Universidad de las Ciencias Informáticas

Facultad 5

Título: Sistema de generación de intérpretes para los laboratorios virtuales.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor: Antonio Luis Rivero Perea.

Tutor: Ing. Gelson Rafael Saurín Ojeda.

Cotutor: Ing. Juan Miguel Rodríguez Sillero.

La Habana

2012

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo al Centro de Informática Industrial de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Antonio Luis Rivero Perea.

_____

Ing. Gelson Rafael Saurín Ojeda

_____

Ing. Juan Miguel Rodríguez Sillero.

_____

Datos de contacto

Ing. Gelson Rafael Saurín Ojeda (grsaurin@uci.cu**)**

Edad: 28 años.

Ciudadanía: cubano.

Categoría Docente: Profesor Instructor.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

Graduado de: Universidad de las Ciencias Informáticas, año 2008.

Ing. Juan Miguel Rodríguez Sillero (jmsillero@uci.cu**)**

Edad: 25 años.

Ciudadanía: cubano.

Categoría Docente: ninguna.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

Graduado de: Universidad de las Ciencias Informáticas, año 2011.

Dedicatoria

A mis padres, por darme tanto amor, por estar siempre a mi lado, por ayudarme a conseguir mis sueños, si he llegado hasta aquí hoy, fue por ellos.

A mis hermanos, todos con diferente carácter, pero queriéndome de igual manera, a Grayde por tranquilizarme con solo oír su voz en el teléfono, a Rolando, el más cariñoso y loco de nosotros, a Luis Manuel que ojalá esté orgulloso de su hermano hoy, y sea un ejemplo a seguir para él.

A mi abuelo por apoyarme y quererme tanto.

A Lizi, por llegar a mi vida y cambiarla completa, por ser mi mejor amiga, mi compañera, por hacerme tan feliz.

Agradecimientos.

Quisiera agradecer a todas aquellas personas que han estado a mi lado y han ayudado a la realización de esta investigación.

A mis padres:

Mi papá que ha sido el mejor ejemplo a seguir, entre todos, y mi mamá protegiéndome más que a ella misma, sonriéndome a veces, aun sintiéndose mal para que no sufriera yo. Gracias, no es la palabra para lo que les debo.

A mis hermanos:

Por apoyarme siempre y tratar de que su hermano cumpliera su sueño.

A Lizi:

Por pensar en mí antes que en ella, por lograr que yo viera la vida de otra forma, por amarme, por comprenderme, por darle color a mis días, gracias.

A mis tíos:

Que de una manera u otra siempre me han ayudado, a Albertico por ser el padre de todos nosotros, y estar siempre pendiente de nuestras necesidades.

A mis amigos:

Muchos de los cuales no pueden estar presentes hoy, pues la distancia nos separa, Tony, Capote, Marcos, Roy, Yusi, a los que han estado a mi lado en los peores momentos, Daisy, Ariel, a los que aun estando lejos, han velado por mí, Yosiel. A Aquellos que en la investigación de este trabajo pasaron noches enteras sin dormir, solo para ayudarme, Jimmi, Gelson.

Gracias a todos.

Resumen

El proyecto Laboratorios Virtuales (LV) del Centro de Informática Industrial realiza el proceso de detección y validación de errores en la aplicación de los laboratorios virtuales. Estos intérpretes son creados para cada ejercicio e incluyen en un núcleo todas las herramientas de validación necesarias. El proyecto no cuenta con un mecanismo capaz de agilizar el proceso de generación de estos intérpretes. Esta necesidad puede influir en el tiempo de desarrollo y en la capacidad de extender el uso de estas herramientas a otras ramas en las que pueden ser utilizadas. Para dar solución a dicho problema se realiza el presente trabajo de diploma que tiene como objetivo principal desarrollar un sistema capaz de agilizar la generación de intérpretes para el proceso de detección y validación de errores de los laboratorios virtuales.

La presente investigación se enmarca en el proceso de compilación, fundamentalmente en las fases de análisis léxico y detección de errores sintácticos. De esta forma, se obtuvo la primera versión de un sistema que agiliza la generación de intérpretes en el proceso de detección y validación de errores. Esta afirmación es respaldada por un conjunto de pruebas realizadas al sistema que validan la efectividad de las funcionalidades implementadas y la agilización del proceso de generación de intérpretes.

Palabras clave:

Gestión de errores, intérpretes, laboratorios virtuales, proceso de compilación.

Summary

The project Virtual Laboratory (VL) of the Industrial Computing Center performs the detection process and validation errors in the application of virtual laboratories. These interpreters are created for each exercise and include a core all necessary validation tools. The project does not have a mechanism to expedite the process of generation of these interpreters. This need can influence the development time and the ability to extend the use of these tools to other branches in which can be used. To solve this problem is made this diploma work has as main objective to develop a system to streamline the generation of interpreters for the process error detection and validation of Virtual Labs.

This research is part of the build process, mainly in the stages of lexical analysis and syntax error detection. Thus, we obtained the first version of a system that speeds the generation of performers in the process of error detection and validation. This assertion is supported by a set of tests the system to validate the effectiveness of the implemented functionality and speed up the process of generation of performers.

Keywords:
Error handling, interpreters, virtual labs, build process.

Índice de contenido.

Índice de tablas.

Índice de ilustraciones.

Introducción

La sociedad actual se ve estrechamente relacionada con el auge de las Tecnologías de la Información y las Comunicaciones (TIC), por lo que su evolución influye fuertemente sobre el proceso de desarrollo de las distintas esferas. La educación es una de las principales áreas donde se evidencia este creciente desarrollo tecnológico que ha revolucionado la enseñanza, incorporándole nuevas formas y métodos de aprendizaje, contribuyendo así al desarrollo cognoscitivo del estudiante.

La utilización de las tecnologías y los escenarios virtuales provee una opción para contrarrestar el déficit de materiales y recursos con los que elaborar ejercicios de laboratorio de forma tradicional. Tratando de recrear un ambiente real, surge la idea de un laboratorio virtual (LV), donde los estudiantes pueden adentrarse en un mundo virtual y aprender a trabajar con recursos ilimitados, lo cual no sería posible en la vida real debido a la carencia de estos.

En la Universidad de las Ciencias informáticas (UCI) se pone en práctica dicha idea, específicamente en la facultad 5, con la creación del proyecto Laboratorios Virtuales que pertenece al Centro de Informática Industrial (CEDIN). Este proyecto ya cuenta con experiencia en la creación de este tipo de software para el aprendizaje.

Para la creación del laboratorio virtual Configuración y Administración de una red de área local, desarrollado por este proyecto, se necesitaba validar varias configuraciones que permitieran servicios de dirección IP[1], FTP[2], DNS[3], etc. Para cumplir con este objetivo, fue necesaria la implementación de un proceso de compilación que tomara como entrada una configuración y realizara a partir de la misma el análisis léxico,

[1] (Protocolo de internet), protocolo no orientado a conexión, usado tanto por el origen como por el destino para la comunicación de datos, a través de una red de paquetes conmutados no fiable y de mejor entrega posible sin garantías.
[2] (Protocolo de transferencia de archivos), protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP.
[3](Sistema de nombres de dominios) forma alternativa de identificar a una máquina conectada a Internet.

sintáctico y semántico, teniendo como resultado que los estudiantes se percataran de los errores cometidos en el mismo momento que efectuaban el proceso de configuración o administración de una red de área local (LAN) virtualmente. De esta forma surge un módulo de intérpretes como respuesta satisfactoria al problema que existía en ese momento. Pero, el módulo obtenido, solo permite detectar errores en los servicios para los cuales fue implementado.

En dicho proyecto el proceso de generación de intérpretes consiste en crear para cada ejercicio un núcleo completo que incluye todas las herramientas de validaciones necesarias; por lo que, su ampliación o la creación de nuevos intérpretes para estas validaciones se hace engorrosa. Un desarrollador para realizar un intérprete ejecuta los siguientes pasos:

Tareas	Tiempo
Re-copiar la aplicación y cambiar los nombres de todos los .h, .cpp, métodos y *namespace*.	12 h
Redefinir los *Tokens*.	1 h
Redefinir el *Visitor* (.h y .cpp).	1 h
Redefinir el analizador Léxico (*lexicalanalizer.h* y *.cpp*).	3 h
Redefinir el Árbol de SA (*AbstractStaticTree*.*h* y *.cpp*).	4 h
Redefinir el analizador sintáctico (*SyntacticAnalyzer.h* y *.cpp*).	12 h
Redefinir el analizador semántico (*SemanticAnalyzer.h* y *.cpp*).	12 h
Analizar dependencias entre clases, para erradicar problemas de no reconocimiento de código.	4 h
Total de horas	49 h

Tabla 1: Pasos del proceso de generación de intérpretes.

Debido a la aceptación que han tenido los laboratorios virtuales, y la necesidad de ampliar la utilización de estos hacia otros campos o áreas del conocimiento, el proceso de detección de errores va más allá de realizar validaciones solamente para estos servicios de red.

El proyecto Laboratorio Virtuales no cuenta con un mecanismo capaz de agilizar el proceso de generación de intérpretes. Esta necesidad puede influir significativamente en el tiempo de desarrollo de estos intérpretes y en la capacidad de extender el uso de las herramientas de validación y detección de errores a otras ramas en las que pueden ser utilizados los LV.

A partir de la **Situación Problemática** anteriormente explicada, surge el siguiente **Problema Científico:** ¿Cómo agilizar el mecanismo de generación de intérpretes en el proceso de validación y detección de errores para la ampliación de los laboratorios virtuales a otras ramas?

Este problema tiene como **Objeto de Estudio** la generación de intérpretes.

Se propone como **Objetivo General** desarrollar una herramienta capaz de agilizar la generación de intérpretes en el proceso de validación y detección de errores de los laboratorios virtuales.

La investigación enmarca su **Campo de Acción**, de acuerdo al Objetivo General planteado, en la generación de Intérpretes en el proceso de validación y detección de errores de los laboratorios virtuales.

Por tanto, se enuncia como **Idea a Defender** que con el desarrollo de esta herramienta se podrá agilizar la generación de intérpretes y contribuir a la ampliación a otras ramas de los procesos de validación que se ejecutan en cada laboratorio virtual.

Conforme a lo anteriormente planteado se definen las siguientes **Tareas de la Investigación**:

- Elaboración del marco teórico a través del estudio del estado del arte de los laboratorios virtuales y las técnicas de compilación.

- Selección de las tecnologías y herramientas libres a utilizar para el desarrollo del sistema de generación de intérpretes.

- Diseño de los componentes que integran el sistema de generación de intérpretes.

- Implementación y prueba de los componentes que integran el sistema de generación de intérpretes, aplicando las normas internacionales que garantizan la calidad requerida del producto.

Durante el desarrollo del presente estudio se hará uso de los siguientes métodos de la investigación:

Métodos Teóricos:

- **Analítico-Sintético:** Este recurso permite realizar el análisis y estudio de las bibliografías existentes sobre el tema en cuestión y lograr obtener de manera sintetizada el contenido necesario y suficiente para la ejecución de la investigación.

- **Histórico-Lógico:** Este método se emplea para conocer la evolución y desarrollo de las etapas principales en el desarrollo de intérpretes.

Métodos Empíricos:

- **Entrevistas:** Método que permite obtener información de los clientes, así como los requisitos funcionales y no funcionales del sistema a desarrollar, logrando así el buen entendimiento entre cliente y desarrollador, y una mejor aceptación del producto.

- **Consultas de materiales bibliográficos:** Este método brinda la posibilidad de reunir información actual sobre el proceso de compilación de intérpretes y las características fundamentales para su generación.

- **Consulta a especialistas:** Las consultas a especialistas se utilizan para obtener criterios válidos sobre el trabajo realizado.

- **Encuestas:** Este método permite determinar la necesidad de crear un sistema que agilice el proceso de generación de intérpretes.

Al concluir este trabajo se espera obtener como **principal resultado** la primera versión de un sistema capaz de agilizar la generación de intérpretes utilizados para el proceso de detección y validación de errores, para el proyecto Laboratorios Virtuales del CEDIN.

El trabajo de diploma está estructurado en 3 capítulos donde se detalla el proceso investigativo realizado:

- **Capítulo 1:** Fundamentación Teórica.
  El marco teórico de la presente investigación engloba las principales características y conceptos de intérpretes y compiladores, así como las fases que los integran, poniendo especial énfasis en las etapas de análisis léxico y detección de errores sintácticos y el uso de herramientas en la gestión de errores. Se investiga sobre las principales herramientas generadoras de intérpretes en la actualidad, sobre las metodologías y herramientas a seleccionar para el desarrollo de la aplicación. Se aplica una encuesta sobre el proceso de generación existente en el proyecto Laboratorios Virtuales.

- **Capítulo 2:** Descripción de la solución propuesta.
  En el presente capítulo se realiza el proceso de modelación del sistema y se ofrece una descripción de sus características. Para ello se especifican las funcionalidades y cualidades con que debe cumplir el sistema y se construyen varios diagramas que posibilitan un mejor entendimiento de su funcionamiento.

- **Capítulo 3:** Implementación y validación del sistema.
  Se describe la implementación de la aplicación y se valida la solución propuesta mediante pruebas realizadas a las funcionalidades, para demostrar la agilización de los procesos existentes de generación de intérpretes.

Capítulo 1: Fundamentación teórica.

Introducción.

En el presente capítulo se exponen un grupo de conceptos y características fundamentales sobre los temas que componen el marco de la investigación. Se definen aspectos relacionados con los laboratorios virtuales, formas de compilación y generadores de compiladores existentes en el momento.

Como parte fundamental de esta investigación, se manifiesta la necesidad de crear un sistema que agilice el proceso de generación de intérpretes en el proceso de detección y validación de errores. La necesidad es evidenciada mediante la aplicación de una encuesta al equipo de desarrolladores del proyecto Laboratorios Virtuales.

Como parte del marco teórico de la investigación se realiza un análisis de los métodos, técnicas y herramientas que se utilizan para el modelado y diseño del sistema, así como las tecnologías que se emplean en el desarrollo de la aplicación.

1.1 Laboratorios virtuales.

Un laboratorio virtual es un espacio electrónico que brinda la posibilidad de desarrollar en él numerosos experimentos alrededor de un área del conocimiento. Algunos de estos experimentos no sería posible llevarlos a cabo en un laboratorio real, ya que presentan un riesgo para el ser humano. Ejemplo de esto sería: un laboratorio virtual para experimentos radiactivos.

La definición de un laboratorio virtual (…) se refiere a la infraestructura, la metodología y las herramientas especiales distribuidas en diferentes partes que permitan a estudiantes e investigadores trabajar en proyectos e investigaciones que les sean comunes (1).

Es posible resumirlos, en una definición propia, como laboratorios modernos que se sustenta en el uso de las nuevas tecnologías de la información: computadoras, uso de Internet e intranet, etc. e incluyen nuevas funcionalidades y herramientas para lograr un

desarrollo de la capacidad cognoscitiva del estudiante de manera flexible.

Todo ello lo hace ser diferente de un laboratorio tradicional, aunque no implica que lo sustituya, sino que se cuenta con una variante de laboratorio más moderno, en línea directa con el mundo a través de la red y a un coste asequible.

1.1.1 Tipos de laboratorios virtuales.

Los Laboratorios Virtuales están clasificados en tres tipos principalmente: de software, para Web y remotos.

- **Laboratorios virtuales software:**

Son aquellos que son desarrollados como un programa de software independiente destinado a ejecutarse en la máquina del usuario, y cuyo servicio no requiere de un servidor web. Es el caso de programas con instalación propia, que pueden estar destinados a plataformas Unix, GNU/Linux, M.S. Windows, etc. e incluso necesitar que otros componentes de software estén instalados previamente, pero que no necesitan los recursos de un servidor determinado (como bases de datos o propuesta de portales de laboratorios virtuales y remotos). Este tipo de laboratorios fueron pensados inicialmente como aplicaciones java accesibles a través de un servidor web, se pueden considerar de este tipo si funcionan localmente y no necesitan recursos de un servidor en concreto.

- **Laboratorios virtuales web:**

Estos se basan en un software que depende de los recursos de un servidor determinado. Esos recursos pueden ser bases de datos, software que requiere ejecutarse en su servidor la exigencia de determinado hardware para ejecutarse, etc. Estos no son programas que un usuario pueda descargar en su equipo para ejecutar de forma independiente.

- **Laboratorios remotos:**

Los laboratorios remotos permiten operar como la palabra indica, remotamente con cierto equipamiento, bien sea, didáctico como maquetas específicas, o industrial, además de poder ofrecer capacidades de laboratorio virtual. En general, estos

laboratorios requieren de equipos servidores específicos que les den acceso a las máquinas a operar de forma remota, y no pueden ofrecer su funcionalidad ejecutándose de forma local.

Otro motivo que hace dependientes estos laboratorios de sus servidores es la habitual gestión de usuarios en el servidor. Estos se basan en instrumentos reales (tarjetas de adquisición de datos, instrumentos de medida, conexiones en interfaces diversas, comunicación de datos).Tienen como ventaja proporcionar un alto nivel de interactividad, en el que el alumno entra en contacto con equipamiento real, en vez de utilizar un equipamiento simulado. Son muy utilizados en la Educación, por ello, es de primer orden tener en cuenta el diseño con que sean implementados.

Las prácticas del proyecto Laboratorios Virtuales son desarrolladas sobre la base de un laboratorio virtual de software donde los usuarios pueden cometer errores propios de las temáticas tratadas en estos. Debido a que las posibles respuestas del usuario dentro del LV presentan “sintaxis, reconocimiento de términos específicos y además un orden semántico”, el método empleado para la validación en el proyecto es mediante el uso de intérpretes.

1.2 Intérpretes.

La comunicación entre los humanos y los ordenadores sería imposible sin la inclusión entre ambos de un traductor[4] que permita la misma. Este toma como entrada un texto escrito en un lenguaje llamado fuente y da como salida otro texto en un lenguaje, denominado objeto (Ilustración 1).

[4]Traducen los programas en código fuente, escritos en lenguajes de alto nivel, a programas escritos en lenguaje máquina.

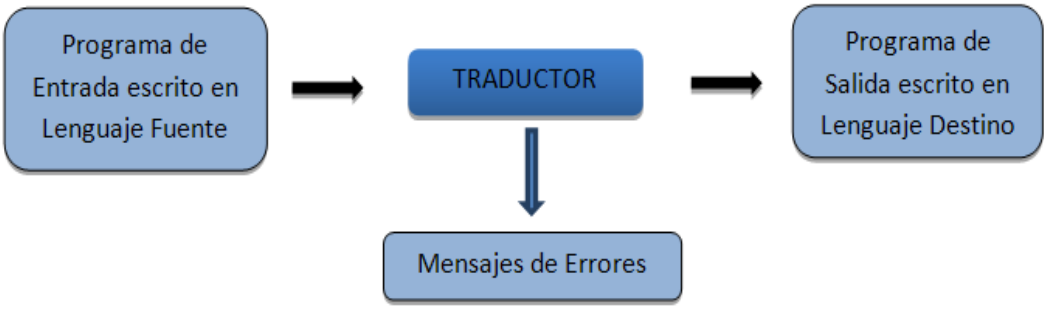


Ilustración 1: Función de un traductor.

Entre los traductores podemos encontrar a los intérpretes, estos no generan un programa equivalente, sino que toman una sentencia del programa fuente en un lenguaje de alto nivel y lo traducen al código equivalente y al mismo tiempo lo ejecutan.

El funcionamiento de un intérprete se caracteriza por traducir y ejecutar, de una en una, las instrucciones del código fuente de un programa, pero, sin generar como salida código objeto (Ilustración2). El proceso que realiza un intérprete es el siguiente: lee la primera instrucción del código fuente, la traduce a código objeto y la ejecuta; a continuación, hace lo mismo con la segunda instrucción; y así sucesivamente, hasta llegar a la última instrucción del programa, siempre y cuando, no se produzca ningún error que detenga el proceso (2).

