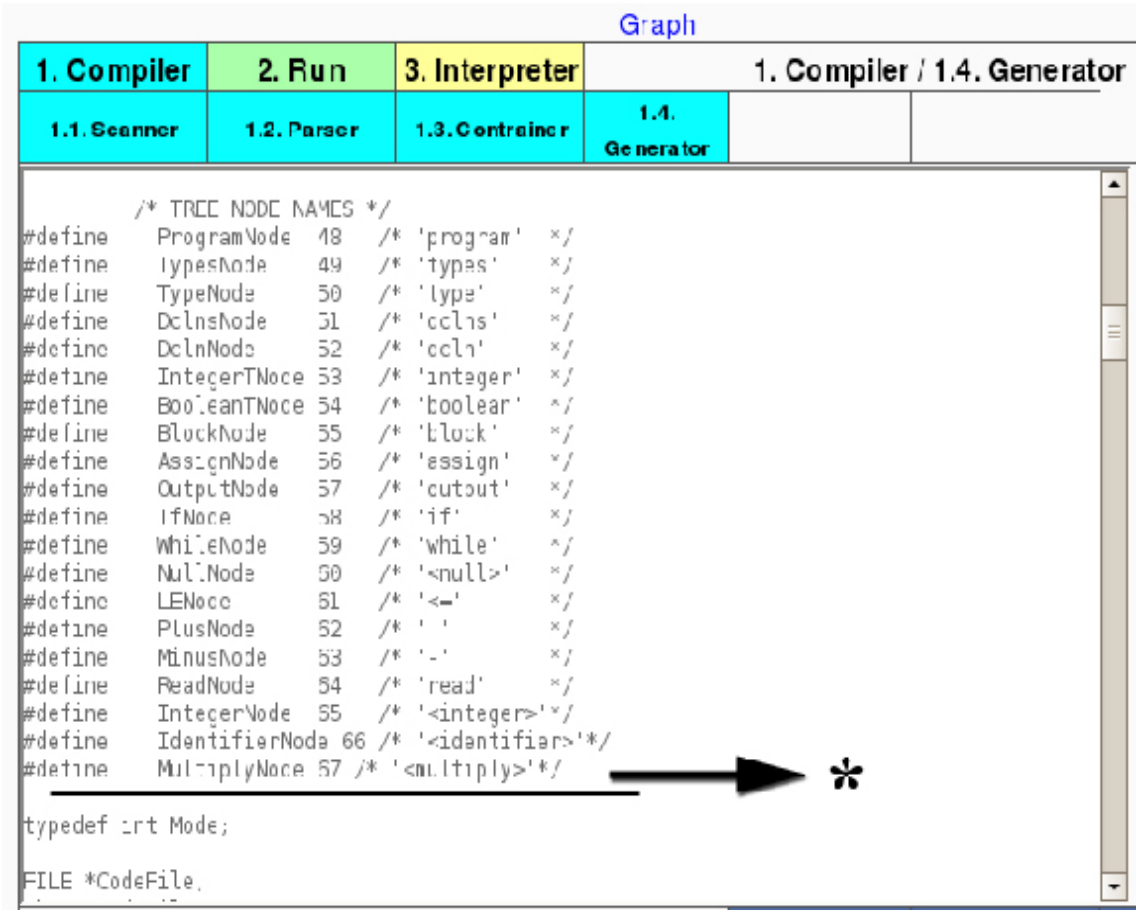

Figura 4.7: Fase 1/4 - Generator - Generación de Código - Parte I.

4.2. Modalidad de Compilación del Compilador

En esta modalidad, la aplicación Web permite compilar un programa fuente de acuerdo a las condiciones definidas en la fase anterior. Para continuar con el esquema del ejemplo de agregar un operador matemático, en esta etapa se partirá del programa fuente que contiene el operador deseado hasta la generación de código correspondiente al mismo.

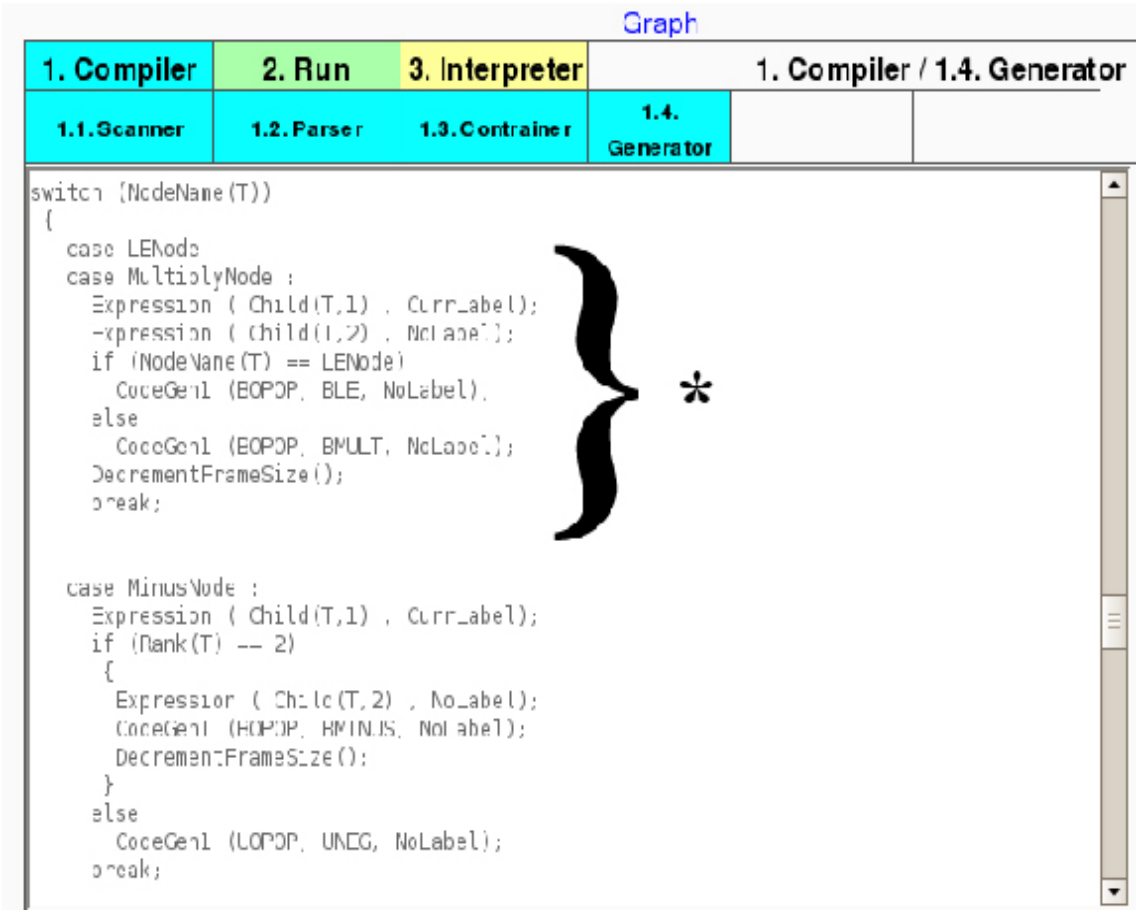

Figura 4.8: Fase 1/4 - Generator - Generación de Código - Parte II.

4.2.1. Edición del Código Fuente

En esta subfase, el usuario puede cambiar y cargar el programa fuente deseado a partir de las condiciones de la fase anterior. Por ejemplo, se puede observar en la figura 4.10 es un programa sencillo que calcula el factorial del número 7 representado en la variable n. También aparecen unas líneas en comentarios; comentarios empiezan con el carácter **#**.

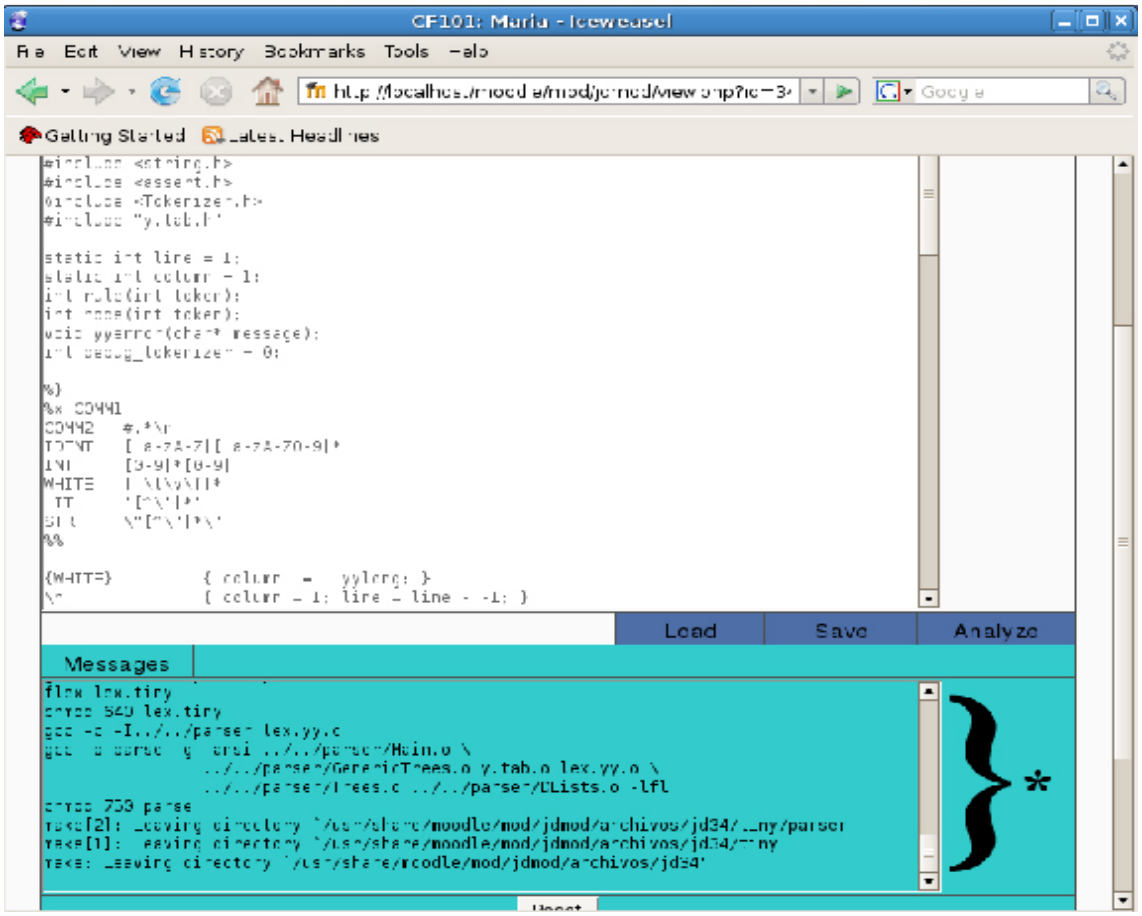

Figura 4.9: Resultados del Análisis de la Fase 1.

4.2.2. Resultados Léxicos

En esta etapa, se muestran los tokens generados a partir del código fuente. Se observa que el operador * encontrado, se ubica en la línea 16 y columna 20, además es identificado internamente con el token 67 (ver fig. 4.11).

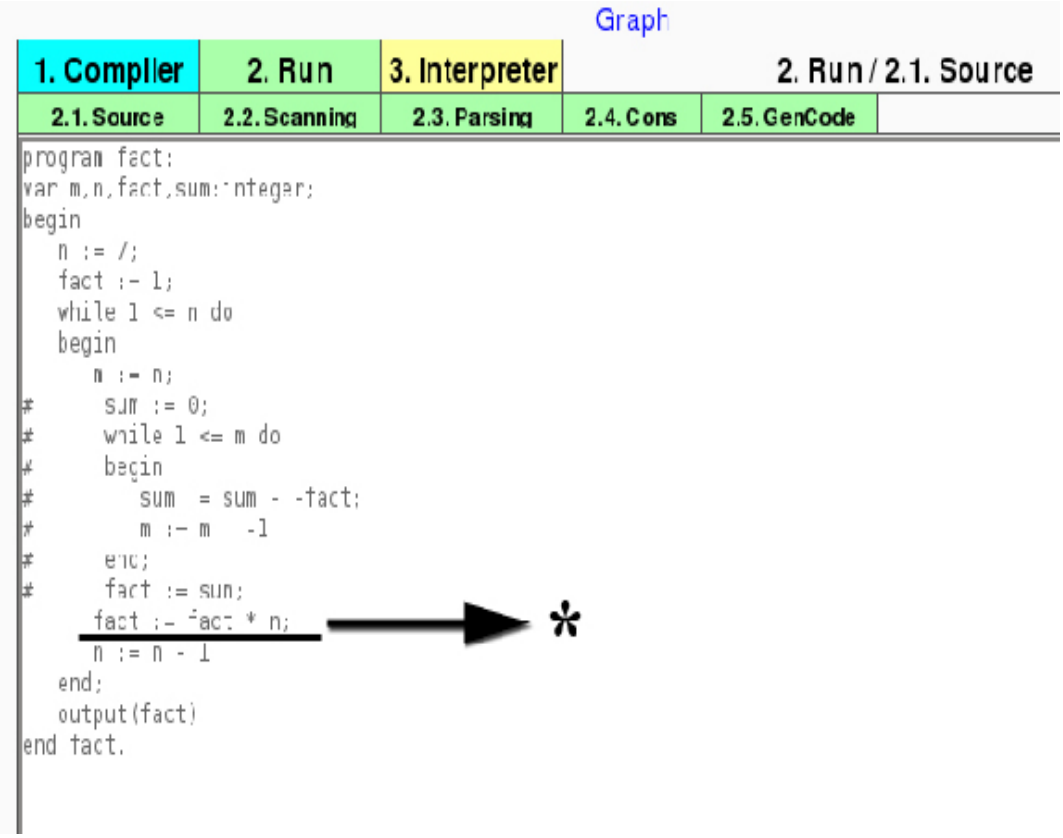

Figura 4.10: Fase 2/1 - Source - Código Fuente.

