INTERPRETE LATÍN PARA LA IMPLEMENTACIÓN DE TRANSACCIONES CORPORATIVAS.

DIEGO ARMANDO ESTUPIÑÁN SÁNCHEZ

JEISSON VICENTE NIÑO CASTILLO

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICO-MECÁNICA

ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

BUCARAMANGA

2012

INTERPRETE LATÍN PARA LA IMPLEMENTACIÓN DE TRANSACCIONES CORPORATIVAS.

DIEGO ARMANDO ESTUPIÑÁN SÁNCHEZ

JEISSON VICENTE NIÑO CASTILLO

Trabajo de grado para optar al título de

Ingeniero de Sistemas

Director

JAIME OCTAVÍO ALBARRACÍN FERREIRA

Doctor en Informática por la Universidad de Oviedo

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICO-MECÁNICA

ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

BUCARAMANGA

2012

DEDICATORIA

A Jesucristo, porque lo que hago es para su gloria.

A mis padres, por su paciencia y perseverancia.

A mi familia, novia y amigos que han ayudado de uno u otra manera para que ésto sea una realidad

Jeisson

A Dios, por brindarme fortaleza durante este proceso.

A mis padres y hermanos por su apoyo, colaboración y respaldo.

A Jeisson, por su trabajo, dedicación y compromiso.

A mis amigos, por su confortación y comprensión.

Diego

AGRADECIMIENTOS

Jeisson

A los profesores que son profesionales y se desempeñan con excelencia en su labor. Y más a aquellos que deciden ‘no amoldarse al mundo actual’.

Al profesor Jaime Albarracín, por su ayuda y constancia.

A mis padres, gracias por su titánico esfuerzo, seguir apoyándome y confiar en mí.

A Diego por su buena labor y disciplina,

Diego

A Dios. Por la salud, seguridad y paciencia para afrontar las diferentes obstáculos que se presentaron durante la realización de este proyecto y en general durante todo el proceso universitario.

A mis Familiares. Porque con su apoyo, esfuerzo y colaboración ha sido posible la realización de este trabajo.

A todas aquellas personas que de una u otra manera contribuyeron a la realización de este proyecto.

TABLA DE CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

LISTA DE ANEXOS

Pág.

.

RESUMEN

TITULO:

INTERPRETE LATIN PARA LA IMPLEMENTACIÓN DE MODELOS EMERGENTES DE DATOS.*

AUTORES:

DIEGO ARMANDO ESTUPIÑAN**
JEISSON VICENTE NIÑO**

PALABRAS CLAVES
Sistema, MODE, Analizador Léxico, Analizador Sintáctico, Intérprete, LATIN.

DESCRIPCIÓN

Observar la realidad como objetos implica que cualquier elemento puede afectar a otro así no pertenezca al mismo conjunto, pero en la realidad no sucede de esta forma, necesariamente deben tener algún tipo de conexión.

Una organización hace parte de la realidad, es decir que pertenece a un sistema. "Un sistema es un conjunto de objetos relacionados entre sí de forma tal que cada uno afecta estructural y funcionalmente a todo el conjunto" también es válido decir que un sistema es un conjunto de clases que interactúan y comparten un método común; empleando estos enunciados, el profesor Albarracín expone en su tesis doctoral el MODE (Modelo de Datos Emergente) que básicamente refuta las teorías en las que mencionan objetos como entes agrupados al azar, además, presenta este modelo que supera los objetos y propone a LATIN (Language Toward Integration) que es el lenguaje capaz de describir sistemas.

En la actualidad no existen lenguajes para describir Sistemas, sino solo estructuras (3GL, DDL o SQL), u Objetos (ODL o OQL). Por lo tanto ha sido ideado y denominado LATIN (Language Toward Integration) el lenguaje adecuado para describir dichos sistemas. Aquí en este trabajo de investigación se especifica los detalles de implementación del Intérprete LATIN.

*Trabajo de Grado. Modalidad: Investigación
**Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería de Sistemas e Informática, Director: Jaime Octavio Albarracín Ferreira.

ABSTRACT

TITLE:

LATIN INTERPRETER FOR THE IMPLEMENTATION OF EMERGING MODELS OF DATA.*

AUTHORS:

DIEGO ARMANDO ESTUPIÑAN SANCHEZ **
JEISSON VICENTE NIÑO CASTILLO **

KEY WORDS

System, MODE, Lexical Analyzer, Parser, Interpreter, LATIN.

DESCRIPTION

Observing reality as objects implies that any element can affect another even if it does not belong to the same set, but in reality it does not happen that way, necessarily some kind of connection must exist.

An organization is part of reality, it means that it belongs to a system. "A system is a set of interrelated objects between themselves so that each one of them affects structurally and functionally the whole set" is also valid to say that a system is a set of classes that interact and share a common method; using these statements, Professor Albarracín presents in his doctoral thesis MODE (Emerging Data Model), which basically refutes theories in which objects randomly grouped as entities are mentioned, also presents this model that exceeds the objects and proposes LATIN (Language Toward Integration) which is the program able to describe systems.

Currently there are no languages to describe systems, but only structures (3GL, DDL or SQL), or Objects (ODL or OQL). Thus LATIN (Language Toward Integration) has been conceived and called the appropriate language to describe such systems. Here in this research the implementation details of LATIN interpreter are specified.

_____
*Working Grade. Mode: Investigation.
** Faculty of Physical-Mechanical Engineering, School of Engineering and Information Systems, Director: Jaime Albarracin Octavio Ferreira.

INTRODUCCIÓN

En el estudio realizado por el profesor Albarracín en su tesis doctoral, muestra las falencias en la aplicación de procesos en las organizaciones, ya que muchas de éstas trabajan con modelos que emplean una estructura digital estándar, pero en ocasiones estos procedimientos son tan robustos que pueden llegan a ser de difícil compresión para el usuario e incluso lo obliga a tener conocimientos previos para su manipulación

El problema no es propiamente la elección del software, simplemente no se ha desarrollado un producto que actúe bajo las condiciones del mundo real, basado en que todo es un sistema y esto a su vez es un conjunto de objetos relacionados entre sí de forma tal que cada uno afecta estructural y funcionalmente a todo el conjunto[1].

Esto fue precisamente la principal inquietud que llevó a desarrollar un proyecto enfocado en describir los pasos que modelen eficientemente las operaciones internas de una empresa, de tal manera que los usuarios puedan utilizar un software simple, sencillo y que produzca resultados coherentes y de fácil compresión.

Teniendo en cuenta la documentación realizada por el Doctor Jaime Octavio Albarracín, a través de la experiencia como profesor y por los estudios realizados en Colombia y en España, propone el lenguaje LATIN (Language Toward Integration) donde expone el esquema para elaborar una herramienta real que permita materializar lo que se está describiendo.

De esta manera surge el "**INTERPRETE LATÍN PARA LA IMPLEMENTACIÓN DE TRANSACCIONES CORPORATIVAS."** con el propósito de especificar este lenguaje y sustentar la elaboración de la gramática de la escritura de LATIN, profundizándola y realizando un esfuerzo porque este, se ajuste lo mejor posible con la realidad de la organización descrita en el MODE, así mismo los analizadores léxico y sintáctico que son los que permiten controlar las entradas digitadas por el usuario y dan paso a la generación de la estructura de los archivos de jerarquía de clases.

[1]ALBARRACIN FERREIRA JAIME OCTAVIO. "Integración de datos y procesos en las organizaciones mediante un Modelo de Datos Reorientado a Objetos", Tesis Doctoral. Universidad de Oviedo. España. 2006.

Básicamente este lenguaje interpreta sistemas y ejecuta transacciones como por ejemplo compras y se obtiene un resultado detallado dependiendo de las operaciones que se realicen. Esto se hace gracias a la descripción desarrollada en C# y SQLite que son los mecanismos que se utilizaron para llevar a cabo este intérprete.

1. PRESENTACIÓN DEL PROYECTO

1.1 DEFINICIÓN DEL PROBLEMA

Los servicios que brinde cualquier sistema dentro de una organización deben mejorar constantemente y adaptarse a cambios que presenten en su entorno, tales como actualizaciones, implementación de nuevos módulos, e incluso la sustitución del mismo si éste no se ajusta completamente a las necesidades de la empresa. Por esta razón el interés de los desarrolladores de software en sacar al mercado la más completa evolución en sistemas y el interés de los investigadores dispuestos a redactar artículos, con el fin de presentar información actual a los medios, de tal forma que sus productos no se tornen anticuados e incluso insuficientes comparados con otros.

Precisamente con el afán de estar innovando y de mostrar una de las labores del ingeniero de sistemas, el profesor Albarracín en su tesis doctoral expone la situación actual en cuanto a bases de datos y manejo de software en las organizaciones. Explica también que las empresas que cuentan con un mecanismo digital para el apoyo y soporte de las transacciones cotidianas, están sujetas a recibir informes de sus propias operaciones, limitadas a códigos y datos que no precisamente conocen algunos usuarios, e incluso a realizar consultas que obliga necesariamente a tener conocimientos previos.

Es por ésto la idea de la construcción de un intérprete del lenguaje LATIN capaz no solo de mostrar información entendible para un desarrollador si no que sea de fácil acceso al usuario y en general para la organización, es decir, que sea competente y muestre en cualquier instancia un detalle completo de las transacciones básicas de la empresa.

1.2 OBJETIVOS

1.2.1 Objetivo General

Implementar el intérprete de un lenguaje para describir sistemas, por medio de transacciones ejemplo, orientadas por la directriz de los modelos emergentes de datos. Este lenguaje será denominado LATIN (Language Toward Integration).

1.2.2 Objetivos Específicos

- Especificar la sintaxis y la semántica del LATIN, requerida para la descripción de sistemas.
- Implementar los analizadores léxico y sintáctico del Interprete LATIN, los cuales permitirán describir y consultar sistemas.
- Implementar la conversión de los sistemas describibles sujetos a las especificaciones propias del LATIN (entradas) a (SQlite) *en cuanto a la estructura del sistema.*
- Implementar la conversión de los sistemas describibles sujetos a las especificaciones propias del LATIN en C# *en cuanto a la funcionalidad del sistema.*
- Implementar la transacción de ejemplo compra de contado (con la orientación del profesor Jaime Albarracín consignada en su libro Modelo de datos emergentes para integrar la organización).
- Implementar la transacción de ejemplo compra a plazo (con la orientación del profesor Jaime Albarracín consignada en su libro Modelo de datos emergentes para integrar la organización).
- Diseñar e implementar una interfaz gráfica en el lenguaje C# que permita interactuar con los analizadores léxico y sintáctico.

1.3 JUSTIFICACIÓN

Porque el profesor Albarracín, no se conformó con cuestionar un paradigma, sino que dio un paso adelante en presentar una posible solución, con su novedoso aporte, dicho paradigma cuestionado no es nada más ni nada menos que el de

objetos (PO), que en la actualidad en el mundo del software es dominante, el profesor lo replantea reorientándolo hacia el paradigma de sistemas (PS), teniendo en mente que los objetos en la realidad, son solo los constituyentes de un sistema; pero no aceptando arbitrariamente cualquier objeto sino, solamente los que se encuentran interrelacionados, es decir, los objetos propios del sistema.

En su tesis, también es planteado que los objetos no existen en la naturaleza, de una manera aislada, desordenada, y al azar, sino que se encuentran organizados en sistemas. Que concebir la realidad como objetos es ver únicamente las partes. Pero ir hasta los sistemas es comprender la verdadera grandeza de la realidad.

En este proyecto se materializó la alternativa propuesta por el profesor Albarracín para construir software apropiado para las organizaciones corporativas. Pero surge un problema para llevar a cabo esta tarea, dado que no existe en la actualidad un leguaje que pueda describir sistemas. Para Los lenguajes de programación de hoy, tales como Visual Basic, C#, J#, C++ e incluso OQL, una clase es como si fuera una entidad con sus respectivos atributos y su propio método. Pero para el modelo de datos reorientado a sistemas planteado, las clases continúan de igual manera con sus atributos y su método, pero la diferencia radica en que existe un método común para todas las clases. Esto permite la integración de todas las entidades (clases) que hacen parte del sistema. Para que dicha integración no sea simplemente en lo estructural como en el caso anterior, sino también en lo funcional como en el caso propuesto y en armonía con el concepto de sistemas. Aquí sistemas es el conjunto de elementos u entidades interrelacionadas de tal forma que cada una afecta estructuralmente y funcionalmente a todo el conjunto.

Por ésto nace la propuesta de un leguaje que permia describir el modelo de sistemas, un lenguaje que integrando objetos permita la descripción de sistemas e impulse la integración de las corporaciones, este lenguaje fue denominado LATIN.

1.4 ALCANCES Y LIMITACIONES

Las acciones que desarrolla el intérprete le permitirá al usuario obtener una completa y detallada información de las transacciones que se realicen dentro de una organización, estableciendo las opciones de actualizar, crear nuevos

sistemas y permitiendo la ejecución de forma ordenada para encontrar información en cualquier instancia de dichos procesos.

Asimismo contiene el intérprete LATIN un componente que realiza el análisis léxico y sintáctico, que proporcionará una ayuda para la detección aproximada de errores, y evitará que se finalice las operaciones defectuosas, interrumpiendo el proceso y proporcionando al usuario mensajes informativos.

Gracias a la acción de depuración por separado, favorecerá la interpretación de la información, ya que se efectúan el ingreso de datos, consulta y ejecución final.

Es importante resaltar que además de contener los botones principales que ejecutan las funciones propias del intérprete, cuenta con la ayuda que explica las instrucciones de uso y funcionalidad de cada botón, de tal manera que simplifica el trabajo del usuario.

Para la comprensión del modelo planteado por el Profesor Jaime Albarracín, se implementaron unas transacciones ejemplo, para que el usuario pueda llegar a ver de una manera, más gráfica el alcance de dicho planteamiento. En estas operaciones el usuario podrá interactuar con el sistema creado con anterioridad con la ayuda del LATIN, pudiendo observar los alcances del modelo.

La ejecución correcta del interprete requiere la existencia previa del Microsoft .NET Framework 4, que se puede descargar gratuitamente de la página del desarrollador y éste se encuentra incluido en los pasos de instalación del interprete. Además de ésto, dependiendo del caso, se puede requerir de permisos especiales del sistema operativo (Windows 7), para la correcta ejecución.

2. MARCO TEÓRICO

2.1 INTRODUCCIÓN A INTÉRPRETES

2.1.1 Concepto De Traductor[2].

A grandes rasgos, un traductor es un programa que lee un programa escrito en un lenguaje, el lenguaje *fuente*, y lo traduce a un programa equivalente en otro lenguaje, el lenguaje *objeto*. Como parte importante de este proceso de traducción, el traductor informa a su usuario de la presencia de errores en el programa fuente. Los traductores engloban tanto a los compiladores (en los que el lenguaje destino suele ser código máquina) como a los intérpretes (en los que el lenguaje destino está constituido por las acciones atómicas que puede ejecutar el intérprete). La figura 1 muestra el esquema básico que compone a un compilador/intérprete.

Figura 1. Esquema preliminar de un traductor.

Es importante destacar la velocidad con la que hoy en día se puede construir un compilador. En la década de 1950, se consideró a los traductores como programas notablemente difíciles de escribir. El primer compilador de Fortran (Formula Translator), por ejemplo, necesitó para su implementación el equivalente a 18 años de trabajo individual (realmente no se tardó tanto puesto que el trabajo se desarrolló en equipo). Hasta que la teoría de autómatas y lenguajes formales se aplicó a la creación de traductores, su desarrollo ha estado plagado de problemas y errores. Sin embargo, hoy día un compilador

[2] JAVA A TOPE: TRADUCTORES Y COMPILADORES CON LEX/YACC, JFLEX/CUP Y JAVACC. Pag 3.

básico puede ser el proyecto fin de carrera de cualquier estudiante universitario de Informática.

Desde los orígenes de la computación, ha existido un abismo entre la forma en que las personas expresan sus necesidades y la forma en que un ordenador es capaz de interpretar instrucciones. Los traductores han intentado salvar este abismo para facilitarles el trabajo a los humanos, lo que ha llevado a aplicar la teoría de autómatas en diferentes campos y áreas concretas de la informática, dando lugar a los distintos tipos de traductores y entre ellos se encuentra el intérprete.

2.1.1.1 Intérpretes

Es un programa en el cual su salida es una ejecución, es decir él recibe como entrada ciertas instrucciones y ejecuta a la vez estas. No se produce un resultado físico (código máquina) sino lógico (una ejecución). Hay lenguajes que sólo pueden ser interpretados, como p.ej. SNOBOL (StriNg Oriented SimBOlyc Language), LISP (LISt Processing), algunas versiones de BASIC (Beginner's All-purpose Symbolic Instruction Code), etc.

Su principal ventaja es que permiten una fácil depuración. Entre los inconvenientes podemos citar, en primer lugar, la lentitud de ejecución , ya que al ejecutar a la vez que se traduce no puede aplicarse un alto grado de optimización; por ejemplo, si el programa entra en un bucle y la optimización no está muy afinada, las mismas instrucciones se interpretarán y ejecutarán una y otra vez, aumentando así el tiempo de ejecución del programa. Otro inconveniente es que durante la ejecución, el intérprete debe residir en memoria, por lo que consumen más recursos.

Algunos lenguajes intentan aunar las ventajas de los compiladores y de los intérpretes y evitar sus desventajas; son los lenguajes pseudointerpretados. En estos, el programa fuente pasa por un pseudocompilador que genera un pseudoejecutable. Para ejecutar este pseudoejecutable se le hace pasar por un motor de ejecución que lo interpreta de manera relativamente eficiente. Esto tiene la ventaja de la portabilidad, ya que el pseudoejecutable es independiente de la máquina en que vaya a ejecutarse, y basta con que en dicha máquina se disponga del motor de ejecución apropiado para poder interpretar cualquier pseudoejecutable. El ejemplo actual más conocido lo constituye el lenguaje

Java; también son pseudointerpretadas algunas versiones de Pascal y de COBOL (COmmon Bussiness Oriented Language). La figura 2 muestra los pasos a seguir en estos lenguajes para obtener una ejecución.

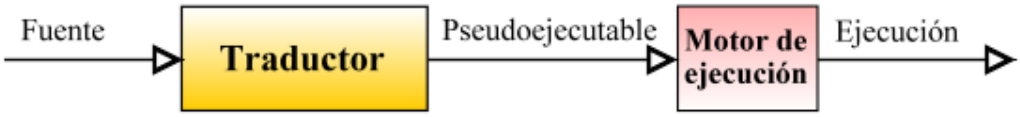

Figura 2. Esquema de traducción/ejecución de un programa interpretado.

2.1.2 Analizador Léxico

Se encarga de buscar los componentes léxicos o palabras que componen el programa fuente, según unas reglas o patrones.

La entrada del analizador léxico podemos definirla como una secuencia de caracteres, que pueda hallarse codificada según cualquier estándar: ASCII (American Standard Code for Information Interchange), EBCDIC (Extended Binary Coded Decimal Interchange Code), Unicode, etc. El analizador léxico divide esta secuencia en palabras con significado propio y después las convierte a una secuencia de terminales desde el punto de vista del analizador sintáctico. Dicha secuencia es el punto de partida para que el analizador sintáctico construya el árbol sintáctico que reconoce la/s sentencia/s de entrada, tal y como puede verse en la figura 3.

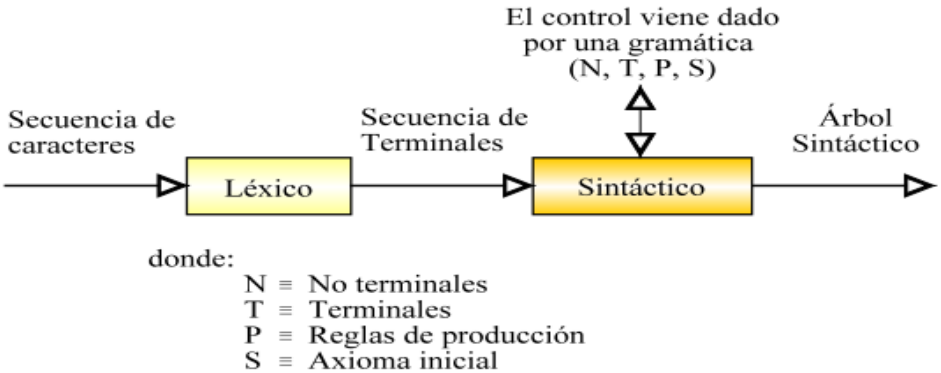

Figura 3. Entradas y salidas de las dos primeras fases de la etapa de análisis Léxico.