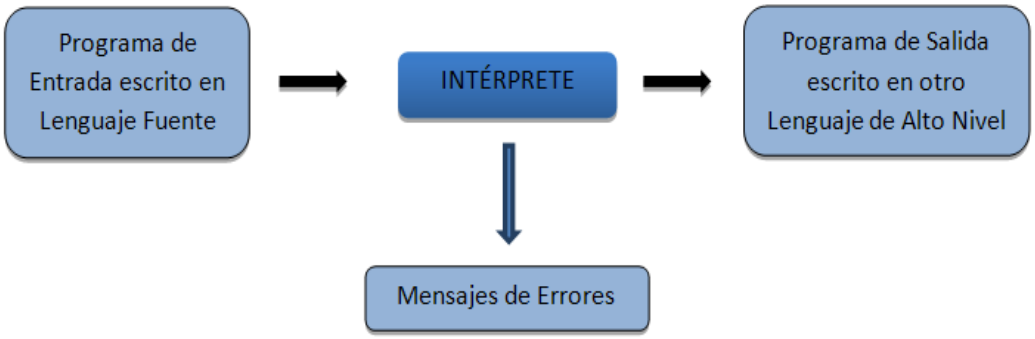


Ilustración 2: Función de un intérprete.

Un compilador[5] es un traductor que convierte un texto escrito en un lenguaje fuente y lo traduce a un programa objeto en código máquina. Tiene como entrada una sentencia en lenguaje formal y como salida un fichero ejecutable, es decir, realiza una traducción de un código de alto nivel a código máquina (3).

La diferencia que existe entre los intérpretes y los compiladores radica en que los intérpretes sólo realizan la traducción a medida que sea necesaria, instrucción por instrucción y normalmente no guardan el resultado de dicha traducción, mientras los compiladores traducen un programa desde su descripción en un lenguaje de programación al código máquina del sistema (4).

La depuración de los programas suele ser más fácil en los intérpretes que en los compiladores puesto que el código fuente está presente durante la ejecución. Estas ventajas pueden incorporarse al compilador mediante la utilización de entornos de desarrollo y depuradores simbólicos en tiempo de ejecución.

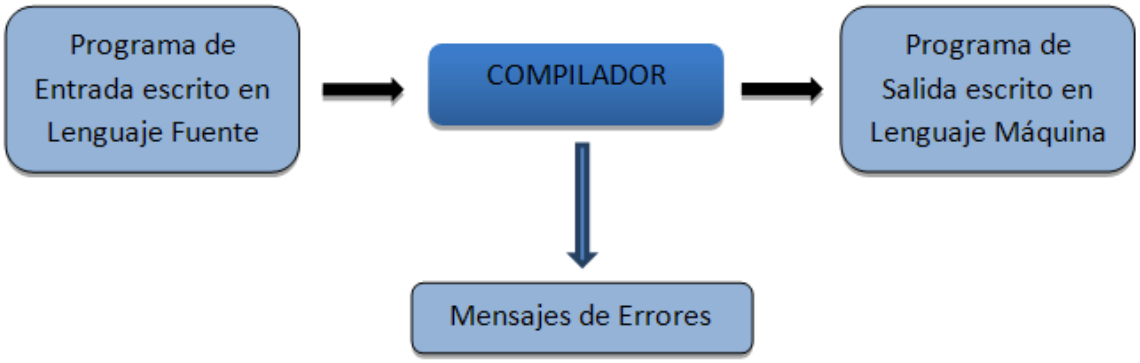


Ilustración 3: Función de un compilador.

El trabajo de un compilador consiste en tomar la cadena fuente del programa,

[5]También se entiende por compilador aquel programa que proporciona un fichero objeto en lugar del ejecutable final.

determinar si es válida desde el punto de vista sintáctico y, a la vez, generar un programa equivalente en un lenguaje que la computadora entienda.

1.2.1 Ventajas de la utilización de intérpretes.

Los intérpretes, por definición, realizan la fase de análisis y ejecución a la vez, los nuevos avances informáticos aumentan la velocidad de procesamiento y capacidad de memoria de los ordenadores. Actualmente, la eficiencia es un problema menos grave y muchas veces se prefieren sistemas que permitan un desarrollo rápido de aplicaciones que cumplan fielmente la tarea encomendada (5).

Entre las ventajas de los sistemas interpretados se encuentran:

- **Sencillos de implementar:** esta característica facilita el estudio de la corrección del intérprete y proporciona nuevas líneas de investigación como la generación automática de intérpretes, a partir de las especificaciones semánticas del lenguaje.
- **Mayor flexibilidad:** los sistemas interpretados permiten modificar y ampliar características del lenguaje fuente. Muchos lenguajes como Lisp[6], APL[7], Prolog[8], etc. surgieron en primer lugar como sistemas interpretados y posteriormente surgieron compiladores.
- **No precisa mantener en memoria todo el código fuente:** este elemento permite su utilización en sistemas de poca memoria o en entornos de red, en los que se puede obtener el código fuente a medida que se necesita.
- **Facilitan la meta-programación:** un programa puede manipular su propio código fuente a medida que se ejecuta. Esto facilita la implementación de sistemas de aprendizaje automatizado.

[6] Familia de lenguaje de programación de computadora de tipo multiparadigma con una sintaxis completamente entre paréntesis.

[7] A Programing Language, lenguaje de programación interpretado orientado a trabajo con matrices.

[8] Lenguaje de programación lógico e interpretado, bastante conocido en el medio de investigación en Inteligencia Artificial.

- **Aumentan la portabilidad del lenguaje:** para que el lenguaje interpretado funcione en otra máquina sólo es necesario que su intérprete funcione en dicha máquina.
- **No existen etapas intermedias de compilación**: los sistemas interpretados facilitan el desarrollo rápido de prototipos, potencian la utilización de sistemas interactivos y facilitan las tareas de depuración.

Sin embargo, los programas interpretados suelen ser más lentos que los compilados debido a la necesidad de traducir el programa mientras se ejecuta. Esta desventaja se equilibra con la flexibilidad, planteada anteriormente, que poseen como entornos de programación y depuración ofreciendo al programa interpretado un entorno no dependiente de la máquina donde se ejecuta el intérprete, sino del propio intérprete, lo que se conoce comúnmente como máquina virtual. Esta característica puede traducirse, por ejemplo, en una mayor facilidad para remplazar partes enteras del programa o añadir módulos completamente nuevos (5).

1.3 Fases del proceso de compilación.

Un proceso de compilación es el encargado de traducir las instrucciones escritas en un determinado lenguaje de programación a lenguaje máquina. Para crear un programa objeto ejecutable, se pueden necesitar otros programas además de un traductor. Un programa fuente se puede dividir en módulos almacenados en archivos distintos. La tarea de reunir el programa fuente a menudo se confía a un programa distinto, llamado preprocesador. El preprocesador también puede expandir abreviaturas, llamadas a macros, a proposiciones del lenguaje fuente.

Un compilador típicamente opera en fases, cada una lleva a cabo una tarea sobre el programa fuente.

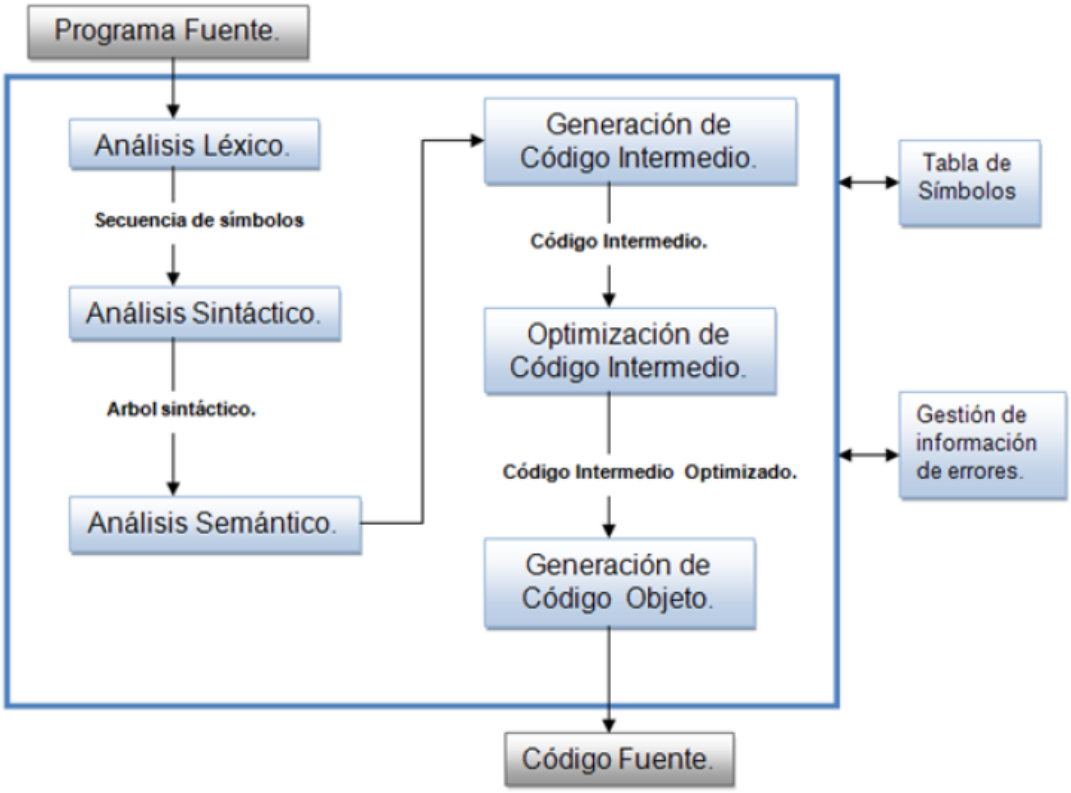


Ilustración 4: Fases del proceso de compilación.

A partir del programa fuente el analizador léxico genera una serie de componentes léxicos (*tokens* en inglés) que son procesados por el analizador sintáctico para construir la estructura de datos (normalmente un árbol jerárquico) que en una etapa más profunda utiliza el analizador semántico. El código intermedio es un código abstracto independiente y que es optimizado para la obtención de código objeto. Este conjunto de operaciones conllevan a la generación del código fuente.

1.3.1 Análisis léxico.

El analizador léxico o scanner lee la secuencia de caracteres del programa fuente, caracter a caracter, y los agrupa para formar unidades con significado propio, estas unidades son los componentes léxicos. Estos componentes léxicos representan (7):

- **Palabras reservadas[9]:** *if*, *while*, *do*, etc.
- **Identificadores:** asociados a variables, nombres de funciones, tipos definidos por el usuario, etiquetas, etc. Por ejemplo: posición, velocidad o tiempo.
- **Operadores[10]:** =, *, +, -, /, ==,>,<,&, !,=.
- **Símbolos especiales:** ; ,( ), [ ], f, g.
- **Constantes numéricas:** estas constantes son literales que representan valores enteros, en coma flotante, etc.
- **Constantes de caracteres:** literales que representan cadenas concretas de caracteres.

La relación de estos componentes léxicos con los estados no finales del sistema, integran en sí el autómata del sistema. Un autómata finito (AF) o máquina de estado finito es un modelo computacional que realiza cómputos en forma automática sobre una entrada para producir una salida.

Este modelo está conformado por un alfabeto, un conjunto de estados y un conjunto de transiciones entre dichos estados. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

[9] Palabra que tiene un significado gramatical especial para ese lenguaje y no puede ser utilizada como un identificador.

[10] Símbolo matemático que indica que debe ser llevada a cabo una operación especificada sobre un cierto número de operandos.

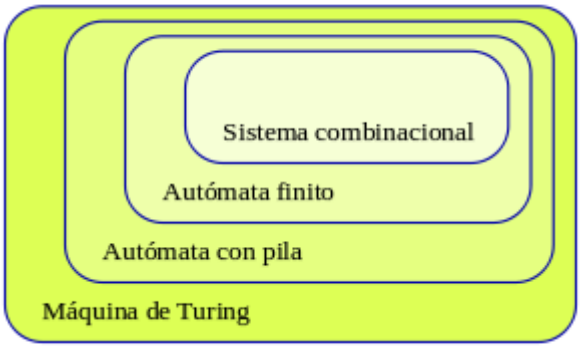


Ilustración 5: Teoría de autómatas.

En el comienzo del proceso de reconocimiento de una cadena de entrada, el autómata finito se encuentra en el estado inicial y a medida que procesa cada símbolo de la cadena va cambiando de estado de acuerdo a lo determinado por la función de transición. Cuando se ha procesado el último de los símbolos de la cadena de entrada, el autómata se detiene en el estado final del proceso. Si el estado final en el que se detuvo es un estado de aceptación, entonces la cadena pertenece al lenguaje

El analizador léxico opera bajo petición del analizador sintáctico devolviendo un componente léxico conforme el analizador sintáctico lo va necesitando para avanzar en la gramática (Ilustración6). Los componentes léxicos son los símbolos terminales de la gramática. Suele implementarse como una subrutina del analizador sintáctico. Cuando recibe la petición del siguiente componente léxico, el analizador léxico lee los caracteres de entrada hasta identificar el siguiente componente léxico (7).

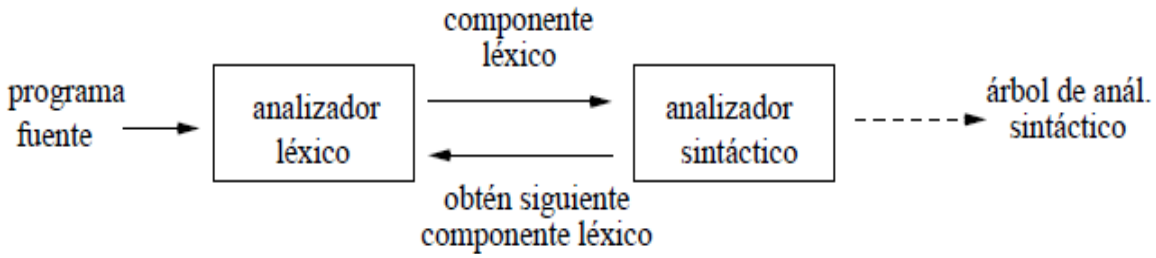


Ilustración 6: Función principal del analizador léxico.

La función principal del analizador léxico es generar una lista ordenada de *tokens*, a partir de los caracteres de entrada. Esos *tokens* son usados por el analizador sintáctico para construir el árbol sintáctico. El analizador léxico es un módulo subordinado al correspondiente del analizador sintáctico.

Además de la anterior que constituye la función principal, el analizador léxico es responsable de otras funciones adicionales (8):

Funciones adicionales	**Ejemplos**
Asistencia en el informe de errores elaborado por el analizador sintáctico.	Cuenta de números de línea con comentarios, macros.
Manejo de algunos errores.	En PASCAL el lexema 0.5 pertenecería al lenguaje y .5 no. Exceder el número de caracteres máximo para un identificador.

Tabla 2: Funciones adicionales del analizador léxico.

1.3.2 Análisis sintáctico.

El análisis sintáctico es la fase del analizador que se encarga de chequear el texto de entrada en base a una gramática dada. Y en caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce. En teoría, se supone que la salida del analizador sintáctico es alguna representación del árbol sintáctico que reconoce la secuencia de *tokens* suministrada por el analizador léxico. En la práctica, el analizador sintáctico también tiene la responsabilidad de (9):

- Acceder a la tabla de símbolos.
- Chequeo de tipos.
- Generar código intermedio.

- Notificar errores cuando estos se producen.

El análisis sintáctico transforma el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada. Un analizador léxico crea *tokens* de una secuencia de caracteres de entrada y son estos *tokens* los que son procesados por el analizador sintáctico para construir la estructura de datos, por ejemplo un árbol de análisis o árboles de sintaxis abstracta.

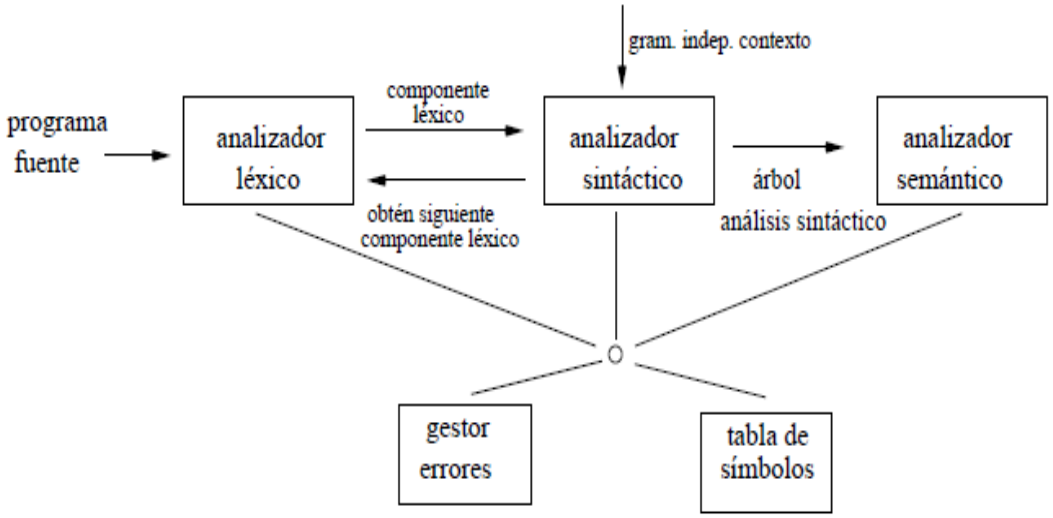


Ilustración 7: Estructura funcional del analizador sintáctico.

Las gramáticas ofrecen ventajas significativas a los diseñadores de lenguajes y a los desarrolladores de compiladores. Entre las ventajas más importantes, citadas por Aguilera y Gálvez en sus escritos sobre el analizador sintáctico, de las gramáticas podemos encontrar las siguientes (9):

- Fácil generar código y detectar errores.
- Las gramáticas son especificaciones sintácticas y precisas de lenguajes de programación.
- A partir de una gramática se puede generar automáticamente un analizador sintáctico.

- El proceso de construcción puede llevar a descubrir ambigüedades.
- Una gramática proporciona una estructura a un lenguaje de programación.
- Es más fácil ampliar/modificar el lenguaje si está descrito con una gramática.

1.3.3 Análisis semántico.

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

Al comprobar la validez semántica de un programa se pudiera decir que se realiza un reconocimiento sintáctico-semántico, pero los errores semánticos no pueden ser detectados por el analizador sintáctico, puesto que se relacionan con interdependencias entre las diferentes partes de un programa que no son reflejadas en un análisis gramatical.

La semántica corresponde al significado asociado a las estructuras formales (sintaxis) del lenguaje. Así, se denomina tradicionalmente “análisis semántico” a todo aquello que forma parte del *front-end*, más allá de lo que la gramática utilizada nos permite (10):

- Tabla de símbolos
- Chequeos de tipos (y otros)
- Generación de representaciones internas

La tabla de símbolos va conteniendo un registro por cada identificador definido por el programador (o predefinidos) añadiéndose información asociada (10):

- Nombre del identificador (diferenciando mayúsculas y minúsculas)
- Categoría (variable, constante, tipo, campo, procedimiento, función, parámetro, clase, etiqueta, módulo, macro, etc.).
- ¿A qué ámbito pertenece? (número).
- ¿Cuál es su tipo? (enlace).

- Otra información (tamaño, ubicación, valor, parámetros o campos enlaces, referencia adelante o no, ámbito que define, etc.).

1.3.4 Gestión de errores.

Si los compiladores o intérpretes tuvieran que procesar solamente programas correctos, su diseño e implementación se simplificaría en buena medida. Pero los programadores escriben programas incorrectos frecuentemente, y un buen intérprete o compilador debe ayudar al programador a localizar e identificar los errores. Los errores en un programa pueden clasificarse en 4 grandes grupos (11):

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

Los **errores léxicos** se detectan cuando el analizador léxico intenta reconocer componentes léxicos y la cadena de caracteres de la entrada no encaja con ningún patrón. Son situaciones en las que usa un carácter invalido (@, $,",>,...), que no pertenece al vocabulario del lenguaje de programación, al escribir mal un identificador, palabra reservada u operador (12).