4.2.3. Resultados Sintácticos

Después de realizar el proceso de identificar los tokens del programa fuente, se procede a analizar la construcción sintáctica generada, que se obtiene en esta etapa. Se observa en primera instancia como la raíz del árbol sintáctico generado del fuente es el token program (ver figura 4.12). También se observa en la siguiente figura (ver fig. 4.13), donde se ubica el operador *.

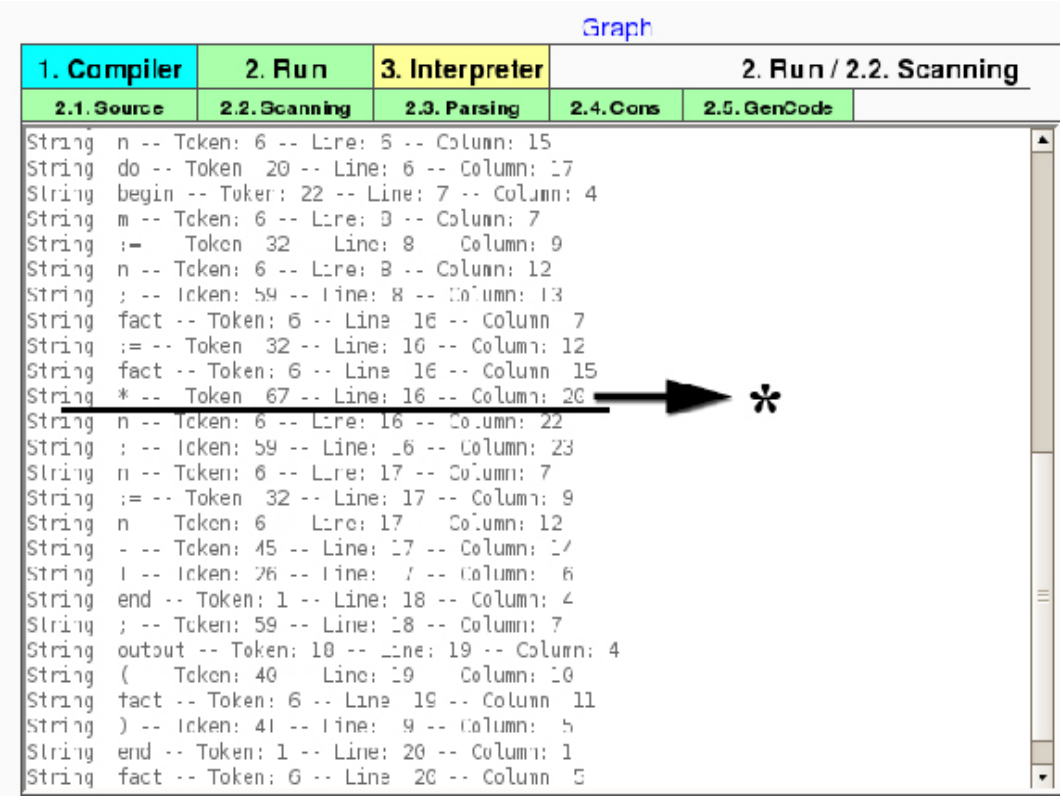

Figura 4.11: Fase 2/2 - Scanning - Tokens.

4.2.4. Resultados Semánticos

El resultado semántico se puede observar en esta etapa, donde a partir de la subfase anterior se obtiene un código equivalente semánticamente al programa fuente expresado en un árbol sintáctico. La secuencia del ejemplo, en la figura 4.14 se observa el proceso de visitar y revisar los nodos del árbol, incluyendo el operador agregado *.

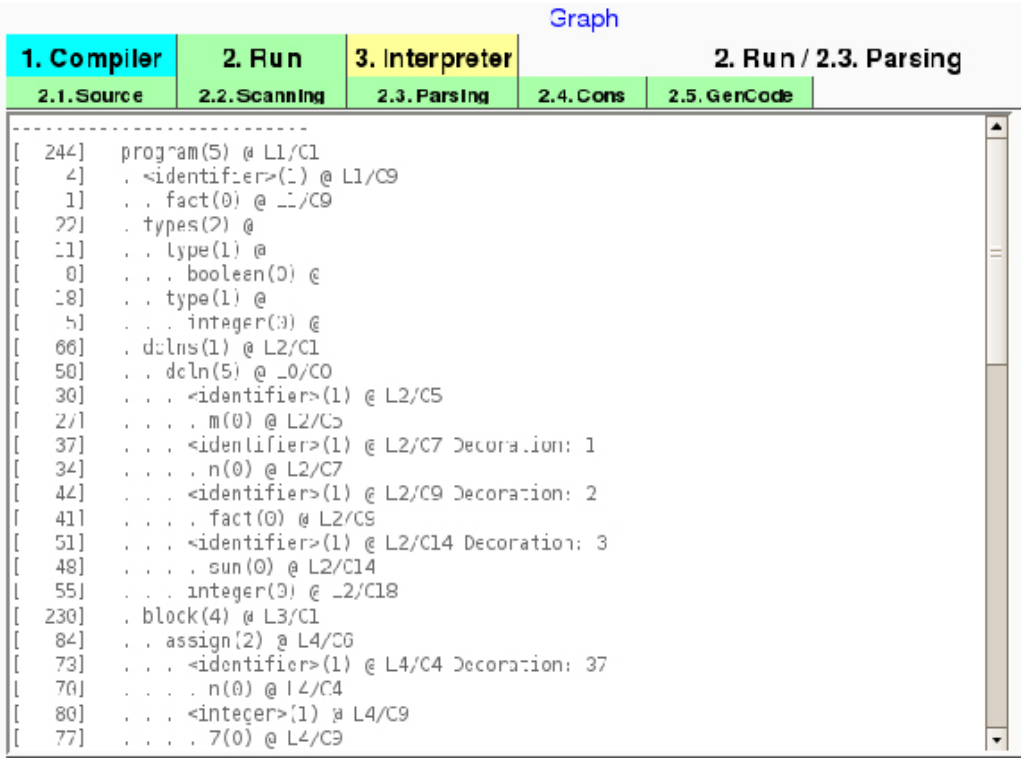

Figura 4.12: Fase 2/3 - Parsing - Árbol Sintáctico - Parte I.

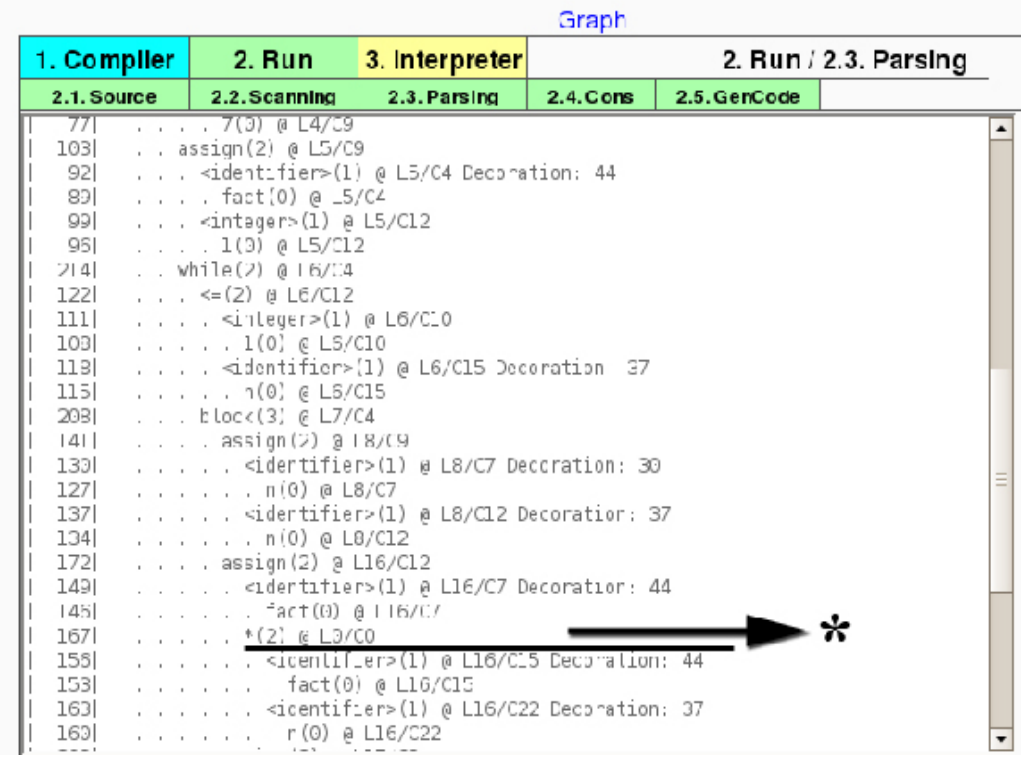

Figura 4.13: Fase 2/3 - Parsing - Árbol Sintáctico - Parte II.

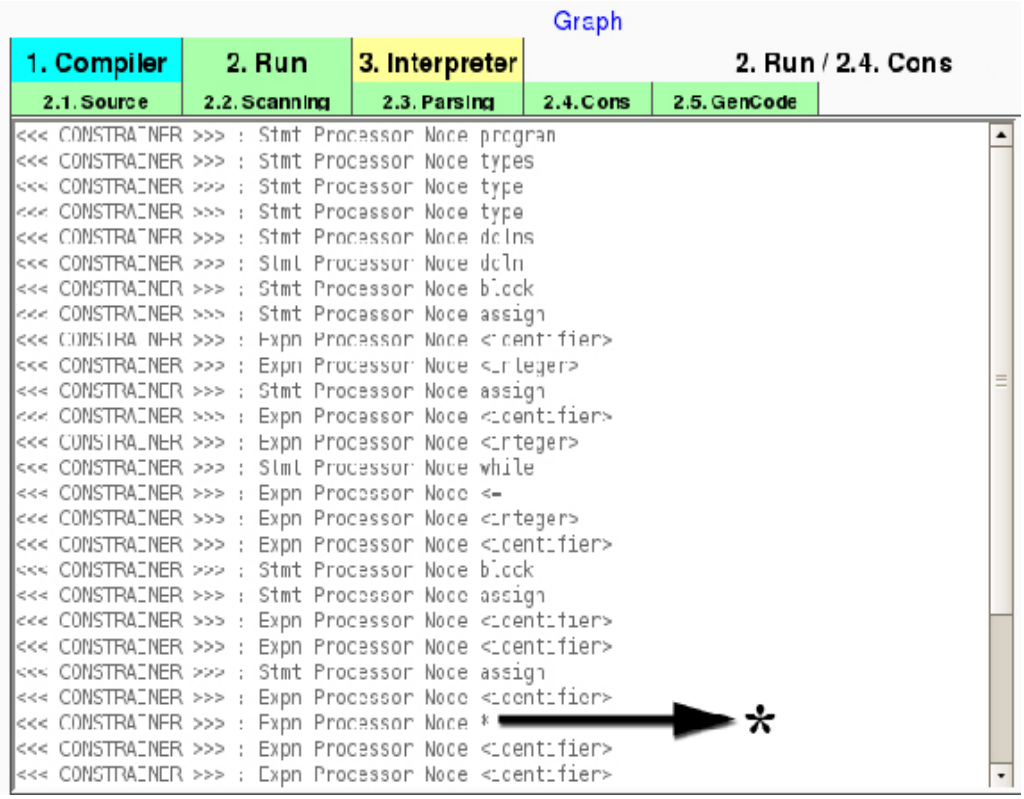

Figura 4.14: Fase 2/4 - Cons - Semántica.

4.2.5. Resultados del Generador de Código

Por último, se presenta la generación de código del programa fuente codificado y se observa en la figura 4.15, el código correspondiente al operador *.

4.2.6. Resultados de la Modalidad

Después de obtener los resultados de esta fase, se procede analizar los resultados de la ejecución de la misma. En esta parte aparecen los errores sintácticos que pueden presentar por el programa fuente definido (ver figura 4.16).