Secuencia de caracteres: son la agrupación de las letras del alfabeto español, números, cambios de línea, tabulaciones, espacios y en general todos los caracteres que tienen representación en el código ASCII.

Secuencia de Terminales: hace referencia a la gramática que recibe el analizador sintáctico; pero también es posible considerar que dicha secuencia es de no terminales si usamos el punto de vista del lexicográfico.

N: un símbolo no terminal es una composición de símbolos no terminales, como es el caso de una fórmula, una expresión condicional y, en general, cualquier parte de un programa que no sea una palabra, un símbolo o un número.

T: un símbolo terminal es un símbolo que tu analizador léxico puede leer directamente, tales como números, símbolos especiales (como los paréntesis, etc.), palabras clave e identificadores (nombres de variables, etc.).

P: es un conjunto de producciones o reglas.

S: es un símbolo inicial y pertenece a los no terminales.

La frase “Secuencia de Terminales” hace referencia a la gramática del sintáctico; pero también es posible considerar que dicha secuencia es de no terminales si usamos el punto de vista del lexicográfico.

El analizador léxico reconoce las palabras en función de una gramática regular de manera que el alfabeto G de dicha gramática son los distintos caracteres del juego de caracteres del ordenador sobre el que se trabaja (que forman el conjunto de símbolos terminales), mientras que sus no terminales son las categorías léxicas en que se integran las distintas secuencias de caracteres. Cada no terminal o categoría léxica de la gramática regular del análisis léxico es considerado como un terminal de la gramática de contexto libre con la que trabaja el analizador sintáctico, de manera que la salida de alto nivel (no terminales) de la fase léxica supone la entrada de bajo nivel (terminales) de la fase sintáctica.

2.1.2.1 Funciones Del Analizador Léxico

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

Esta interacción suele aplicarse convirtiendo al analizador léxico en una subrutina o corrutina del analizador sintáctico. Recibida la orden "Dame el siguiente componente léxico "del analizador sintáctico, el léxico lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico, el cual devuelve al sintáctico según el formato convenido (ver figura 4).

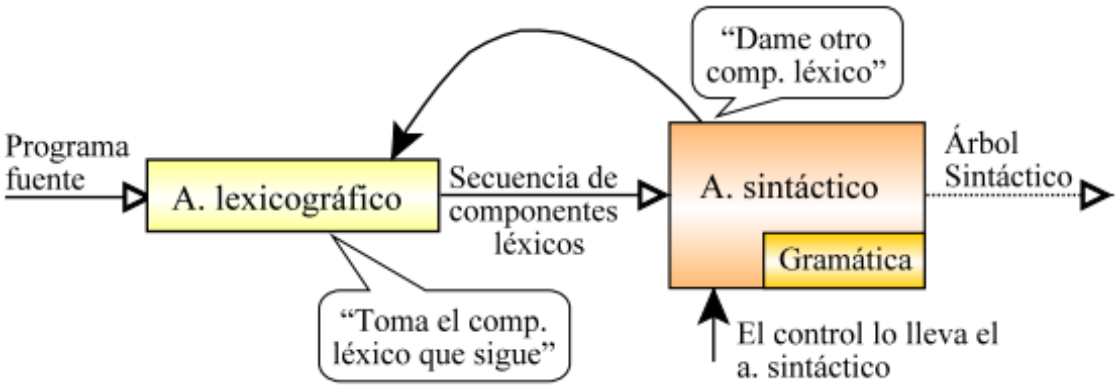

Figura 4. Interacción analizador léxico con el analizador sintáctico

Además de esta función principal, el analizador léxico también realiza otras de gran importancia, a saber:

- Eliminar los comentarios del programa.
- Eliminar espacios en blanco, tabuladores, retorno de carro, etc., y en general, todo aquello que carezca de significado según la sintaxis del lenguaje.
- Reconocer los identificadores de usuario, números, palabras reservadas del lenguaje, etc., y tratarlos correctamente con respecto a la tabla de símbolos[3] (solo en los casos en que este analizador deba tratar con dicha estructura).
- Llevar la cuenta del número de línea por la que va leyendo, por si se produce algún error, dar información acerca de dónde se ha producido.
- Avisar de errores léxicos. Por ejemplo, si el carácter '@' no pertenece al lenguaje, se debe emitir un error.

2.1.3 Analizador Sintáctico

[3] Es una estructura de datos donde cada símbolo en el código fuente de un programa está asociado con información tal como la ubicación, tipo de datos y el ámbito de cada variable

La tarea del analizador sintáctico es determinar la estructura sintáctica de un programa a partir de los tokens[4] producidos por el analizador léxico y, ya sea de manera explícita o implícita, construir un árbol de análisis gramatical o árbol sintáctico que represente esta estructura. De este modo, se puede ver el analizador sintáctico como una función que toma como su entrada la secuencia de tokens producidos por el analizador léxico y que produce como su salida el árbol sintáctico.[5]

Secuencia de tokens $\xrightarrow{\text{analizador sintactico}}$ *arbol sintactico*

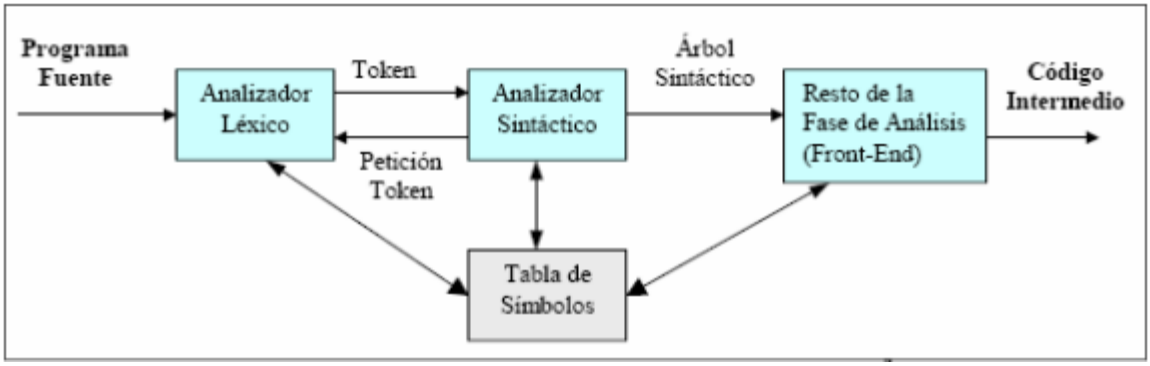

Figura 5. Esquema general del análisis sintáctico.

Esencialmente un árbol sintáctico se pertenece a una sentencia, acata a una gramática, y forma una representación que se utiliza para detallar el proceso de derivación de dicha sentencia. La raíz del árbol es la sentencia inicial y, según nos convenga, lo dibujaremos en la cima o en el fondo del árbol.
Además de determinar si la gramática establecida por el lenguaje se respeta, el analizador sintáctico realiza las siguientes funciones:

- Acceder a la tabla de símbolos para armar el árbol sintáctico.
- Revisar los tipos.
- Generar código intermedio.
- Generar errores cuando se producen.

[4] Es una agrupación de caracteres.
[5] Construcción de compiladores principios y practica - Kenneth C Louden; Pag. 96

① E	→	E + E
②	|	E * E
③	|	num
④	|	id
⑤	|	(E)

Figura 6. Gramática[6] ambigua que reconoce expresiones aritméticas

Veamos un ejemplo utilizando la gramática del cuadro 1; presumamos que hay que reconocer la cadena (a + b) * a + b que el analizador léxico nos ha suministrado como (id + id) * id + id y vamos a construir el árbol sintáctico, que sería el de la figura 7.

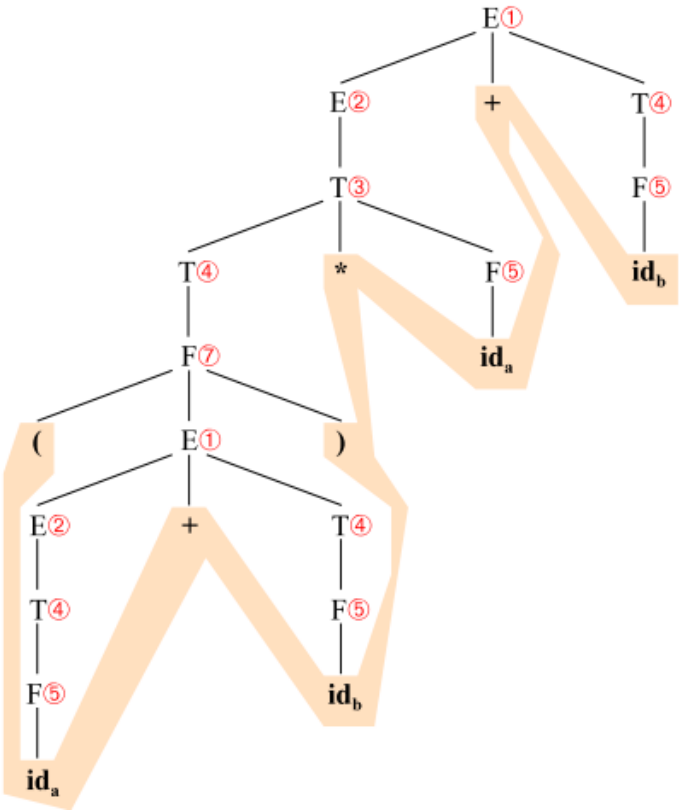

Figura 7. Árbol sintáctico correspondiente al reconocimiento de la a b a b secuencia (id + id) * id + id, que se halla marcada de color.[7]

[6] La gramática es el estudio de las reglas y principios que regulan el uso de las lenguas y la organización de las palabras dentro de una oración.
[7] JAVA A TOPE: TRADUCTORES Y COMPILADORES CON LEX/YACC, JFLEX/CUP Y JAVACC. Pag. 55

Según la aproximación que se tome para construir el árbol sintáctico se desprenden dos tipos o clases de analizadores:

- Descendentes: parten del axioma inicial, y van efectuando derivaciones a izquierda hasta obtener la secuencia de derivaciones que reconoce a la sentencia. Pueden ser:
  - Con retroceso.
  - Con funciones recursivas.
  - De gramáticas LL (1).

- Ascendentes: Parten de la sentencia de entrada, y van aplicando derivaciones inversas (desde el consecuente hasta el antecedente), hasta llegar al axioma inicial. Pueden ser:
  - Con retroceso.
  - De gramáticas LR (1).

Para generar el analizador sintáctico se utilizó la versión 0.1.1941 de C#Cup que es una herramienta que recibe como entrada un archivo con las reglas gramaticales, los símbolos terminales y no terminales, así como también algunos métodos definidos por el usuario para ser implementados con las clases CS. Estos generadores de Analizadores léxico y sintáctico respectivamente son usados en lenguajes como C y JAVA en la actualidad, no solo para sistemas operativos Windows, sino que también hay versiones para sistemas Unix.

Los archivos creados por C#Cup son: sym.cs y parser.cs. El primero contiene los símbolos utilizados en la gramática y el segundo contiene el analizador sintáctico con las reglas gramáticas, los símbolos terminales y no terminales así como también los métodos utilizados para la traducción que permiten generar estructuras de los archivos de jerarquía de clases la cual se explica más adelante

Una de las ventajas de utilizar los generadores de analizadores léxico y sintácticos, C#Flex / C#Cup, es la compatibilidad que entre estos existe, lo que facilitó de cierta manera el desarrollo de los analizadores léxico y sintáctico para el intérprete.

2.1.4 Definición De La Estructura De Datos

Si se desea tener una correcta comunicación con cada una de los componentes del intérprete, se es necesaria la generación de diferentes archivos de datos, que facilitaran el almacenamiento de la información relevante de cada una de las clases creadas. De igual manera, el intérprete genera un archivo temporal que es usado al momento de la creación de las sistemas para indicarle al usuario cuales sistemas fueron creados en dicho análisis.

La gramática definida para el analizador sintáctico está compuesta por reglas gramaticales que contienen terminales ("tokens" regresados por el analizador léxico) y no terminales. Los terminales pueden tener asociado un valor, como por ejemplo los terminales NAME, COMA, MAS, MENOS, NUMERIC entre muchos otros que están declarados en el anexo A, cuyo valor asociado puede ser el nombre de un atributo, clase o un sistema. De forma similar si desea asociar un valor a un no terminal, se debe declararlos en la especificación de la gramática de tipo integer si el valor asociado es un entero o tipo Object para cualquier tipo, sea string, integer o cualquier otro.

2.2 REPRESENTACIÓN DE LA JERARQUÍA DE SISTEMAS

2.2.1 Definición Del Sistema

Una clase, según el estándar ODMG, es definida como un grupo de objetos compartiendo el mismo conjunto de atributos, interrelaciones y métodos. La definición de sistema en este proyecto, amplía el concepto de clase ODMG, ya que abarca y supera la entidad individual implícita en la anterior definición.

En efecto, aquí un sistema es planteado más allá de una tabla con atributos, interrelaciones y métodos compartidos por todas sus tuplas. Aquí, un sistema (System) es un conjunto de clases relacionadas e inter-actuantes según un mismo método, el cual no es particular a cada clase sino que surge común a todas ellas.

Esta jerarquía está representada por el súpersistema, el sistema y el subsistema, definido desde el nivel superior al inferior respectivamente. Si se

muestra la partición desde el segundo nivel se obtiene tanto el aspecto estructural como el funcional para cada subdivisión del sistema.

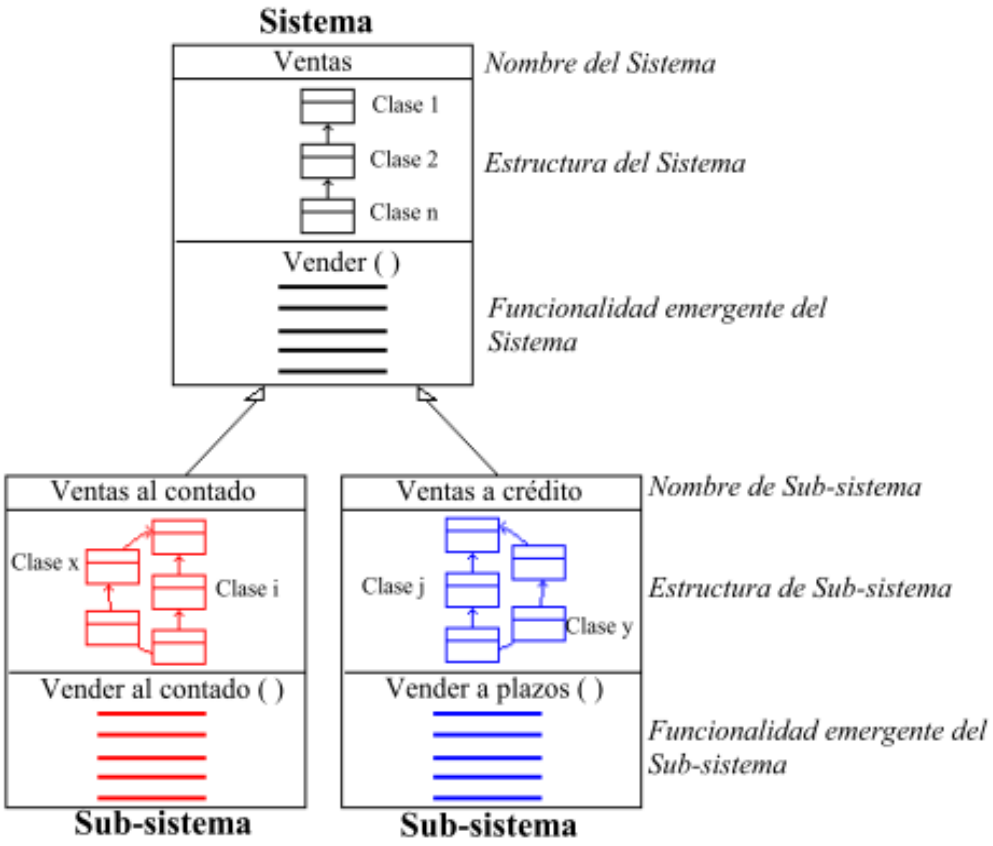

Figura 8. Jerarquía de Sistemas

2.2.2 Codificación (Esqueleto) En Latín De La Descripción De La Jerarquía De Sistemas

Para ilustrar esto se utilizara un ejemplo, el cual corresponde a una compra a plazos de insumos agropecuarios.

```
Super_System Universal
{
  {
```

```
  class Mayor
  {
    // Define sus atributos
       Id_cuenta integer not null primary key
      Nom_cuenta Varchar (100)
      Saldo integer not null
      Debito integer not null
      Credito integer not null
  // Mayor no tiene claves foráneas
  }
  class Diario
  {
  // Define sus atributos
     Id_cuenta integer not null foreign key
     Id_transaccion integer not null primary key
      Fecha date
      …
  }
// Define su método universal  "Realizar Transacción"
   Void Realizar_Transaccion (…)
}// Fin definición de Súper Sistema Universal

System Compras
{
 {
// Hereda de Universal sus clases
// Define además clase Mercancía propia de este sistema
   class Mayor
  {
     Id_cuenta integer not null primary key
    ….
  }
      class Mercancía
  {
// Define sus atributos…
      Id_cuenta integer not null foreign key
      Id_mercancia integer not null primary key
      Nom_mercancia Varchar (100)
      Existencias integer not null
      …
  }
```

```
    class Diario
  {
  // Define sus atributos
     Id_cuenta integer not null foreign key
      Id_transaccion integer not null primary key
       …
  }
 // Define clave foránea de Diario referida a Mercancía
// Hereda "Realizar Transacción" y completa  "Comprar"
Void insertar_entrada_mercancia_en_diario ()
Void sumar_a_existencia_en_mercancia ()
Void sumar_entrada_mercancia_en_mayor ()
}// Fin definición de sistema Compras

Sub_System CompraPlazos
// Define además clases propias de este subsistema
{
   {
class Mayor
  {
    // Define sus atributos
      Id_cuenta integer not null primary key
      Nom_cuenta Varchar (100)
      Saldo integer not null
      Debito integer not null
      Credito integer not null
  // Mayor no tiene claves foráneas
  }
  class Mercancía
  {
  //  Define sus atributos…
      Id_cuenta integer not null foreign key
      Id_mercancia integer not null primary key
      Nom_mercancia Varchar (100)
      Existencias integer not null
      …
  }
Class Proveedores
```

```
 {
  // Define los atributos
   Id_proveedor integer not null primary key
   Id_cuenta integer not null foreign key
   Nom_proveedor Varchar (100)
   Direccion Varchar (120)
   Telefono integer
   …
 }
  Class Facturas
 {
  // Define los atributos
     Id_factura integer not null primary key
     Id_proveedor integer not null foreign key
     Fecha Date
     …
  }
  Class Detalles
 {
  // Define los atributos
     Id_factura integer not null foreign key
     Nom_producto Varchar (120)
     Cant integer not null
     Valor_U integer not null
     …
 }
 class Diario
  {
  // Define sus atributos
     Id_cuenta integer not null foreign key
      Id_transaccion integer not null primary key
       …
  }
}
Void insertar_deuda_proveedor_en_diario ()
Void insertar_nueva_factura_proveedor ()
Void sumar_deuda_proveedor_en_mayor ()
Void incrementar_deuda_proveedor ()
Void insertar_entrada_mercancia_en_diario ()
Void sumar_a_existencia_en_mercancia ()
Void sumar_entrada_mercancia_en_mayor ()
}; // Fin definición de subsistema CompraPlazos
```

2.3 TECNOLOGÍA USADA EN EL PROYECTO

2.3.1 Aplicación De Escritorito

En una aplicación de escritorio normalmente no iniciamos sesión por cada aplicación que usemos, sólo se inicia sesión una vez cuando prendemos el sistema operativo, asumiendo que vamos a abrir una aplicación para ver nuestra lista de tareas:

1. El usuario carga la aplicación.
2. La aplicación (el código), se conecta a la base de datos y recupera la información del usuario.
3. La aplicación muestra al usuario la información solicitada.