Errores léxicos típicos son:

1. **Nombre ilegales de identificadores:** un nombre contiene caracteres inválidos.
2. **Números incorrectos:** un número contiene caracteres inválidos o no está formado correctamente, por ejemplo 3,14 en vez de 3.14 o 0.3.14.
3. **Errores de ortografía en palabras reservadas:** caracteres omitidos, adicionales o cambiados de sitio, por ejemplo la palabra *while* en vez de *hwile*.
4. **Fin de archivo:** se detecta un fin de archivo a la mitad de un componente léxico.

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

que se ha seguido no es correcta.

Los **errores semánticos** se producen durante el proceso de análisis semántico, detectándose al comprobar que cada operando involucrado en las operaciones sea de un tipo permitido por la misma o según las reglas semánticas que deba cumplir el lenguaje fuente.

Los **errores lógicos** están relacionados con el cumplimiento de condiciones en un ambiente dado, o con redundancia en el código. Ejemplo de ellos pudieran ser que se hace referencia a una variable no declarada aún. Un intérprete o compilador que se detenga ante el primer error que se encuentre no es muy eficaz. El tratamiento de los errores en cualquiera de las fases debe cumplir con los siguientes requisitos:

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

1.3.5 Generación de código intermedio.

La mayoría de los compiladores generan código como parte del proceso de análisis sintáctico, esto es debido a que requieren del árbol de sintaxis y si este no va a ser construido físicamente, entonces deberá acompañar al analizador sintáctico al barrer el árbol implícito. En lugar de generar código ensamblador directamente, los compiladores generan un código intermedio que es más parecido al código ensamblador, las operaciones por ejemplo nunca se hacen con más de dos operandos.

Al no generarse código ensamblador el cual es dependiente de la computadora específica, sino código intermedio, se puede reutilizar la parte del compilador que genera código intermedio en otro compilador para una computadora con diferente procesador cambiando solamente el generador de código ensamblador al cual llamamos *back-end*, la desventaja obviamente es la lentitud que esto conlleva (13).

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.

El código intermedio es un código abstracto independiente de la máquina para la que se generará el código objeto. El código intermedio ha de cumplir dos requisitos importantes: ser fácil de producir a partir del análisis sintáctico, y ser fácil de traducir al lenguaje objeto. Esta fase puede no existir si se genera directamente código máquina, pero suele ser conveniente emplearla.

1.3.6 Optimización de código intermedio.

La fase de optimización de código consiste en mejorar el código intermedio, de modo que resulte un código máquina más rápido de ejecutar. Esta fase de la etapa de síntesis es posible sobre todo si el traductor es un compilador (difícilmente un intérprete puede optimizar el código objeto). Hay mucha variación en la cantidad de optimización de código que ejecutan los distintos compiladores.

La optimización de código intermedio se puede realizar (6):

- **A nivel local:** sólo utilizan la información de un bloque básico para realizar la optimización.
- **A nivel global:** que usan información de varios bloques básicos.

El término optimización de código es inadecuado ya que no se garantiza el obtener, en el sentido matemático, el mejor código posible atendiendo a maximizar o minimizar una función objetivo (tiempo de ejecución y espacio). El término de mejora de código sería más apropiado que el de optimización (6).

Además de la optimización a nivel de código intermedio, se puede reducir el tiempo de ejecución de un programa actuando a otros niveles: a nivel de código fuente y a nivel de

código objeto (6).

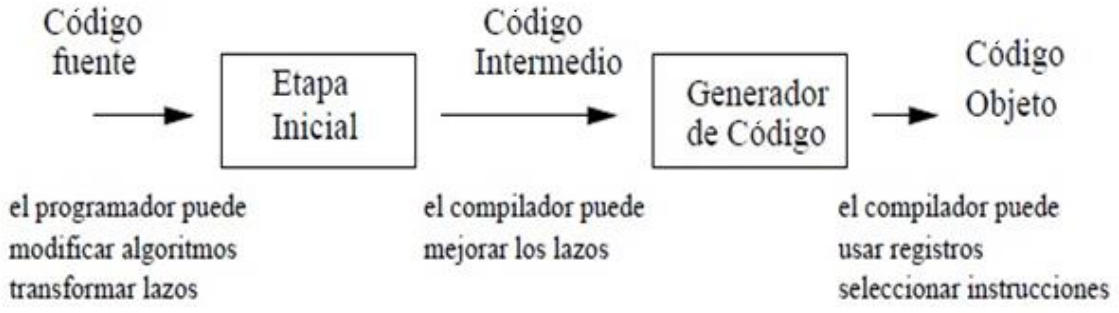


Ilustración 8: Niveles para la optimización del código.

La fase de generación de código objeto se encarga de la obtención del programa nativo usando el juego de instrucciones específico de la máquina o CPU objeto y el formato para archivos ejecutables del sistema operativo. Entre otras cosas, también se le asignan direcciones definitivas a las rutinas y variables que componen el programa (13).

1.4 Herramientas generadoras de intérpretes.

En la actualidad no es posible pensar en la realización computacional de un procesador de lenguajes, como por ejemplo, compiladores, intérpretes, etc., sin utilizar una herramienta instrumental, tal como un generador de intérpretes. Este tipo de herramientas existe desde hace años, y han ido evolucionando junto con la propia evolución de la ciencia de la computación, encontrándonos en la actualidad muchos y diferentes tipos de generadores de compiladores, que en general se corresponden con los entornos de desarrollo de aplicaciones y con los paradigmas de la programación (15).

Un generador de analizadores es un programa que acepta como entrada la especificación de las características de un lenguaje L y produce como salida un

analizador para L. La especificación de entrada pude referirse a la lexicografía, la sintaxis o la semántica; el analizador resultante servirá para analizar las características especificadas.

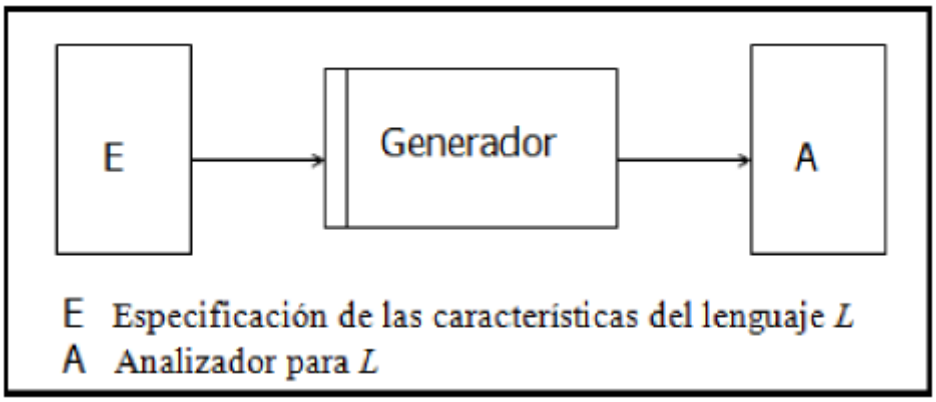


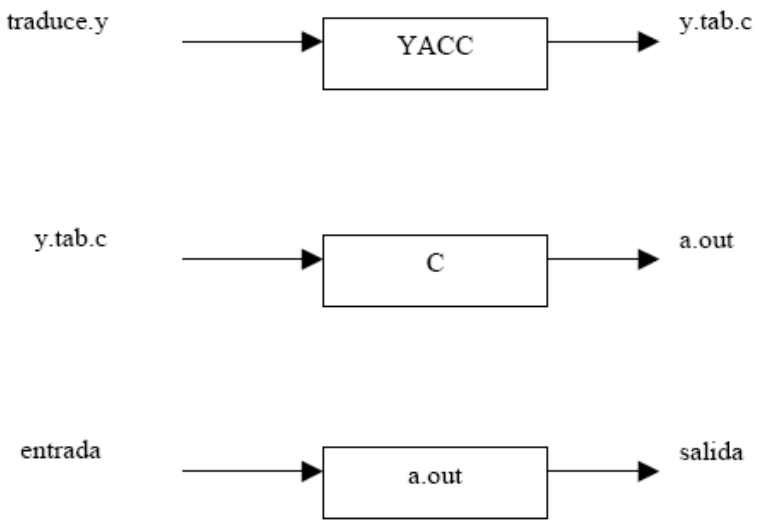
Ilustración 9: Esquema de un generador de analizadores.

Para crear un generador de código se deben hacer muchas de las tareas que realizan los compiladores; algunas de estas tareas son: la búsqueda de patrones, la escritura de código, el análisis sintáctico, el análisis léxico y la optimización de código.
A continuación aparecen algunas de las herramientas más utilizadas:

Yacc: es una herramienta para generar analizadores sintácticos. Las siglas del nombre significan *Yet Another Compiler Compiler*, es decir, “Otro generador de compiladores más” (16).

Puesto que el analizador sintáctico generado por Yacc requiere un analizador léxico, se utiliza a menudo conjuntamente con un generador de analizador léxico, en la mayoría de los casos Lex o Flex, alternativa del software libre (16).

Sin embargo un analizador sintáctico de Yacc no puede funcionar por sí solo, sino que necesita un analizador léxico externo para funcionar.



.

Ilustración 10: Construcción de un traductor usando Yacc.

Lex: es un generador de analizadores léxicos, que permite crear programas autónomos que analizan léxicamente la entrada y la procesan según las necesidades del usuario que trabaja con ella. Lex genera código fuente en C, a partir de una serie de especificaciones escritas en lenguaje Lex. El código C generado contiene una función llamada yylex(), que localiza cadenas en la entrada (lexemas) que se ajusten a uno de los patrones léxicos especificados en el código fuente Lex, realizando las acciones asociadas a dicho patrón. Ylex() puede llevar a cabo cualquier tipo de acciones ante un determinado patrón y, en particular, puede comportarse como un analizador léxico (16).

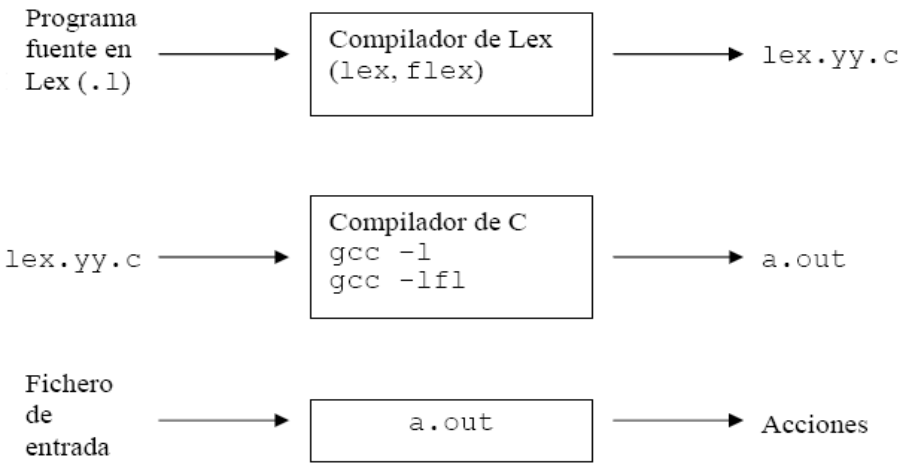


Ilustración11: Construcción de un traductor utilizando Lex.

Lex y Yacc son herramientas que propician la generación de analizadores eficientes, incluso más de lo que se pudieran hacer de manera manual, siendo estos, analizadores ascendentes que reconocen la mayor parte de los lenguajes.

Por otra parte, estas herramientas incrementan la dificultad del trabajo de depuración, puesto que los errores que se cometen en el fichero de especificación son solo visibles en el analizador generado. Mezclan las especificaciones sintácticas con las semánticas. En el caso de Yacc necesita usar herramientas externas para que le provean los *tokens* necesarios. Estas características son la base para no recomendar su utilización en el proceso de generación de intérpretes para el proyecto de Laboratorios Virtuales.

JavaCC (*Java Compiler Compiler*): es una herramienta para generar programas escritos en lenguaje Java; acepta como entrada una especificación de un determinado lenguaje y produce como salida un analizador sintáctico para ese lenguaje. En la manera más simple de funcionamiento, la especificación proporcionada define las características sintácticas y lexicográficas de un lenguaje y se genera un analizador léxico-sintáctico del lenguaje especificado; pero también es posible completar una especificación léxico-sintáctica con la inclusión adecuada de código para que el programa generado llegue a

ser un analizador completo del lenguaje (17).

El analizador sintáctico obtenido es, en general, LL (k): descendente y determinista con la consulta de k símbolos por adelantados; si la gramática proporcionada cumple con la condición LL (1); se genera un analizador sintáctico descendente-predictivo-recursivo (17).

JavaCC es una herramienta de nueva generación que permite generar analizadores léxicos y sintácticos con una buena integración. Incluye además la herramienta JJTREE para generar árboles sintácticos. Son muy buenas características para un generador de analizadores, pero la complejidad de los ficheros de especificación de la misma la convierte en una herramienta no idónea en el proceso de generación de intérpretes para el proyecto de Laboratorios Virtuales, además de generar los analizadores en lenguaje java, el cual no es utilizado por el proyecto.

1.5 Metodologías de desarrollo de Software.

Una metodología de desarrollo de software es el conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software (18).

Las metodologías se clasifican en robustas y ágiles. Las metodologías robustas son aquellas en las que se hace énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este sistema es efectivo en proyectos de gran tamaño (respecto a tiempo y recursos). Ejemplo de ellas son:

- RUP: *RationalUnifiedProcess* (Proceso Unificado de Desarrollo).
- MSF: *Microsoft Solutions Framework* (Marco de Soluciones Microsoft).

Las metodologías ágiles están orientadas para proyectos pequeños donde es más importante construir un buen equipo que construir el entorno. Ejemplo:

- XP: *Extreme Programming* (Programación Extrema).
- *Family of CrystalMethodologies* (Familia de Metodologías Crystal).

1.5.1 Programación extrema (XP).

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico (19). Se muestran a continuación algunas de sus características:

- El juego de la planificación: hay una comunicación frecuente el cliente y los programadores.
- Entregas pequeñas: producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema, tardando estas no más de 3 meses.
- Diseño simple: se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- Pruebas: la producción de código está dirigida por las pruebas unitarias.

1.5.2 Proceso unificado de desarrollo (RUP).

RUP organiza los proyectos en términos de flujos de trabajo y fases, las cuales consisten de una o más iteraciones. En cada iteración el énfasis en cada flujo de trabajo variará a lo largo del flujo de vida.
Se apoya en tres principios básicos (20):

- Dirigido por casos de uso: Los casos de uso dirigen y controlan el proceso de desarrollo en su totalidad.
- Centrado en la arquitectura: Es la pieza clave que permite comprender el sistema, organizar el desarrollo y hacer evolucionar el software.

- Proceso iterativo e incremental: El desarrollo se plantea de manera progresiva, de tal modo que se atenúen los riesgos y se planteen las cuestiones en el instante en el que estamos capacitados para resolverlas.

1.6 Herramientas CASE.

Se puede definir a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. Como es sabido, los estados en el ciclo de vida de desarrollo de un software son: Investigación preliminar, análisis, diseño, implementación e instalación (21). Las propuestas principales dentro del proyecto es el uso de Visual Paradigm y Rational Rose. A continuación se exponen algunas de las características principales de estas herramientas.

Visual Paradigm.

Entre sus características están:

- Software libre.
- Disponibilidad en múltiples plataformas (Windows, Linux).
- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo

Rational Rose Enterprise.

Entre las características principales de Rational se pueden destacar:

- Admite como notaciones: UML, OMT y Booch.

- Permite desarrollo multiusuario.
- Genera documentación del sistema.
- Disponible en múltiples plataformas.

1.7 Lenguaje Unificado de Modelado UML.

UML ofrece soporte para clases, clases abstractas, relaciones, comportamiento por iteración, empaquetamiento, entre otros. Estos elementos se pueden representar mediante nueve tipos de diagramas, que son: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de desarrollo (22).

UML ("*Unified Modeling Language*") está consolidado como el lenguaje estándar en el análisis y diseño de sistemas de cómputo. Mediante UML es posible establecer la serie de requerimientos y estructuras necesarias para plasmar un sistema de software previo al proceso intensivo de escribir código.

Este lenguaje está compuesto por tres elementos fundamentales, los elementos, los diagramas y las relaciones (23):

- Elementos: abstracciones que constituyen los bloques básicos de construcción.
- Diagramas: es la representación de un conjunto de elementos: visualizan un sistema desde diferentes perspectivas.
- Relaciones: permiten ligar los elementos.

1.8 Lenguajes de programación.

Un lenguaje de programación es un sistema de instrucciones lógicas que pueden ser utilizadas para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente.

Entre los lenguajes de programación se encuentran:

C#: es un lenguaje de programación que toma las mejores características de lenguajes prexistentes como Visual Basic, Java o C++ y las combina en uno solo. Su compilador

es el más depurado y optimizado de los incluidos en el *.NET Framework SDK* (24).

C++: es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.
Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales.
C++ es un lenguaje orientado a objeto, adopta una visión muy cercana al lenguaje máquina. En un principio se destinó a escribir sistemas operativos, pero sus características le abrieron perspectivas nuevas.
El lenguaje está formado por instrucciones muy explícitas, cortas, cuya duración de ejecución puede preverse con antelación en el momento de escribir el programa.

1.7 Entorno de desarrollo integrado

Un entorno de desarrollo integrado, llamado también IDE (sigla en inglés de *Integrated Development Environment*), es un programa informático compuesto por un conjunto de herramientas de programación. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario (GUI). Entre ellos se encuentran:

Microsoft Visual Studio.NET: Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones web ASP.NET, servicios web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C++, Visual C# y Visual J# utilizan el mismo entorno de desarrollo integrado (IDE), que les permite compartir herramientas y facilita la creación de soluciones en varios lenguajes. Estos lenguajes aprovechan las funciones de *.NET Framework*, que ofrece acceso a

tecnologías clave para simplificar el desarrollo de aplicaciones web ASP y servicios web XML (25).

QT Creator: Es un entorno de desarrollo (IDE) multiplataforma muy completo, mediante las librerías QT (*Quasar Technologies*).
Principales características de QtCreator (26)**:**

- Posee un avanzado editor de código C++.
- Además soporta los lenguajes: C#/.NET Languages (Mono), Python: PyQt y PySide, Ada, Pascal, Perl, PHP y Ruby.
- Posee también una GUI integrada y diseñador de formularios.
- Herramienta para proyectos y administración.
- Ayuda sensible al contexto, integrada.
- Depurador visual.
- Resaltado y auto-completado de código.
- Soporte para refactorización de código.

1.9 Análisis de los principales problemas en la generación de intérpretes.

Como parte de la presente investigación, se aplicó una encuesta, (Anexo1: Plantilla de encuesta realizada a los desarrolladores que integran el proyecto Laboratorios Virtuales.), a un grupo de 25 personas del equipo de trabajo del proyecto (muestra representativa), con el fin de conocer las características, efectos y consecuencias del proceso actual de generación de intérpretes (Tabla 3).

Resultados de la encuesta.

RESULTADOS DE LA ENCUESTA		
	No.	**%**

Insatisfechos con el proceso actual	23	92
Principales dificultades		
a. Pérdida de tiempo	21	84
b. Introducción de errores	13	52
c. Incremento del tiempo de máquina	20	80
d. Baja productividad del equipo de trabajo	19	76
e. Disminución de la moral del equipo de trabajo	17	68
Consideran necesario desarrollar una herramienta para la generación de intérpretes.	23	92

Tabla 3: Resultados de la encuesta.

El análisis de los resultados obtenidos por la encuesta aplicada a los desarrolladores del proyecto Laboratorios Virtuales, refleja el impacto negativo, efecto del engorroso proceso de generación de intérpretes actual. De estos resultados se puede concluir, que esta práctica causa pérdida de tiempo, que se traduce en retrasos innecesarios en el cumplimiento de los hitos y metas programadas, para la liberación de productos y funcionalidades importantes disminuyendo la eficiencia del trabajo. Se impone como criterio general (92 %), la necesidad de desarrollar un sistema para la generación de intérpretes que agilice la utilización de recursos y reduzca considerablemente el tiempo de desarrollo.