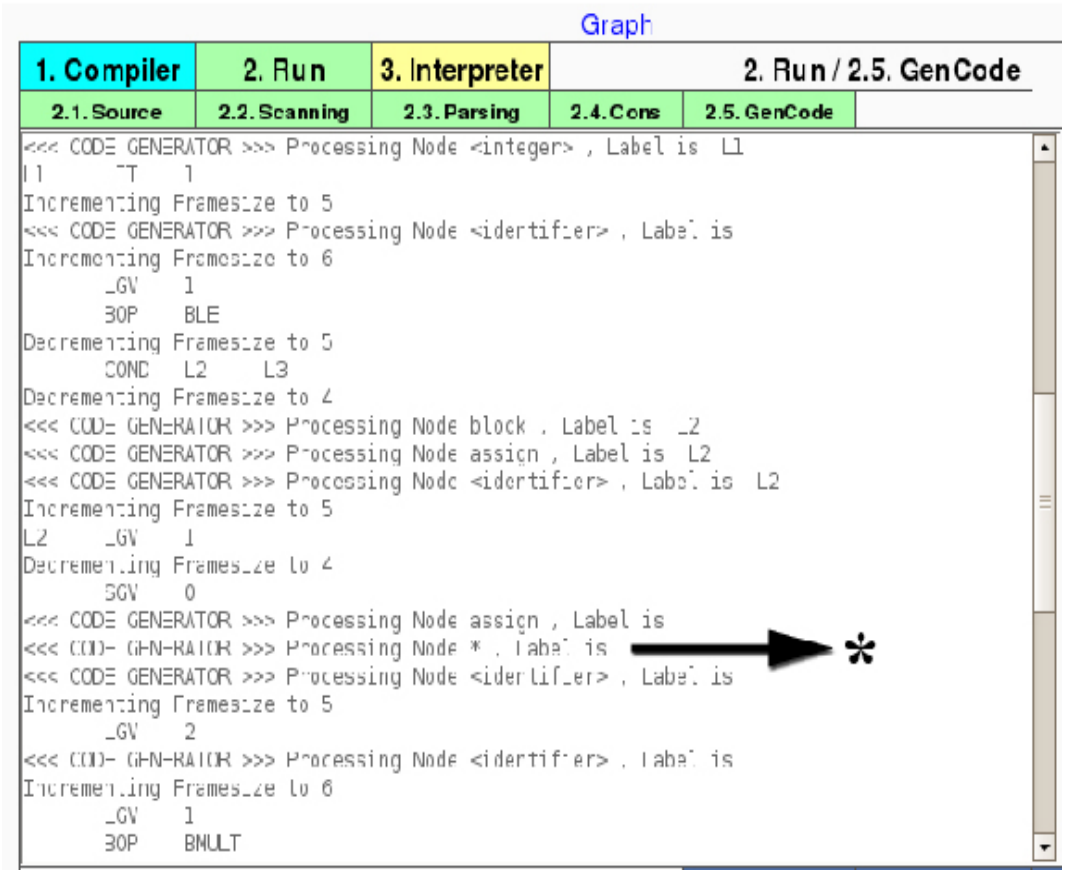

Figura 4.15: Fase 2/5 - GenCode - Generación de Código.

4.3. Modalidad de Ejecución del Compilador

En esta fase, finalmente se observar el resultado de ejecutar el programa fuente obteniendo la respuesta a través de frame de los mensajes.

4.3.1. Ejecución del Código de Máquina

Esta subfase se observa el código de máquina generado, el cual se puede interpretar para obtener una salida (ver figura 4.17).

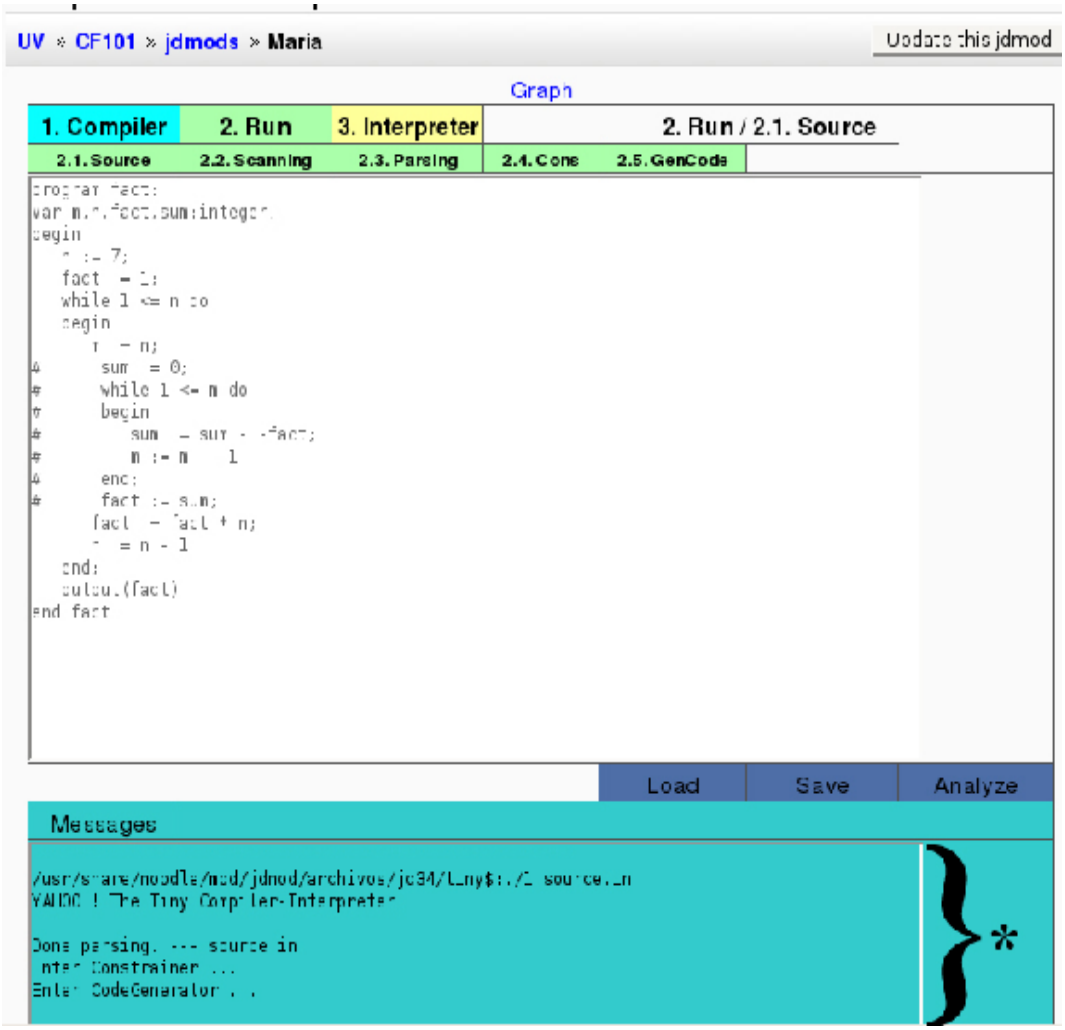

Figura 4.16: Resultados del Análisis de la Fase 2.

4.3.2. Resultados de la Modalidad

Finalmente, se observa el resultado final, 5040, que es el factorial de 7 (figura 4.18).

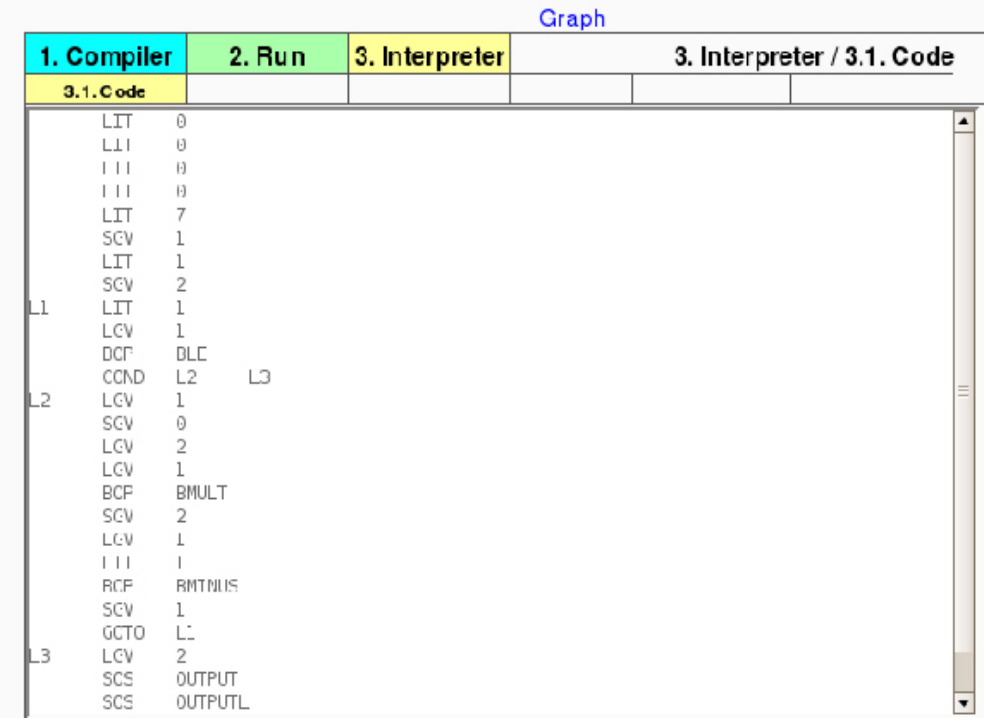

Figura 4.17: Fase 3/1 - Código de Máquina del Compilador.

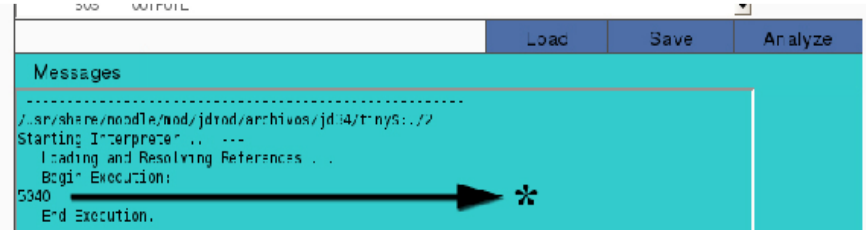

Figura 4.18: Resultados del Análisis de la Fase 3.

Capítulo 5

Evolución de las Iteraciones de la Aplicación Web

En el presente capítulo, se explica la metodología utilizada para desarrollar el software presentado en el capítulo anterior. Se definieron cuatro (4) iteraciones de acuerdo a los objetivos planteados. A continuación se detallan cada una de las iteraciones desarrolladas y las actividades aplicadas en ellas, pero antes se destacan las decisiones de diseño iniciales que se tomaron para realizar la aplicación Web.

5.1. Decisiones de Diseño Iniciales

El software desarrollado por Bermúdez en el 2003 [12], sirvió como base para realizar todo el desarrollo bajo la licencia de software libre GPL. Este software usa las herramientas LEX, YACC y GCC [13], por ello en este desarrollo se mantiene estas herramientas y se utilizan otras como Moodle. Para la elección de Moodle, se realizó una comparación

con algunas herramientas para la educación a distancia, tales como OLAT, eCollege, Web-Study Course Management System y ATutor [8]. En esta comparación se puede resaltar que Moodle es una herramienta bajo el licenciamiento de software libre GPL, su interfaz gráfica es agradable para el usuario inal, permite configurarse totalmente, entre otras razones. Además, se destaca que Moodle esta instalado y funcional en la Universidad Nacional Experimental del Táchira (UNET).

A continuación se describen las cuatro iteraciones del desarrollo de la aplicación Web. Estas iteraciones surgieron de acuerdo a los productos obtenidos y objetivos definidos en cada una de ellas.

5.2. Primera Iteración

Siempre que se desea desarrollar una aplicación, esta debe empezar por un bosquejo inicial, por lo tanto, a partir de reuniones previas y asignaturas cursadas con el Dr. Manuel Bermúdez, se definieron los lineamientos iniciales.

5.2.1. Formulación

Para la formulación, se comienza por el objetivo general y los objetivos específicos del proyecto, permitiendo definir la motivación del proyecto así como también un conjunto de metas necesarias para consolidar el Sistema de Escritura de Traductores vía Web.

La principal motivación de esta aplicación es que permitirá a todos los estudiantes de computación, conocer todas las fases del proceso de compilación a través de una herramien-

ta vía Web, donde el coordinador del curso podrá realizar un seguimiento del mismo sin importar el lugar donde se encuentre.

El desarrollo de este proyecto es necesario porque actualmente existe una desmotivación por el curso de Compiladores a nivel mundial en las carreras de Computación [12], debido a muchos factores, pero el principal es el patrón que ha existido por varios años en dicho curso. El patrón es el libro del Dragón Rojo de Aho [3]. Este método es excesivamente teórico, matemáticamente formal y tedioso, y ha logrado que los estudiantes de computación no puedan contar con un aprendizaje ideal de la teoría de compiladores. Las demás metodologías de enseñanza que se han propuesto, el ciento por ciento de todas ellas, estan plasmadas en los libros existentes y son prácticamente una copia del modelo de Aho [12].