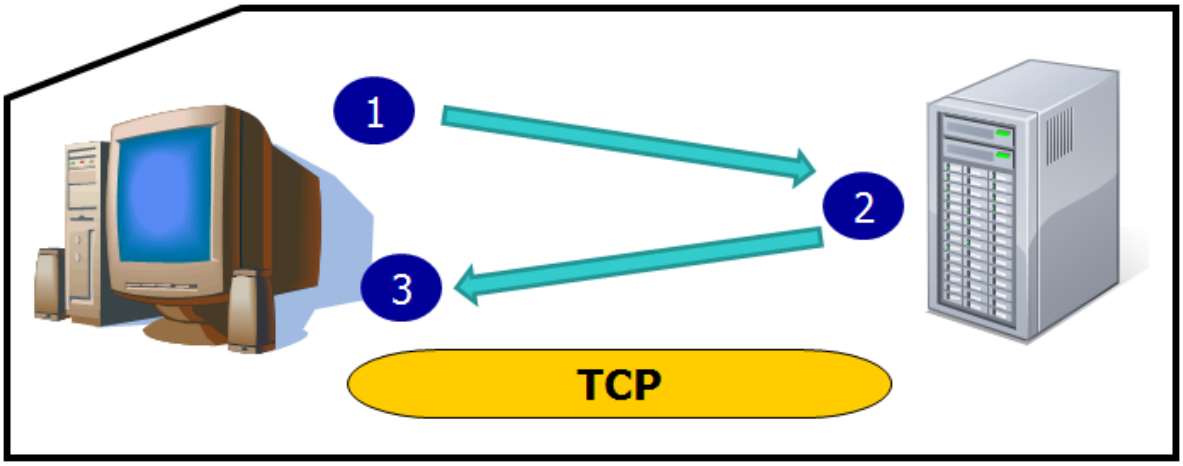

Figura 9. Aplicación de escritorio

2.3.2 Entorno De Desarrollo

Con la idea inicial de plasmar la tesis del Doctor Albarracín en un marco no puramente teórico, sino poder dejar plasmada la idea en un plano practico, y así poder deslumbrar los pros que ofrece lo postulado por el profesor, para ello se desarrolló este interprete.

Partiendo de esta idea, se fue un poco más allá, no solo contemplando la idea de una herramienta atrapada en un solo entorno de desarrollo, se pensó en crear versiones del mismo, en diferentes entornos de programación y tecnologías.

Se encargó realizar el proyecto en C# como una alternativa al ya realizado bajo el lenguaje de programación java. Llamado INTÉRPRETE LATIN PARA LA IMPLEMENTACIÓN DE MODELOS EMERGENTES DE DATOS elaborado por JULIÁN RICARDO CAMARGO GONZÁLEZ, YULIANA MAYERLY SOTO CARREÑO Trabajo de grado para optar al título de Ingeniero de Sistemas UIS.

.NET Framework 4[8]. Microsoft Visual C# es un lenguaje de programación diseñado para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. C# es simple, eficaz, con seguridad de tipos y orientado a objetos. Con sus diversas innovaciones, C# permite desarrollar aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C.

Visual Studio admite Visual C# con un editor de código completo, plantillas de proyecto, diseñadores, asistentes para código, un depurador eficaz y fácil de usar, además de otras herramientas. La biblioteca de clases .NET Framework ofrece acceso a una amplia gama de servicios de sistema operativo y a otras clases útiles y adecuadamente diseñadas que aceleran el ciclo de desarrollo de manera significativa.

Visual C#[9]. Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

2.3.3 Herramientas De Desarrollo

Para el análisis, diseño e implementación del intérprete, se utilizaron las siguientes herramientas software

SQLiteStudio. Es un administrador de base de datos SQLite con las excelentes características.

[8] *msdn.microsoft.com/en-us/.../w0x726c2.aspx*
[9] http://msdn.microsoft.com/es-es/library/kx37x362.aspx

C#Lex[10]. Es una versión en C# del popular constructor léxico de JLex. JLex es la contraparte de Java para generar analizadores léxicos, popular en los sistemas UNIX. Se convirtió JLex a un lenguaje C# .NET para así poder aprovechar todas sus características también la plataforma .NET

C#Lex construye automáticamente un analizador léxico (señalizador) a partir de una especificación predefinida, que descompone una secuencia de caracteres en tokens (símbolos).

C# Cup[11]. Es una versión de C# del popular CUP basado en Java. C # como una representante de la familia de las lenguas. NET se utiliza con mayor frecuencia en los negocios y para fines educativos. Es una herramienta para la construcción automática de analizadores sintácticos LR que son de tipo ascendente y que construyen el árbol sintáctico de las hojas hacia la raíz en este lenguaje será útil en el futuro. La diferencia entre c# y Java reside en las diferentes bibliotecas del sistemas dentro de los lenguajes específicos. Por lo tanto, no era muy difícil de traducir CUP existente de Java para la plataforma. NET. Esto hace posible cambiar la plataforma de desarrollo con un mínimo esfuerzo de adaptación en los archivos existentes de taza de especificación.

ReSharper[12]. Es una herramienta diseñada para el IDE de Microsofot Visual Studio.NET y brinda a los desarrolladores de esta plataforma un editor de C# mejorado. Por ejemplo, una mejor herramienta de resaltado de errores, asistencia de codificación avanzada con un asistente y completado de código más inteligente, navegación y búsqueda, pruebas unitarias, edición de aplicaciones ASP.NET, editor de scripts para la herramienta de construcción NAnt, etc.

2.4 METODOLOGÍA

2.4.1 Modelo En Cascada

Para realizar el intérprete LATIN se propone el modelo en cascada, debido a que se necesita un orden riguroso en el proceso para el desarrollo del

[10] www.seclab.tuwien.ac.at/projects/cuplex/lex.htm
[11] http://www.seclab.tuwien.ac.at/projects/cuplex/cup.htm
[12] http://www.jetbrains.com/resharper/

software, de tal forma que al iniciar cada fase, necesariamente deba esperar que finalice para continuar con el procedimiento de la siguiente etapa.

En la figura 10 se resumen las etapas de elaboración del proyecto, obteniendo como resultado el intérprete.

2.4.2 Razones Para La Elección De La Metodología

La elección de esta metodología se debe a las siguientes razones:

- Sabiendo la complejidad de la realización del intérprete, lo más conveniente era establecer un método sencillo que siguiera los pasos establecidos.

- Esta metodología permite avanzar en procedimientos posteriores, ya que las fases se establecen al principio, lo que facilito el proceso de desarrollo.

- Establece criterios de entrada y salida en cada fase claramente definidos**.**

- Este proyecto tiene una visión internacional y era necesario obtener un software de calidad y esta metodología lo propicia gracias a las ventajas y características propias de este modelo.

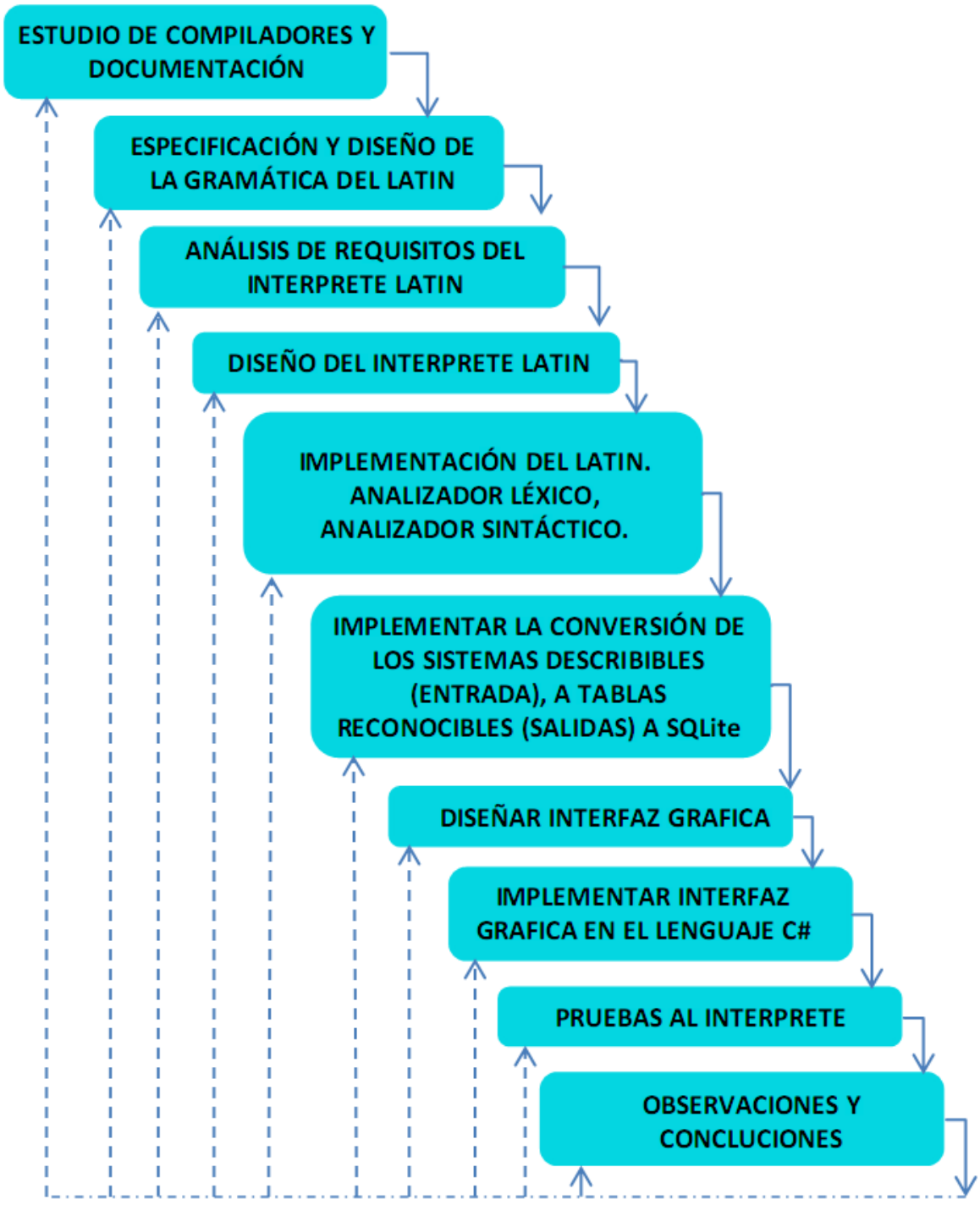

Figura 10. Principales etapas de la elaboración de LATIN

3. DESARROLLO DE LA ME METODOLOGÍA

3.1. ACTIVIDADES DESARROLLADAS

3.1.1 Estudio De Compiladores Y Documentación.

En el transcurso de la carrera, fueron realmente pocas las herramientas, lecturas, clases o cualquier cosa que pudiera darnos tan siquiera alguna pista de cómo realizar lo que para nosotros, era una tremenda tarea la realización de este intérprete.

Con nada más que unas clases vistas de la materia compiladores, tomada como electiva, y apostándole al nuevo conocimiento, se inició la capacitación, lectura y sobre todo la investigación, en todo lo que nos concerniera. Lecturas de lo planteado por el profesor Albarracín, familiarizarnos con el modelo, aprender el manejo de las nuevas herramientas, los pasos que sigue en el desarrollo de este tipo de aplicaciones.

3.1.2 Especificar Gramática Del Lenguaje Latín.

El profesor Jaime Albarracín facilitó una gramática que fue concebida, pensando ser implementada para un modelo de datos orientado a objetos. Se encargó inicialmente estudiar, analizar y verificar la posibilidad de migrar esta gramática a un modelo orientado a sistemas. También se solicitó que la parte estructural del modelo fuera administrada por un motor de bases de datos orientado a objetos, dicho motor fue el db4o.

Para implementar el motor de bases de datos db4o se vio la necesidad de hacer una investigación y un estudio de la gramática facilitada para que la parte funcional y la estructural del modelo pudieran tener una correcta interacción. Esta tarea fue imposible para los plazos y recursos que se disponen actualmente (esto será detallado posteriormente en la sesión 3.1.5).

Abandonando esta idea de poder concebir el modelo con un motor de bases de datos orientado a objetos, se tomó la decisión junto al director del proyecto de elegir la posibilidad de que la parte estructural del LATIN fuera manejada por un

motor de bases de datos relacional. Inicialmente con el primer motor se realizó una depuración y mejoramiento de la gramática, corrigiendo redundancia, suprimiendo palabras reservadas que no se estaban utilizando, implementando nuevos árboles para que se ajustaran a los nuevos requerimientos, que eran los de implementar sistemas, supersistemas y agregando nuevas palabras reservadas solicitadas para el modelo, tales palabras son addo (insert), muto (update), eligo (select), deleo (delete), ex (where).

3.1.2.1 Valores Literales

A continuación se describe la estructura del lenguaje implementado para los componentes.

- **Variables:** Una variable comienza con una letra y puede estar seguida de uno o más caracteres alfa – numéricos. También acepta el carácter "_" o guión de piso.

  Variable = [A–Za–z_] [A–Za–Z_0 - 9]*

  Ejemplo: Cantidad_1, _conexión, Compra_contado, compra_Credito.

- **Números:** Los enteros se representan como una secuencia de dígitos. Los de coma flotante utilizan "." Como separador decimal.

  Número = 0| [1-9][0-9]*| [0-9]+ "." [0-9]*| "."[0-9]*

  Ejemplo: 101, 0.5, 6, .65

- **Fecha:** El rango permitido es de '1000-01-01' a '9999-12-31'. El formato es "YYYY – MM- DD".

  Fecha= [1-9] [0-9] [0-9] [0-9] "-" [0-1] [0-9] "-" [0-3] [0-9]

  Ejemplo: 2012-01-04, 2000-04-27

- **Hora:** El rango permitido es de '00:00:00' a '23:59:59'. El formato es "HH:MM:SS".26

  Hora = [0-2] [0-9] ":" [0-5] [0-9] ":" [0-5] [0-9]

  Ejemplo: 17:15:10, 08:24:48

- *Valores Null:* cuando se encuentre este valor significa que “no existen datos”. Solo se debe escribir null en minúscula, no está permitido la combinación de mayúsculas y minúsculas.

3.1.2.2 Caracteres Permitidos

Estos son aquellos caracteres especiales que todo lenguaje de programación permite al momento de definir una instrucción, siendo los más comunes los siguientes: (, *,), ‘,<, +, /, {, &, >, -, (,}, =).

CARÁCTER	SIGNIFICADO
+	Signo mas
-	Signo menos
*	Multiplicación
/	División
;	Punto y coma
,	Coma
.	Punto
:	Dos puntos
‘	Comilla sencilla
>	Mayor que
<	Menor que
=	Igual que
(	Paréntesis izquierdo
)	Paréntesis derecho
}	Llave izquierda
}	Llave derecha
&	Ampersand (compuerta lógica Y)

Tabla 1. Caracteres permitidos

3.1.2.3 Palabras Reservadas

Son aquellas que son únicas y propias del lenguaje, no siendo posible definir una variable, constante, u objeto con el nombre de estas palabras reservadas. Se aceptan combinaciones en mayúscula y minúscula solo la primera letra de la palabra en mayúscula

PALABRAS RESERVADAS INTERPRETE LATIN			
avg	all	count	decimal
double	exists	deleo	is
min	max	numeric	primary
set	sum	muto	and
any	between	by	check
class	date	default	disctinct
drop	escape	float	char
byte	foreign	in	addo
Integer	into	key	like
method	not	null	or
precision	real	references	eligo
smallint	some	show	time
unique	values	varchar	ex
bool	extend	class	System
Subsystem	having	Supersystem	from

Tabla 2. Palabras reservadas

3.1.2.4 Sintaxis Iníciales

Solo aceptan un comando de entrada por vez y siempre deberá terminar en ";". La sintaxis de entrada puede ser para:

- Definición de súper sistema, clases.
- Definición de sistema, clases y métodos.
- Definición de subsistema, clases y métodos.
- Manipulación y ejecución de los métodos.

3.1.2.6 Definición De Sentencias SYDL

Estas son las que permiten la definición de jerarquías de clase y sus elementos estructurales.

- **Definición de Súper Sistema**

```
Supersystem nombre_supersistema  [cuerpo_supersistema];

    cuerpo_supersistema :
    { [clase1, clase 2,…clase n]
    }
```

En donde esta definición de súper sistema solo se puede crear una vez, entendiendo que esta creación es particular para cada empresa según las reglas del modelo del profesor Albarracín

• **Definición de Sistema**

```
System nombre_sistema [herencia] [cuerpo_sistema];
herencia            :  extend nombre_sistema_padre
cuerpo_sistema
                      { [clase1, clase 2,…clase n]
                      [definición_metodo]
                       }
```

En donde system *nombre_sistema* crea un nuevo sistema. Este sistema puede heredar las clases y el método de otro sistema con el comando *extend nombre_sistema_padre*. Los nombres *nombre_sistema* y *nombre_sistema_padre* pueden ser literales de tipo variable.

- **Definición de clases**

```
definicion_clase        :/* vacio */
                        |class nombre_clase
                        {
                        [listado_atributos]
                        }
                        [definición_clase]

listado_atributos       : /* vacio */
                        | [elemento_clase]
                        | [listado_atributos], [elemento_clase]

elemento_clase          : [def_columna]
                        | [def_interrelacion]

def_columna             :
                        [columna][tipo_dato][opc_def_columna]

tipo_dato               :  varchar (numero)
                        | numeric
                        | numeric (numero)
                        | numeric (numero, numero)
                        | decimal
                        | decimal (numero)
                        | decimal (numero, numero)
                        | integer
                        | smallint
                        | float
                        | float (numero)
                        | real
                        | char
                        | double precision
                        | date
                        | time
                        | byte
                        | bool

opc_def_columna         : /* vacio */
                        | [opc_def_columna][listado_opc_columna]
```

```
listado_opc_columna      :  not null
                         | not null unique
                         | not null primary key
                         | default numero
                         | default null
                         | default variable
                         | check ([condicion_busqueda])
                         | references nombre_class
                         | references
                          nombre_class ([listado_columna])

def_interrelacion        : unique ( [listado_columna] )
                         |primary key ([listado_columna])
                         |foreign key ([listado_columna])
                         references nombre_clase
                         |foreign key ([listado_columna])
                         references nombre_tabla
                         ([listado_columna])
                         |check ([condicion_busqueda])

listado_columna          : nombre_columna
                         | [listado_columna], nombre_columna


condicion_busqueda       : [condicion_busqueda]
                         or [condicion_busqueda]
                         | [condicion_busqueda]
                         and [condicion_busqueda]
                         |not [condicion_busqueda]
                         | ([condicion_busqueda])
                         | [predicado]

predicado                : [comparacion_predicado
                         | [like_predicado]
                         | [entre_predicado]
                         | [en_predicado]
                         | [prueba_para_nul
                         | [todo_o_algun_predicado]
                         | [prueba_existencia]
```

```
entre_predicado          : [escalar_exp] not
                         between [escalar_exp]
                         and [escalar_exp]
                         | [escalar_exp] between [escalar_exp]
                         and [escalar_exp]

like_predicado           : [escalar_exp] not
                         like [atom] [opt_escape]
                         | [escalar_exp] like
                         [atom] [opt_escape]

opt_escape               : /* vacio */
                         | escape [atom]

prueba_para_null         : [column_ref] is not null
                         | [column_ref]  is null

en_predicado             : [escalar_exp] not in (subquery)
                         | [escalar_exp] in (subquery)
                         | [escalar_exp] not
                         in ([atom_commalist])
                         | [escalar_exp] in ([atom_commalist])

atom_commalist           : [atom]
                         | [atom_commalist], [atom]

todo_o_algun_predicado   : [escalar_exp] [comparacion]
                         [any_all_some] subquery

any_all_some             : any
                         | all
                         | some

prueba_existencia        : exists [subquery]
                         subquery : (ELIGO [opt_all_distinct]
                         [selection] [clase_exp])

subquery          :  (eligo[opt_all_distinct][selection][table_exp])
```

```
selection                  : [escalar_exp_commalist]
                           |*

table_exp                 :[from_clause][opt_ex_clause][opt_group_by_clause]
                          [opt_having_clause]


from_clause               :from  [table_ref_commalist]


opt_all_distinct          :/*vacio*/
                          | all
                          |distinct


atom                      : parameter_ref
                          | numero
                          | &variable
                          | ‘ nombre ‘
                          | ‘ literal_hora ‘
                          | ‘ literal_fecha ‘
                          | &numero
                          | ‘ ‘


comparación               :=
                          |<>
                          |<
                          |>
                          |<=
                          |>=


escalar_exp               : [escalar_exp] + [escalar_exp]
                          | [escalar_exp] - [escalar_exp]
                          | [escalar_exp] * [escalar_exp]
                          | [escalar_exp] / [escalar_exp]
                          | + [escalar_exp]
                          | - [escalar_exp]
                          | [atom]
                          | column_ref
                          | [function_ref]
                          | ([escalar_exp])


function_ref              : [ammsc] ( * )
                          : [ammsc] (distinct column_ref)
```

```
                              : [ammsc] (all [escalar_exp])
                              : [ammsc] ([escalar_exp])

ammsc                         : avg
                              |min
                              |max
                              |sum
                              |count
```

Como se puede ver en el listado de atributos (listado_atributos) de la gramática, esta mantiene la sintaxis SQL normal al momento de crear los atributos de las clases.