Conclusiones parciales.

El análisis realizado se orientó fundamentalmente al estudio de los principales elementos que permiten la implementación de un generador de intérpretes, el conocimiento de sus clasificaciones, arquitectura y componentes. Se realizó un estudio de las principales herramientas generadoras de analizadores, donde se evidencian sus negativas con relación a las necesidades del proyecto de Laboratorios Virtuales, por lo

que surge la necesidad de desarrollar un sistema que agilice la generación de intérpretes en el proyecto de Laboratorios Virtuales.

Un generador de intérpretes debe ser capaz de brindar los procedimientos básicos que integran el proceso de compilación. A partir del programa fuente el analizador léxico genera una serie de componentes léxicos que son procesados por el analizador sintáctico para construir la estructura de datos (normalmente un árbol jerárquico) que en una etapa más profunda utiliza el analizador semántico. Luego son llevadas a cabo un grupo de operaciones que confluyen en la generación del código fuente.

Se hace alusión a metodologías de desarrollo de software, herramientas y entornos, dentro de los cuales están los seleccionados para la elaboración de dicho trabajo de diploma. Con el objetivo de identificar los principales problemas relacionados con la generación actual de intérpretes en el proceso de detección y validación de errores, que afectan el rendimiento de la producción, se aplicó un cuestionario a una muestra representativa de los desarrolladores del proyecto. Esta encuesta reflejó la necesidad de desarrollar un sistema que agilizara el proceso estudiado.

Capítulo 2: Fundamentos de la solución propuesta.

Introducción

En este capítulo se establece una caracterización de la metodología seleccionada para el diseño e implementación del módulo. Se realiza una descripción de la plantilla de casos de prueba usada, y del diseño de la misma. Se establece una descripción detallada de las funcionalidades requeridas, además de definir los artefactos generados en el proceso de diseño e implementación del módulo.

2.1 Propuesta de solución.

Esta tesis tiene como propósito general, lograr el desarrollo de un sistema que permita la agilización de la generación de intérpretes en el proceso de detección y validación de errores para el proyecto Laboratorios Virtuales. Tras el estudio realizado de las herramientas existentes para la generación y modificación de intérpretes, se llegó a la conclusión de que estas no son capaces de integrarse con las aplicaciones desarrolladas dentro del proyecto, puesto que algunas no generan el intérprete con el lenguaje utilizado, tal es el caso del Yacc que genera un intérprete en lenguaje C, y depende además de un analizador léxico externo. El sistema propuesto en este trabajo de diploma, tiene como salida un analizador sintáctico escrito en lenguaje C++ y no necesita ninguna herramienta externa para su utilización.

Un generador de intérpretes debe ser capaz de garantizar los procedimientos básicos que integran el proceso de compilación, el sistema propuesto se centra en la generación de las fases de análisis léxico y detección de errores sintácticos. Para lograr dicho objetivo el sistema permite al desarrollador la configuración de un analizador léxico, donde se definen todos los estados y sus transiciones, creando así el autómata general del sistema. Permite además configurar el analizador sintáctico, brindándole al desarrollador la posibilidad de adicionar los no terminales y relacionarlos entre ellos o con *tokens* para crear de esta forma las reglas gramaticales del lenguaje. Como producto final se obtiene un intérprete escrito en lenguaje C++, que realiza la detección

de errores sintácticos a partir de los *tokens* generados por el analizador léxico del sistema. El esbozo de la propuesta del sistema de generación de intérpretes para los laboratorios virtuales se muestra a continuación:

Ilustración 12: Esbozo de la interfaz gráfica de usuario, ventana principal.

Barra de menú: aparecerán las operaciones principales para trabajar con el proyecto, tales como crear un proyecto nuevo, abrir uno ya existente y guardar un proyecto. Además tendrá la opción generar código fuente.

Área de secciones: está compuesta por dos secciones, las cuales son mostradas por la interfaz como dos etiquetas, la primera está dedicada a la entrada de elementos que conforman el analizador léxico, y la segunda a la introducción de la gramática que será reconocida por el analizador sintáctico.

Ilustración 13: (Sección "Analizador Léxico").

La sección Analizador Léxico brindará la posibilidad de introducir los elementos lexicográficos para definir internamente el autómata del sistema, cuyos datos serán mostrados al desarrollador a través de las tablas de Estados, Palabras Reservadas y Transiciones, siendo esta última la encargada de mostrar con que caracter se pasará de un estado a otro.

Cuenta con un área llamada botones de control para efectuar las operaciones de gestión de los elementos introducidos, es decir, contiene una serie de botones que permiten adicionar y eliminar los datos introducidos por el desarrollador.

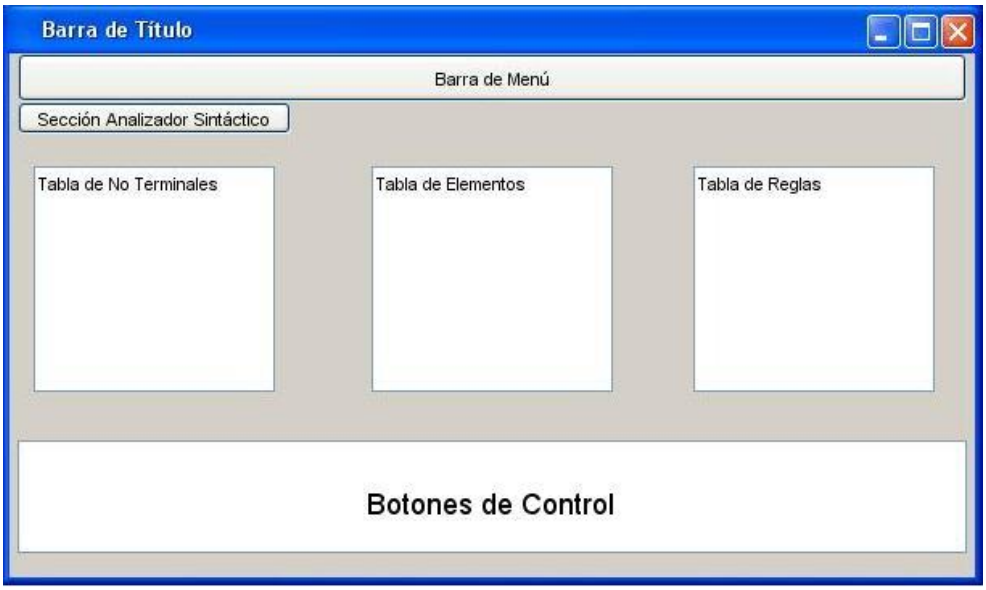


Ilustración 14: Sección "Analizador Sintáctico"

La sección Analizador Sintáctico contará con las tablas No Terminales, Elementos y Reglas, y se encargará de la sintaxis del sistema. En la primera tabla el sistema mostrará los no terminales adicionados por el desarrollador, en la segunda tabla, se mostrarán los elementos (los estados finales y los no terminales), de los cuales depende un no terminal anteriormente seleccionado. En la tercera tabla se mostrará la regla asociada a la dependencia descrita en la segunda tabla.

2.2 Herramientas y tecnologías utilizadas.

A partir de la investigación realizada en el Capítulo 1 y teniendo en cuenta las herramientas y tecnologías utilizadas en el proyecto, se decide la selección de las siguientes herramientas:

Como metodología de desarrollo de software a utilizar se escoge **RUP** ya que es una metodología para el desarrollo de software orientado a objetos. Es una metodología dirigida por casos de uso y centrada en la arquitectura.

La herramienta CASE que se selecciona es el **Visual Paradigm**, ya que es una herramienta libre y multiplataforma que soporta el ciclo de vida completo del desarrollo del software. Además genera código en una amplia gama de lenguajes, entre ellos C++.

El proyecto como lenguaje de programación utiliza **C++**, que es un lenguaje de programación orientado a objeto, lo cual es esencial para incrementar la productividad, calidad y reutilización del software. Ofrece gran riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además es libre y robusto.

Entre los IDE estudiados se escogió **QT Creator**, este es libre y multiplataforma. Posee una extensa biblioteca de clases y herramientas, las cuales se encuentran bien documentadas, lo que es de gran ayuda en el desarrollo del software.

2.3 Modelo del dominio.

Un modelo del dominio captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema; se considera en RUP un subconjunto del llamado modelo de objetos del negocio. Este modelo posee un bajo nivel de estructuración del negocio y está centrado en tecnologías informáticas.

Un modelo del dominio es una representación de las clases conceptuales del mundo real, no de componentes de software. Este tipo de modelo tiene como objetivo principal ayudar a comprender los conceptos con los que deberá trabajar el sistema de Generación de Intérpretes. Estos conceptos se representan en el siguiente diagrama (Ilustración 15):

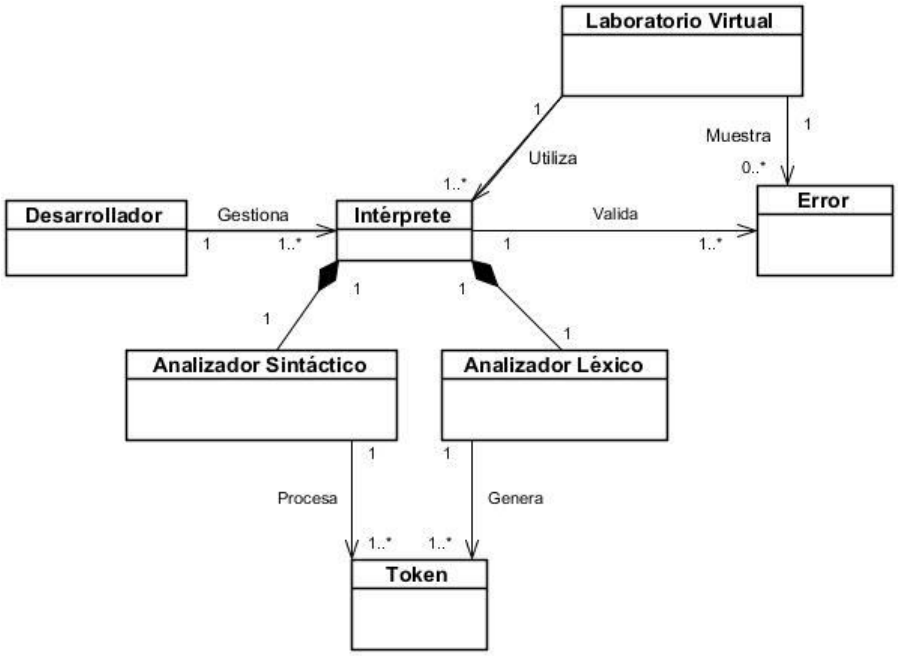


Ilustración 15: Modelo del dominio.

2.3.1 Descripción de las clases conceptuales que conforman el modelo de dominio.

A continuación se describen los conceptos que se han identificado y que se utilizan en el Modelo de Dominio (Ilustración 15):

Desarrollador: actor que utilizará el sistema propuesto para gestionar intérpretes, es el encargado de la creación y modificación de las fases de análisis léxico y sintáctico.

Intérprete: clase conceptual que contiene todos los procesos de interpretación (validación de errores), que van a ocurrir dentro del componente. Está compuesto por los analizadores léxico y sintáctico que serán generados por el sistema desarrollada.

Laboratorio Virtual: clase que utiliza los elementos que componen al intérprete en el

proceso de validación de errores.

***Token*:** componentes léxicos, con significado propio, que son generados por el analizador léxico y procesados por el analizador sintáctico.

Analizador léxico: forma parte del intérprete y es el encargado de la generación de los componentes léxicos.

Analizador sintáctico: forma parte del intérprete y procesa los *tokens* generados por el analizador léxico para garantizar que la sintaxis sea correcta.

Error: se muestra dentro del Laboratorio Virtual y es validado por el intérprete.

2.4 Requerimientos del sistema.

"Un requerimiento es una condición que tiene que ser alcanzada por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente" (27).

Los requisitos con que debe cumplir un sistema se clasifican en funcionales y no funcionales. Los funcionales son capacidades o condiciones que el sistema debe cumplir, mientras que los no funcionales son propiedades o cualidades que hacen al producto atractivo, usable, rápido y confiable.

2.4.1 Requerimientos funcionales.

El sistema para la generación de intérpretes debe ser diseñado para garantizar el proceso de validación de errores en la creación de Laboratorios Virtuales, para cumplir con dicha función debe garantizar las siguientes funcionalidades:

RF 1. Adicionar estados: el sistema debe ser capaz de adicionar los estados del

autómata.

RF 2. Mostrar estados: el sistema debe ser capaz de mostrar los estados en la tabla que les corresponde.

RF 3. Eliminar estados: el sistema debe ser capaz de eliminar los estados una vez que han sido mostrados.

RF 4. Adicionar palabra reservada: el sistema debe ser capaz de adicionar las palabras reservadas del sistema.

RF 5. Mostrar palabra reservada: el sistema debe ser capaz de mostrar las palabras reservadas en su tabla correspondiente.

RF 6. Eliminar palabra reservada: el sistema debe ser capaz de eliminar las palabras reservadas una vez que han sido mostradas.

RF 7. Adicionar transición: el sistema debe ser capaz de adicionar las transiciones entre los diferentes estados del autómata.

RF 8. Mostrar transición: el sistema debe ser capaz de mostrar las transiciones en su tabla correspondiente.

RF 9. Eliminar transición: el sistema debe ser capaz de eliminar las transiciones entre los diferentes estados del autómata.

RF 10. Adicionar no terminal: el sistema debe ser capaz de adicionar los no terminales.

RF 11. Mostrar no terminal: el sistema debe ser capaz de mostrar los no terminales en la tabla que le corresponde a estos.

RF 12. Eliminar no terminal: el sistema debe ser capaz de eliminar los no terminales.

RF 13. Adicionar elemento: el sistema debe ser capaz de adicionar los elementos (terminales y no terminales).

RF 14. Mostrar elemento: el sistema debe ser capaz de mostrar los elementos en la tabla correspondiente a estos.

RF 15. Eliminar elemento: el sistema debe ser capaz de eliminar los elementos anteriormente adicionados.

RF 16. Adicionar regla: el sistema debe ser capaz de adicionar las diferentes reglas gramaticales.

RF 17. Mostrar regla: el sistema debe ser capaz de mostrar las reglas en su tabla correspondiente.

RF 18. Eliminar regla: el sistema debe ser capaz de eliminar las diferentes reglas gramaticales anteriormente adicionadas.

RF 19. Generar código fuente: el sistema debe ser capaz de generar el código fuente correspondiente a las clases que el desarrollador señale necesarias en la aplicación.

RF 20. Crear nuevo proyecto: el sistema debe ser capaz de crear un nuevo proyecto.

RF 21.Abrir proyecto existente: el sistema debe ser capaz de abrir un proyecto ya creado.

RF 22. Guardar proyecto: el sistema debe ser capaz de poder guardar el proyecto en la dirección definida por el desarrollador.

2.4.2 Requerimientos no funcionales.

Los requisitos no funcionales, no describen funcionalidades dentro del sistema, sino cualidades con las que debe cumplir. A continuación se muestran los requisitos no funcionales del sistema:

Requerimientos de Usabilidad

RNF 1. El sistema debe tener una interfaz gráfica de usuario funcional con buena utilización de los elementos de diseño y adecuada combinación de colores.

RNF 2. El sistema deberá ser utilizado por usuarios capacitados para su interacción.

Requerimientos de Hardware del Sistema

RNF 3. Se recomienda que las computadoras donde se utilice el sistema posean 256 MB de memoria RAM como mínimo, para garantizar la integración con los laboratorios virtuales desarrollados por el proyecto.

Restricciones en el Diseño y la Implementación.

RNF 4. El código debe cumplir con los estándares de codificación establecidos por el equipo de desarrollo.

Requerimiento asociado al Licenciamiento.

RNF 5. Se debe garantizar que el sistema se desarrolle bajo los principios del software libre y por tanto cualquier componente de software que se utilice también lo debe ser.

2.5 Modelo de Casos de Uso del sistema.

El Modelo de Casos de Uso del Sistema es aquel en el que se representan los actores del sistema, los Casos de Uso y sus relaciones.

Haciendo uso de las facilidades que brinda UML, se puede proceder a la captura de los requisitos funcionales del sistema y determinar cómo será utilizado desde el punto de vista del actor, para así construirlo sobre la base de sus necesidades.

2.5.1 Actor del sistema.

El actor es una entidad externa del sistema que de alguna manera participa en la

historia del caso de uso. Por lo regular, estimula el sistema con eventos de entrada o recibe algo de él. Los actores están representados por el papel que desempeñan en el caso (28).

El desarrollador constituye el actor del sistema para la generación de intérpretes interactuando con el sistema en el proceso de detección y validación de errores.

Actor	Descripción
Desarrollador	Interactúa con el sistema para ejecutar las funcionalidades de este último.

Tabla 4: Actor del sistema.

2.5.2 Diagrama de Casos de Uso del sistema.

El Caso de uso es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso. Los casos de uso son historias o casos de utilización de un sistema, no son exactamente los requerimientos, ni las especificaciones funcionales, sino que ejemplifican e incluyen tácticamente los requerimientos en las historias que narran (28).

La siguiente Ilustración muestra la forma en que el actor del sistema (desarrollador) interactúa con el sistema para constribuir a que se ejecuten los requisitos funcionales definidos con anterioridad, a través de un diagrama de casos de uso (Ilustración 16).

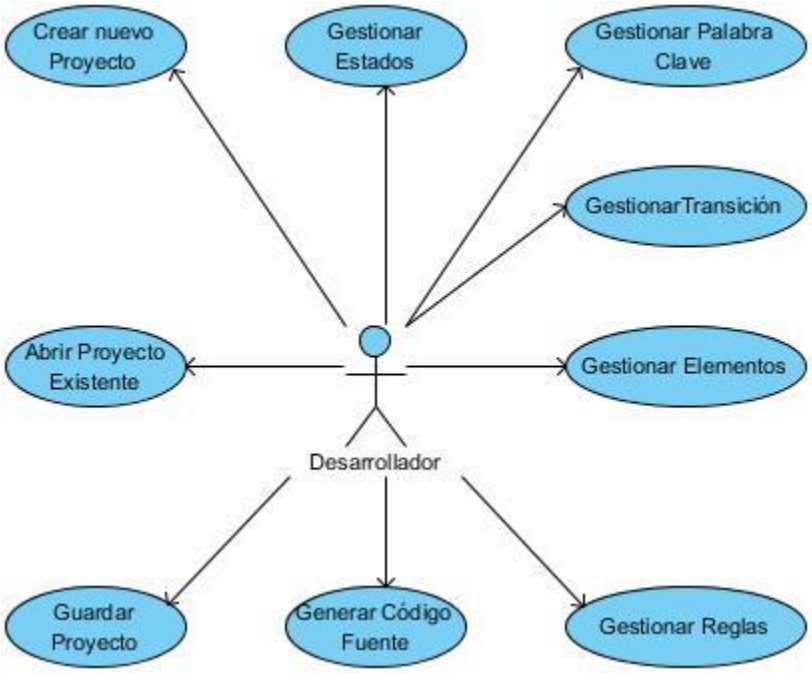


Ilustración 16: Diagrama de casos de uso del sistema.

2.5.3 Descripción de los casos de uso arquitectónicamente significativos.

Cada caso de uso tiene una descripción de las funcionalidades que ejecutará el sistema propuesto como respuesta a las acciones del usuario. Algunos casos de uso mitigan los mayores riesgos del proyecto, tienen mayor importancia para el usuario, y permiten cubrir todas las funcionalidades significativas. A estos, que son relevantes para la arquitectura, se les denomina casos de uso arquitectónicamente significativos.

Los flujos operacionales de los casos de uso arquitectónicamente significativos aparecen argumentados en las tablas ubicadas en los anexos (Anexo 2).