Luego de exponer las razones del por qué del desarrollo de un sistema de escritura de traductores, se destacan las siguientes metas informativas:

- Se conocerán todas las fases de un compilador.
- El sitio proporcionará a los usuarios una descripción técnica y de uso de la herramienta.
- Podrá contar con un entorno constructivista donde se encontrará material digital que es colocado por el instructor del curso y por estudiantes, previa autorización del instructor.

Las metas antes expuestas nos indican cual será el enfoque de contenido y de información para el usuario final. Pero también se deben definir cuales serán las metas aplicables que orientarán sobre la habilidad que tendrá la WebApp:

- El usuario elabora sus propias reglas léxicas, semánticas y sintácticas.
- Se puede ejecutar desde cualquier parte del mundo, sólo se necesita una conexión de Internet y un navegador.
- Se encuentra en un entorno de aprendizaje a distancia, donde los usuarios pueden mejorar el conocimiento sobre el área, a través de todas las actividades con las que cuenta Moodle.

El ámbito del sistema estará definido en desempeño en los cursos de Compiladores e Intérpretes de la Universidad Nacional Experimental del Táchira (UNET) y de la Universidad de Florida (UF) en el futuro cercano, pretendiendo despertar el interés de los estudiantes y profesores de computación en profundizar en el tema.

5.2.2. Planificación

Antes de comenzar a elaborar el proyecto, se realizó un estimado del tiempo de desarrollo, el cual fue de seis meses aproximadamente. Este tiempo estipulado se basó en el desarrollo de cada una de las iteraciones, pero en realidad se desarrolló en aproximadamente un año. Las principales causas fueron:

- Nunca se cumplió con la dedicación diaria de cuatro horas según lo planificado, debido a que se realizaban otras actividades.
- Al principio, las revisiones con el tutor se planificaron por Internet por problemas de distancia, pero gracias a que se pudo coincidir con él en varios eventos académicos, se lograron avances significativos en la elaboración del trabajo de grado, los cuales no estaban previstos de esa manera.

- Hubo retrasos en el desarrollo del sistema propuesto porque Moodle tiene una escasa documentación y existen pocos trabajos de desarrollo de software en Moodle.

5.2.3. Análisis

El objetivo es poder identificar el espectro del contenido que se desea aplicar para los estudiantes, y se toman como base el trabajo de Bermúdez [13]. Por ello se analizan todos los subprocesos que él desarrolla, los cuales son:

- Análisis léxico y sintáctico a través de herramientas libres como LEX y YACC.
- Análisis de semántica estática, incluyendo revisión de tipos de los operadores.
- Generación de código de máquina para el código fuente deseado, basado en las reglas anteriores.
- Visualización la interacción de cada uno de los subprocesos, a través de los archivos generados por los mismos.

5.2.4. Ingeniería

La ingeniería de un proyecto Web es muy importante, ya que en ésta se diseña todo el proyecto. Para esta iteración se cuenta con el diseño de la interfaz de usuario a través de un bosquejo desarrollado en papel.

5.2.5. Evaluación del Cliente

El principal cliente por los momentos, es el Dr. Manuel Bermúdez que por más de veinte años ha trabajado en el área. El siempre estuvo involucrado en cada una de las actividades,

logrando de esta manera cubrir con la planificación de la primera iteración.

5.2.6. Productos Obtenidos

Se realizó un bosquejo en papel del sistema descrito. Con este producto se dío por finalizada esta iteración, para luego dar paso a una iteración donde se comienzó a implementar el sistema propuesto.

5.3. Segunda Iteración

En esta iteracion se implementó una primera versión del sistema de acuerdo a los resultados obtenidos hasta ese momento. A continuación se explica cada una de las actividades que se realizaron de acuerdo a la planificación.

5.3.1. Planificación

Luego de la primera iteración, se logró un primer bosquejo de la aplicación Web. Ahora se realiza el diseño y la implementación del software. Para ello, se conoce que la herramienta debe ser vía Web, de manera que cualquier usuario pueda conectarse a ella sin importar la distancia. Es importante resaltar que todo software que se implemente a través de la red, debe considerar la seguridad como un factor primordial porque cualquier persona puede atacar el sistema desarrollado, sobre todo si se usará a través de Internet, tal y como se planteó desde los objetivos del proyecto. Por eso se debe pensar en varias alternativas, tomando como decisión desarrollar el sistema en una primera etapa sin pensar en su seguridad, ya que la misma se adaptará a un sistema de educación a distancia y con una

plataforma con mecanismos de seguridad comprobados como es el caso de Moodle [15].

5.3.2. Análisis

Para crear el modelo del proyecto, se tomó en cuenta que el desarrollo será vía Web y con herramientas de Software Libre como es el caso de PHP [19] y MySQL [16]. El análisis se estructuró de la siguiente manera:

Análisis de la interacción:

- Se debe mantener y en lo posible mejorar, el esquema de trabajo realizado en el trabajo de Bermúdez.
- Debe funcionar vía Web, logrando un entorno más interactivo.
- Debe permitir importar reglas definidas por otras personas.

Análisis funcional:

- Obtener archivos que permitan identificar cada una de las fases de un proceso de compilación. Por ejemplo, el análisis léxico muestra los tokens generados a partir del código fuente.
- Poder ejecutar el código fuente, es decir, que exista entrada, proceso y salida.
- Poder modificar todas los cinco componentes del proceso: la parte léxica, sintáctica, semántica, generación de código, y ejecución.

Análisis de configuración:

- Primordialmente que sea multiplaforma.
- Para el instructor del curso, el sistema debe tener la capacidad de manejar múltiples usuarios y así poder verificar sus avances.
- Funcional para varios idiomas, inicialmente en inglés y español.

5.3.3. Ingeniería

Cuando se habla del aplicar la ingeniería para aplicaciones Web, evidentemente se hace referencia al diseño de la aplicación. El diseño esta enmarcado en tres partes: el arquitectónico, de navegación y de interfaz.

La aplicación dentro de Moodle, esta diseñada arquitectónicamente bajo una estructura lineal, es decir, la sucesión de interacciones es predecible. Esta sucesión se presenta en cinco fases, en las cuales no se podría avanzar si no esta definida la anterior. Las fases son: análisis léxico, análisis sintáctico, análisis semántico, generación de código, y ejecución.

Una vez definida la arquitectura Web, se estructuran las rutas de navegación para que el usuario pueda acceder al contenido y a los servicios de la WebApp. Esto es llevado a cabo según dos criterios:

1. Se definió una semántica para todos los usuarios, ya que el objetivo es conocer el proceso de compilación. Además, este usuario podrá realizar cualquier cambio a cualquier fase en forma independiente de otros usuarios.
2. El segundo criterio tiene que ver con la mecánica, es decir, como se logra aplicar la semántica. Por ser una aplicación Web, los enlaces existentes en los botones o celdas

de tablas, permiten a los usuarios cambiar a la fase deseada. También cuenta con un menú que permite cargar el código fuente y configurar aspectos importantes del entorno gráfico.

Finalmente, se define el diseño de la interfaz de usuario, el cual estará influenciado por la apariencia del estilo del Moodle donde este ubicado. A pesar de ello, se tomaron en cuenta algunos componentes de la interfaz Web [18], los cuales fueron:

- Tablas.
- Menús.
- Botones.
- Colores agradables.
- Varias ventanas o *frames* para presentar los diferentes servicios.

5.3.4. Generación de Páginas

Luego de un diseño basado en el análisis previo, se generaron las diferentes páginas que trabaja con cada unas de las fases. Todas ellas se realizaron con los lenguajes de programación: PHP, JavaScript [10] y AJAX [2]. No se definieron plantillas, ya que solamente se tiene un marco de trabajo principal y se llaman algunas funciones de AJAX y las funciones definidas por el programador. Además como la WebApp se ejecuta sobre un servidor Apache [4] bajo el sistema operativo GNU/Linux [6], existen acciones a través de botones que permiten ejecutar comandos en consola de GNU/Linux, y mostrar sus resultados. A continuación se muestra (ver figura 5.1) la aplicación obtenida en esta iteración:

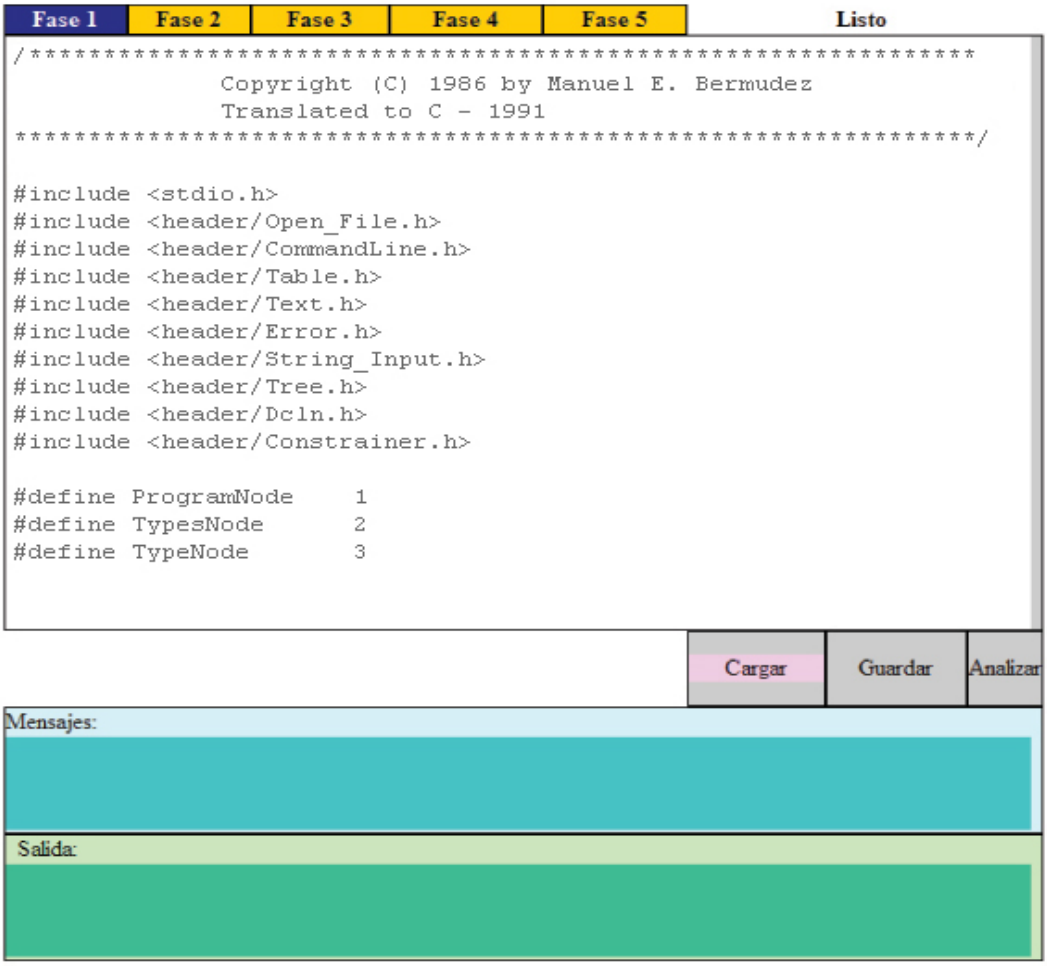

Figura 5.1: Primera Versión del Sistema Elaborado

5.3.5. Pruebas

Las pruebas del sistema desarrollado, no se realizaron al finalizar del desarrollo del mismo, sino todo lo contrario. A medida que el sistema fue evolucionando, se fueron aplicando las pruebas pertinentes. El proceso de pruebas para está iteración esta basado en pruebas de contenido, pruebas de la interfaz usuario y pruebas de navegación.

Pruebas de Contenido

Se aplican detectaron errores tipográficos, gramáticales, errores en la consistencia del contenido, inexactitudes en las representaciones gráficas y referencias cruzadas.

Errores sintácticos: se revisaron todas las páginas y los archivos de configuración de idiomas, ya que la aplicación se puede cambiar de idioma (por ejemplo de ingles a español)a través de un archivo donde se cargan todos los nombres de las etiquetas.

Errores semánticos: para detectarlos se revisó en base a la estructura del contenido, donde la información presentada no es precisa ni concisa, porque la información no está bien estructurada. Por ello se recomienda cambiarla para que cada fase se pueda profundizar y entender mejor el proceso de compilación.

Pruebas de la Interfaz

Mecanismos de la interfaz: para poder realizar una interacción con una aplicación Web, esta debe ocurrir a través de mecanismos que facilite el lenguaje de programación utilizado. En nuestro caso se usaron los siguientes: vínculos, formas y funciones (guiones).

- Vínculos: se probaron todos los vínculos, logrando el objetivo propuesto. Solamente faltó la acción de Analizar que no estaba implementada en ese momento.

- Formas: la aplicación presenta una forma general, la cual contiene algunas etiquetas o variables, las cuales fueron revisadas una a una, tanto su ID como sus características. Luego se verificó que la información transmitida al servidor llegará en forma correcta, y por último se revisaron todos los valores que tenían por defecto las variables.

- Funciones: se revisaron todas las funciones realizadas en AJAX y Javascript, desde su implementación hasta su impacto en la interfaz de usuario, es decir, se observan individualmente y luego se observan en forma integral.