Si una clase ya ha sido implantada en otro sistema no se debe colocar los atributos. En su caso se debe escribir:

class *nombre_class* **{}**

Si es preciso agregar nuevos atributos, se colocaran a la clase generada en el padre. Se debe de prestar atención de no, añadir nuevas propiedades con nombres iguales a los que ya se han definido anteriormente. Si este es el caso, el analizador tornara un error indicando que ya existe el campo que se intenta crear. Para el caso de la herencia de clases hay que tener cuidado que si la clase ya existe pero no se declaró en la clase padre el sistema intentara crear dicha clase pero como ella ya existe el sistema retornara un error.

3.1.2.7 Definición De Sentencias OQL

- **Consultar clases**

```
consulta_clases         : [eligo_statement]
                        | [deleo_statement_searched]
                        | [addo_statement]
                        | [muto_statement_searched]
                        | [drop_statement]

eligo_statement         : eligo [opt_all_distinct]
                        | [selection] nombre_class
```

```
deleo_statement_searched :deleo from opt_ex_clause

addo_statement          : addo into nombre_clase [opc_column_commalist]
                         [value_or_query_spec]

muto_statement_searched  : muto nombre set assignment_commalist opt_ex_clause


drop_statement          : drop class [parametro_lista]

parametro_lista       : [addo_parametro]
                         |[parametro_lista] ,[ addo_parametro]

addo_parametro          :/*vacio*/
                         |nombre
                         |null

opt_ex_clause           :/*vacio*/
                         |[ex_clause]

ex_clause               :ex [condicion_busqueda]

opc_column_commalist   :/*vacio*/
                         |(listado_columna)

value_or_query_spec     : values ( addo_atom_commalist)
                         | [query_spec]

query_spec       :eligo [opt_all_distinct][selection]nombre[table_exp]


table_exp        :[from_clause][opt_ex_clause][opt_group_by_clause]
                 [opt_having_clause]

from_clause            :from  [table_ref_commalist]

table_ref_commalist    : [table_ref]
                          |[table_ref_commalist], [table_ref]

table_ref              : nombre_tabla


opt_group_by_clause    :/*vacio*/
                         |group by [column_ref_commalist]

opt_having_clause       :/*vacio*/
                         |having [condicion_busqueda]
```

```
column_ref_commalist  : [column_ref]
                      | [column_ref_commalist] , [column_ref]

column_ref           : table
                      |nombre1. nombre2 . nombre3

table                : nombre
                      |nombre . nombre


escalar_exp_commalist : [escalar_exp]
                      |[escalar_exp_commalist,] [escalar_exp]


assignment_commalist  : [assignment]
                      |[assignment_commalist,] [assignment]

assignment            :nombre COMPARACION [escalar_exp]
                      |nombre COMPARACION null
                      |nombre COMPARACION [subquery] [escalar_exp]
```

3.1.3 Análisis De Requisitos Y Diseño Del Interprete LATIN

3.1.3.1 Requisitos Funcionales

- Análisis léxico, en donde el intérprete hace un exploración de la gramática clasificándola en tokens reconocibles para el analizador sintáctico.
- Análisis sintáctico, en donde se agrupen los componentes léxicos del código fuente en frases gramaticales que el intérprete utiliza para sintetizar la salida.
- Un Control de errores aproximado de la gramática declarada por el usuario que impida la ejecución incorrecta del intérprete y dé una señal cercana de la ubicación del error.
- Una interfaz gráfica en donde se ilustre la capacidad y el potencial del modelo de datos reorientado a sistemas en una organización. Tomando como ejemplo dos transacciones cotidianas, realizadas por la mayoría de las industrias, como lo son compra a contado y compra a crédito.
- En esta interfaz se debe realizar las validaciones más comunes en cada campo de la factura.

3.1.3.2 Requisitos No Funcionales

- Interfaz amigable, donde el usuario pueda interactuar y explorar, en un marco práctico el modelo planteado. Siendo este entorno lo más sencillo de manipular, tratando de asemejar otros editores de código que un usuario intermedio haya utilizado
- El intérprete debe de usar un motor de bases de datos relacional que manipule la parte estructural del LATIN.
- Un software lo más portable posible (su tamaño y funcionalidad), que no se encuentre sujeto a aspectos tecnológicos, como lo puede ser el poco espacio disponible en disco duro, o la ausencia de otras herramientas informáticas que impidan la ejecución correcta del programa.

3.1.3.3 Elaboración De Los Casos De Usos

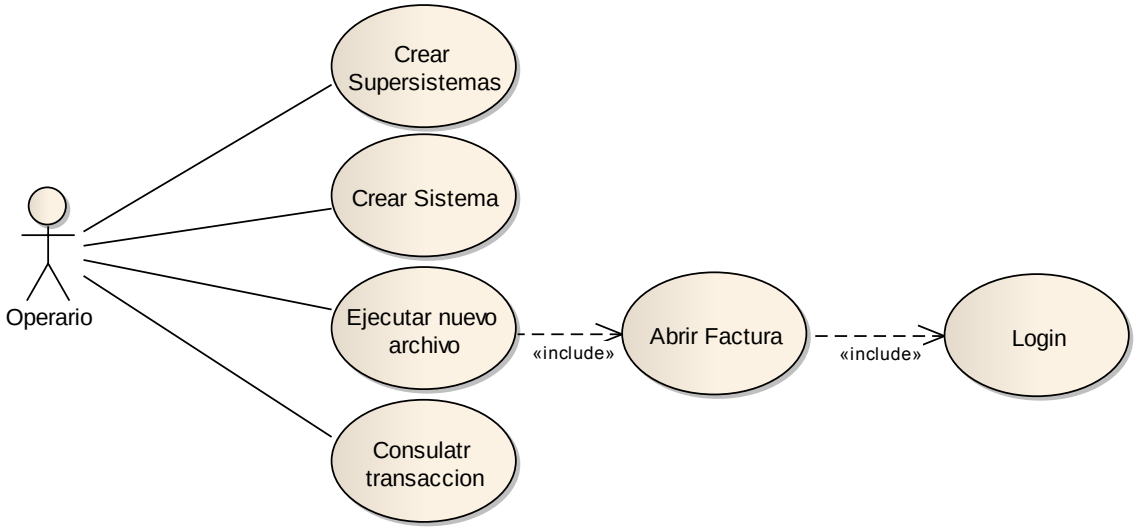

Figura 11. Diagrama caso de uso de la interfaz del intérprete

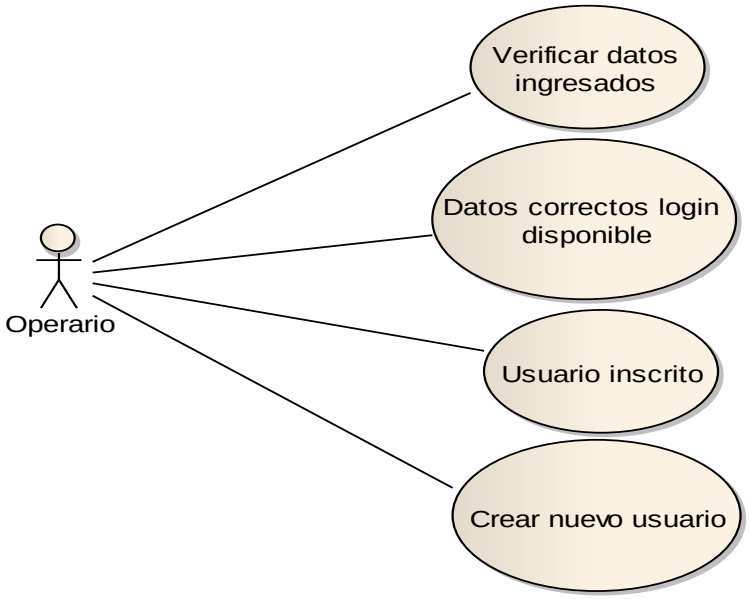

Figura 12. Diagrama caso de uso de login

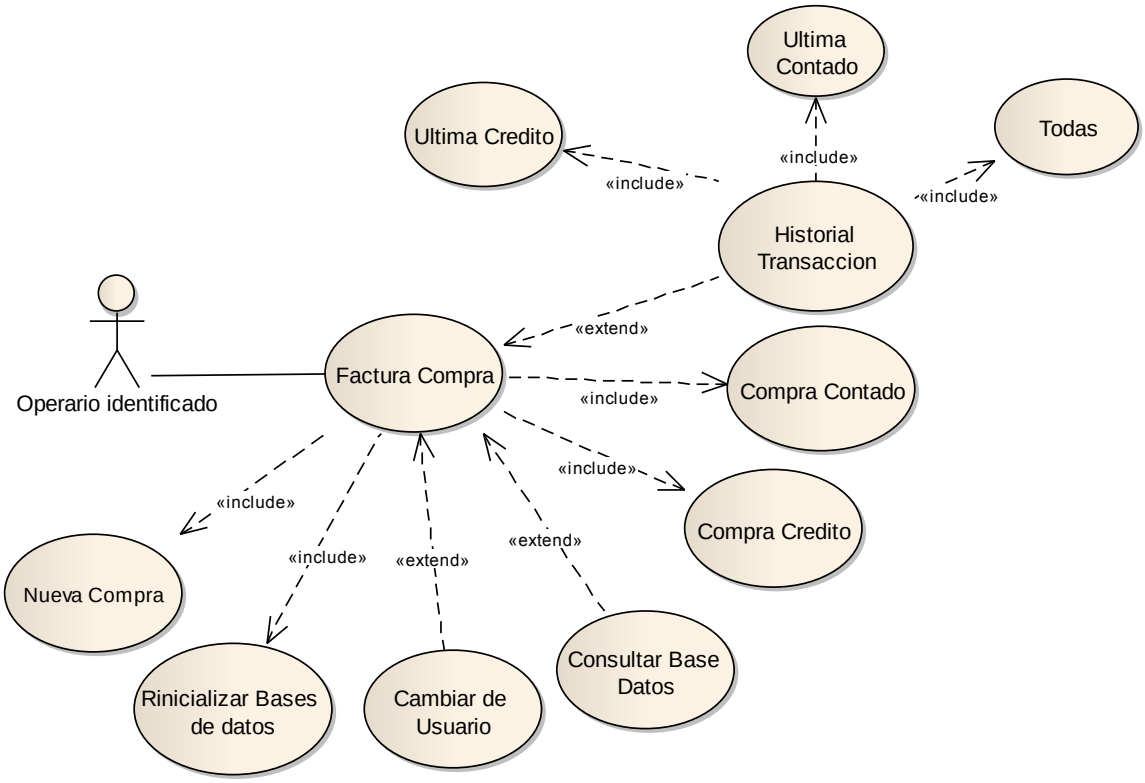

Figura 13. Diagrama caso de uso de la interfaz de Compra Mercancía

3.1.3.4 Diagrama De Clases.

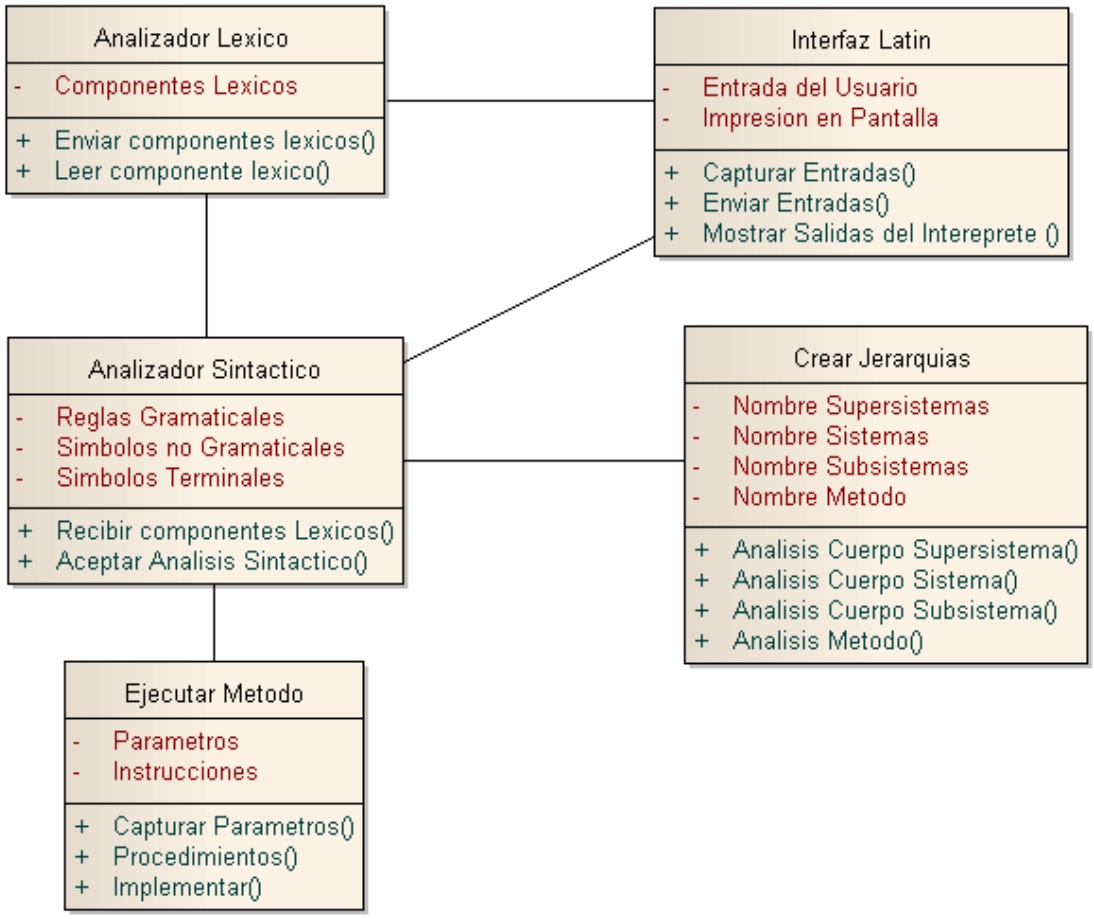

Figura 14. Diagrama de Clases

3.1.4 Implementar Analizadores Léxico Y Sintáctico

3.1.4.1 Analizador Léxico

La construcción de este analizador se realizo mediante un generador de analizadores léxicos. En este caso, se utiliza un programa especial que tiene como entrada pares de la forma (expresión regular, acción). El generador crea todos los autómatas finitos, los convierte a autómata finito determinista, y lo

implementa en C#. El programa C# así generado se compila y se genera un ejecutable que es el analizador léxico de nuestro lenguaje LATIN. El generador utilizado fue C#Lex.exe para entorno MS-DOS.

A continuación se muestra los pasos realizados en el entorno MS-DOS para obtener el analizador léxico del intérprete LATIN.

```
C#Lex.exe lexico.lex
```

En donde C#Lex.exe es el ejecutable para generar el analizador léxico, y el Lexico.lex contiene los caracteres reconocibles para la gramática.

La siguiente es la salida del generador C#Lex.exe

```
D:\Jeisson\proyecto\ultimo>C#Lex.exe lexico.lex
Processing first section -- user code.
Processing second section -- JLex declarations.
Processing third section -- lexical rules.
Creating NFA machine representation.
NFA comprised of 729 states.
Working on character classes
.....................................................................
.............
NFA has 56 distinct character classes.
Creating DFA transition table.
Working on DFA states
.....................................................................
.............
Minimizing DFA transition table.
383 states after removal of redundant states.
Outputting lexical analyzer code.
```

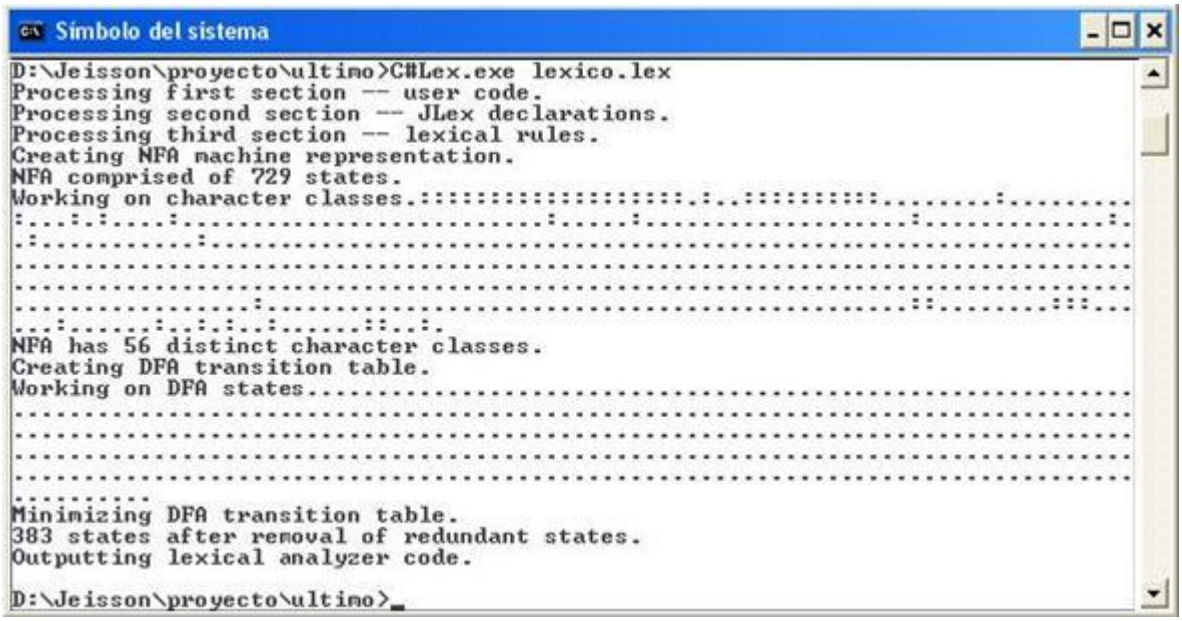

Figura 15. Creación del analizador léxico con C#Lex.exe

Este proceso genero la siguiente clase lexico.lex.cs, en la carpeta en donde se encuentra el ejecutable.

3.1.4 2 Analizador Sintáctico

Se usó el generador de analizadores sintáctico C#Cup.exe, el cual recibe como entrada fundamentalmente una gramática independiente del contexto, las reglas gramaticales, los símbolos terminales y no terminales, así como también algunos métodos definidos por el usuario para ser implementados. Para más información sobre esta gramática ver el Anexo A.