2.6 Plantilla de casos de prueba.

Para el diseño de casos de pruebas se utilizó la plantilla establecida por Calisoft para la UCI. La misma está compuesta por una hoja de presentación; donde se especifica el

nombre del proyecto, del producto, del caso de prueba, además cuenta con una tabla de control de versiones en donde se registra: la fecha, versión, descripción y autor.
En la hoja de Casos de Prueba; se especifican las condiciones de ejecución, que no es más que las condiciones que se deben cumplir para que se pueda realizar el requisito, se puntualiza el requisito a probar y se pasa a llenar una tabla compuesta por:

Escenario: se especifica el número y el nombre.

- Descripción: descripción del escenario de prueba.
- Variables: son los campos con que cuenta el requisito.
- Respuesta del sistema: se describe la respuesta esperada del sistema.
- Flujo central: pasos a desarrollar para probar la funcionalidad que se indicó.

En la hoja de Variables se registra en una tabla compuesta por:

- Número: se especifica el número de las variables con que cuenta el caso de prueba.
- Nombre de campo: se especifica el nombre de campo de entrada.
- Clasificación: se clasifica según el componente de diseño utilizado.
- Valor nulo: se especifica si el campo puede ser nulo o no.
- Descripción: se especifica una breve descripción de los datos que deben introducirse.

La siguiente Ilustración es solo un fragmento de la plantilla, para más detalles vea la carpeta Anexos que se adjunta a la investigación.

Escenario	Descripción	Respuesta del sistema	Flujo Central
Nombre del escenario donde se realiza el caso de prueba.	Descripción del escenario de prueba.	Se escribe el resultado que se espera la realizar la prueba.	Pasos a desarrollar para indicar la funcionalidad que se indicó.

Tabla 4: Fragmento de plantilla de casos de prueba.

2.6.1 Diseño de los casos de prueba.

El sistema fue sometido a varias pruebas basadas en la ejecución, revisión y retroalimentación de sus funcionalidades. Estas pruebas son realizadas mediante modelos de prueba cuyo objetivo es evaluar cada una de las opciones con que cuenta la solución informática y en adición, permiten detectar funciones incorrectas.

Conclusiones parciales.

En este acápite se ha realizado la selección de la metodología y herramientas que serán utilizadas en el desarrollo de la aplicación. Se efectúa un modelado del sistema a través de diferentes diagramas para una mejor comprensión de este. Se describe además lo que va a ser la solución propuesta.

Capítulo 3: Implementación y análisis de los resultados.

Introducción

En este capítulo se establece el estándar de codificación, así como la distribución física del sistema, mostrando el diagrama de componentes correspondiente. Se efectúa además la descripción de los casos de pruebas de la aplicación.

3.1 Estándar de codificación.

3.1.1 Nombres.

- Los nombres de las clases son sustantivos singulares.
- Los nombres de clases y objetos deben reflejar qué hacen y no cómo lo hacen.
- Escoger nombres lo suficientemente largos que expresen correctamente el sentido de lo que se quiere, pero evitando manejar nombres que dificulten la labor de implementación.
- Evitar nombres que permitan una interpretación subjetiva (evitar ambigüedad y asegurar abstracción).
- Evitar redundancia no repitiendo nombres de clases en sus elementos.
- Dado que los nombres generalmente son el producto de concatenar varias palabras, se debe emplear la primera palabra en minúscula, mayúscula para denotar la letra de inicio de cada una de las palabras restantes por las que esté formado y minúscula para las letras intermedias en el caso de los nombres de métodos y funciones. Para el caso de los nombres de variables y atributos debe aplicarse la misma convención.
- Las variables booleanas deben contener la palabra *is* en su nombre.
- Los nombres de constantes deben contener solo letras mayúsculas.
- Minimizar el uso de abreviaturas. En caso de ser requeridas, se debe ser consistente en su uso y cada abreviatura debe significar solo una cosa.

- Los nombres de los métodos son frases que incluyen verbos.
- Los nombres de los atributos y parámetros son frases con sustantivos.

3.1.2 Codificación.

- Se establece un tamaño de indentacion[11] estándar de tres espacios, sin tabulaciones.
- Alinear secciones del código.
- Alinear verticalmente llaves de apertura y cierre.
- Usar espacios antes y después de los operadores que el lenguaje de programación permita.
- Emplear líneas en blanco para organizar el código, permitiendo crear párrafos de código para una mejor lectura.
- Evitar colocar más de una sentencia por línea.
- Emplear constantes en sustitución de números o cadenas de caracteres literales.
- Minimizar el alcance de las variables para evitar confusión y facilitar el mantenimiento.
- Emplear cada variable y rutina solo para un propósito.
- Evitar el uso de variables públicas, sustituirlas por variables privadas y métodos que provean el valor de tal variable, para mantener el encapsulamiento.
- Emplear las letras i, j, k, l, m, p, q, r para contadores en ciclos.
- Mantener la modularidad del código bajo el criterio de la lógica que encierra, no exagerar la modularidad.
- Emplear correctamente los tipos de ciclos: si es al menos una vez usar *do-while*,

[11] Espacio o sangría que se pone a la derecha de cada línea de código.

si es ninguna o más veces usar *while*, y si se conoce el número exacto de ciclos usar *for*.

- Inicializar todas las variables.
- Emplear líneas en blanco para separar pasos lógicos (Ej.: declaraciones, lazos).

3.2 Diagrama de clases del diseño.

El diagrama de clases captura la estructura lógica del sistema y las clases que constituyen el modelo. Es un modelo estático, describiendo lo que existe y qué atributos y comportamiento tiene. Los diagramas de clases son los más útiles para ilustrar las relaciones entre las clases e interfaces. Las generalizaciones, las agregaciones y las asociaciones son todas valiosas para reflejar la herencia, la composición o el uso y las conexiones respectivamente. A continuación se muestra el diagrama de clases del diseño que compone el sistema de la solución propuesta.

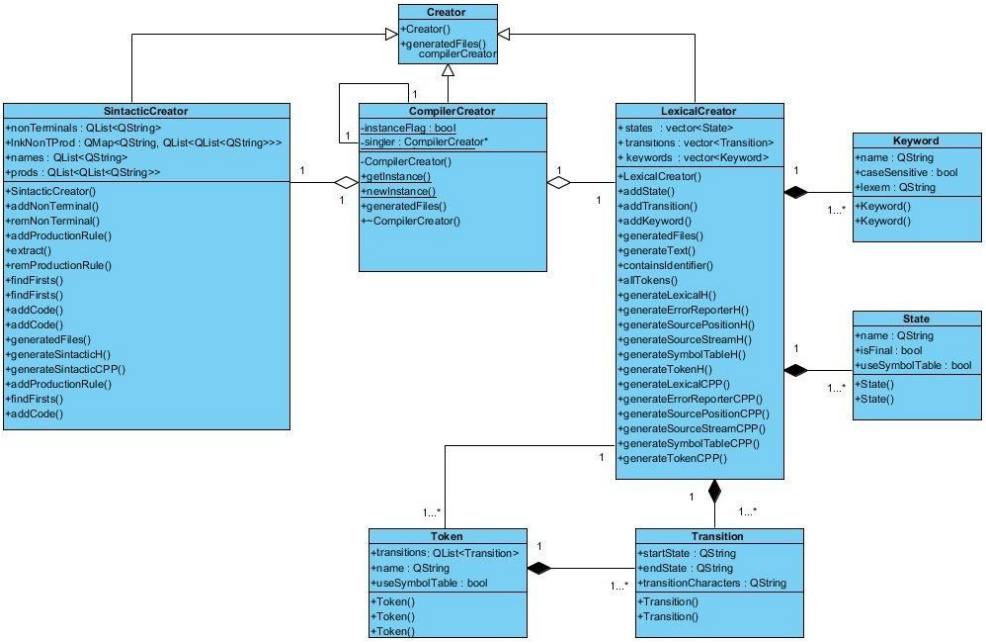


Ilustración 17: Diagrama de clases del diseño del paquete intérprete.

Ilustración 18: Diagrama de clases del diseño del paquete GUI.

A continuación se mostrará una breve descripción de las clases más importantes del sistema:

Creator: Clase que mediante el polimorfismo define métodos que son implementados en sus clases hijas, garantizando la posterior incorporación de la fase semántica.

CompilerCreator: Es la clase controladora del sistema, mediante la agregación a esta de las clases *LexicalAnalizer* y *SintacticAnalizer,* se conforma el intérprete del sistema.

LexicalAnalizer: La función de esta clase consiste en generar el código fuente correspondiente a la fase de análisis léxico. Y proveer mediante la gestión de las clases *State*, *Transition* y *Keyword* los *tokens* del sistema.

SintaxAnalizer: Clase encargada de generar el código fuente de las clases *SintaxAnalizer.h* y *SintaxAnalizer.cpp,* gestionando anteriormente los no terminales y las reglas gramaticales del sistema.

3.3 Diagrama de componentes.

Un diagrama de componentes ilustra los fragmentos de software, controladores embebidos, etc. que conformarán un sistema. Este tipo de diagrama tiene un nivel de abstracción más elevado que un diagrama de clase. Usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución.

Un componente es una pieza de software que describe un conjunto de servicios que son usados solo por medio de interfaces bien definidas. El siguiente diagrama (Ilustración 19) describe las interacciones existentes entre los principales componentes que integran la solución, así como sus dependencias más significativas.

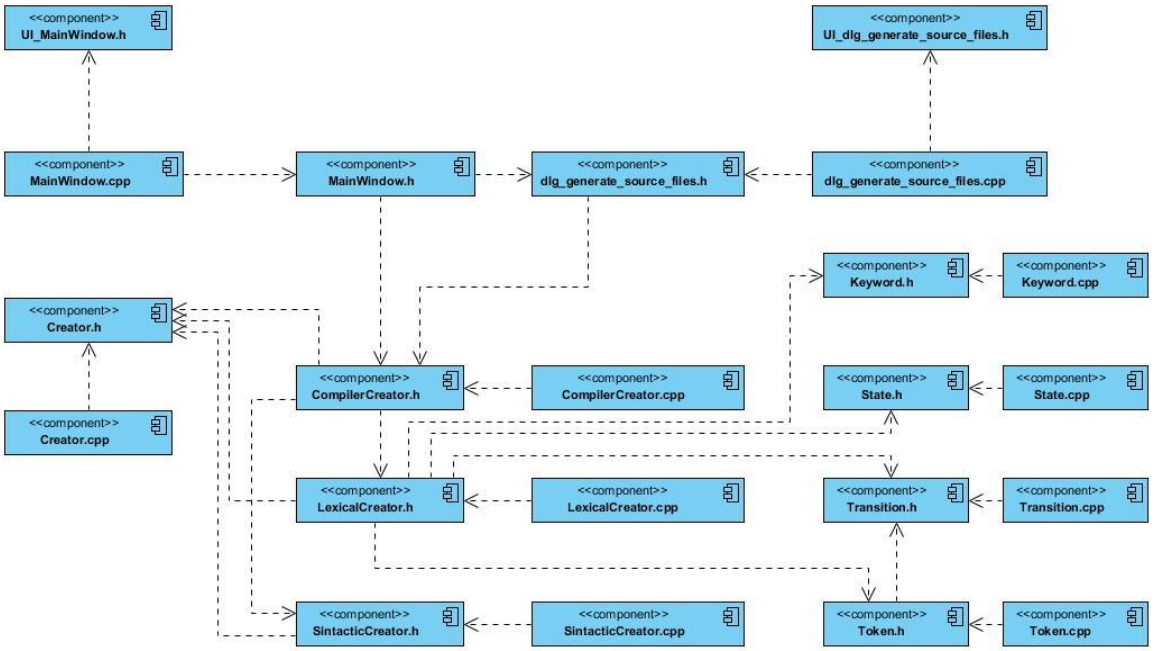


Ilustración 19: Diagrama de componentes.

3.4 Prueba del sistema.

Para validar el correcto funcionamiento del sistema, se tomó como prueba la integración de las clases generadas por la aplicación al módulo de intérpretes para el laboratorio virtual: Configuración y administración de una red de área local. Dicho módulo contaba con diferentes configuraciones de servicios tales como: TCP/IP, DNS, PROXY, FTP y SAMBA. Este último fue seleccionado para comprobar que las clases generadas por el sistema, que integradas al módulo permitirían la detección de errores sintácticos a la gramática que anteriormente se haya definido para dicho servicio.

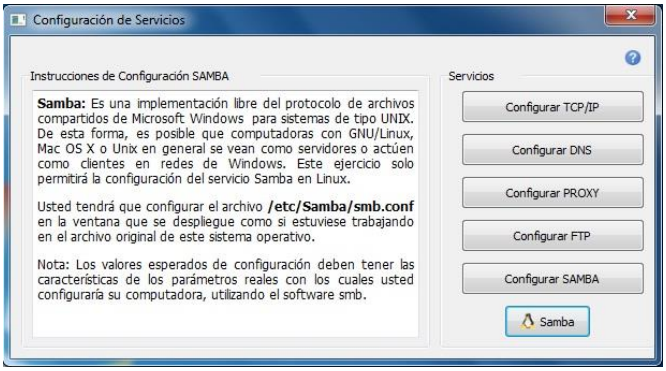


Ilustración 20: Configuración de servicios.

El módulo de intérpretes para el laboratorio virtual: Configuración y administración de una red de área local presenta una serie de botones con cada servicio por configurar y un área de texto a la izquierda donde muestra al usuario las instrucciones para la configuración del servicio seleccionado, en este caso se seleccionó la configuración del servicio SAMBA. Para la configuración de este, se solicita al usuario que inserte la configuración deseada o la cargue desde un fichero, especificando su dirección. A continuación se muestra una configuración del servicio SAMBA no válida, en el campo de *create_mask* insertando letras donde deberían ir solo números.

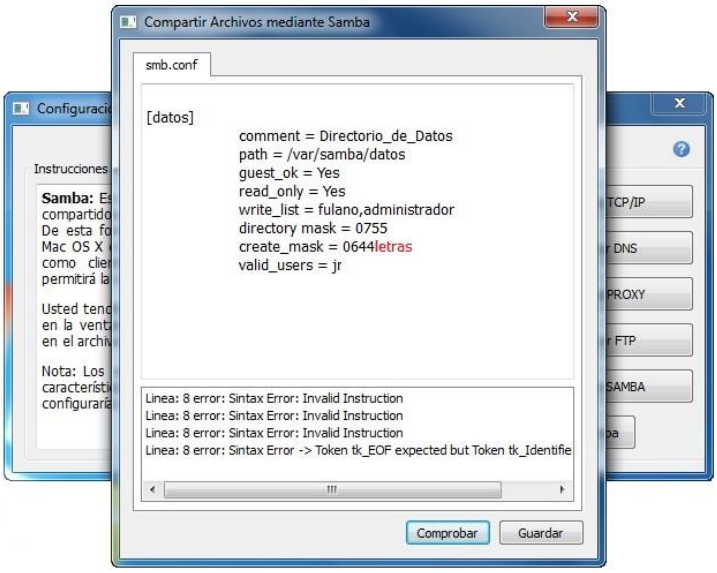


Ilustración 21: Configuración del servicio SAMBA con errores.

A partir de la inserción de elementos no válidos para la gramática definida, se muestran los errores de sintaxis correspondientes.

A continuación se cargará un fichero con una configuración válida para el servicio SAMBA.

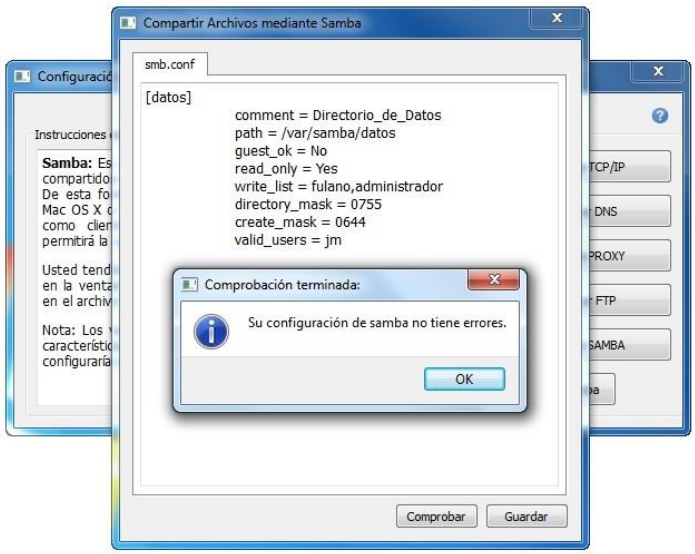


Ilustración 22: Configuración del servicio SAMBA sin errores.

Conclusiones parciales.

El estándar de codificación explicado en este capítulo es aplicado en diversos centros de nuestra universidad, el centro de Informática Industrial es uno de ellos, esto justifica el empleo del estándar de codificación utilizado en el sistema de generación para los Laboratorios Virtuales. Las pruebas realizadas evidencian le veracidad de las diversas funcionalidades que provee el sistema.

Conclusiones

Luego de terminado el proceso de investigación, desarrollo y validación de la aplicación realizada, se arriba a las siguientes conclusiones:

- El análisis de las principales herramientas generadoras de intérpretes en la actualidad muestra el constante desarrollo de las mismas, mejorando sus funcionalidades y rendimiento; pero dichas herramientas presentan negativas, descritas en el capítulo 1, cuando se trata de incorporarlas al proceso de detección y validación de errores del proyecto Laboratorios Virtuales. Para darle fortaleza al problema para el cual se desarrolló esta aplicación, se realizó una encuesta a desarrolladores del proyecto sobre el proceso de generación de intérpretes. En esta encuesta quedó evidenciada la necesidad de un sistema que agilizara dicho proceso.

- El modelado del sistema brindó una visión de la estructura del software desarrollado, así como la interacción entre sus clases.

- Los casos de prueba realizados al sistema al concluir el desarrollo del software, muestran el correcto funcionamiento de los requerimientos del sistema.

Anteriormente el desarrollador para crear un intérprete tenía que copiar uno ya elaborado, y realizar a partir de este un proceso de modificación a cada una de las clases, definiendo los métodos, los cuales emitían errores en variadas ocasiones, ya que había que tener en cuenta la dependencia de clases del intérprete anterior. Este proceso de generación de intérpretes era engorroso, por lo que surgió como necesidad agilizarlo.

Se obtiene con el presente trabajo de diploma un sistema que agiliza el proceso de generación de intérpretes existente en el proyecto Laboratorios Virtuales. El software desarrollado brinda una interfaz gráfica de usuario, mediante la cual, el desarrollador llena los campos referentes a los estados, las palabras reservadas, las transiciones, los

terminales, los no terminales, las reglas, y genera a partir de estos, el código fuente de las clases de un analizador léxico y un analizador sintáctico; las que acopladas a un nuevo proyecto realizan el análisis léxico y la detección de errores sintácticos a sentencias de código con gramática LL 1.

Recomendaciones.

Se recomienda para un posterior seguimiento del presente trabajo de diploma, que se le añadan al software desarrollado funcionalidades como:

- Reconocer dada una gramática, si la misma es LL1, y en caso de no serlo, poder convertirla en una que sí lo sea.
- Continuar el desarrollo del software incorporándole la fase de análisis semántico.
- A partir de los estados definidos y sus transiciones en el analizador léxico, mostrar el dibujo del autómata correspondiente.

Referencias bibliográficas.