Semántica de la interfaz: como se mencionó anteriormente, existía una incongruencia ya que no se lograba comprender cómo es el proceso de compilación. Por eso se recomiendó en la siguiente iteración profundizar en el análisis del sistema, para afianzar el significado o el objetivo principal del trabajo propuesto.

Compatibilidad: se probó la aplicación en los siguientes ambientes:

- Tres máquinas distintas con el sistema operativo Windows XP y con el navegador Firefox.

- Tres sistemas operativos: Windows XP, Debian y Centos.

- Tres navegadores: Firefox, Internet Explorer y Nestcape.

Pruebas de Navegación

Sintaxis de navegación: la herramienta de Dreamweaver 8 [7] cuenta con la opción de un comprobador de vínculos que permite revisar de forma automatizada todos los vínculos del sistema desarrollado, en la cual todos funcionaban según lo previsto.

Semántica de navegación: se revisaron todas las unidades semánticas de navegación (USN), un conjunto de estructuras de información y navegación relacionadas que tienen un objetivo específico. En esta iteración las USN revisadas fueron:

- Fase 1: Análisis léxico.
- Fase 2: Análisis sintáctico.
- Fase 3: Análisis semántico.
- Fase 4: Generación de código.
- Fase 5: Resultados.

5.3.6. Evaluación del Cliente

El cliente no se sintió satisfecho ya que la WebApp no cumplía con los objetivos planteados, que es poder comprender todo el proceso de compilación y tener el control de cada una de las subfases del proceso. Para esto se agregó acceso al código fuente. También en el sistema no existía una interacción con el cliente, que permitiera compilar los cambios realizados.

5.3.7. Productos Obtenidos

Se obtuvo una aplicación preliminar, con varias fallas de fondo así como de forma, como el uso de colores no agradables y limitaciones en la interactividad con el usuario. Por ello finalizó esta iteración, porque fue necesario desarrollar una versión que permitiera entender el todo el proceso de compilación.

5.4. Tercera Iteración

Luego de un resultado poco satisfactorio en la iteración anterior, se realizaron varias actividades que pretendieron mejorar el sistema desarrollado hasta ese momento.

5.4.1. Planificación

Después de los resultados de la iteración anterior, se replantearon algunas cosas. La primera fue que se debe realizar un análisis que permita integrar todos los subprocesos que comprenden un compilador. Además la aplicación debía mejorar su aspecto gráfico y debía permitir una mejor interoperatividad.

5.4.2. Análisis

Una vez definidos los lenguajes de programación, se redefinió el modelo diseñado. El análisis se estructuró de la siguiente manera:

Análisis de la interacción:

- Profundizar en las fases planteadas anteriormente.

- Permitir que el cliente tenga acceso a los códigos fuentes de los diversos lenguajes.
- Funcionar vía Web, logrando un entorno más interactivo.
- Permitir importar reglas definidas por otras personas.

Análisis funcional:

- Se redefinieron las fases, las cuales ahora trabajarán sobre los archivos que permiten las construcciones léxicas, sintácticas y semánticas, y archivos de generación de código.
- Se obtuvieron archivos como respuestas a estas construcciones, que permiten identificar cada una de esas fases.
- Se permitío ejecutar el código fuente, es decir, que exista entrada, proceso y salida.
- Se permitío modificar todas las condiciones del proceso, en la parte léxica, sintáctica, semántica, generación de código y de ejecución.

Análisis de configuración: el análisis realizado en las etapas anteriores se mantuvo, porque se cumplió con las expectativas planteadas, las cuales eran:

- Multiplaforma.
- Múltiples usuarios.
- Funcional para varios idiomas.

5.4.3. Ingeniería

Se debió plantear un diseño totalmente distinto al anterior. Por ello se plantó el un modelo con las siguientes fases:

- Compiler: permite al cliente manipular los códigos fuentes que generan las reglas del análisis léxicos, sintácticos, semánticos y la generación de código.
- Run: permite al cliente ejecutar las fases de un compilador a partir de un programa fuente de acuerdo a las condiciones definidas en la etapa anterior.
- Interpreter: el cliente obtiene la salida de la ejecución de su programa así como un archivo en un código de bajo nivel.

De igual forma el sistema esta diseñado arquitectónicamente bajo una estructura lineal, con una sucesión predecible de interacciones. En esta sucesión las fases diseñadas no permiten al usuario avanzar si no está definida la anterior. Los criterios restantes correspondientes al diseño de navegación y al diseño de configuración se mantuvieron tal y cual estaban definidos en la iteración anterior.

5.4.4. Generación de Páginas

Luego del diseño referente a esta iteración, se generaron las diferentes páginas que trabaja con cada unas de las fases y subfases. Todas ellas se realizaron con los lenguajes de programación: PHP, JavaScript y AJAX. A continuación se muestra (ver figura 5.2) la aplicación obtenida en esta iteración:

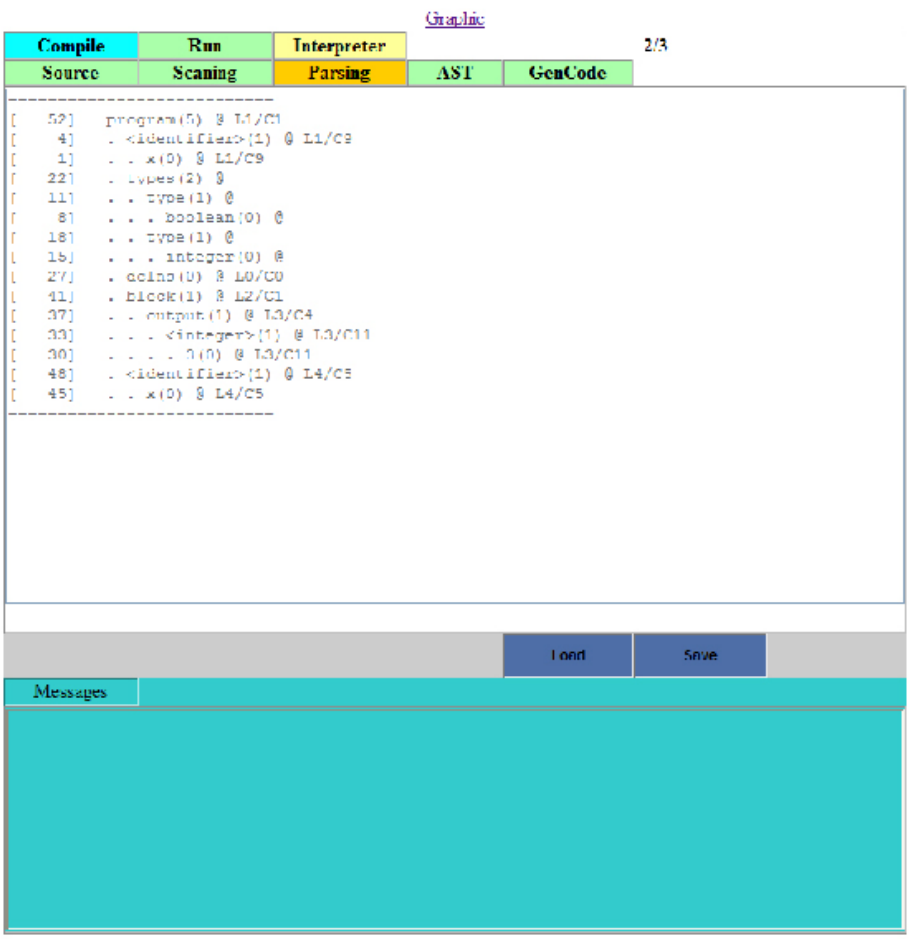

Figura 5.2: Segunda Versión del Sistema Elaborado.

5.4.5. Pruebas

Nuevamente se aplicaron las pruebas de la iteración anterior, para verificar la sintaxis y semántica del proyecto elaborado. Sin embargo se realizaron pruebas de configuración y semántica de la interfaz desarrollada.

Pruebas de Contenido

Se aplicaron para detectar errores tipográficos, gramaticales, errores en la consistencia del contenido, inexactitudes en las representaciones gráficas y referencias cruzadas.

Errores sintácticos: se revisaron todas las páginas y los archivos de configuración de idiomas.

Errores semánticos: se revisaron en base a la estructura del contenido, donde la información presentada ahora se considera coherente.

Pruebas de la interfaz

Mecanismos de la interfaz: para poder realizar una interacción con una aplicación Web, esta debe ocurrir a través de mecanismos que faciliten el lenguaje de programación utilizado. En nuestro caso se usaron los siguientes: vínculos, formas y funciones (guiones).

- Vínculos: se probaron todos los vínculos, logrando el objetivo propuesto para esta iteración.
- Formas: la aplicación presentaba una forma general, la cual contenía algunas etiquetas o variables, las cuales fueron revisadas una a una. También se verificó la información transmitida al servidor y los valores por defecto. Además se probó la entrada de información cuando se corre el código fuente del lenguaje implementado, a través de una etiqueta y un boton para ingresar la información.
- Funciones: se revisaron todas las funciones realizadas en AJAX y Javascript.

Semántica de la interfaz: la aplicación Web desarrollada ahora contaba con tres (3) fases y nueve (9), subfases las cuales fueron analizadas una a una, primero de forma independiente, y luego de forma integral.

Compatibilidad: se probó la aplicación en los siguientes ambientes:

- Dos máquinas distintas con el sistema operativo Windows XP.
- Tres sistemas operativos: Windows XP, Debian y Centos.
- Tres navegadores: Firefox, Internet Explorer y Nestcape.

Pruebas de Navegación

Sintaxis de navegación: nuevamente la herramienta de Dreamweaver 8 sirvió de ayuda para comprobar todos los vínculos del sistema desarrollado. Todos funcionaron según lo previsto.

Semántica de navegación: se revisaron todas las USN, que en esta iteración fueron:

- Fase 1: Compiler.
  - Scanner.
  - Parser.
  - Contrainer.
  - Generator.
- Fase 2: Run.

  - Source.
  - Scanning.
  - Parsing.
  - Constrain.
  - GenCode.

- Fase 3: Interpreter.
  - Code.

Pruebas de la Configuración

Se comprobó el funcionamiento de la aplicación, tanto del lado del servidor y del lado del cliente. Para comprobar los conflictos en el lado del servidor, se realizaron las siguientes tareas:

- Se instaló el cliente en el servidor y se comprobó su funcionamiento.
- Se verificaron los archivos del sistema, directorios y datos del sistema.
- Se verificó si la instalación de un cortafuego (firewall) puede incidir en la aplicación.
- Las funciones de la aplicación en el lado del servidor funcionaron apropiadamente.

Después de comprobar que no existían conflitos del lado del servidor, se debe verificó del lado del cliente, el uso de diferentes navegadores. Para este caso, el comportamiento fue el esperado, incluyendo la prueba de la aplicación en tres sistemas operativos.

5.4.6. Evaluación del Cliente

La evaluación fue satisfactoria porque ahora se podía percibir todo el proceso de compilación Además se logró tener acceso a los códigos que generan las condiciones generales de un compilador. Para la siguiente iteración se exige la seguridad y el manejo de usuarios por parte de la aplicación.

5.4.7. Productos Obtenidos

Se obtuvi una herramienta Web que permite comprender el proceso de compilación de un lenguaje de programación. Finalizada esta iteración, se debe realizar otra, con la integración de la herramienta de Moodle que reforzará las características faltantes al proyecto propuesto.

5.5. Cuarta Iteración

En esta iteración, se integró el sistema generado hasta el momento con la aplicación de Moodle. Se realizaron todas las tareas necesarias, y se volvió a comprobar la ausencia de todos los errores previstos en las iteraciones anteriores.

5.5.1. Planificación

Finalizada esta etapa importante del desarrollo de la WebApp (el corazón del sistema), se pasó a incorporarlo a el ambiente de Moodle. Para esto se tomaron previsiones en cuanto a:

- Las diversas versiones existentes.

- La escasa documentación del código.
- Incompatibilidad con el código a incorporar.
- Elaboraración de los manuales de usuario y del sistema.

La aplicación final deberiá ser ejecutada sobre Moodle. Por ello la herramienta elaborada hasta la iteración anterior debia ser la misma que apareciera en Moodle, lo cual no requirió las actividades de análisis, ingeniería y generación de páginas. La figura 5.3 presenta el trabajo realizado.

5.5.2. Pruebas

En esta actividad se aplicaron las mismas pruebas de la iteración anterior pero sobre la aplicación montada en Moodle, permitiendo verificar que todo lo elaborado esta funcionaba correctamente. Además, se elaboraron las mismas pruebas con dos usuarios distintos, pero con los mismos privilegios que permite Moodle.

Pruebas de Contenido

Se aplicaron para detectar errores tipográficos, gramaticales, errores en la consistencia del contenido, inexactitudes en las representaciones gracias y referencias cruzadas.

Errores sintácticos: se revisaron todas las páginas y los archivos de configuración de idiomas.

Errores semánticos: para detectarlos se debió revisar en base a la estructura del contenido, donde la información presentada ahora se considera coherente.