Una vez realizado este archivo necesario para C#Cup se procede a generar usando la siguiente línea de comandos desde la consola de Windows (MS-DOS):

```
> C#Cup.exe sintactico.cup
```

En donde C#Cup.exe es el ejecutable para generar el analizador sintáctico, y el sintactico.cup contiene la gramática del LATIN.

La siguiente es la salida del generador C#Cup.exe

```
------- C# CUP v0.1 Parser Generation Summary -------
```

```
  0 errors and 34 warnings
  114 terminals, 83 non-terminals, and 200 productions declared,
  producing 386 unique parse states.
  34 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
  Code written to "parser.cs", and "sym.cs".
----------------------------------------------------- (v0.1)
```

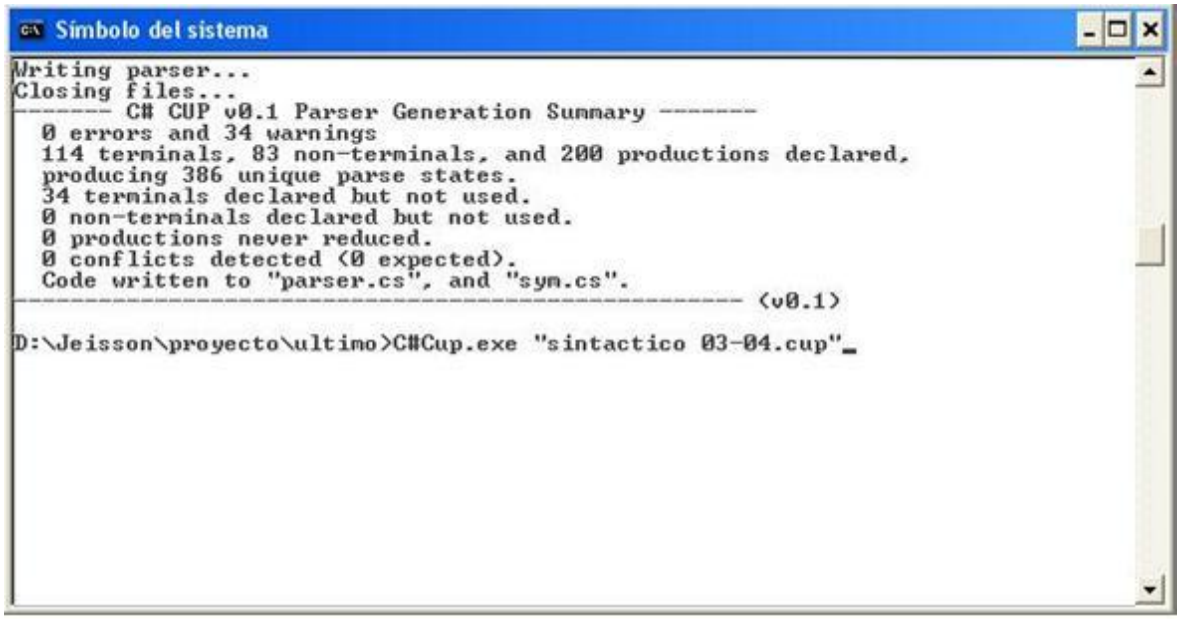

Figura 16. Creación del analizador sintáctico con C#Cup.exe

Esta operación genera dos archivos uno de ellos es parser.cs que contiene la clasificación de las reglas de producción de la gramática y el otro es sym.cs es la clase que contiene los símbolos terminales del analizador sintáctico.

3.1.5 Implementar La Conversión De Los Sistemas Descritos, A Tablas Reconocibles A Sqlite.

No queriendo desconocer el trabajo hecho, ni meter en saco roto el tiempo y el esfuerzo realizado; pero más que mostrar e informar cual fue el trabajo efectuado, se quiere dejar una base para futuras investigaciones, que puedan tomar como fundamento lo acá realizado y seguir avanzando en el mejoramiento, profundización y perfeccionamiento del modelo.

Figura 17. Proceso del Latín

Una de las principales características del intérprete, es que después de la etapa del análisis léxico y sintáctico, se produce una separación de lo funcional (métodos) con lo estructural (clases o tablas dependiendo del lenguaje de consulta escogido). Se concibió el proyecto pensando en que esta parte estructural fuese manipula con el motor de bases de datos orientado a objetos DB4O. En la figura 17 se muestra cual fue la lógica inicial del proyecto.

En las etapas iníciales del trabajo, lo que en el plan de desarrollo de este libro hemos venido tratando de explicar, no nos encontramos con problemas que nos pusiera en riego la continuación del proyecto, surcamos por estas etapas con satisfacción y animo por encontrarnos con lo que seguía. Cuando llegamos a esta etapa (3.1.5) fue en donde comenzaron los problemas, nuestra gramática previamente había sido debidamente adaptada, para que el intérprete pudiera interactuar correctamente con db40, pero en el momento de hacer algunas pruebas de funcionamiento, que consistían en comprobar si el intérprete estaba reconociendo y realizando las tareas que se suponía que debía hacer, que era realizar algunos conversiones, y crear algunos archivos, entre ellos era el de implantar las clases que el usuario estaba definiendo en su sistema pero los objetos creados a partir de estas clases no eran reconocidos por db4o.

Aclarando que toda lo que corresponde con la creación y cualquier tipo de acción que realizara el LATIN, siempre será en tiempo de ejecución, puesto que no hay una estructura fija o predeterminada, con la cual poder trabajar, sino que es el usuario final quien va a dar la estructura que desea para su sistema. Entendiendo que la naturaleza de las empresas puede tener una variedad impórtate en sus funciones. Entonces debido a esto el intérprete tenía que tomar la estructura del sistema dado (sus clases que varían según las empresas) y crear dichos archivos, todo esto mientras se está ejecutando. Fue precisamente en esto, en donde encontramos fallas importantes en cuanto a la manipulación de dichas clases creadas en tiempo de ejecución porque no se podían guardar en un motor de base de datos orientado a objetos.

En términos más coloquiales, ¿qué era lo que se podía hacer con db4o y el Latin? Simplemente nada. Después de una etapa de investigación y consulta, pudimos llegar a crear clases en tiempo de ejecución, pero clases creadas de esta manera, no pertenecen al proyecto (pre compilado), porque estos archivos son creados en una etapa de pos compilación del proyecto, estas clases las intentamos crear con un espacio de nombre de C# que es: System.CodeDom.Compiler. Que contiene tipos para administrar la generación y compilación de código fuente en los lenguajes de programación compatibles.

Luego con estos archivos .cs generados pero no compilados buscamos la manera de manipularlos, que era usar sus métodos y constructores, para esto se utilizó el patrón Reflexión que proporciona objetos (de tipo Type) que encapsulan ensamblados, módulos y tipos. Se puede utilizar la reflexión para crear dinámicamente una instancia de un tipo, enlazar el tipo a un objeto existente u obtener el tipo a partir de un objeto existente, e invocar sus métodos o tener acceso a sus campos y propiedades. Si utiliza atributos en el código, la reflexión le permite tener acceso a ellos.

Con esta librería y con este patrón pudimos generar una clase externa y manipular sus métodos. En el anexo C se encuentra un ejemplo bastante sencillo de esto.

La respuesta a porque db4o y el LATIN no pudieron interactuar correctamente fue porque los objetos creados con la ayuda de Reflexión no pudieron ser manipulados correctamente por el motor. Debido a que db4o necesita objetos generados de manera convencional y no objetos creados dinámicamente (pos compilado y almacenado en memoria). Al enviarle una orden para que db4o manipule estos objetos simplemente no realiza ninguna operación.

Después de este tropiezo en el cual se perdió mucho tiempo y recursos pero que se ganó conocimiento, se tomó la decisión junto con el que era en ese entonces el codirector y que ahora es nuestro director, de abandonar la idea de que el LATIN, en su parte estructural fuera manipulada por db4o y empezar a buscar otro tipo de motor. Se nos dio la libertar de buscar una herramienta que manipulara la parte estructural del intérprete.

Anteriormente se había pensado en que el intérprete pudiera descomponer la parte estructural de sistemas a objetos. Pero como ya se explicó anteriormente la dificultad que se encontró con esto, se vio la necesidad de migrar de un motor orientado a objetos a uno relacional.

Uno de los alcances que se quería tener con el intérprete era poder difundir ampliamente a través de páginas web o cualquier otro medio. Viendo esta necesidad y después de un estudio concluimos que se requería un motor de bases de datos relacional que fuera liviano pero que también fuera robusto. Entre las opciones que había la mejor que a nosotros junto con el director nos pareció, fue SQLite ya que entre sus características se encuentra: cero configuración por parte de un administrador, implementa la mayoría del estándar SQL92, su pequeño tamaño (menos de 350KB completamente configurado o menos de 200KB con características opcionales omitidos.), su implementación es de libre uso y genera un único archivo donde se encuentra toda la base de datos.

Luego se inició la etapa de prueba donde se realizaron pequeños ejemplos con este motor para comprobar si SQLite podría acoplarse perfectamente al LATIN. Este proceso nos arrojó resultados satisfactorios y nos dio vía libre para continuar con el desarrollo del intérprete, con el nuevo motor. Se vio la necesidad de regresar a revisar la gramática para eliminar el código que era requerido por db4o e incrustar el nuevo código para el uso de SQLite.

Ahora pasamos de convertir sistemas - objetos a sistemas – tablas por el nuevo motor, pero antes no debemos olvidar que para llegar a este paso se debe cumplir completamente con la lógica del LATIN.

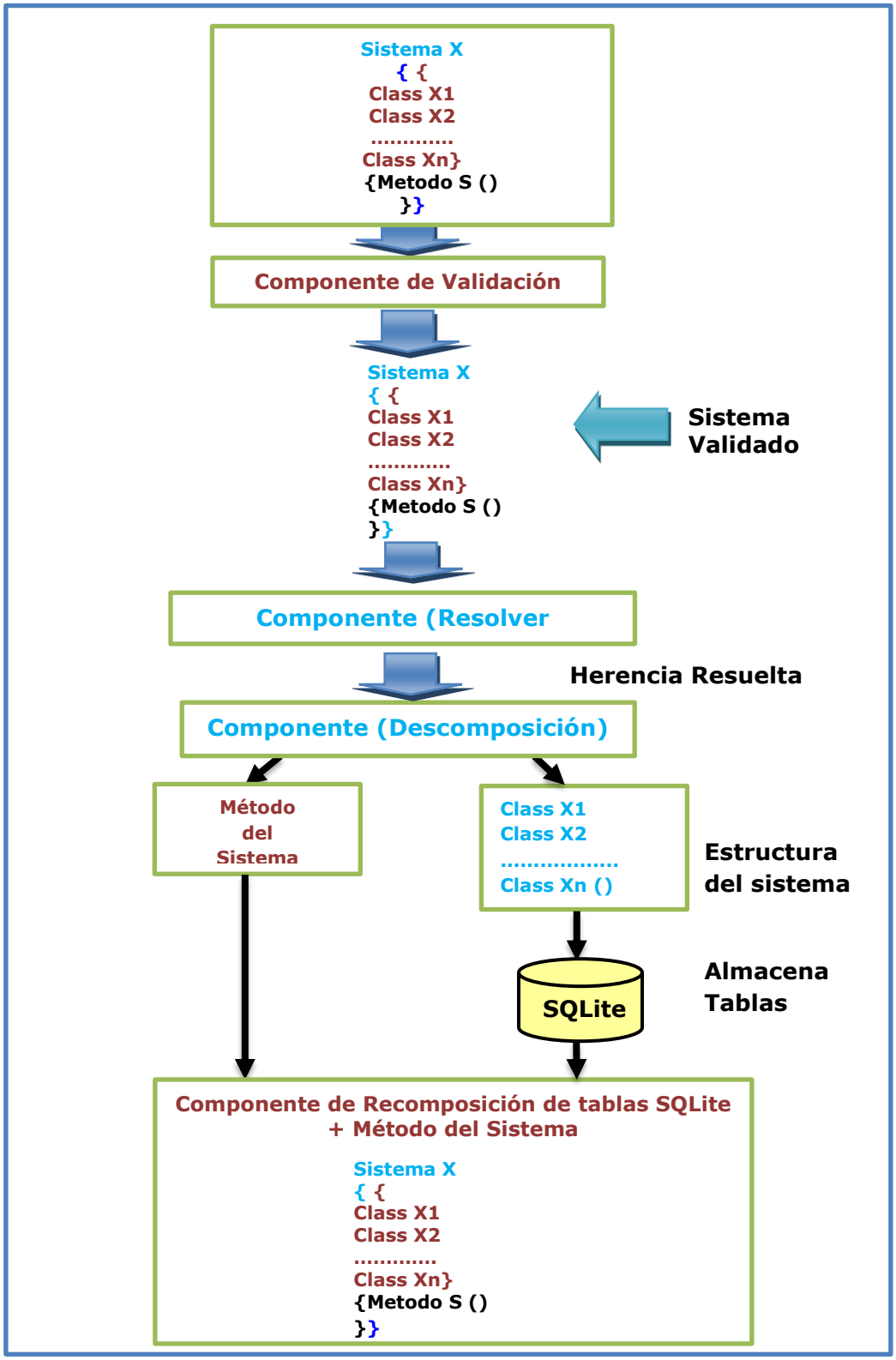

Figura 18. Lógica de Traducción de Sistemas

1. *Componente de validación*

En este primer paso el LATIN, reconoce y valida por medio de su gramática, la definición del sistema, el cual está compuesto de clases y de un método en común a esas clases. Dicha validación consistirá en reconocer la definición de los atributos junto con sus tipos de dato. También deberá reconocer el cumplimiento de las reglas de integridad referentes a claves primarias y foráneas, y demás restricciones inherentes al LATIN. Y de la misma manera debe validar la pertenencia del método a ese sistema.

2. *Componente para resolver herencia*

En este segundo paso el LATIN resuelve, el problema de la herencia que se presenta cuando se define un sistema dentro de una jerarquía de sistemas (Subsistemas, sistemas, Supersistema).

3. *Componente de descomposición*

Aquí separa o descompone las clases del método común, es decir, las clases quedan como estructuras separadas del método común. Esta separación se hace para construir una entrada apropiada, para los motores de Bases de Datos Relacionales y almacenando apropiadamente los métodos comunes de estas tablas.

4. *Componente de reagrupación*

El siguiente paso es reconstruir nuevamente cada una de las clases del sistema con su respectiva porción método. Es decir en este momento cada clase tendrá su propia funcionalidad, aunque tales clases no pueden ser independientes entre sí, puesto que deben ejecutarse todas en simultaneidad gracias a la ejecución del método común.

3.1.6 Diseñar E Implementar Interfaz Grafica

Buscando un estilo y un diseño tradicional, se ha creado una interfaz gráfica de usuario muy sencilla e intuitiva y sobre todo muy fácil de usar, bastante parecidas a los editores de textos más populares, WordPad y Bloc de notas.

La interfaz del intérprete de los componentes de análisis léxico-sintáctico y la ejecución de las instrucciones, es básicamente un editor de texto que cuenta con las características necesarias para hacer definiciones y consultas sobre bases de datos bajo el nuevo paradigma manejado, “Reorientado a

Sistemas". Permitiendo visualizar de manera sencilla y práctica el propósito del presente proyecto.

La interfaz sintetiza toda la parte del proyecto que debe ser tangible para el usuario final. El propósito de la creación de ésta, es el de proveer una interacción más amigable entre la persona y el ordenador, mejorar la facilidad de uso del sistema generador de bases de datos y por supuesto, superar la típica interfaz de línea de comandos. Siguiendo esta línea de ideas de presentar algo mucho más "palpable" para el usuario, se implementa a la par un sistema ejemplo, donde se podrá interactuar, de una manera más familiar, con el intérprete y así poder visualizar los alcances que posee este nuevo paradigma.

Durante toda la fase de creación de la interfaz gráfica, el diseño se aferró a ser totalmente consistente y coherente, usando y basándonos en el paradigma WIMP (ventanas, iconos, menús y dispositivos de interfaz humano), que está actualmente vigente en la mayor parte del mercado computacional. Por tanto, la aplicación permite el reconocimiento intuitivo a través de signos, normalmente familiares al usuario, facilita la manipulación de los símbolos de modo que su representación da orientación sobre qué tipo de objeto es y qué tipo de acciones podemos realizar con él, posibilita establecer relaciones lógicas entre datos que de otra forma serían complicadas de expresar, comprender y ejecutar.

3.1.6.1 Presentación De La Interfaz

A continuación se presenta la vista general de la interfaz, lista para ser utilizada, donde se encuentra cada uno de los componentes que integran el software. (El funcionamiento de los botones será explicado con detalle en el capítulo 5 de este libro).

Se presenta una imagen en la que se ve el diseño global de la interfaz y resta expresar que solo se hizo un enfoque muy general sobre los atributos y operaciones de cada clase implementada en la aplicación.