1. Art applications for crowds. **Rudomin, Dr Isaac, Millán, Dr. Erik and Díaz, Marissa.** s.l. : The Knowledge Engineering Review., 2008. 23-399-412.
2. Compiladores, el comienzo: Tecnología didactica. **Crespo, Héctor Guerra.** México : s.n. 970-94054.
3. IV, Conferencia 8 de programación. **Aprendizaje, Fase de analisi semántico. Entorno Vistual de.** Universidad de las Ciencias Informáticas. : s.n., 2011.
4. **Yapura, Gustavo Daniel.** Investigación, estudio del funcionamiento e implementación de intérpretes y compiladores. [Online] [Cited: noviembre 20, 2011.] [http://www.investigacion.frc.utn.edu.ar/labsis/Publicaciones/InvesDes/Compiladores/caratula.html].
5. Intérpretes y diseño de lenguajes de programación. **Labra Gayo, Jóse Emilio.** 2004.
6. **Aho, A.V., Sethi, R., Ullman, J.D.**Compiladores: principios, técnicas y herramientas, Tema 8, 9, 10. s.l. : s.l. : addison-Wesley Iberoamericana, 1990.
7. Tema 2. Analizador Léxico. **autores, Colectivo de.** 2012.
8. Análisis Léxico. **Botía Blaya, Juan A.** Madrid, España. : s.n., 2012.
9. **Aguilera Sierra, María del Mar and Galvez Rojas, Sergio.** Análisis Sintáctico. Lenguajes y Ciencias de la Computación. [Online] 2009. . [Cited: Abril 12, 2012.] http://www.lcc.uma.es/~galvez/ftp/tci/tictema3.pdf
10. **Gálvez., José Fortes.** Análisis Semántico. [Online] Febrero 07, 2002. . [Cited: Abril 02, 2012.] http://serdis.dis.ulpgc.es/~ii-pl/ftp/transp/old/tr-asem-ver.pdf.
11. EcuRed. [Online] [Cited: Abril 15, 2012.], http://www.ecured.cu/index.php/Teor%C3%ADa_de_compiladores
12. **Scrib. Compiladores 1, Segundo Semestre.** Scrib. Compiladores 1, Segundo Semestre. . [Online] 2010. [Cited: Abril 18, 2012.] http://www.es.scribd.com/doc/34492076/Analisis-lexico.
13. Diseño de compiladores. **Garrido, J. Iñesta, Moreno, F. and Pérez, J.. s.l. : Universidad de Alicante, 2002.** s.l. : s.l. : Universidad de Alicante, 2002., 2002.
14. Módulo de Intérpretes para el laboratorio virtual Configuración y Administración de servicios de una red de área local del proyecto PROLAVI. **Vilató, José Alfredo.** Trabajo de diploma :

Universidad de las Ciencias Informáticas : s.n., 2011.

15. Herramientas YACC y BISON. **Ronceros, Ing. Mirko Manrique.**
16. Scribd. Lex y Yacc. [Online] [Cited: Abril 28, 2012.] http://es.scribd.com/doc/38204603/LEX-Y-YACC.
17. Procesadores de Lenguajes. [Online] [Cited: Febrero 16, 2012.] www.uhu.es/470004004/practicas/practica04.htm.
18. **Quesada, Juan Antonio López.**Fundamentos de Ingeniería del software. España : s.n.
19. . **Beck, K.**Extreme Programming Explained. s.l. : Embrace Change, Pearson Education, 1999.
20. **Fernández, Carlos Alberto Fernández y.**Proceso Unificado Rational para desarrollo de software. Huajuapan de León. Oaxaca : s.n., 2000.
21. Scribd. Capítulo 1 , Herramientas CASE. [Online] [Cited: Marzo 6, 2012.] http://es.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE.
22. **Mediavilla, Elena.**Programación orientada a objetos. 2007.
23. **Contreras, Brígida.**Análisis y Diseño de Sistemas, UML. República Boliviariana de Venezeuela. : s.n.
24. **Seco, José Antonio González.**El lenguaje de programación C#.
25. Introducción al Visual Studio. [Online] [Cited: Marzo 8, 2012.] http://msdn.microsoft.com/es-es/library/fx6bk1f4%28v=vs.80%29.aspx.
26. Diseño y Desarrollo Web, Internet y Teconología. Qt Creator – Completo entorno de desarrollo multiplataforma. [Online] 10 06, 2010. [Cited: Abril 04, 2012.] http://pixelcoblog.com/qt-creator-completo-entorno-de-desarrollo-multiplataforma/.
27. Perfil del ingeniero de requerimientos. Seminario de Tecnología de la Información y Software, 2007. **Castillo, Jaime F.** 2007.
28. UML y patrones. **Carig, Larman.**México : 1era Edición, 1999.

¡Error! No se encuentra el origen de la referencia.¡Error! No se encuentra el origen de la referencia.¡Error! No se encuentra el origen de la referencia.¡Error! No se encuentra el origen de la referencia.**Anexos.**

Anexo 1. Plantilla de encuesta realizada a los desarrolladores que integran el proyecto Laboratorios Virtuales.

Encuesta para la valoración de la actual generación de intérpretes en el proceso de detección y validación de errores del proyecto Laboratorios Virtuales.

Estimado usuario: Con el objetivo de ayudarlos a hacer más eficiente su trabajo, estamos haciendo un diagnóstico sobre los problemas relacionados con la generación de intérpretes en el proceso de detección y validación de errores del proyecto Laboratorios Virtuales.

Le pedimos responda con la mayor veracidad el siguiente cuestionario:

Tabla 5: Plantilla de encuesta.

1. Señale su calificación profesional:

   - ☐ Estudiante
   - ☐ Profesor
   - ☐ Especialista
   - ☐ Otra calificación

2. Diga si usted ha realizado o a formado parte del proceso de detección y validación de errores para la creación de un laboratorio virtual:

   ○ Si ○ No

3. ¿Se encuentra satisfecho con el procedimiento actual de generación de intérpretes en el proceso de detección y validación de errores del proyecto Laboratorios Virtuales?

○ Si ○ No

4. Señale las dificultades encontradas durante el procedimiento actual de generación de intérpretes en el proceso de detección y validación de errores del proyecto Laboratorios Virtuales.

- ☐ Pérdida de tiempo.
- ☐ Introducción de errores.
- ☐ Incremento del tiempo de máquina.
- ☐ Baja productividad del equipo de trabajo.
- ☐ Disminución de la moral del equipo de trabajo.
- ☐ Otra calificación

5. ¿Considera necesaria la creación de un sistema para la generación de intérpretes, que optimice la utilización de recursos y reduzca considerablemente el tiempo de desarrollo?

○ Si ○ No

Anexo 2. Descripción de los casos de uso más significativos.

Caso de Uso	Generar código fuente

Resumen	El caso de uso se inicia cuando el desarrollador selecciona generar código fuente.
Precondiciones	
Referencias	R1, R2, R3, R4, R5,R6, R7, R8, R9,
Prioridad	Crítica

Flujo normal de eventos	
Acción del Actor	Respuesta del Sistema
1. El caso de uso se inicia cuando el desarrollador selecciona generar código fuente. 2. El desarrollador señala las clases que quiere generar y selecciona la opción generar.	1.1. El sistema muestra un diálogo dando al desarrollador la posibilidad de escoger las clases que quiere generar. 2.1. El sistema genera las clases seleccionadas por el desarrollador terminando así el caso de uso.

Flujo Alterno	
Acción del Actor	Respuesta del Sistema

Poscondiciones	

Tabla 6: Caso de uso generar código fuente.

Caso de Uso	Adicionar estados.
Resumen	El caso de uso se inicia cuando el desarrollador llena el campo

	"estado" y selecciona la opción adicionar.
Precondiciones	
Referencias	R1.
Prioridad	Crítica

Flujo normal de eventos	
Acción del Actor	Respuesta del Sistema
1. El caso de uso se inicia cuando el desarrollador llena el campo "Nombre Estado", especifica a través de cheksbox si el estado es final, si lleva tabla de símbolos, luego selecciona la opción "Adicionar".	1.1. El sistema muestra en una tabla el estado agregado, si es final y si posee tabla de símbolos terminándose así el caso de uso.

Flujo Alterno	
Acción del Actor	Respuesta del Sistema

Poscondiciones	

Tabla 7: Caso de uso Adicionar Estados.

Caso de Uso	Eliminar estados.

Resumen	El caso de uso se inicia cuando el desarrollador selecciona un estado anteriormente agregado a la tabla, y escoge la opción “Eliminar”.
Precondiciones	El estado debe haber sido adicionado anteriormente.
Referencias	R3.
Prioridad	Crítica

Flujo normal de eventos	
Acción del Actor	Respuesta del Sistema
1. El caso de uso se inicia cuando el desarrollador selecciona un estado anteriormente agregado a la tabla, y escoge la opción “Eliminar”.	1.1. El sistema elimina el estado seleccionado, terminándose así el caso de uso.

Flujo Alterno	
Acción del Actor	Respuesta del Sistema

Poscondiciones	

Tabla 8: Caso de uso Eliminar Estados.

Caso de Uso	Adicionar palabra reservada.

Resumen	El caso de uso se inicia cuando el desarrollador llena el campo “Nombre” correspondiente a palabra reservada, y selecciona la opción “Adicionar”.
Precondiciones	Debe haber sido agregado primero el estado Identifier.
Referencias	R4.
Prioridad	Crítica

Flujo normal de eventos	
Acción del Actor	Respuesta del Sistema
1. El caso de uso se inicia cuando el desarrollador llena el campo “Nombre” correspondiente a palabra reservada, y selecciona la opción “Adicionar”.	1.1. El sistema adiciona la palabra reservada mostrándola en la tabla de palabras reservadas, terminándose así el caso de uso.

Flujo Alterno	
Acción del Actor	Respuesta del Sistema

Poscondiciones	

Tabla 9: Caso de uso Adicionar Palabra Reservada.

Caso de Uso	Eliminar palabra reservada.
Resumen	El caso de uso se inicia cuando el desarrollador selecciona una palabra reservada de la tabla, y escoge la opción “Eliminar”.
Precondiciones	Debe existir al menos una palabra reservada en la tabla.
Referencias	R6.
Prioridad	Crítica

Flujo normal de eventos	
Acción del Actor	Respuesta del Sistema
2. El caso de uso se inicia cuando el desarrollador selecciona una palabra reservada de la tabla, y escoge la opción “Eliminar”.	1.2. El sistema elimina la palabra reservada seleccionada, terminándose así el caso de uso.
Flujo Alterno	
Acción del Actor	Respuesta del Sistema

Poscondiciones	

Tabla 10: Caso de uso Eliminar Palabra Reservada.

Caso de Uso	Adicionar transición
Resumen	El caso de uso se inicia cuando el desarrollador escoge de los estados adicionados al sistema, un estado inicial y uno final, especifica con que caracteres se efectúa la transición y selecciona la opción “Adicionar”.
Precondiciones	Debe existir al menos un estado.
Referencias	R7.
Prioridad	Crítica

Flujo normal de eventos	
Acción del Actor	Respuesta del Sistema
1. El caso de uso se inicia cuando el desarrollador escoge de los estados adicionados al sistema, un estado inicial y uno final, especifica con que caracteres se efectúa la transición y	1.1. El sistema adiciona la transición mostrándola en una tabla de transiciones, terminando así el

selecciona la opción “Adicionar”.	caso de uso.
Flujo Alterno	
Acción del Actor	Respuesta del Sistema
Poscondiciones	

Tabla 11: Caso de uso Adicionar Transición.

Caso de Uso	Eliminar transición
Resumen	El caso de uso se inicia cuando el desarrollador selecciona una o más transiciones de la tabla y escoge la opción “Eliminar”.
Precondiciones	Debe existir al menos una transición.
Referencias	R9.
Prioridad	Crítica
Flujo normal de eventos	
Acción del Actor	Respuesta del Sistema
1. El caso de uso se inicia cuando el desarrollador selecciona una o mas	1.1. El sistema elimina la transición seleccionada,

transiciones de la tabla y les modifca los campos de la misma.	terminando así el caso de uso.
Flujo Alterno	
Acción del Actor	Respuesta del Sistema
Poscondiciones	

Tabla 12: Caso de uso Eliminar Transición.

Anexo 3. Diseño de casos de prueba establecidos para la validación del sistema.

Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 1 Gestionar Estado.	1. El desarrollador llena el campo “Nombre” hace clic en el botón Adicionar, ubicado en la sección de estados.	1.1 El sistema agrega el nuevo estado, y lo muestra en la tabla superior ubicada en la sección de estados.	• Acceder al sistema. • Acceder a la sección de Estados.

	2. El desarrollador selecciona el estado deseado, y hace clic en el botón Eliminar.	2.1 El sistema elimina el estado seleccionado, y actualiza el visual dejando de mostrarlo en la tabla.	

Tabla 13: CP 1 Gestionar Estados.

Escenario	Descripción	Respuesta del sistema	Flujo Central
EC 2 Gestionar Palabra Reservada.	1. El desarrollador llena el campo "Nombre" hace clic en el botón Adicionar, ubicado en la sección de Palabras Reservadas.	1.1. El sistema agrega la palabra reservada, y la muestra en la tabla superior ubicada en la sección de Palabras Reservadas.	• Acceder al sistema. • Acceder a la sección de Palabras Reservadas.
	2. El desarrollador selecciona la palabra reservada deseada, y hace clic en el botón Eliminar.	**2.1.** El sistema elimina la palabra reservada seleccionada, y actualiza el visual dejando de mostrarlo en la tabla.	

Tabla 14: CP 2 Gestionar Palabra Reservada.

Escenario	Descripción	Respuesta del sistema	Flujo Central
EC 3 Gestionar Transición.	El desarrollador escoge el estado inicial , el estado final, según los	El sistema agrega la transición, y la muestra en la tabla superior ubicada en	• Acceder al sistema. • Acceder a la sección de

	estados que ya han sido agregados, especifica con que carácter se efectúa la transición, y hace clic en el botón Adicionar, de la sección de Transición.	la sección de Transición.	Transición.
	El desarrollador selecciona la transición deseada, y hace clic en el botón Eliminar.	El sistema elimina la transición seleccionada, y actualiza el visual dejando de mostrarla en la tabla.	

Tabla 15: CP 3 Gestionar Transición.

Escenario	Descripción	Respuesta del sistema	Flujo Central
EC 4 Gestionar No Terminal.	El desarrollador llena el campo nombre de la sección No Terminal, y hace clic en el botón Adicionar, de la misma sección.	El sistema agrega el no terminal, y lo muestra en la tabla superior ubicada en la sección de No Terminal.	• Acceder al sistema. • Acceder a la sección de No Terminal.
	El desarrollador selecciona el no terminal deseado, y hace clic en el botón Eliminar de dicha sección.	El sistema elimina el no terminal seleccionado, y actualiza el visual dejando de mostrarlo en la tabla.	

Tabla 16: CP 4 Gestionar No Terminal.

Escenario	Descripción	Respuesta del sistema	Flujo Central
EC 5 Gestionar Elemento.	El desarrollador escoge entre los terminales y los no terminales cual será	El sistema agrega el elemento, y lo muestra en la tabla superior ubicada en	• Acceder al sistema. • Acceder a la sección de

	el elemento que desea agregar a la regla y hace clic en el botón Adicionar, de la sección Elemento.	la sección de Elemento.	Elemento.
	El desarrollador selecciona el elemento deseado, y hace clic en el botón Eliminar de dicha sección.	El sistema elimina el elemento seleccionado, y actualiza el visual dejando de mostrarlo en la tabla.	

Tabla 17: CP 5 Gestionar Elemento.

Escenario	Descripción	Respuesta del sistema	Flujo Central
EC 5 Gestionar Regla.	El desarrollador seleccionará en el el campo regla un no terminal, luego selecciona de la tabla de elementos el elemento que desea asociar adicha regla y hace clic en el botón Adicionar Regla.	El sistema agrega la regla, y la muestra en la tabla superior ubicada en la sección de Regla.	• Acceder al sistema. • Acceder a la sección de Elemento. • Acceder a la sección de Regla.
	El desarrollador selecciona la regla deseada, y hace clic en el botón Eliminar de la sección Regla.	El sistema elimina la regla seleccionada, y actualiza el visual dejando de mostrarla en la tabla.	

Tabla 18: CP 6 Gestionar Regla.

Escenario	Descripción	Respuesta del sistema	Flujo Central
EC 7 Generar Código	El desarrollador despliega el menú opciones y hace clic	El sistema muestra un diálogo, pidiendo al desarrollador que	• Acceder al sistema. • Acceder al

Fuente.	en generar código fuente. El desarrollador selecciona las clases que desea generar y hace clic en el botón Generar. El desarrollador especifica la dirección y hace clic en el botón *choose*.	seleccione las clases que desea generar. El sistema muestra una ventana, solicitando la dirección donde generar esas clases. El sistema crea una carpeta *Compiler* donde genera las clases seleccionadas por el desarrollador.	menú opciones. • Acceder al diálogo que especifica cuales clases desea generar. • Acceder a la ventana que solicita la dirección de donde generar las clases seleccionadas. • Acceder a la dirección especificada.

Tabla 19: CP 7 Generar Código Fuente.

Escenario	Descripción	Respuesta del sistema	Flujo Central
EC 8 Crear Nuevo Proyecto	El desarrollador marca el menú Archivos, y hace clic en Nuevo Proyecto.	Si no se habia efectuado ninguna acción en la aplicación, el sistema se quedaría tal como estaba, si ya se hubiera realizado al menos una acción, el sistema se resetea, volviendo al estado inicial.	• Acceder al sistema. • Acceder al menú Archivos. • Acceder al campo Nuevo Proyecto.

Tabla 20: CP 8 Abrir Proyecto Existente.

Escenario	Descripción	Respuesta del sistema	Flujo Central
EC 9 Abrir Proyecto Existente.	El desarrollador marca el menú Archivos, y hace clic en Abrir Proyecto.	El sistema muestra una ventana solicitando la dirección de donde cargar el archivo de	• Acceder al sistema. • Acceder al menú Archivos.

	El desarrollador especifica la dirección donde se encuentra el proyecto que desea abrir y hace clic en el botón Abrir.	extención .ccr. El sistema carga el fichero seleccionado, mostrando en la aplicación todos los campos de la misma.	• Acceder al campo Abrir Proyecto. • Acceder a la dirección donde se encuentra el proyecto.

Tabla 21: CP 9 Crear Nuevo Proyecto.

Escenario	Descripción	Respuesta del sistema	Flujo Central
EC 10 Guardar Proyecto	El desarrollador marca el menú Archivos y hace clic en Guardar Proyecto.	El sistema muestra una ventana solicitando la dirección donde desea guardar el proyecto.	• Acceder al sistema. • Acceder al menú Archivos. • Acceder al campo Guardar Proyecto. • Acceder a la ventana de solicitud de dirección. • Acceder a la dirección donde se guarda el archivo .ccr.
	El desarrollador especifica la dirección donde desea guardar el proyecto y hace clic en el botón Guardar.	El sistema guarda en la dirección especificada un fichero.ccr con los datos del proyecto.	

Tabla 22: CP 10 Guardar proyecto.

Anexo 4. Resultados obtenidos tras la aplicación de los casos de prueba establecidos para el sistema.

CP: Caso de prueba.

EC: Escenario del caso.

CP 1 Gestionar estados.

EC 1 Gestionar estados.

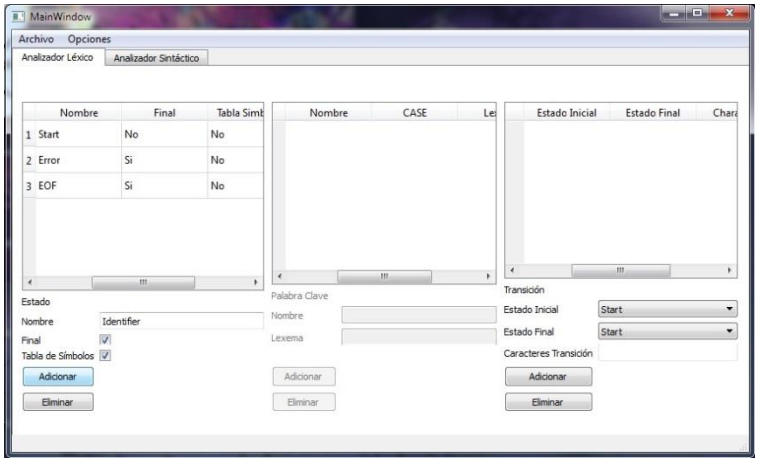


Ilustración 23: Acción del desarrollador (Adicionar estado).