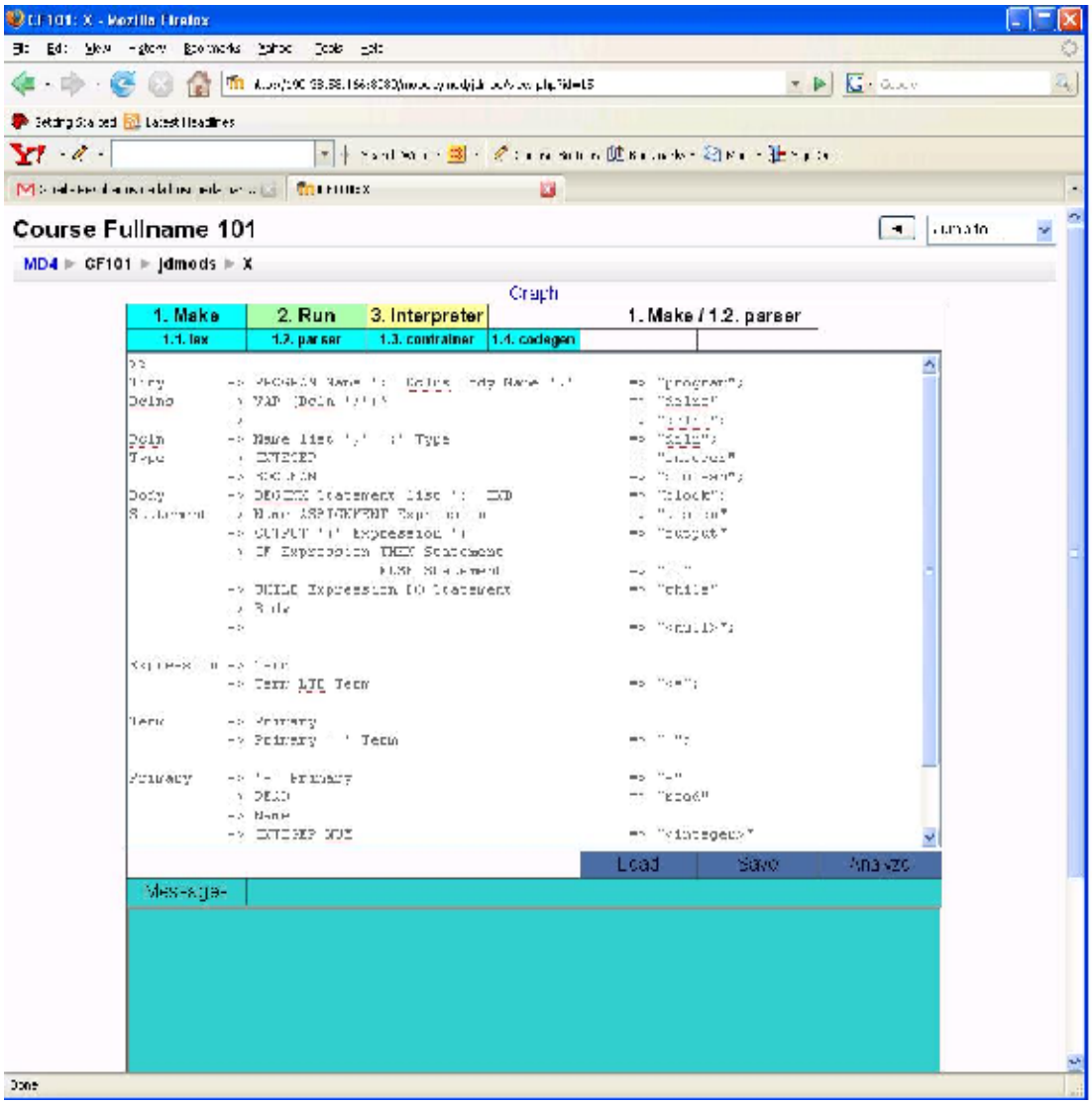

Figura 5.3: Versión Final del Sistema Elaborado.

Pruebas de la Interfaz

Mecanismos de la interfaz: se usaron los siguientes mecanismos: vínculos, formas y funciones (guiones).

- Vínculos: se probaron todos los vínculos, logrando el objetivo propuesto para esta it-

eración. También se logró el enlace de Moodle a la actividad del Sistema de Escritura de Traductores.

- Formas: se verificó la forma general con todas sus características.
- Funciones: se revisaron todas las funciones realizadas en AJAX y Javascript.

Semántica de la interfaz: la aplicación Web desarrollada ahora contaba con tres (3) fases y con nueve (9) subfases, las cuales fueron analizadas una a una, primero en forma independiente y luego en forma integral.

Compatibilidad: se probó nuevamente la aplicación en los siguientes ambientes:

- Dos máquinas distintas con el sistema operativo Windows XP.
- Tres sistemas operativos: Windows XP, Debian y Centos.
- Tres navegadores: Firefox, Internet Explorer y Nestcape.

Pruebas de Navegación

Sintaxis de navegación: se comprobaron todos los vínculos del sistema desarrollado.

Semántica de navegación: se revisaron todas las siguientes USN:

- Fase 1: Compiler.
  - Scanner.
  - Parser.
  - Contrainer.

    - Generator.

- Fase 2: Run.

    - Source.
    - Scanning.
    - Parsing.
    - Cons.
    - GenCode.

- Fase 3: Interpreter.

    - Code.

Pruebas de la Configuración

Se comprobó de nuevo el funcionamiento de la aplicación, tanto del lado del servidor como del lado del cliente. Para ello se revisaron los siguientes posibles conflictos:

Conflictos en el lado del servidor:

- Se instaló el cliente en el servidor y se comprobó su funcionamiento.
- Se verificaron los archivos del sistema, directorios y datos del sistema relacionados.
- Se verificó si la instalación de un cortafuego (firewall) puede incidir en la aplicación.
- Las funciones de la aplicación en el lado del servidor funcionaron correctamente.

Conflictos en el lado del cliente: se verificó el uso de diferentes navegadores y su comportamiento fue el esperado, incluyendo la prueba de la aplicación en tres sistemas operativos.

5.5.3. Evaluación del Cliente

Finalmente se logró una aplicación robusta con aceptación por parte del cliente. Los manuales de usuario y del sistema, también tuvieron una revisión satisfactoria.

5.5.4. Productos Obtenidos

Se obtuvo el objetivo principal planteado, un Sistema de Escritura de Traductores vía Web.

Conclusiones

- El Sistema de Escritura de Traductores desarrollado se caracteriza por estar implementado bajo un ambiente Web, donde no importa la plataforma donde se encuentre en cliente (navegador). Siempre que se tenga acceso desde Internet (servidor con acceso a Internet), el usuario podrá hacer uso de la herramienta desde cualquier parte del mundo. El modelo del compilador desarrollado se implementó bajo el paradigma de programación imperativo, que hace referencia a un lenguaje de recorrido completo que soporta las siguientes características: tipos de datos, estructuras de control, asignación de elementos, comandos de entradas y salidas, y procesamiento de caracteres, entre otras.

- Una vez analizado el Sistema de Escritura de Traductores desarrollado por Bermúdez, se implementó un sistema vía Web bajo un modelo lógico adaptado al sistema original. Este modelo lógico contiene en tres fases. La primera permite al usuario implementar las reglas léxicas, sintácticas, semánticas y la generación de código. La segunda fase permite compilar el programa fuente deseado y generar los archivos correspondientes a las subfases anteriores. La última fase permite ejecutar el código objeto generado.

- Ante el deseo de contribuir al desarrollo de Software Libre en el mundo y acogiéndonos a los lineamientos del gobierno Venezolano, el desarrollo del proyecto se realizó bajo la filosofía del Software Libre, logrando obtener un producto académico de calidad y fortaleciendo la lucha en las Universidades Venezolanas por promover el Software Libre.

- El proyecto realizado se ejecutó en cuatros fases. En cada una de ellas se realizaron

las pruebas pertinentes de acuerdo a la metodología aplicada, alcanzando el objetivo propuesto: una herramienta Web que mejore la pedagogía en la enseñanza de los Compiladores. Siguiendo la filosofía del Software Libre, el acceso al código fuente está disponible en un archivo ubicado en la sección de archivos del curso creado en Moodle, así como en la página personal del Dr. Bermúdez y del autor de este proyecto.

Recomendaciones

- Es necesario fomentar la investigación en el campo de los Compiladores e Intérpretes. Para ello se recomienda crear una mayor concientización sobre las ventajas que los estudiantes de computación y profesores puedan obtener, al dominar un tema en el cual la pedagogía es limitada y la bibliografía de calidad es escasa.

- Se recomienda colocar el proyecto en un sitio Web donde una comunidad de desarrollo de Software Libre, pueda tener acceso a él y mejorarlo en pro del estudio de los compiladores.

- Se recomienda introducir el proyecto desarrollado en la comunidad de Moodle, de manera que la comunidad pueda sugerir cambios que no afecten las versiones futuras que realice esta comunidad.

- Se recomienda realizar pruebas con los estudiantes de los cursos de Compiladores de la Universidad Nacional Experimental del Táchira y de la Universidad de Florida, de manera que surja una siguiente versión después del uso en estos cursos.

- En el curso creado de Compiladores en Moodle, se recomienda incentivar a través de materiales didácticos a los diferentes participantes, a estudiar e investigar sobre el tema de los compiladores, y lograr concretar el concepto de aprendizaje constructivista que promueve Moodle.

- Se recomienda para la siguiente versión investigar, y en tal caso desarrollar, un proceso bajo GNU/Linux que permita al software implementado, el ingreso de información a medida que se ejecuta el código objeto del programa fuente deseado.

Referencia Bibliográficas

[1] ACM Computing Curricula, Final Draft. 15 Diciembre del 2001. http://www.computer.org/education/cc2001/final. http://www.sigcse.org/cc2001/.

[2] Ajax. Página Web. 2007. http://es.wikipedia.org/wiki/AJAX

[3] Alfred Aho, Ravi Sethi y Jeffrey Ullman. Compiladores - Principios, técnicas y herramientas. *Primera Edición. Editorial Addison Wesley*. Wilminton, Estados Unidos. 1990.

[4] Apache. Página Web. 2007. http://www.apache.org/.

[5] Culebro, Goméz y Torres. Software libre vs. Software propietario. 2007. http://www.rebelion.org/docs/32693.pdf.

[6] Debian - GNU/Linux. Página Web. 2007. http://www.debian.org/.

[7] Dreamweaver 8. Página Web. 2007. http://www.adobe.com/dreamweaver/.

[8] EduTools - Course Management System (CMS). Página Web. 2007. http://www.edutools.info/itemlist.jsp?pj=4.

[9] FIRST, German National Research Center for Information Technology. Compiler Tools. Página web. Septimbre 2006. http://catalog.compilertools.net/

[10] JavaScript Página Web. 2007. http://es.wikipedia.org/wiki/JavaScript

[11] Manuel Bermúdez. Course Principles of Programming Languages. Página web, Septiembre 2006. http://www.cise.ufl.edu/class/cop5555su06/index.html

[12] Manuel Bermúdez. Modernización de la Enseñanza de la Traducción, XI Congreso Iberoamericano de Educación Superior en Computación CIESC-03. September 29 - October 3, La Paz, Bolivia.

[13] Manuel Bermúdez. Traductor Writing System (TWS). Página web. 2003. http://www.cise.ufl.edu/class/cop5555su06/language.notes.html

[14] Ministerio de Ciencia y Tecnología (MCT). Libro Amarillo del Software Libre. *Primera Edición.. MCT*. Caracas, Venezuela. 2004.

[15] Moodle - Sistema para la Administración de Cursos. http://www.moodle.org. 2007

[16] MySQL. Página Web. 2007. http://www.mysql.com/.

[17] Richard Stallman. Software libre para una sociedad libre. *Traficantes de Sueños.* España. 2004.

[18] Roger Pressman. Ingeniería del Software. *McGraw Hill. Quinta Edición.* Madrid, España. 2002.

[19] PHP. Página Web. 2007. http://www.php.net/.

[20] Proyecto GNU. Página Web. 2007. http://www.gnu.org.

[21] Sergio Galves y Miguel Mora. Traductores y Compiladores. *Universidad de Malaga. Primera Edición.* Madrid, España. 2005.

[22] The Institute of Electrical and Electronics Engineers. Página Web. 2007. http://www.ieee.org.

[23] Tim Berners-Lee. Proposal for a HyperText Project. CERN. Primera Edición. 1990.

[24] Tucker A. y Noonan R. Programming Languages, Principles and Paradigms. *Editorial McGraw Hill.* Estados Unidos. 2003.

[25] Universidad Pedagógica Experimental Libertador (UPEL). Manual de Trabajos de Grado de Especialización y Maestría y Tesis Doctorales. *UPEL. Tercera Edición.* Caracas, Venezuela. 2003.

[26] Wikipedia La Enciclopedia Libre. http://es.wikipedia.org/wiki.

Apéndice A

Manual del Usuario

El sistema de escritura de traductores se encuentra implementado bajo una actividad perteneciente a Moodle. El acceso se realiza a través de un navegador Web, el recomendado es FireFox de Mozilla. La dirección de la página sería local (localhost), ya que la aplicación está instalada en la misma máquina. Primero se se solicita el login y el password del usuario. Una vez que sean correctos, se selecciona el curso de Compiladores existente. La presentación se muestra en la figura A.1.