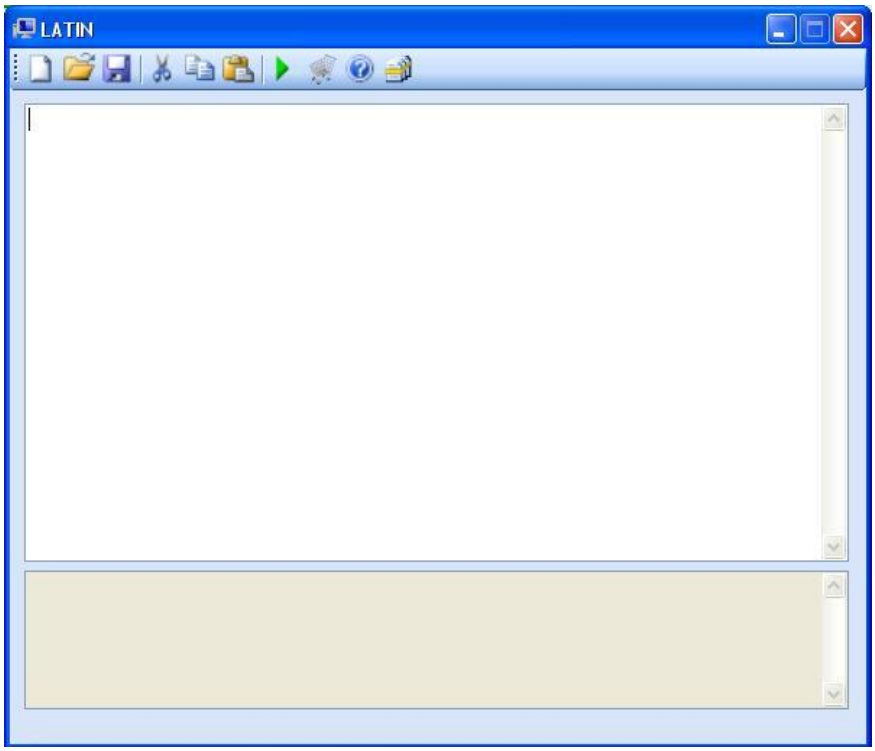

Figura 19. Interfaz gráfica del intérprete del componente de análisis

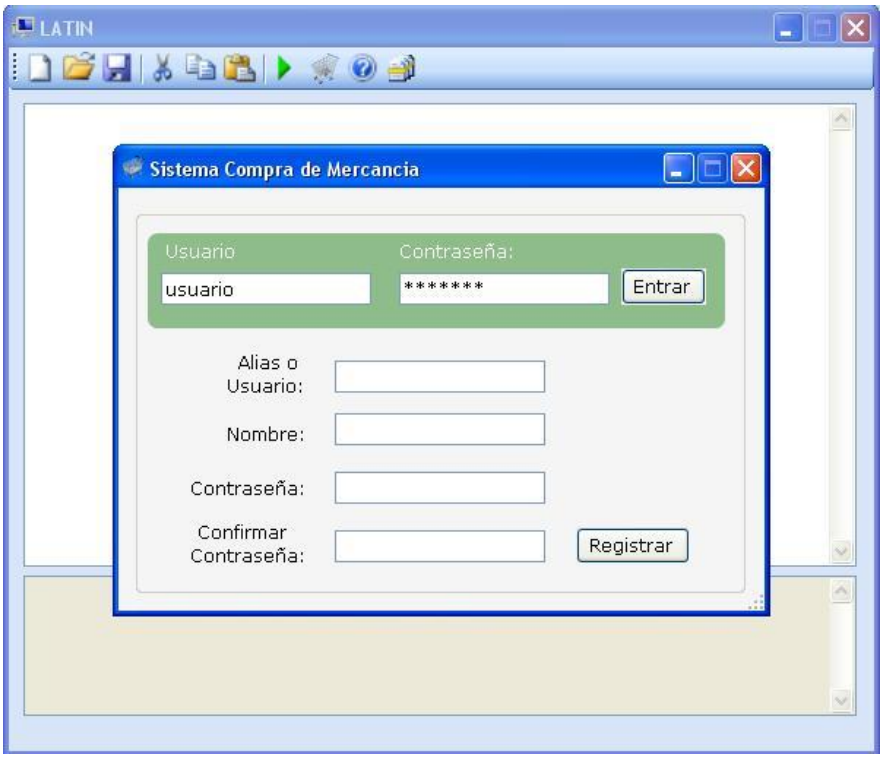

Figura 20. Interfaz gráfica del componente del login de compra

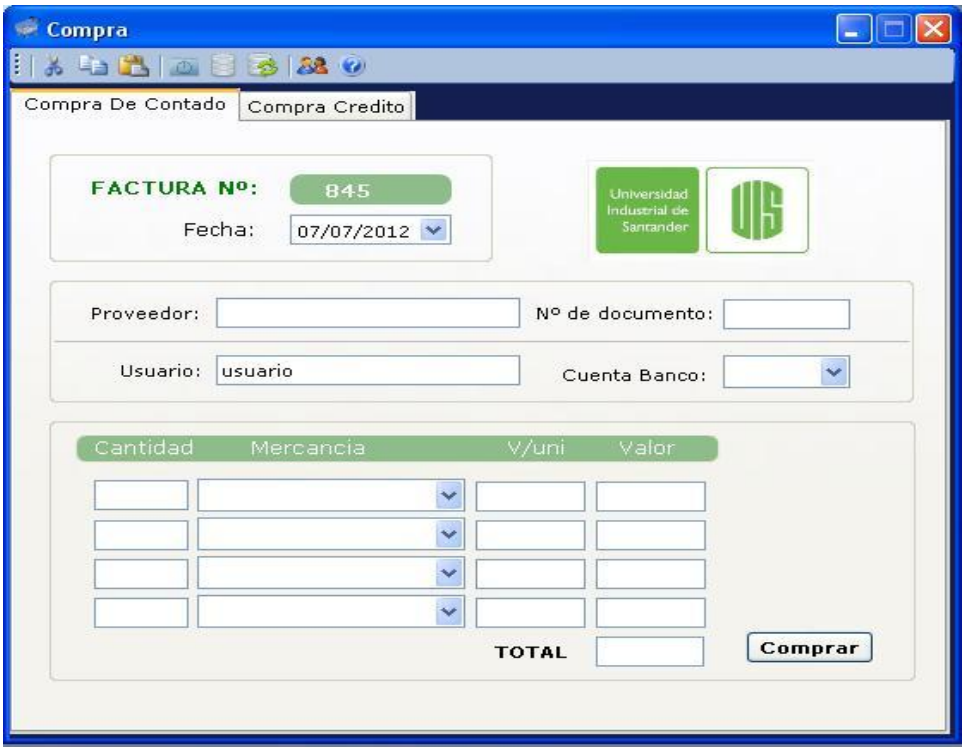

Figura 21.Interfaz gráfica del componente compra contado

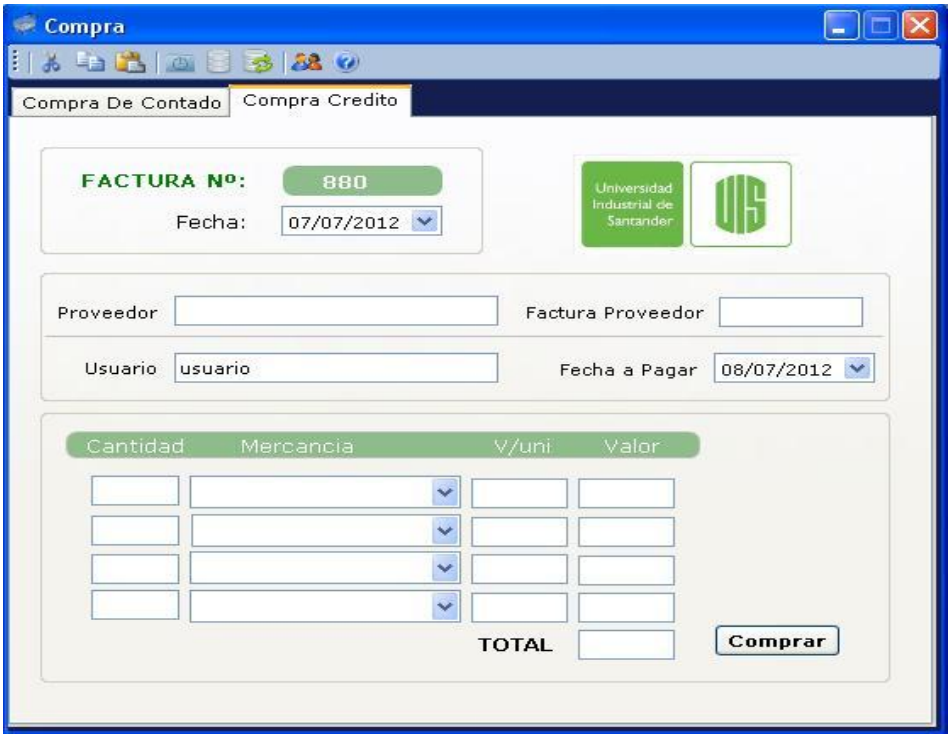

Figura 22. Interfaz gráfica del componente compra crédito

3.1.7 Pruebas Al Intérprete Latin

Durante el desarrollo de cada uno de los componentes del intérprete para el Latin se aplicaron las pruebas respectivas según las necesidades del mismo.

3.1.7.1 Prueba Analizador Léxico Y Sintáctico

Una vez obtenidos los analizadores léxico y sintáctico gracias a la ayuda de los de los generadores C#Lex y C#Cup respectivamente ellos crean las siguientes clases lexico.lex.cs,parser.cs y sym.cs. que son los analizadores léxico y sintáctico. Se procede a compilar y ejecutar todo el proceso analizador.

Para implementar los analizadores léxico y sintáctico se realizar, primero la clase léxico.lex.cs debe recibir las instrucciones(es el lenguaje latin) a analizar y segundo la clase parser debe recibir los tokes clasificados por el léxico.lex.cs a continuación se muestra el código en c# para implementar estos dos analizadores:

```
…
parser ThisParser;
Yylex ThisScanner;
ThisScanner = new Yylex(streamReader);
ThisParser = new parser(ThisScanner);
ThisParser.parse();
…
```

Para esta prueba primero, se le proporcionó al analizador una gramática correcta con palabras reservadas erróneas, luego una gramática errónea con palabras reservadas correctas.

A continuación se implementó un *supersistema* universal con una gramática correcta pero con un error en el uso de la palabra reservada *class* que en cuyo lugar fue clase. Allí podemos observar que nos arroja una aproximación de en qué línea puede estar el error.

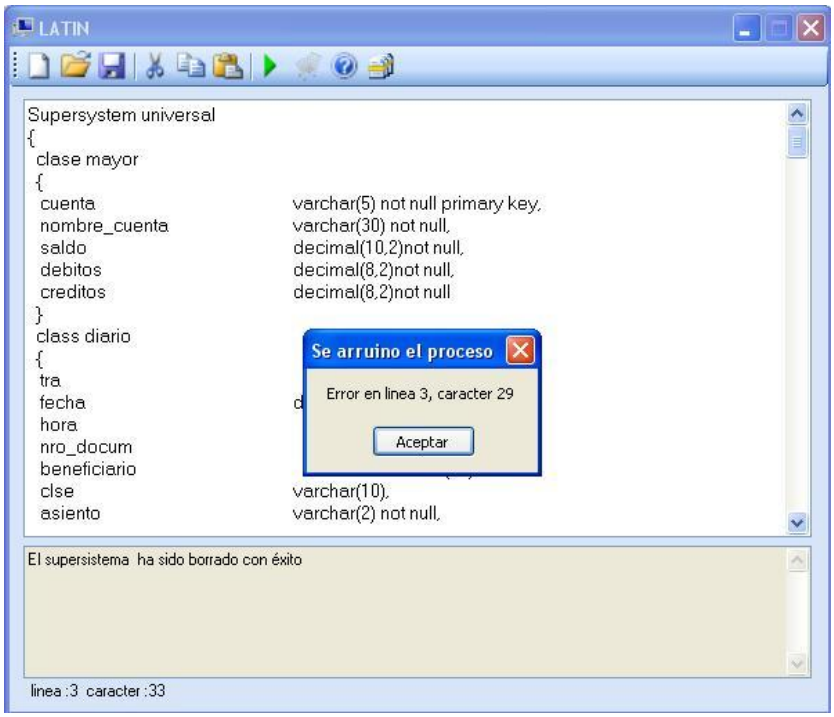

Figura 23. Pruebas, Error léxico

A continuación se implementó un *supersistema* universal con una gramática errónea y con un uso de las palabras reservadas correcto. Allí podemos observar que nos arroja una aproximación de en qué línea puede estar el error.

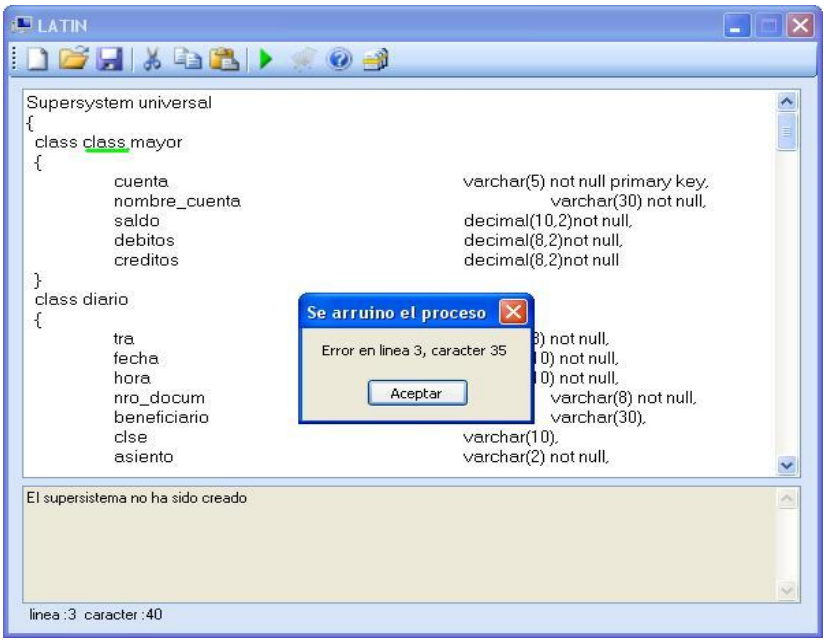

Figura 24. Pruebas, Error sintáctico

3.1.7.5 Prueba De La Interfaz De Usuario.

Las pruebas realizadas a la interfaz de usuario se desarrollaron en conjunto a las pruebas realizadas anteriormente, esto debido a la constante interacción con la misma al momento de realizar las pruebas. Permitiendo así un chequeo constante de los elementos desarrollados para la interfaz.

Se aplicaron pruebas de ejecución con cada uno de los botones de la barra de herramientas. Así como también de captura de información desde el área de texto principal.

En cada prueba desarrollada sobre la interfaz se lograba mejorar la experiencia del usuario al momento de interactuar con el intérprete mostrando en cada mejora mayor intuición en su presentación.

4. TRANSACCIONES EJEMPLO

4.1 COMPRA AL CONTADO

Supóngase una instancia de la transacción “compra al contado, nacional, de mercancía para la venta” la cual podría ser la transacción de código 1, que evolucionaría a través de tres macros: Por Ventas (actualizando el auxiliar de mercancías), por Tesorería (actualizando el auxiliar de bancos) y por Contabilidad (actualizando los libros diario y mayor), suponiendo que el pago se hace al momento de recibir la mercancía y que no hay IVA ni otros conceptos, para simplificar. Las entidades del MODE requeridas así como la forma en que evolucionan las transacciones, están determinadas por las reglas del negocio particulares de cada organización.

Además se supone que la transacción de este ejemplo ha sido compactada mediante reingeniería y ha sido también computarizada, por lo que su evolución ocurre dentro del hardware y corresponde a un solo programa de computador y solo uno. No puede corresponder a varios programas porque esto equivaldría a mantener fragmentada la transacción, oponiéndose así al espíritu de la reingeniería.

Con el presente ejemplo sólo se pretende ilustrar el funcionamiento fundamental del MODE desde el punto de vista meramente conceptual, sin adentrarse en detalles técnicos, y sin recurrir a lenguajes de programación, de descripción (DDL o ODL) o de consulta (SQL o OQL), porque directamente no sirven para el MODE.

Por otro lado, una organización no procesa abstracciones sino transacciones concretas, por lo que también hay que suponer que la transacción en cuestión ocurrió por la compra de 200 cuchillos por €2400 (a €12 c/u), en la ferretería “Aceros”, a quien se le pagó con un cheque del banco Nacional. Sin embargo, hay que advertir que mientras para los señores de Contabilidad dicha transacción es reducida a los asientos contables mostrados adelante, para este libro significa la actualización de la base de datos, tabla por tabla, en términos de CPU, como se muestra detalladamente en la figura 25. Aunque en términos de usuario de carne y hueso la transacción sucederá por completo y al instante, gracias primero a la reingeniería y segundo gracias a la informática.

Para simplificar el tipo y la definición del sistema y de los subsistemas, se ignoran otras variedades de compras posibles, como se puede intuir. Ahí existe la entidad “Mercancía” pero no existen entidades del MODE tales como “Seguros” y “Vehículos”, los cuales también son susceptibles de ser comprados. Por esto podrían estar presentes abriendo posibilidades de otras modalidades de compras. Excepto que la compra de seguros y/o de vehículos esté a cargo de un macro diferente, según lo señalen las reglas del negocio.

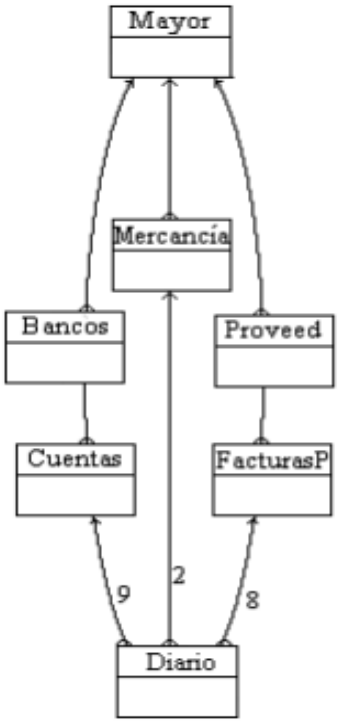

Figura 25. Sistema Tipo compras

Volviendo a la transacción del caso, lo que frecuentemente hacen las empresas (aún mediante computadores) es aplazar el registro de los asientos contables en Contabilidad después del registro de los demás datos en los otros macros, según la naturaleza de la transacción ejecutada. Aquí por el contrario, un único programa de computador asumirá en la misma ejecución, el registro íntegro de toda la transacción invocándolo como un método (sea “comprar al contado mercancía nacional” el nombre de tal método), respetando así la naturaleza atómica de la transacción. Para obtener las tablas primero se realiza una inserción (addo) en la Bitácora o libro Diario para

cada uno de los siguientes asientos contables, generando cada asiento una tupla.

Código cuenta	*Nombre cuenta*	*Debito*	*Crédito*
300	*Existencias comerciales*	*2400*	
572	*Bancos*		*2400*

Se ve así que el "Diario" (Figura 25) contiene dos registros insertados (apuntados por las dos flechas horizontales de la extrema derecha) cuya fecha es el 15 de diciembre de 2005 y cuya hora es 10:34: Uno por cada asiento y cada uno por €2400. El primer registro es el asiento débito sobre la cuenta mayor 300, "Existencias comerciales", y el segundo es el asiento crédito sobre la cuenta mayor 572, "Bancos". Nótese, sin embargo, que estas cuentas no aparecen por ninguna parte en el "Diario". ¿Cómo sabe entonces el programa que estas son las cuentas a debitar y a acreditar y no otras?

Diario

Tra	Fecha	Hora	Nro_docu	Clse	Portador	Usuario	Asiento	D/C	Valor	Cant
1	20051215	10:34	12345	Fa	Empresa	Fulano	1	Db	2400	200
1	20051215	10:34	10263	Ch	Aceros	Fulano	2	Cr	2400	Null

FK1	FK2	FK3	FK4	FK5	FK6	FK7	FK8	FK9	
Nulls	1F36	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	←
Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	3849	←

Mercancías

Código	Nombre	Exist.	Valor_existencia	cuenta_c
1F36	Cuchillos	200+100	2400+1000	300
2U71	Enjalmas	50	1000	300
2F72	Clavos	20	20	300

Cuentas

cuenta_b	ch_gir	v_gir	ch_con	v_con	saldo	nit_bco
3849	1	2400	0	0	50000	90318
7000					5000	90100

Bancos ↓

nit_bco	nombre	teléfono	dirección	saldo	cuenta_c
90318	Nacional	2479853	Cl. 20 #3-10	50000	572
90100	Exterior	31297119	Kra. 7 #4-80	5000	572

Mayor ↓ ↓

cuenta	nombre_cuenta	saldo	débitos	créditos
300	Existencias comerciales	x	x1 + 2400	
572	Bancos	y		y2 + 2400

Tabla 3. Actualización de tablas de la base de datos en la compra al contado.

Precisamente la propia estructura del MODE tiene la respuesta. Porque para poder acceder al registro del archivo "Mayor" cuya clave es 300, "Existencias comerciales", primero debe ser accedido el registro cuya clave es 1F36, cuchillos, en el archivo "Mercancías" cuya clave foránea "cuenta_c" si es 300. Por eso, en el primer registro del "Diario" aparece como clave foránea 2 ("FK2") el código de mercancía 1F36 y no el código de cuenta mayor 300. Porque de acuerdo al MODE, la ruta referencial número 2 está formada por las tablas "Diario"→"Mercancías"→"Mayor", lo que significa que para llegar hasta el "Mayor" debo pasar primero por "Diario" y por "Mercancías". El archivo de "Mercancías" es auxiliar del archivo "Mayor" ya que explica y detalla los datos generales contenidos en él, en la cuenta 300. Si por ejemplo, desde el período contable anterior ya existían los 100 cuchillos, las 50 enjalmas y las 20 cajas de clavos, el valor del saldo "x" en el "Mayor" sería 2020, en el instante antes de ocurrir la presente transacción.

En general, el MODE obliga a que los programas se muevan de lo detallado a lo general por cada ruta referencial, desde el "Diario", el receptáculo más extendido en detalles, hasta el "Mayor", el receptáculo más resumido y conciso, igual a como sucede en la realidad. Así cada asiento sigue su propia ruta referencial, metiéndose primero en el "Diario", pasando luego por el auxiliar respectivo y finalmente llegando al "Mayor", siempre de sur a norte, de polo a polo. De este modo, si una transacción es contabilizada con dos asientos, se moverá por dos rutas referenciales, si con tres, por tres, y así sucesivamente.