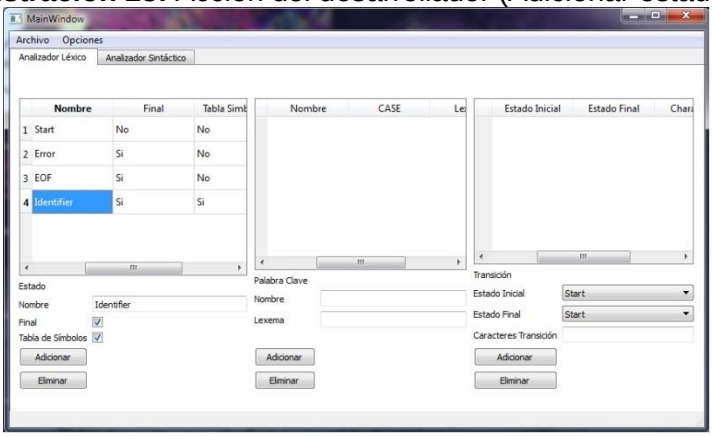


Ilustración 24: Acción del sistema (Adiciona y muestra estado).

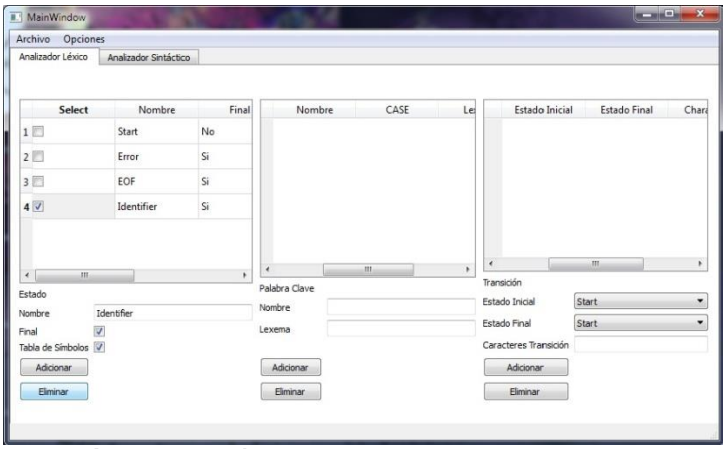


Ilustración 25: Acción del desarrollador (Eliminar estado).

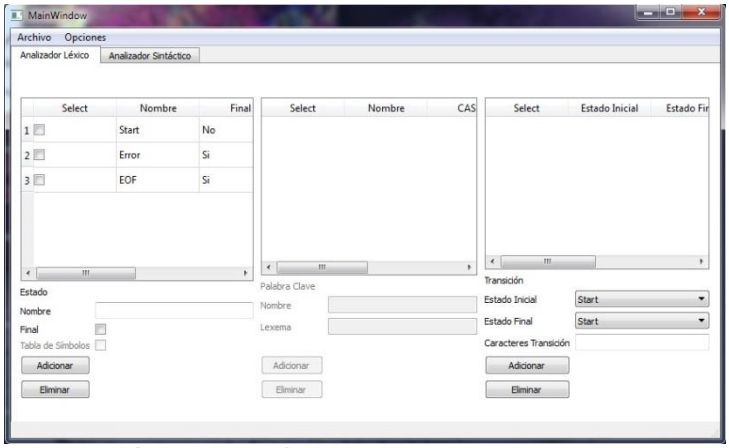


Ilustración 26: Acción del sistema (Elimina estado).

CP 2 Gestionar palabra reservada.

EC 2 Gestionar palabra reservada.

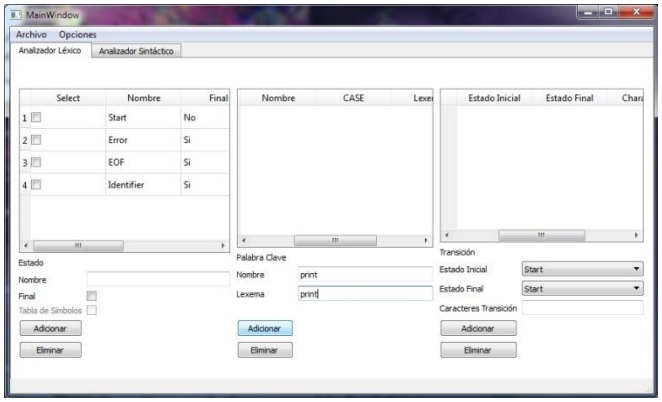


Ilustración 27: Acción del desarrollador (Adicionar palabra reservada).

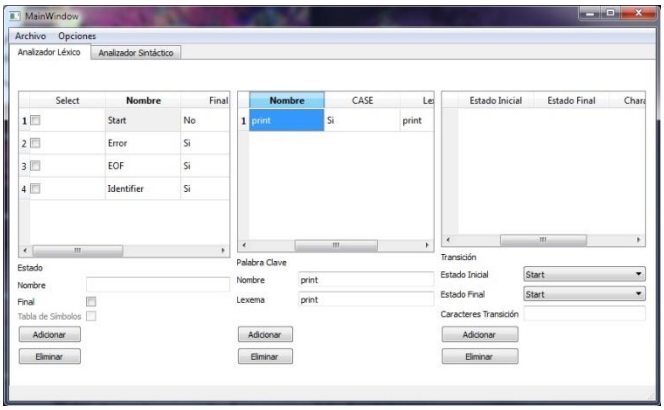


Ilustración 28: Acción del sistema (Mostrar palabra reservada)

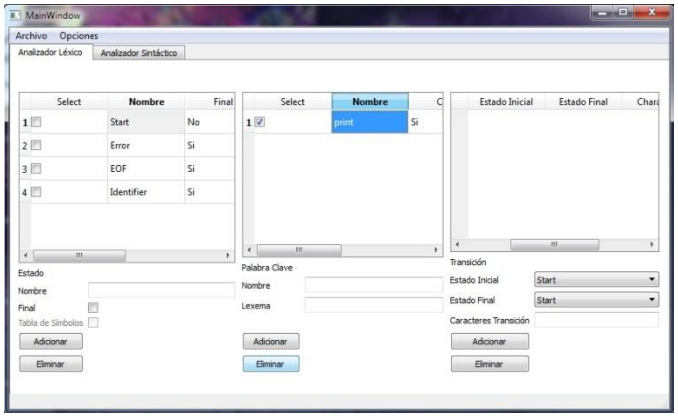


Ilustración 29: Acción del desarrollador (Eliminar palabra reservada).

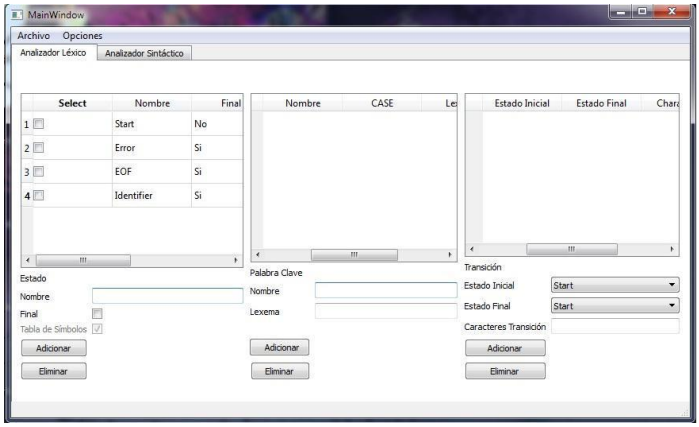


Ilustración 30: Acción del sistema (Elimina palabra reservada).

CP 3 Gestionar transición.

EC 3 Gestionar transición.

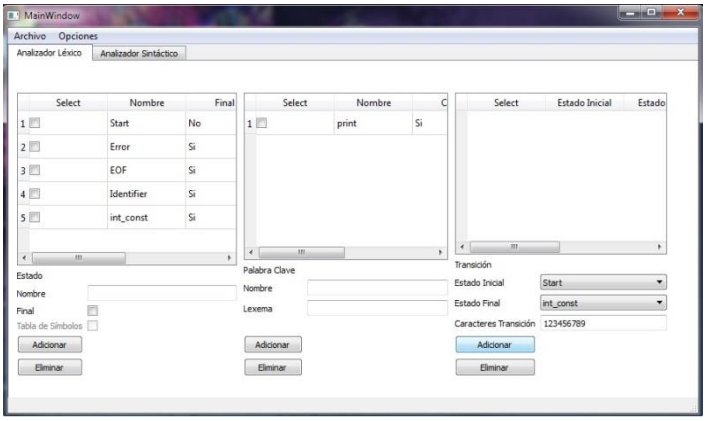
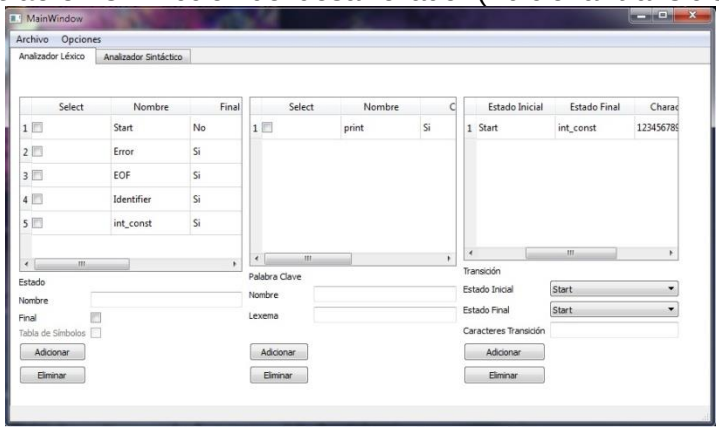


Ilustración 31: Acción del desarrollador (Adicionar transición).



.

Ilustración 32: Acción del sistema (Adiciona y muestra transición).

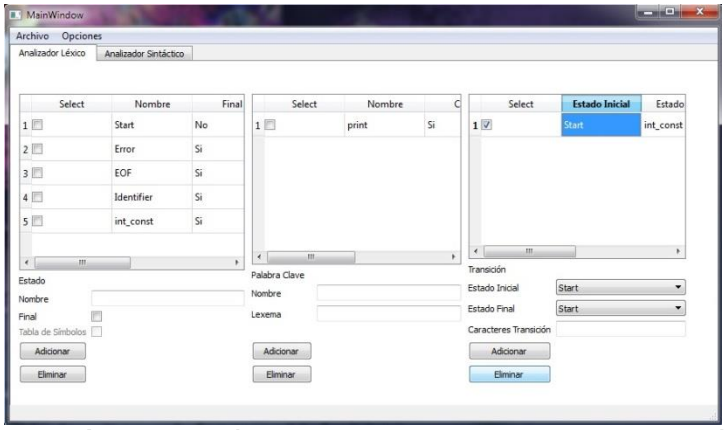


Ilustración 33: Acción del desarrollador (Eliminar transición).

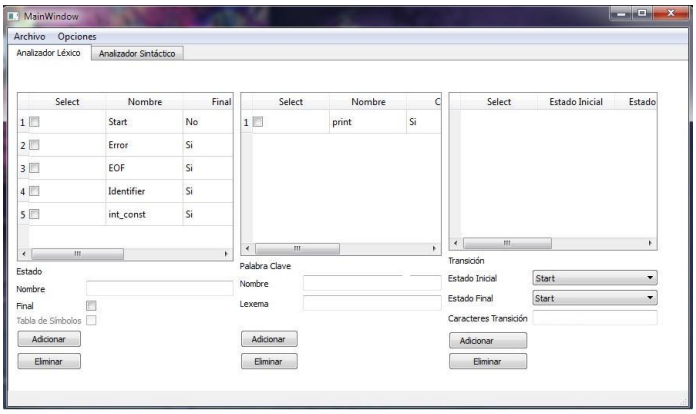


Ilustración 34: Acción del sistema (Elimina transición).

CP 4 Gestionar no terminal.

EC 4 Gestionar no terminal.

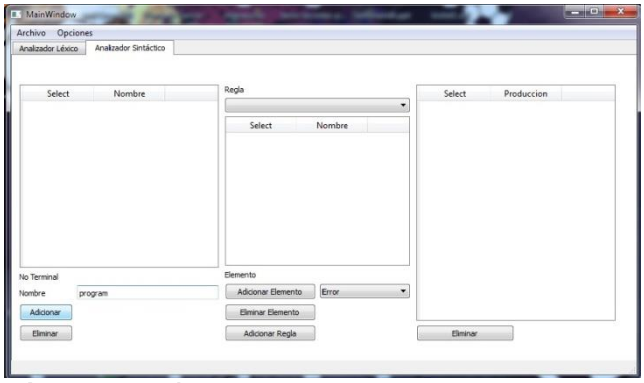


Ilustración 35: Acción del desarrollador (Adicionar no terminal).

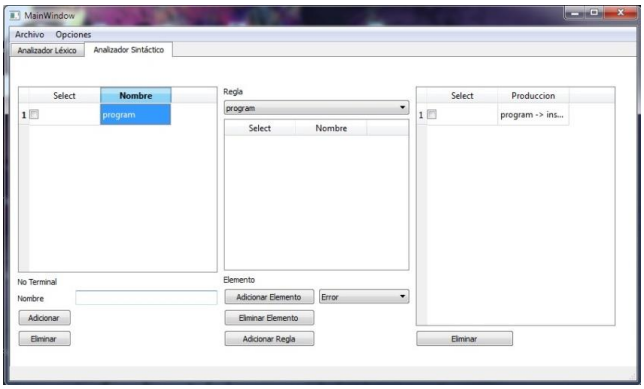


Ilustración 36: Acción del sistema (Adiciona y muestra no terminal).

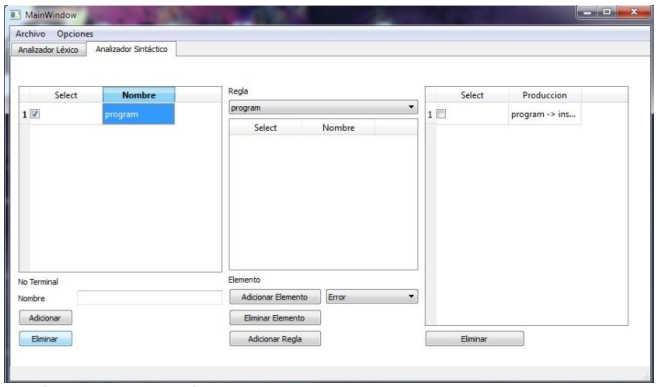


Ilustración 37: Acción del desarrollador (Eliminar no terminal).

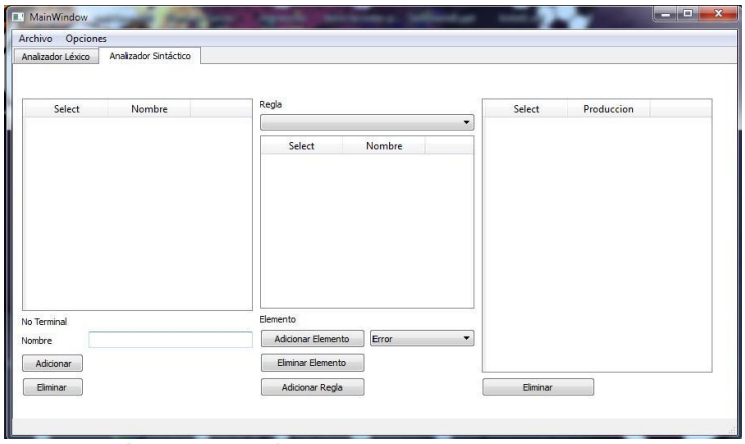


Ilustración 38: Acción del sistema (Elimina no terminal).

CP 5 Gestionar elemento.

EC 5 Gestionar elemento.

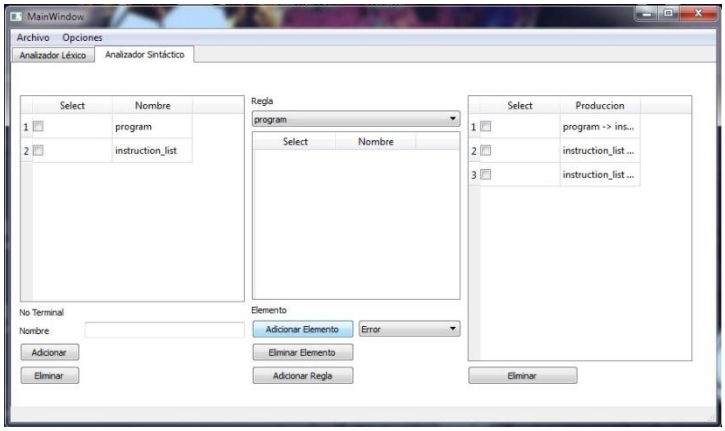


Ilustración 39: Acción del desarrollador (Adicionar elemento).

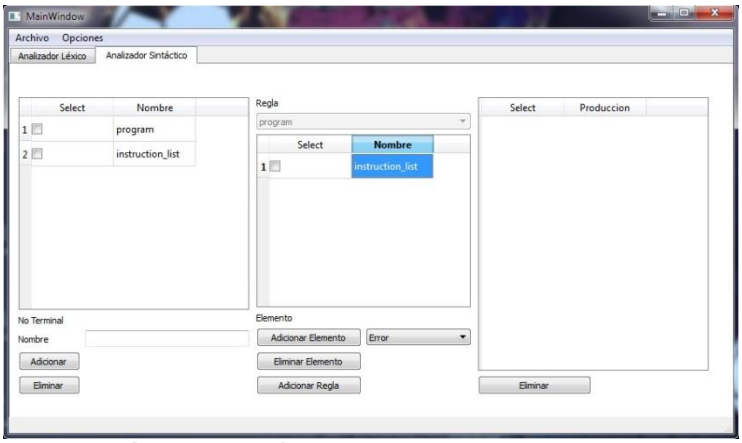


Ilustración 40: Acción del sistema (Muestra elemento).

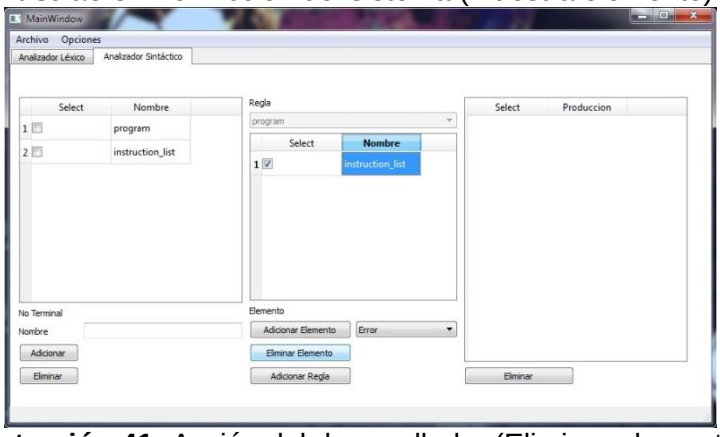


Ilustración 41: Acción del desarrollador (Eliminar elemento).

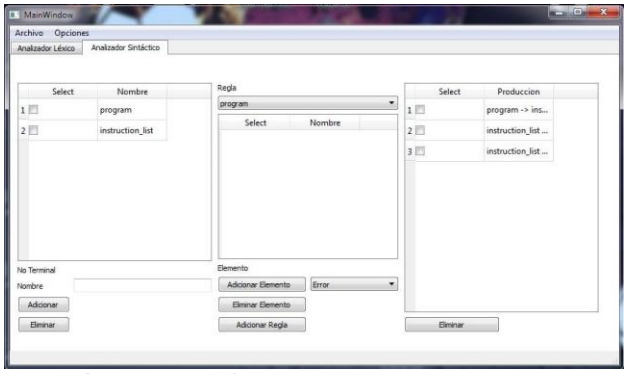


Ilustración 42: Acción del sistema (Elimina elemento)

CP 6 Gestionar regla.

EC 6 Gestionar regla.

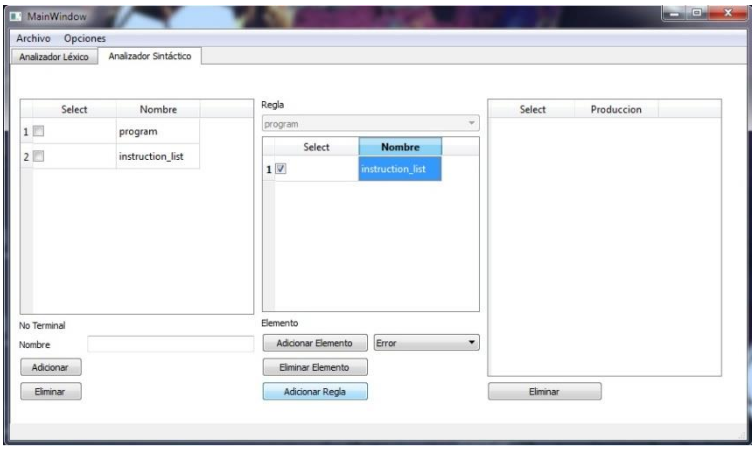


Ilustración 43: Acción del desarrollador (Adicionar regla).