Se selecciona la actividad del curso de Compiladores para poder usar el compilador desarrollado (ver figura A.2). Este compilador debe estar anexado en el curso correspodiente.

El sistema consta de tres fases principales distribuidas de la siguiente manera:

- Fase 1 - Compiler: en este módulo se definen las reglas del compilador , es decir, como funcionará el analizador léxico (a través del comando LEX en GNU/Linux) y el analizador sintáctico (a través del comando YACC en GNU/ Linux). También

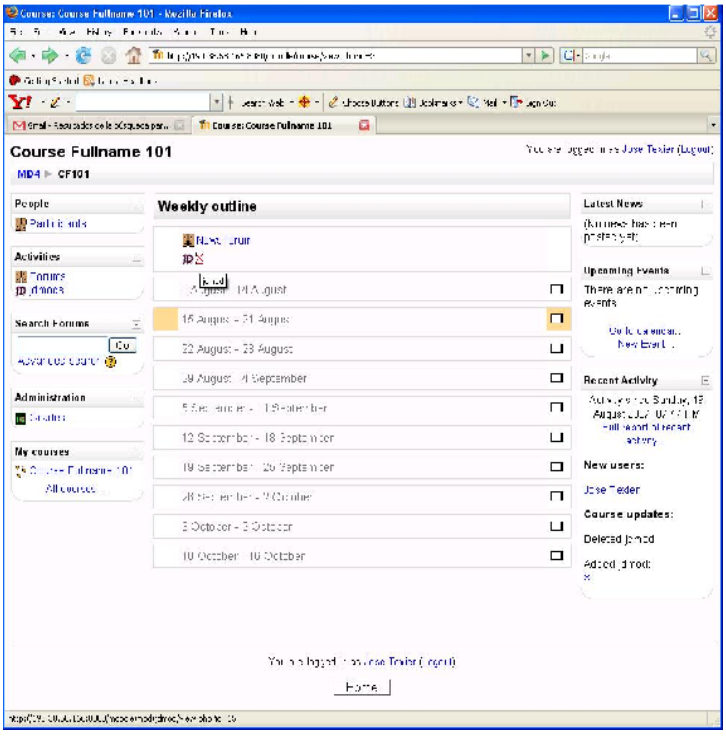

Figura A.1: Acceso al Sistema Vía Moodle

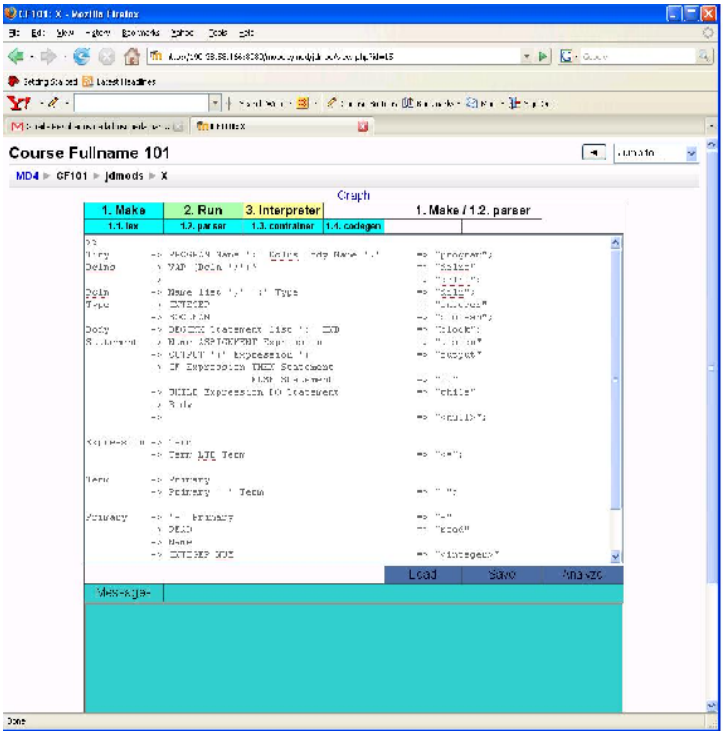

Figura A.2: Sistema Desarrollado

permite definir como será la construcción del árbol de sintaxis abstracta, la revisión de semántica estática y la generación de código.

- Fase 2 - Run: a partir de un programa fuente, se obtienen sus tokens, un árbol sintáctico, su revisión semántica y el código de máquina.

- Fase 3 - Interpreter: este módulo permite ejecutar el código de máquina generado y mostrar la secuencia de pasos de la ejecución del programa.

La distribución general del sistema es la siguiente:

- Fase 1: Compiler.
  - Scanner (ver figura A.3).
  - Parser (ver figura A.4).
  - Contrainer (ver figura A.5).
  - Generator (ver figura A.6).
- Fase 2: Run.
  - Source (ver figura A.7).
  - Scanning (ver figura A.8).
  - Parsing (ver figura A.9).
  - Cons (ver figura A.10).
  - GenCode (ver figura A.11).
- Fase 3: Interpreter - Code (ver figura A.12).