El funcionamiento para el otro asiento, el de la cuenta 572 es similar. Aquí también antes de acceder al registro del "Mayor" cuya clave es 572, "Bancos", debo acceder en la tabla "Cuentas" al registro con clave 3849, que es la cuenta bancaria de donde se ha girado el cheque 10263. Por eso en el "Diario" aparece como clave foránea 9 ("FK9") el número de la cuenta bancaria 3849, y no el código de cuenta mayor 572. En el MODE la ruta 9 está formada por las tablas "Diario"→"Cuentas"→"Bancos"→"Mayor", lo que significa que para llegar hasta el "Mayor" debo pasar primero por el "Diario", por "Cuentas" y por "Bancos". El "Diario" contiene todos los cheques girados de cada cuenta bancaria. "Cuentas" contiene los saldos de cada cuenta bancaria y "Bancos" el saldo total de todas las cuentas tenidas en cada banco. Finalmente "Bancos" es el auxiliar del "Mayor" ya que explica y detalla los saldos globales contenidos en la cuenta mayor 572. Por ejemplo, si desde el período contable anterior esta compra es la primera compra, el saldo "y" en el "Mayor" sería 55000 euros, antes de registrar la transacción.

La estructura esencial del "Diario" es la misma de un libro diario como el usado en la contabilidad general de cualquier empresa, excepto que no contiene directamente las cuentas de contabilidad sino las claves foráneas a través de las cuales se llega a ellas en el "Mayor". Esta escalera de claves foráneas a través de las cuales se conocen las cuentas mayores de contabilidad, funciona lo mismo para auxiliares como "Mercancías", "Cuentas" y "Bancos", y le da cohesión al MODE. Pero también la estructura del "Mayor" es la misma del libro mayor usado en la contabilidad general de cualquier empresa.

El MODE en todo momento está expresando lo relacionado con las transacciones sucedidas en la organización, desde los más finos detalles, hasta la generalidad requerida por los altos niveles administrativos. Su lectura general, de muchos a uno, al menos por las dos rutas referenciales usadas en este ejemplo, es como sigue. Por la ruta dos: En el "Diario" hay varios asientos que corresponden a un artículo negociado en una transacción, y a la forma de negociación. Además, en la tabla de "Mercancías" muchos artículos son totalizados en una sola cuenta mayor, la de inventarios en el "Mayor". Por la ruta nueve: En el "Diario" hay varios asientos que están relacionados con una sola cuenta bancaria, expresando sus movimientos detallados. Muchas cuentas bancarias en la tabla de "Cuentas" pueden pertenecer a un mismo banco y sus saldos sumados son el saldo del banco respectivo. El saldo de

cada banco es a su vez totalizado en una sola cuenta mayor, la de “Bancos”, en el “Mayor”.

Sobre cada tabla mostrada en la figura 25 se ven unas flechitas apuntando verticalmente hacia abajo (excepto en el “Diario”) y otras apuntando horizontalmente hacia la izquierda en el “Diario”, mostrando las primeras los campos de cada tabla que serán actualizados (muto) por la transacción y las segundas los registros a ser insertados (addo) por la transacción.

Si la compra considerada en esta transacción involucrara a más artículos además de los cuchillos, por cada artículo se generaría un asiento en el “Diario” y por cada asiento de estos el programa navegaría repetidamente por la ruta referencial número 2, tantas veces como artículos se hubiesen comprado. De este modo, el funcionamiento del MODE lleva a mayor exactitud que la contabilidad convencional que suele ser llevada en las empresas.

A continuación se muestra como crear el Supersistema, con todos sus componentes (sistemas y clases) y la ejecución del método *compra al contado de mercancía* de la consulta hecha en LATIN:

```
Supersystem universal
{
  class mayor
  {
     cuenta                    varchar(5) not null primary key,
     nombre_cuenta             varchar(30) not null,
     saldo                     decimal(10,2)not null,
     debitos                   decimal(8,2)not null,
     creditos                  decimal(8,2)not null
  }
  class diario
  {
     tra                  varchar(3) not null,
     fecha                varchar(10) not null,
     hora                 varchar(10) not null,
     nro_docum            varchar(8) not null,
     beneficiario         varchar(30),
     clse                 varchar(10),
     asiento              varchar(2) not null,
     d_c                  varchar(2),
     valor                numeric(8,2) not null,
```

```
    cantidad            integer
  }
};


System compra_mercancia extend universal
{

  class mercancia
  {
     codigo                    varchar(5) not null primary key,
     nombre                    varchar(30) not null,
     existencia                integer not null,
     valor_existencia          decimal(10,2) not null,
     cuenta_c                  varchar(5) not null,
     foreign key(cuenta_c)     references mayor(cuenta)
  }

  class diario
 {
      cajena2                  varchar(5),
     foreign key(cajena2)      references mercancia(codigo)
 }

method comprar
(cod_trans__,fecha__,hora__,nro_docum__,beneficiario__,clse__,asie
nto__,deb_cred__,valor__,cod_mercancia__,cantidad__)
{

     addo into diario values
     (&cod_trans__,'fecha__','hora__',&nro_docum__,'beneficiario__
     ','clse__',&asiento__,
     'deb_cred__',&valor__,'cod_mercancia__');

     muto mercancia set
     existencia = existencia + cantidad__,
     valor_existencia = valor_existencia + valor__
     ex codigo = 'cod_mercancia__';

     muto mayor set
     debitos = debitos + valor__
     ex cuenta = ( eligo cuenta_c from mercancia ex codigo =
     'cod_mercancia__' ) ;
 }
```

```
};

Subsystem compra_mercancia_contado extend compra_mercancia
{
   class bancos
    {
     nit_bco              varchar(14) not null primary key,
     nom_banco            varchar(30) not null,
     telefono             varchar(15),
     direccion            varchar(30),
     saldo                decimal(10,2),
     cuenta_c             varchar(5) not null,
     foreign key(cuenta_c)      references mayor(cuenta)
    }
   class cuentas
    {
     cuenta_ban           varchar(15) not null primary key,
     cheque_gir           smallint not null,
     valor_giro           decimal(10,2) not null,
     cheque_con           smallint,
     valor_con            decimal(10,2),
     saldo                decimal(10,2) not null,
     nit_banco            varchar(14) not null,
     email                varchar(30),
     foreign key(nit_banco)     references bancos(nit_bco)

    }
   class diario
    {
     cajena9              varchar(15) references
     cuentas(cuenta_ban)
    }
   method comprar
(ch_gir__,beneficiario2__,asiento2__,deb_cred2__,cuenta_ban__ )
 {
     addo into diario values
     (&cod_trans__,'fecha__','hora__',&nro_docum__,'beneficiario2_
     _','clse__',&asiento2__,
     'deb_cred2__',&valor__,null,&cuenta_ban__);

     muto cuentas set
     cheque_gir = cheque_gir + ch_gir__,
     valor_giro = valor_giro + valor__,
     saldo = saldo - valor__
```

```
      ex cuenta_ban = cuenta_ban__;

      muto bancos set
      saldo = saldo - valor__
      ex nit_bco = ( eligo nit_bco from cuentas ex cuenta_ban =
      cuenta_ban__) ;

      muto mayor set
      creditos = creditos + valor__
      ex cuenta = ( eligo cuenta_c from bancos ex nit_bco = (eligo
      nit_bco from cuentas ex cuenta_ban = cuenta_ban__));
 }
};
```

El comando a utilizar para implementar el método de la clase compra de contado:

```
compra_mercancia_contado ( 1, 2008-01-14,21:09:00, 10263,
empresa, fulano, 1, d, 2400, f36, 200,1, aceros,2,c,3849);
```

Pero antes debe estar ingresada la siguiente información en la base de datos tanto para la compra a contado como a credito:

```
"insert into mayor values(300,'existencia comercial',100,0,0)";
"insert into mayor values(572,'bancos',100,0,0)";
"insert into mayor values(400,'proveedores',0,0,0)";
"insert into mercancia values('1f36','cuchillos',100,1000,300)";
"insert into mercancia values('2u71','enjalmas',50,500,300)";
"insert into mercancia values('2f72','clavos',20,300,300)";
"insert into mercancia values('2t41','martillos',50,300,300)";
"insert into proveedores values(91300,'aceros',6351500,'cra 123#3 -1',700,400)";
"insert into facturasp values(78,2400,'05/05/2011',91300)";
"insert into bancos values(90318,'Nacional',2472020,'cll 20#3-10',50000,572)";
"insert into bancos values(90100,'Exterior',3103030,'cra 7#4-80',5000,572)";
"insert into cuentas values(3849,1,2400,0,0,5000,90318,'')";
"insert into cuentas values(7000,0,0,0,0,500,90100,'')";
```

Insertar objeto en clase “Diario”, para modificar objeto 1F36 en clase “Mercancías” con Valor 2400 y Cant 200, y objeto 3849 en clase “Cuentas” con cheque 10263, y objeto 90318 en clase “Bancos” con cheque 10263 de cuenta 3849.

Consulta de instancia transaccional “Compra al contado de mercancía nacional”:

Compra al contado de mercancía nacional,
hecha el año 2005, mes 12, día 15, a la hora 10:34,
pagada con el cheque 10263 de la cuenta 3849 del banco Nacional,
a la ferretería Aceros,
en cantidad de 200 cuchillos
por valor de €2400,
y registrada por Fulano.

La consulta de sistemas es de semántica natural, directa, y semejante a la descripción de la transacción inscrita sobre el libro Diario, que manualmente llevaban las empresas antiguamente. Sin embargo, esta prosa solo serviría para la consulta de una compra específica, pero no funcionaría para consultar todas las compras de determinado período.

4.2 COMPRA A PLAZOS

Otra circunstancia se presenta cuando una supuesta transacción se interrumpe para diferir en el tiempo algunas partes. En este caso, realmente no se trata de una sino de dos o más transacciones. Por ejemplo, si en la compra que se acaba de ver su pago no se hubiera realizado al momento de recibir la mercancía sino en una fecha posterior, se originaría otra transacción complementaria que podría ser el “Pago de facturas a proveedores nacionales de mercancía”. Entonces, al momento de recibir la mercancía no se afectarían los bancos sino las cuentas por pagar a los proveedores, siendo los asientos los siguientes:

Código cuenta	*Nombre cuenta*	*Debito*	*Crédito*
300	*Existencias comerciales*	*2400*	
400	*Proveedores*		*2400*

Esto quiere decir que ahora la transacción, por decir algo la transacción 0, sería la “Compra nacional de mercancía a crédito”, que es otro subsistema de compras, y se moverá por las rutas referenciales 2 y 8

en lugar de hacerlo por las rutas 2 y 9 como antes sucedió con la "Compra nacional de mercancía al contado". Por lo tanto, ahora el "Diario" tiene insertados también dos registros, uno para cada asiento. El de los cuchillos (1F36) es igual, correspondiendo a la misma clave foránea 2 (FK2), pero el otro corresponde ahora a los proveedores, y por eso el valor que está en la clave foránea 8 (FK8), 78, es el número de la factura remitida por el proveedor. De aquí en adelante, la transacción navegará por el MODE con un asiento por la ruta 2 y con el otro por la ruta 8. Las tablas que participan ahora son el "Diario", "Mercancías", "FacturasP" (facturas de proveedor), "Proveedores" y "Mayor", como aparecen en la figura 25.

Diario

Tra	Fecha	Hora	Nro_doc	Clse	Portador	Usuario	Asiento	D/C	Valor	Cant
0	20051215	10:34	78	Fa	Empresa	Fulano	1	Db	2400	200
0	20051215	10:34	10000^{15}	Cu	Aceros	Fulano	2	Cr	2400	Null

FK1	FK2	FK3	FK4	FK5	FK6	FK7	FK8	FK9	
Nulls	1F36	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	←
Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	78	Nulls	←

Mercancías ↓ ↓

Código	Nombre	Exist.	Valor_existencia	cuenta_c
1F36	Cuchillos	200+100	2400+1000	300

FacturasP

nro_fac	valor_fac	fecha_fac	nit_prov	
78	2400	20051215	91300	←

Proveedores ↓

nit_prov	nombre	teléfono	dirección	saldo	cuenta_c
91300	Aceros	2348789	K 32 #3-01	7000	400

Mayor ↓ ↓

cuenta	nombre_cuenta	saldo	débitos	créditos
300	Existencias comerciales	x	x1 + 2400	
400	proveedores	u		u2 + 2400

Tabla 4. Actualización de tablas de la base de datos en la compra a crédito.

A continuación se presenta, la sintaxis para creación del sistema *compra mercancía a plazo.* Aclarando que esto es para crear desde el sistema compra.

```
Supersystem universal
{

  class mayor
  {
   cuenta             varchar(5) not null primary key,
   nombre_cuenta      varchar(30) not null,
   saldo              decimal(10,2)not null,
   debitos            decimal(8,2)not null,
   creditos           decimal(8,2)not null
  }

  class diario
  {
   tra                varchar(3) not null,
   fecha              date not null,
   hora               time not null,
   nro_docum          varchar(8) not null,
   beneficiario       varchar(30),
   clse               varchar(10),
   asiento            varchar(2) not null,
   d_c                varchar(1),
   valor              decimal(8,2) not null
  }
};

System compra_mercancia extend universal
{
   class mercancia
  {
   codigo                  varchar(5) not null primary key,
   nombre                  varchar(30) not null,
   existencia              integer not null,
   valor_existencia        decimal(10,2) not null,
   cuenta_c                varchar(5) not null,
   foreign key(cuenta_c)   references mayor(cuenta)
  }

  class diario
```

```
 {
  cajena2                  varchar(5),
  foreign key(cajena2)     references mercancia(codigo)
 }

  method comprar
(cod_trans__,fecha__,hora__,nro_docum__,beneficiario__,clse__,asie
nto__,deb_cred__,valor__,cod_mercancia__,cantidad__)
 {
  addo into diario values
(&cod_trans__,'fecha__','hora__',&nro_docum__,'beneficiario__','cl
se__',&asiento__,'deb_cred__',&valor__,'cod_mercancia__');

   muto mercancia set
   existencia = existencia + cantidad__,
   valor_existencia = valor_existencia + valor__
   ex codigo = 'cod_mercancia__';

   muto mayor set
   debitos = debitos + valor__
   ex cuenta = ( eligo cuenta_c from mercancia ex codigo =
   'cod_mercancia__' ) ;
 }
};

Subsystem compra_mercancia_plazos extend compra_mercancia
{
class proveedores
 {
  nit_prov      numeric not null primary key,
  nombre        varchar(30) not null,
  telefono      numeric,
  direccion     varchar(30),
  saldo         decimal(10,2) not null,
  cuenta_c      varchar(5) not null,
  foreign key(cuenta_c)    references mayor(cuenta)
 }

class facturasp
 {
  nro_fact           varchar(5) not null primary key,
  valor_fact         numeric not null,
  fecha_fact         date not null,
  nit_provF          numeric not null,
```

```
  foreign key(nit_provF)   references proveedores(nit_prov)
 }


class diario
 {
  cajena8       varchar(5),
  foreign key(cajena8)     references facturasp(nro_fact)
 }
 method comprar
   (beneficiario2__,asiento2__,deb_cred2__,nro_fact__,nit_prov__ )
 {
   addo into diario values
   (&cod_trans__,'fecha__','hora__',&nro_docum__,
   'beneficiario2__','clse__',&asiento2__,'deb_cred2__',
   &valor__,null,&nro_fact__);

   addo into facturasp values
   (&nro_fact__,&valor__,'fecha__',&nit_prov__);

   muto proveedores set
   saldo = saldo + valor__
   ex nit_prov = nit_prov__ ;

   muto mayor set
   creditos = creditos + valor__
   ex cuenta = ( eligo cuenta_c from proveedores
   ex nit_prov = nit_prov__ );
 }
};
```

El comando a utilizar para implementar el método de la clase compra a plazos

```
compra_mercancia_plazos(0,2008-01-14,21:09:00,10000,empresa,
fulano,   1,   d,   2400,   f36,   200,empresa, 2,c,78,91300);
```

4.3 IMPLEMENTACIÓN GRAFICA DE TRANSACCIONES EJEMPLO

Siguiendo la lógica descrita anteriormente, se procede a llevar a un marco práctico, lo ya planteado, que es la creación de la implementación de un ejemplo, de una de las transacciones más recurrentes en el grueso de la industria.

Dejando aquí plasmado, la conquista de unos de los objetivos propuesto al inicio del proyecto, y además la visualización más familiar de las bondades del Latin.

4.3.1 Implementación Gráfica De La Transacción Compra Al Contado

Con lo descrito ya en la sesión 4.1 se procede a ejecutar este sistema. A continuación la imagen de los resultados obtenidos.

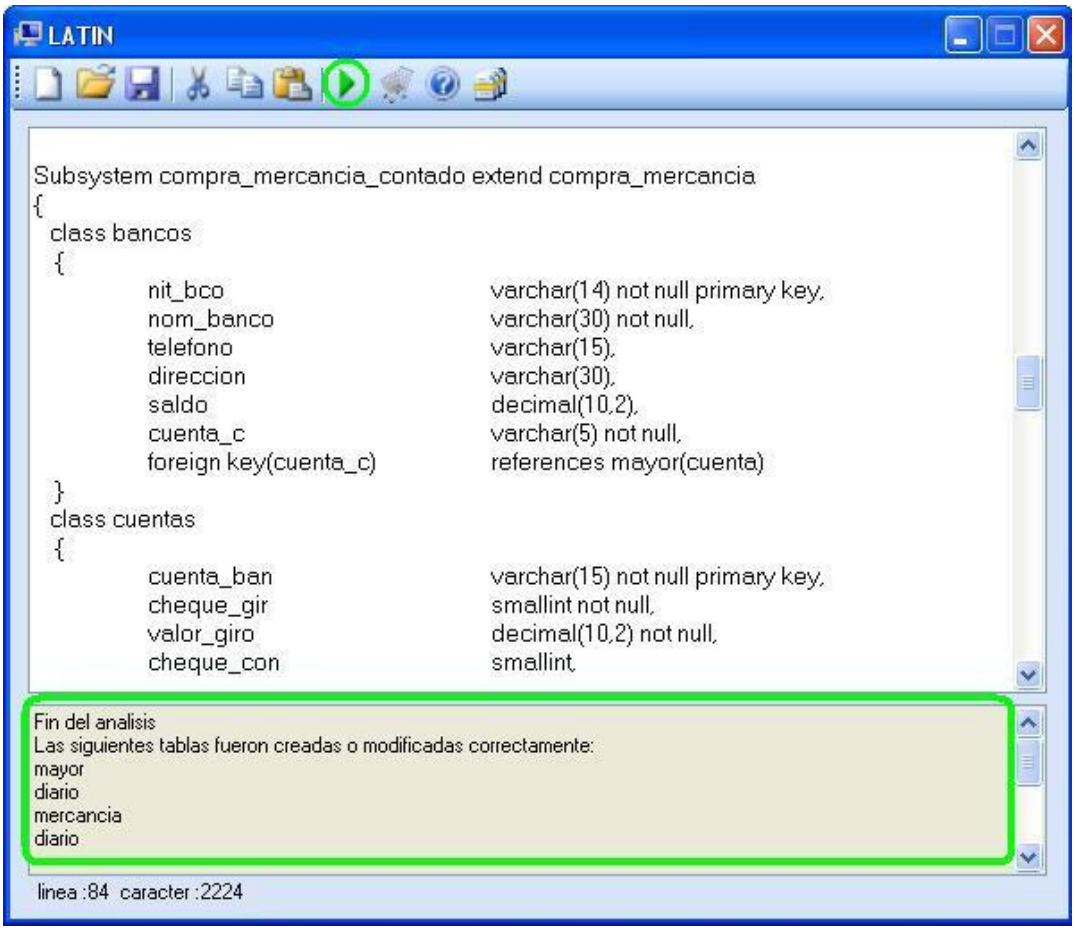

Figura 26. Análisis léxico-Sintáctico Transacción ejemplo

En el recuadro inferior, se presentan los resultados (exitosos en este caso), donde se especifican las acciones realizadas, sea de creación o de modificación (repetida). Con esta acción se ha realizado el análisis y creado el Supersistema *universal* junto con el sistema *compra_mercancia* y el subsistema que nos interesa para el ejemplo que es *compra_mercancia_contado.*

Lo siguiente que se hace es, la ejecución del método del subsistema para poder realizar una transacción de esta naturaleza, se puede realizar desde el editor, pero sabiendo que antes se debe de llenar las tablas para poder ser así manipuladas. O la otra opción que es la que vamos a desarrollar a continuación, que es por medio de una transacción con la ayuda de la interfaz factura.