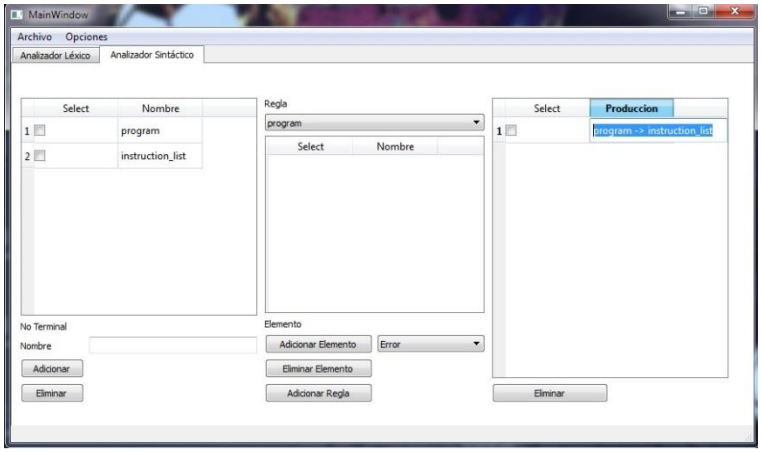


Ilustración 44: Acción del sistema (Adiciona y muestra regla).

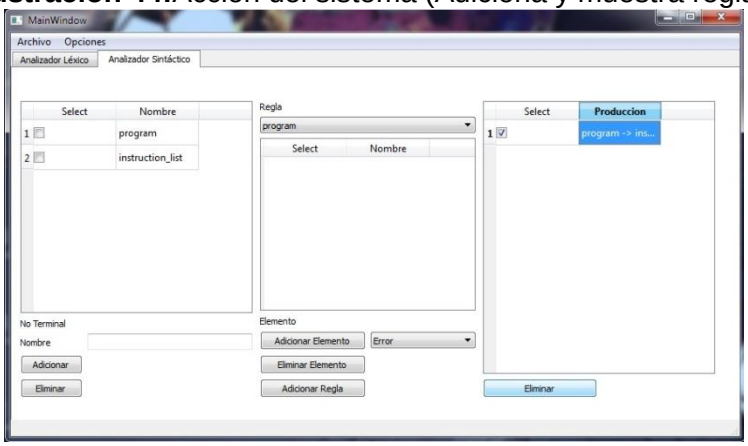


Ilustración 45: Acción del desarrollador (Elimina regla).

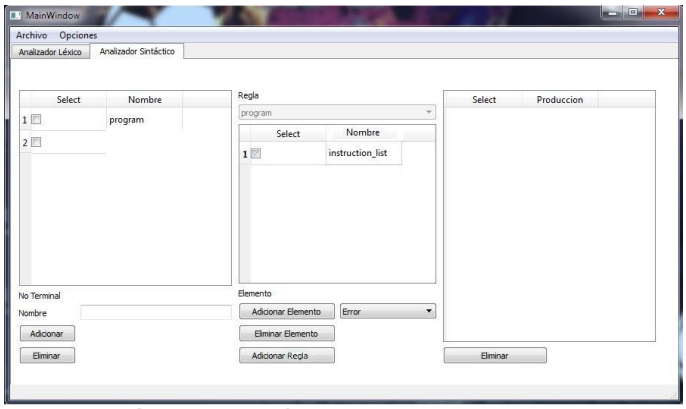


Ilustración 46: Acción del sistema (Elimina regla).

CP 7 Generar código fuente.

EC 7 Generar código fuente.

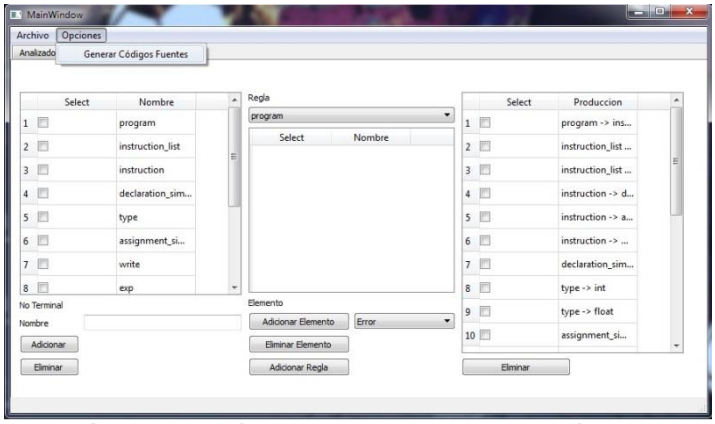


Ilustración 47: Acción del sistema (Generar código fuente).

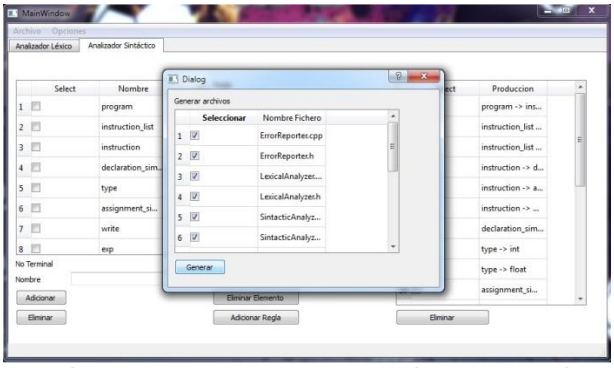


Ilustración 48: Acción del sistema (Muestra diálogo), acción del desarrollador (Selecciona clases).

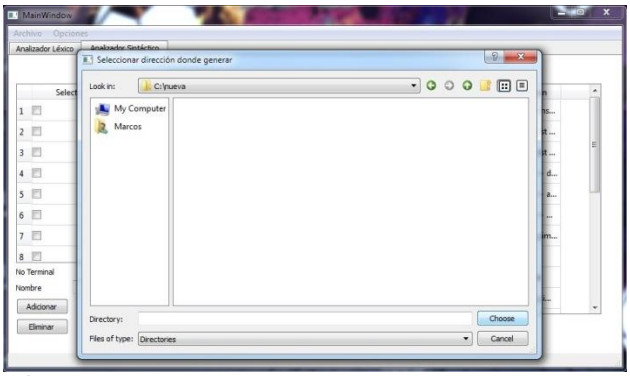


Ilustración 49: Acción del sistema (Muestra ventana de solicitud de dirección), acción del desarrollador (Especifica dirección donde generar las clases).

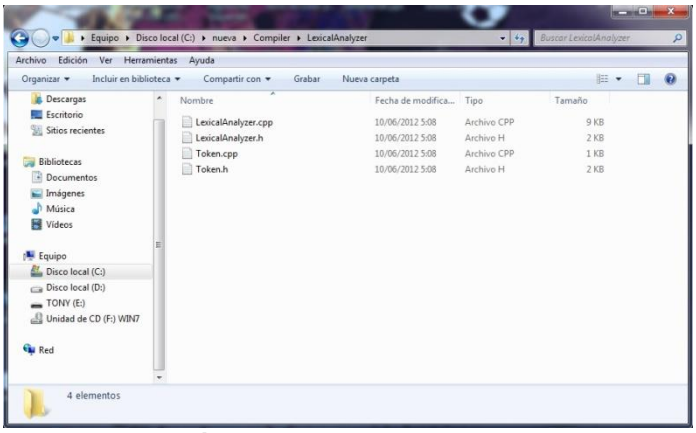


Ilustración50: Directorio *LexicalAnalizer.*

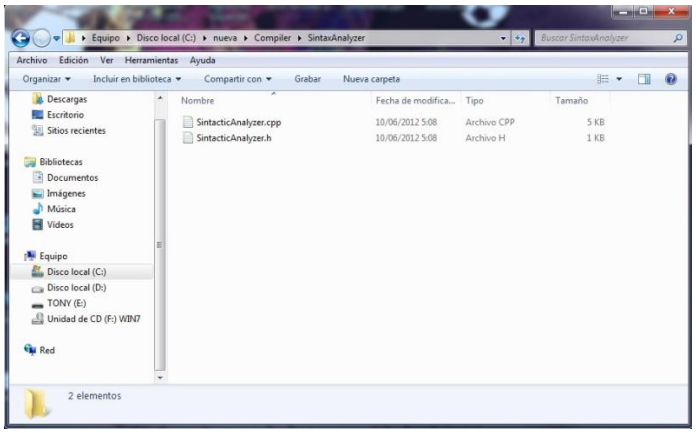


Ilustración51: Directorio *SintacticAnalizer.*

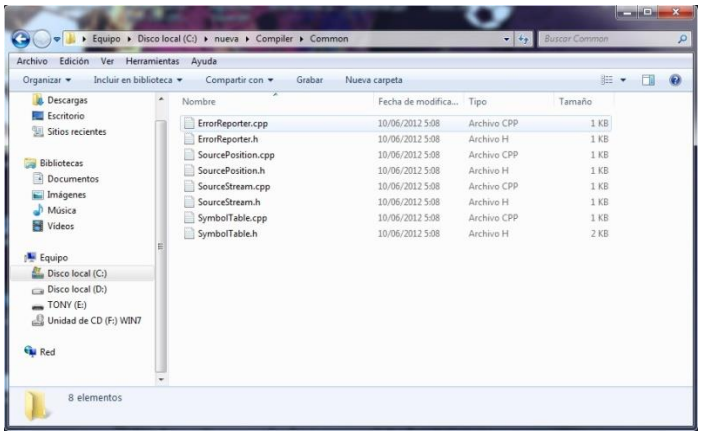


Ilustración 52: Carpeta *Common.*

CP 8 Abrir nuevo proyecto.

EC 8 Abrir nuevo proyecto.

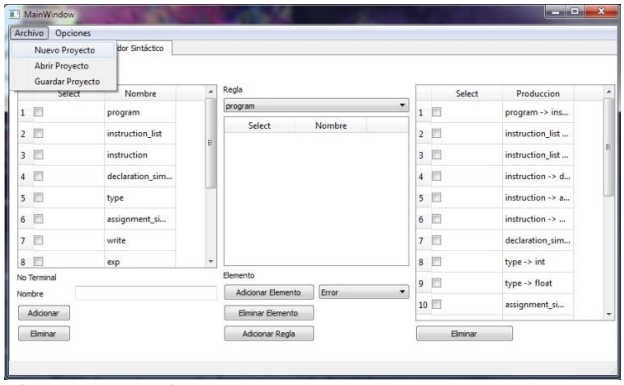


Ilustración 53: Acción del desarrollador (Abrir nuevo proyecto).

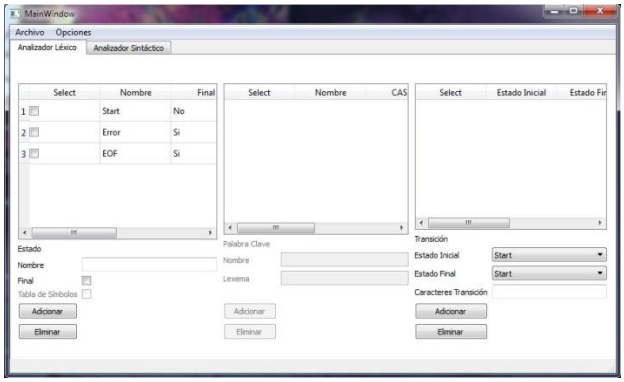


Ilustración 54: Acción del sistema (Abre proyecto vacío).

CP 9 Abrir proyecto existente.

CE 9 Abrir proyecto existente.

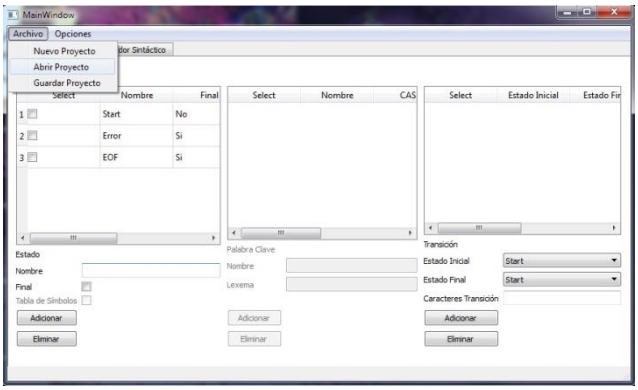


Ilustración 55: Acción del desarrollador (Abrir proyecto existente).

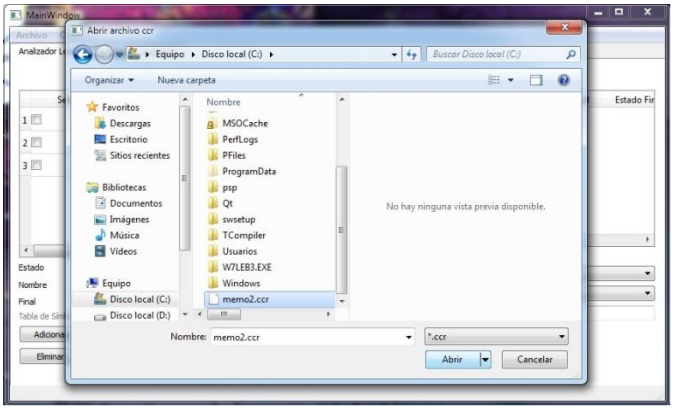


Ilustración 56: Acción del sistema (Muestra ventana solicitando dirección donde se encuentra el proyecto). Acción del sistema: Abre el proyecto seleccionado.

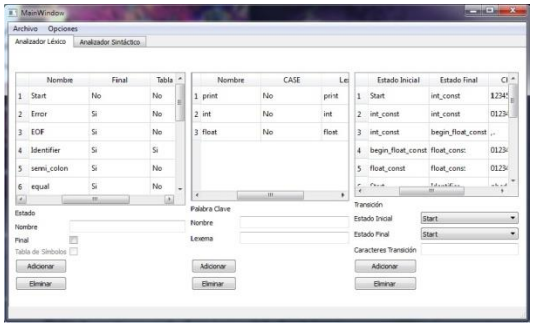


Ilustración57: Analizador léxico.

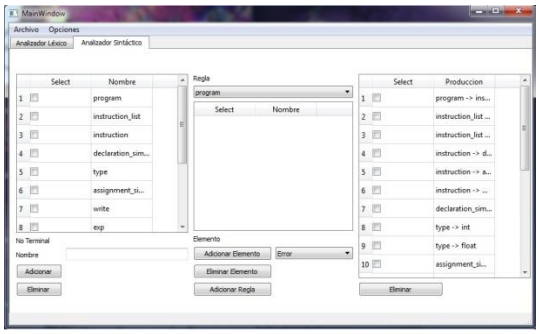


Ilustración 58: Analizador sintáctico.

CP 10 Guardar proyecto.

EC 10 Guardar proyecto.

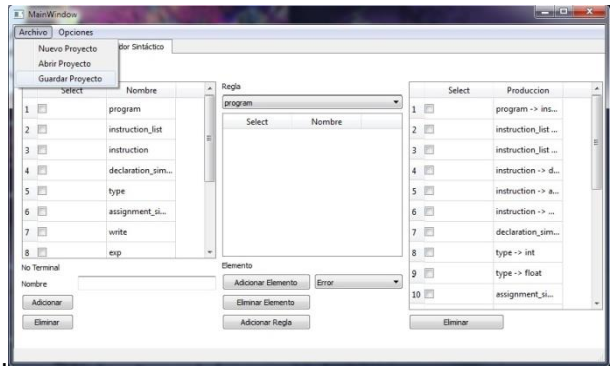


Ilustración 59: Acción del desarrollador (Guardar proyecto).

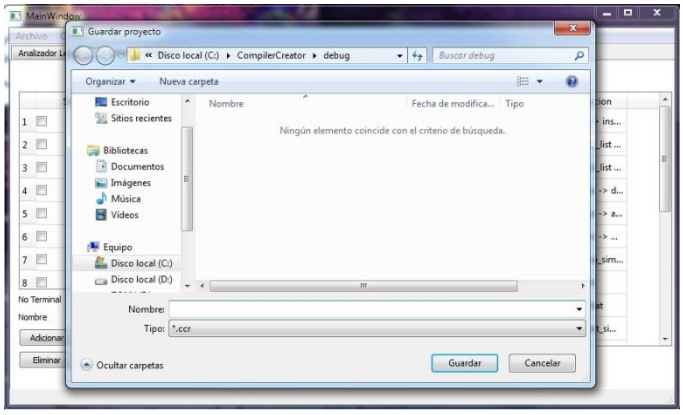


Ilustración 60: Acción del sistema (Muestra Ventana de solicitud de dirección).

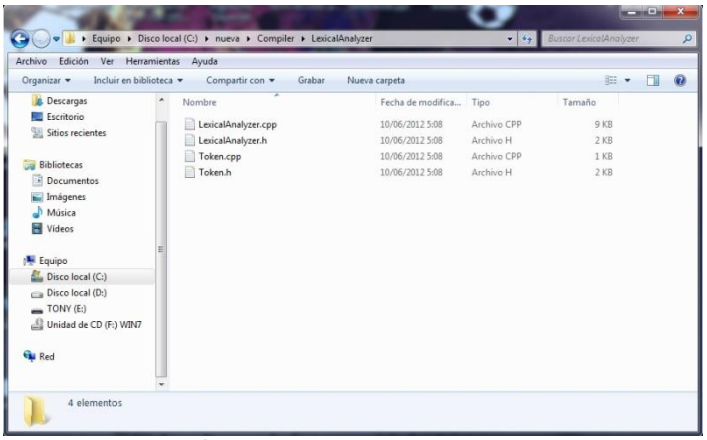


Ilustración 61: Directorio *LexicalAnalizer*.

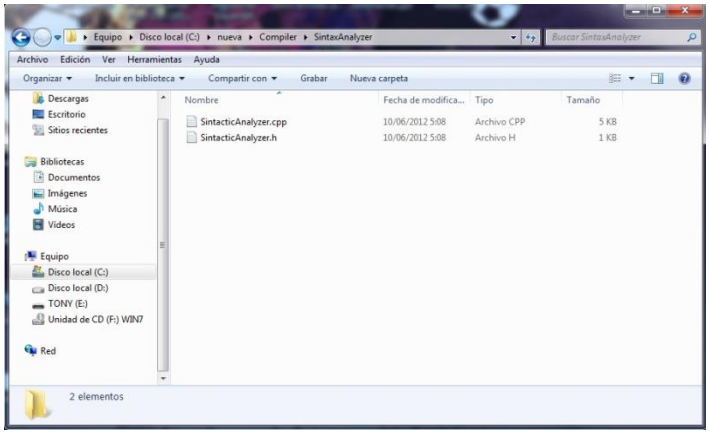


Ilustración 62: Directorio *SintacticAnalizer*.

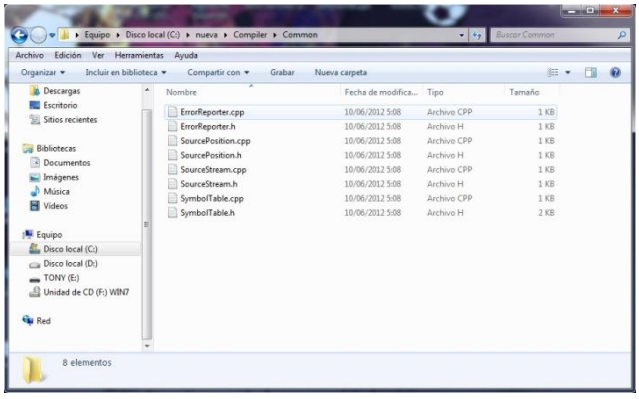


Ilustración 63: Directorio *Common***.**