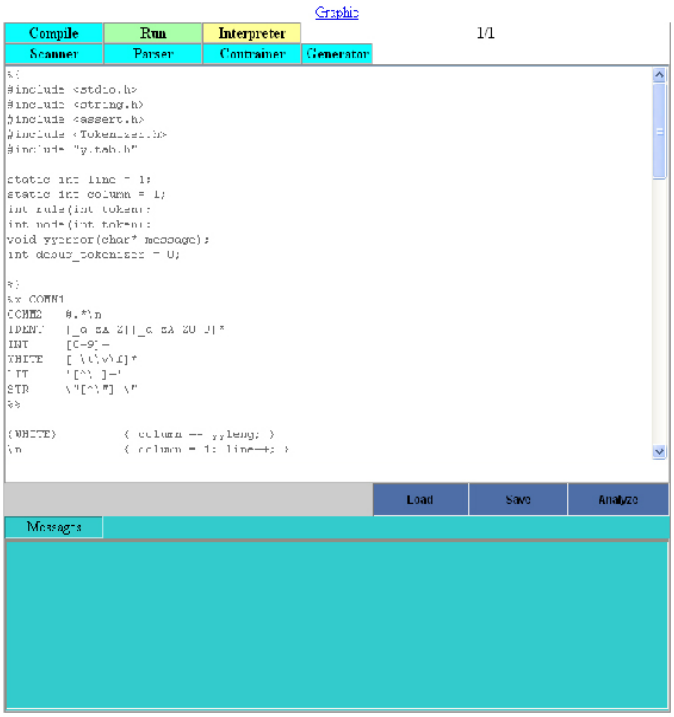

Figura A.3: Fase 1/1 - Scanner - Análisis Léxico

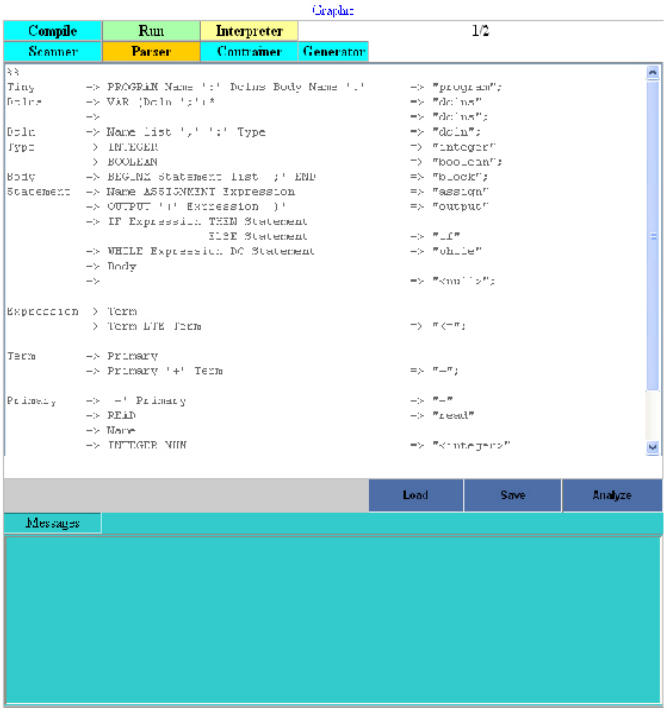

Figura A.4: Fase 1/2 - Parser - Análisis Sintáctico

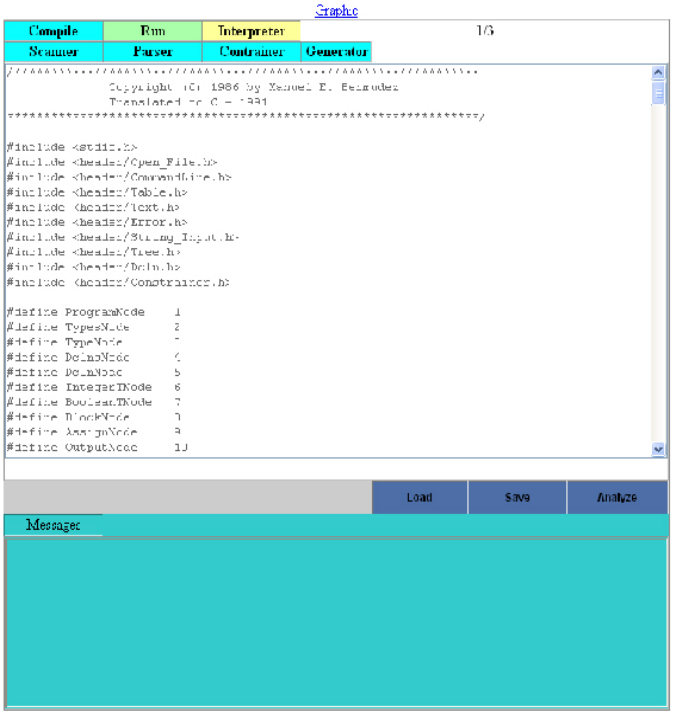

Figura A.5: Fase 1/3 - Constrainer - Análisis Semántico

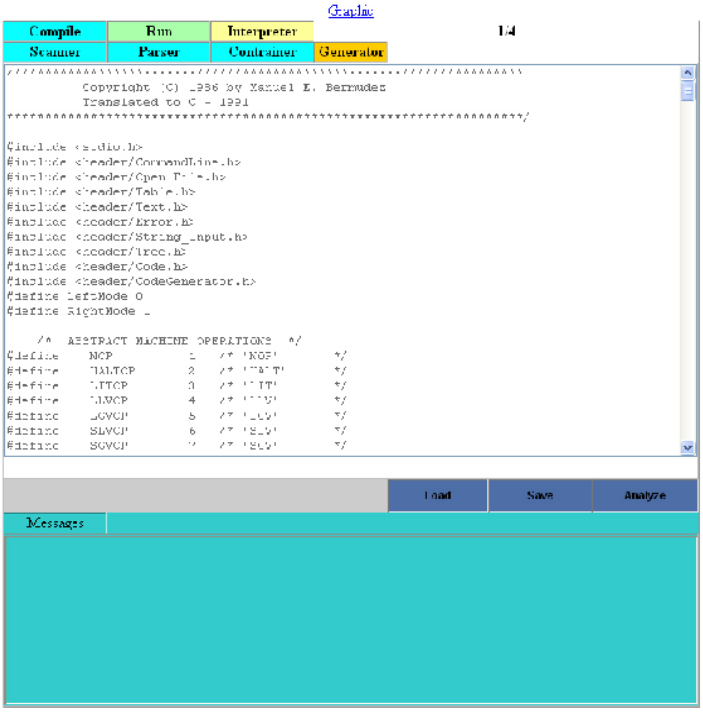

Figura A.6: Fase 1/4 - Generator - Generación de Código

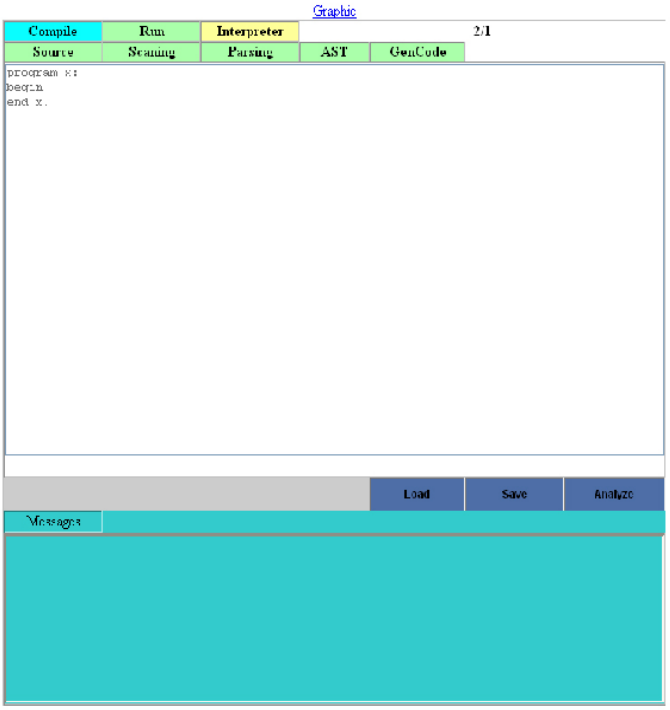

Figura A.7: Fase 2/1 - Source - Código Fuente

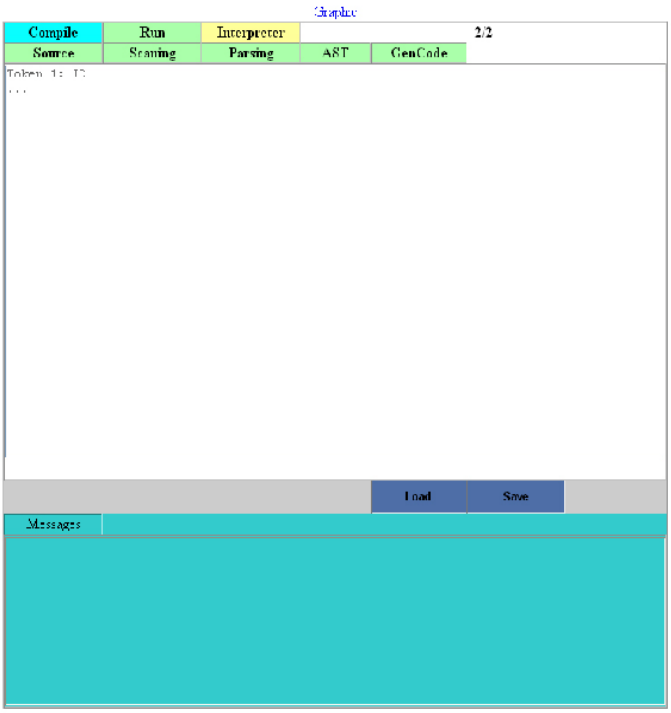

Figura A.8: Fase 2/2 - Scanning - Tokens

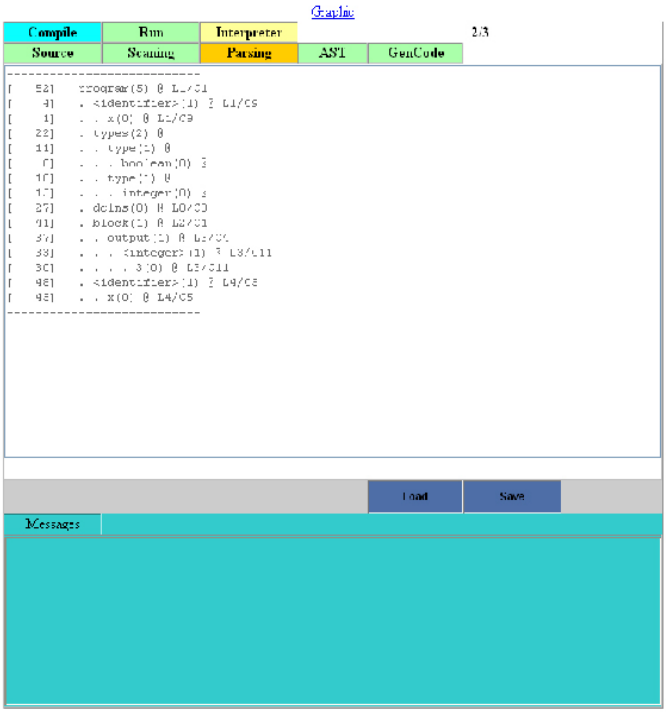

Figura A.9: Fase 2/3 - Parsing - Árbol Sintáctico

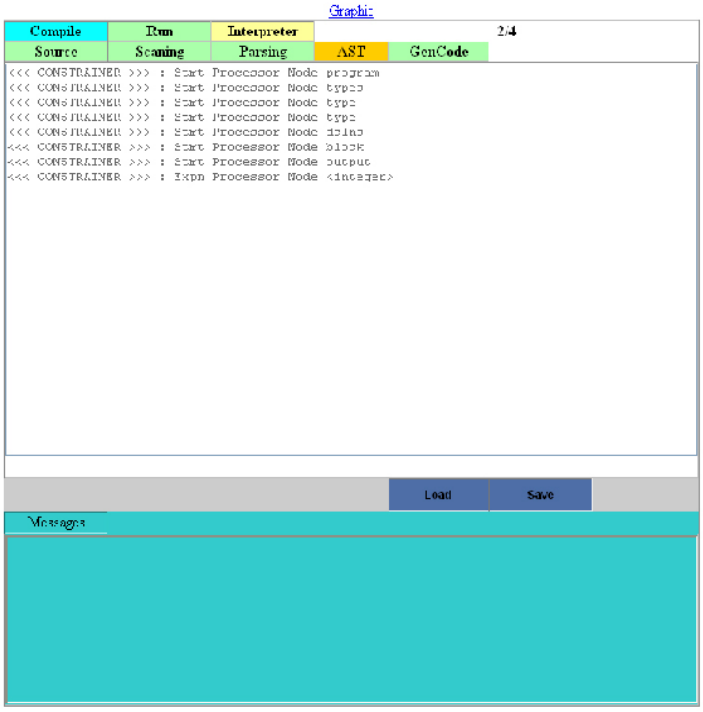

Figura A.10: Fase 2/4 - AST - Semántica

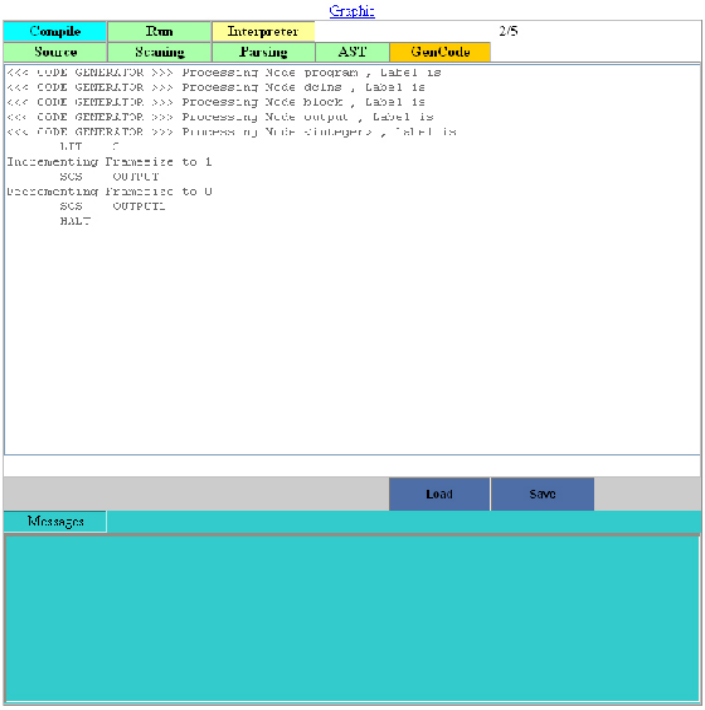

Figura A.11: Fase 2/5 - GenCode - Lenguaje de Máquina

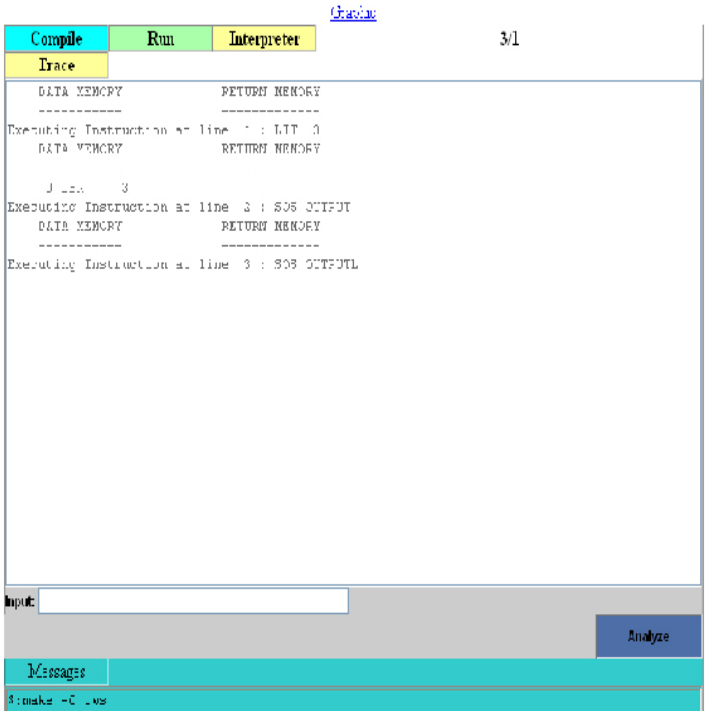

Figura A.12: Fase 3/1 - Trace - Secuencia de Ejecución del Programa

Apéndice B

Manual del Sistema

Este manual permite a cualquier usuario, instalar el Sistema de Escritura de Traductores para colocarlo al servicio de un grupo de estudiantes o personas interesadas en entender y mejorar los conocimientos en el tema de los Compiladores.

Requerimientos hardware : se necesita tener un computador que sirva de servidor para poder instalar la herramienta Web. Los requerimientos de hardware mínimos son:

- Procesador: Superior Intel Pentium IVI a 1.600 MHz.
- Memoria RAM: 1.024 MB.
- Configuración del Monitor: a color de 24 bits.

Requerimientos software : para poder usar la herramienta primero debe estar instalado Moodle (versión 1.6) en servidor Web. Pero antes Moodle debe encontrar las siguientes condiciones de Software en el servidor:

- El Sistema Operativo Debian GNU/Linux que debe tener instalado los comandos de LEX y YACC. Además debe tener una versión superior al GCC 3.9.

- Un servidor Web. La mayoría de los usuarios usan Apache, pero Moodle funciona con cualquier servidor Web que soporte PHP, como el IIS (Internet Information Server) de las plataformas Windows. En este caso se recomienda Apache.

- PHP versión superior a 4.3.0. PHP 5 está soportado a partir de Moodle 1.4.

- Una base de datos: MySQL o PostgreSQL, que están completamente soportadas y recomendadas para su uso con Moodle. MySQL es la elección preferida. MySQL 4.1.16 es la versión mínima para trabajar con Moodle 1.6.

- Agregar un módulo a Moodle colocando el Software desarrollado en la carpeta de *mod* de Moodle.

Luego de tener instalado los requerimientos de software anteriores, se debe configurar Moodle de manera que exista el curso de Compiladores en Moodle.

Como crear un nuevo curso en Moodle : se asume que Moodle ahora está funcionando correctamente. Ahora se deben seguir los siguientes pasos:

- Seleccione **Crear un nuevo curso** desde la página **Admin** (o desde los enlaces de administración en la página principal).

- Complete el formulario prestando especial atención al formato del curso. En este momento no tiene que preocuparse demasiado por los detalles, pues todo puede ser cambiado después por el profesor.

- Presione **Guardar cambios**; aparecerá un nuevo formulario en el que puede asignar profesores al curso. Desde este formulario sólo pueden añadirse cuentas de usuarios existentes, si necesita una cuenta para un profesor debe pedirle al profesor que cree su cuenta él mismo o créela usted utilizando la opción **Añadir nuevo usuario** en la página Admin.

- Una vez hecho esto, el curso está listo para ser personalizado y puede accederse al mismo a través del enlace **Cursos** en la página principal.

- Para obtener más detalles sobre la creación de cursos, vea el **Manual del Profesor de Moodle** [15].

- Por último, agregar el módulo al curso creado a través de la asignación de nuevas actividades.

Estructura del sitio Moodle : se presenta un breve resumen de los contenidos del directorio Moodle, que sirven para orientar al usuario que esté realizando la instalación:

- **config.php** - contiene la configuración fundamental. Este archivo no viene con Moodle - lo creará el instalador del sistema.

- **install.php** - el script que ejecutará para crear el archivo config.php.

- **version.php** - define la versión actual del código de Moodle.

- **index.php** - la página principal del sitio.

- **admin/** - Código para administrar todo el servidor.

- **auth/** - Módulos para la autenticación de usuarios.

- **blocks/** - Módulos para los pequeños bloques laterales contenidos en muchas páginas.
- **calendar/** - Código para manejar y mostrar eventos de calendario.
- **course/** - Código para presentar y gestionar los cursos.
- **doc/** - Documentación de ayuda de Moodle. (Por ejemplo esta página).
- **files/** - Código para presentar y gestionar los archivos cargados.
- **lang/** - Textos en diferentes idiomas, un directorio por idioma.
- **lib/** - Librerías del código fundamental de Moodle.
- **login/** - Código para manejar las entradas y creación de cuentas.
- **mod/** - Todos los módulos de los cursos de Moodle.
- **pix/** - Gráficos genéricos del sitio.
- **theme/** - Paquetes de temas/pieles para cambiar la apariencia del sitio.
- **user/** - Código para mostrar y gestionar los usuarios.

Donde descargar los archivos

- Moodle en http://moodle.org/download/
- Debian GNU/Linux en http://www.debian.org/
- Apache en http://www.apache.org/

- PHP en http://www.php.net/

- MySQL en http://www.mysql.com/