Ya creado de manera correcta el subsistema, se habilita el botón de la factura, para así poder ingresar y realizar las transacciones necesarias. En los siguientes gráficos se muestra una operación ejemplo, con la ayuda de este módulo.

Como primer paso luego de haber dado clic en el icono de la factura (carrito de compra), con esta operación se llenara la base de datos, con los datos descritos en la figura 6, y dará el control a la ventana de login, done se nos solicitar unos datos, para así tener registro del usuario que realiza la operación. Por defecto aparecerá un usuario y su respectiva contraseña que son: systema y systema. Pero se encuentra la posibilidad de registrar un nuevo usuario.

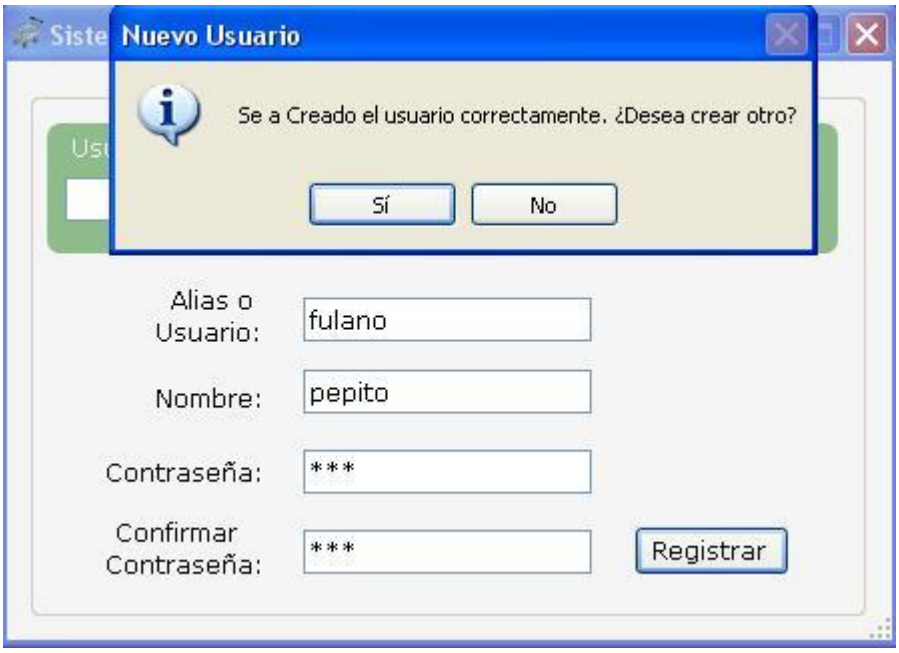

Figura 27. Registro de Nuevo Usuario

Luego de haber realizado un correcto login, pasa a la ventana donde se encuentra propiamente lo que denominamos factura o el modulo compra (figura 28). Ya estando ahí, se procede con el ingreso de datos solicitados para la realización de la operación.

Figura 28. Operación ejemplo de la interfaz Compra contado

En sesiones posteriores se explicara más a fondo cada uno de los campos requeridos. Por ahora, para el ejemplo se realizó una compra al contado, a la Ferretería Aceros que ha dado un documento como certificado de la compra, para este caso es 10263, y que ha sido registrada la compra por el usuario fulano y ha realizado el pago de la cuenta de banco 3849. La compra es por los ítems mostrados en la figura 25.

En este módulo se encuentra la posibilidad de hacer consultas a las transacciones realizadas, y con este tendremos a la mano una bitácora de lo que se ha comprado, de contado, crédito o generar el informe de todas las transacciones.

.

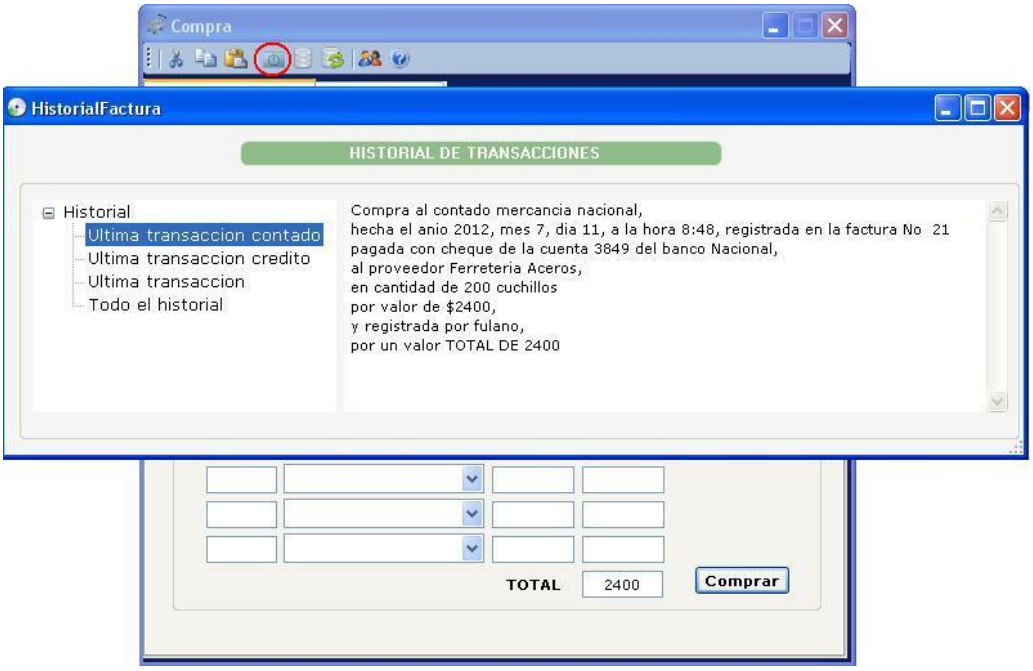

Figura 29. Consulta de historial de transacciones

Para los usuarios ya un poco más avanzados, se encuentra la opción de hacerle un seguimiento a la base de datos, y así observar las alteraciones realizadas por medio de las transacciones y la ejecución del método *compra_mercancia_contado.*

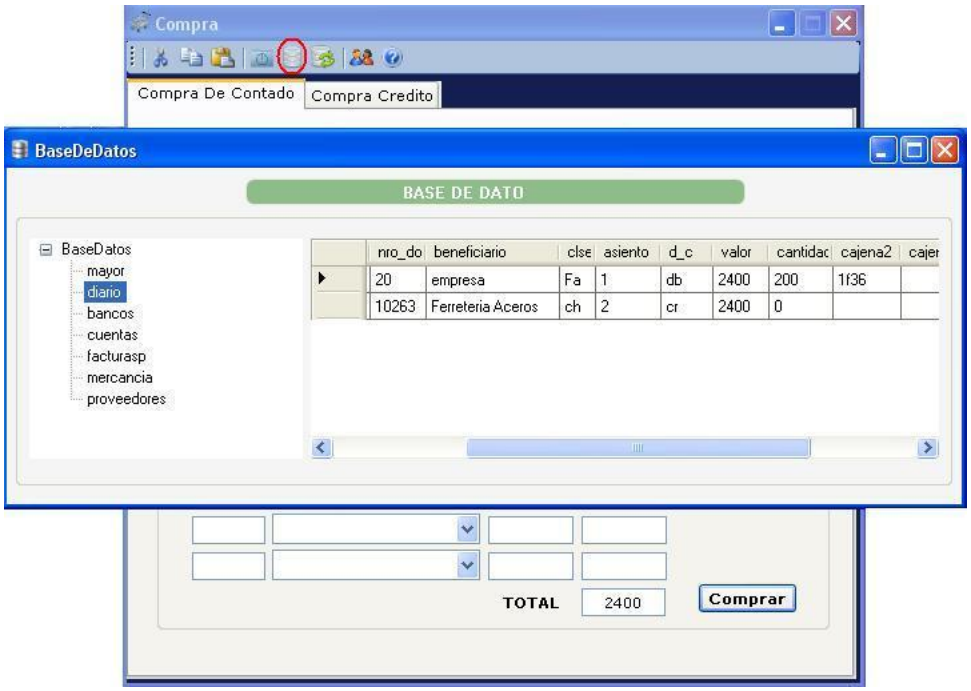

Figura 30. Consulta de base de datos a la tabla diario

4.3.2 Implementación Gráfica De La Transacción Compra A Plazos

Con la creación del subsistema compra_mercancia_plazos del sistema *compra_mercancia*, da paso a la posibilidad de interactuar con el módulo de compra de mercancías a plazo (crédito). A continuación se presenta la transacción ejemplo realizada en este módulo.

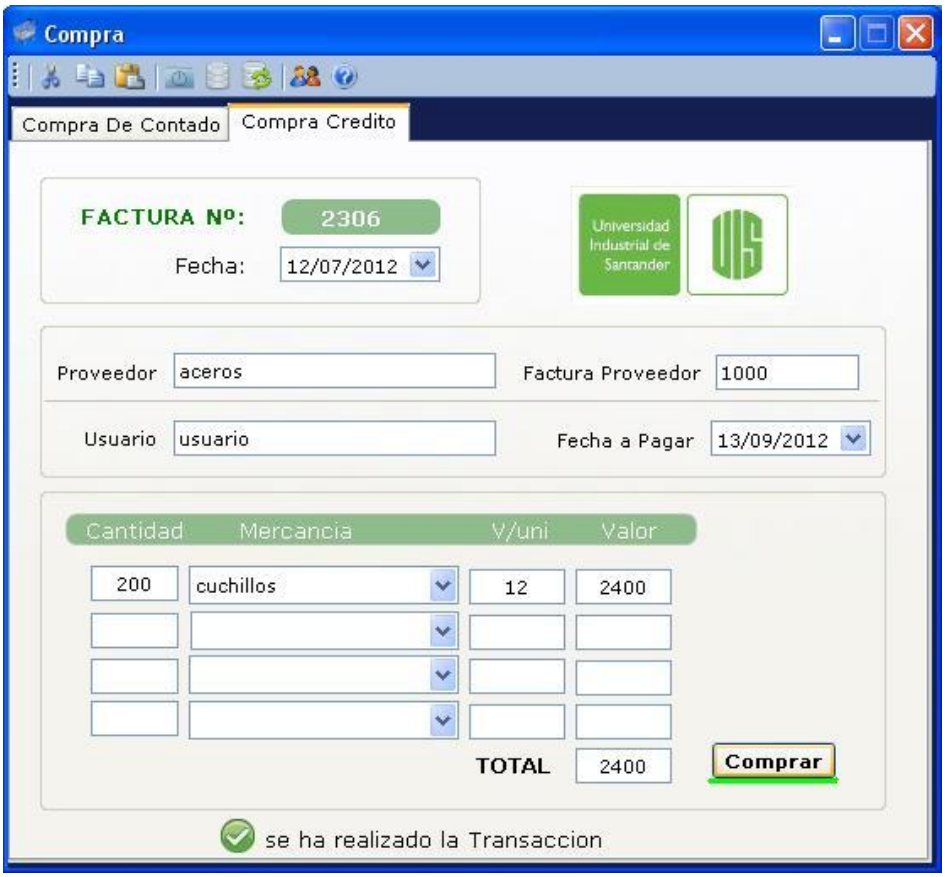

Figura 31. Operación ejemplo de la interfaz Compra Crédito

A simple vista esta operación, asemeja en aspecto a la compra de contado, pero en esencia, no son semejantes, dado que navegan por rutas referenciales diferentes. La transacción a contado, lo hace por la ruta 2,9 (ver figura 25), mientras que Compra a Crédito lo hace por la 2,8. Alterando un conjunto de entidades diferentes y en síntesis, dejando el pago de la mercancía, adquirida para ser realizado al futuro (nótese la fecha de la factura y la fecha por pagar).

Al igual que Compra a contado, esta transacción también brinda las mismas posibilidades, para poder consultar el resultado de la operación, tanto como a nivel de la bitácora que se lleva por cada compra, como obviamente una consulta a nivel de base de datos.

A continuación, se mostrara estas dos consultas.

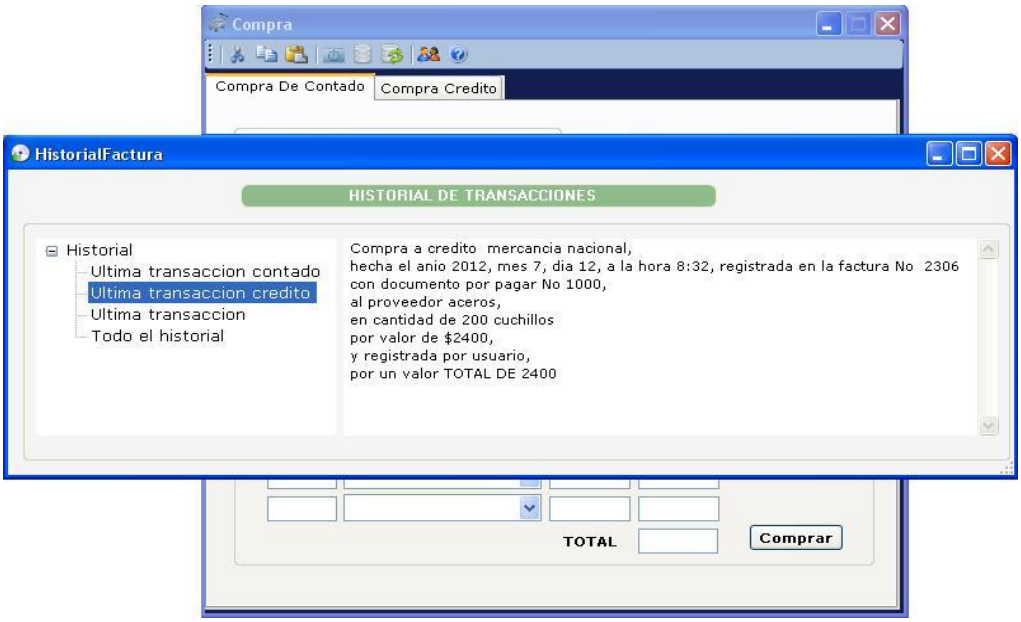

Figura 32. Historial transacción crédito

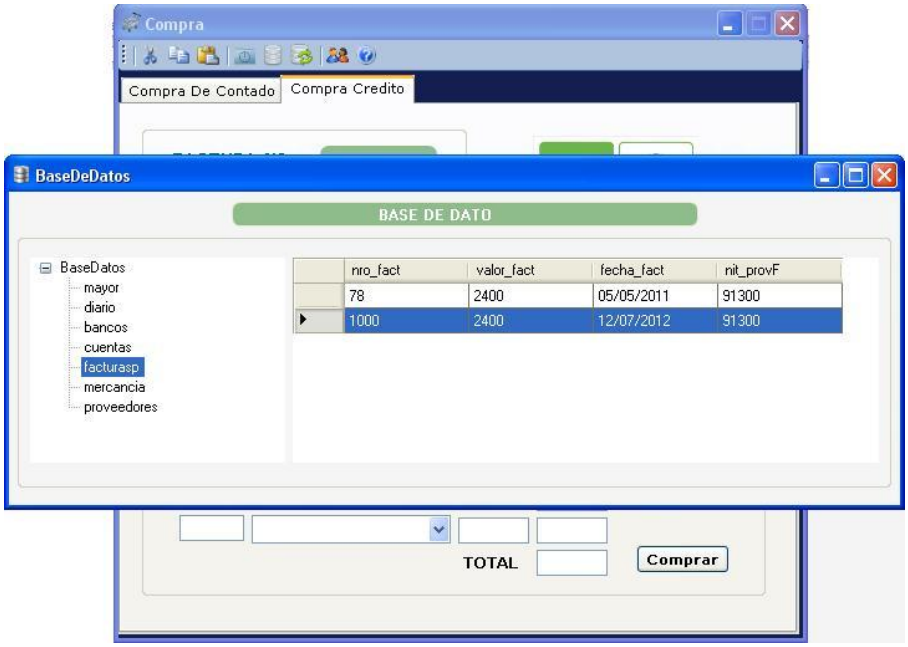

Figura 33. Consulta de base de datos a la tabla facturasP

5. MANUAL DE USUARIO

5.1 INSTRUCCIONES DE USO DE LOS ELEMENTOS DE LA INTERFAZ

Se debe sentar un precedente, dejando en claro que se presentaron sugerencias al director del proyecto, sobre algunas reglas del negocio, pero siempre se presentó oposición y negación al cambio.

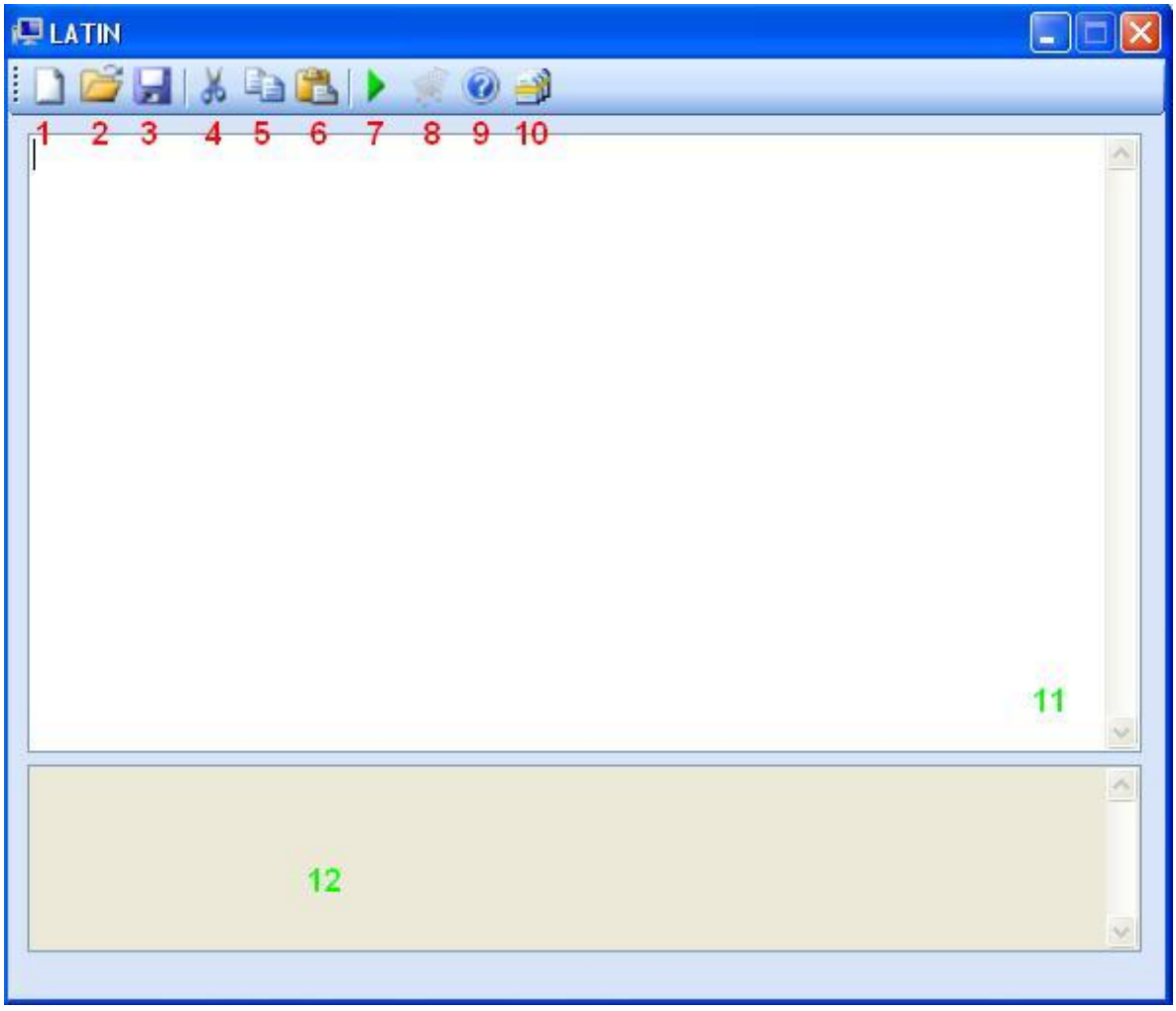

Figura 34. Manual de la interfaz principal del Latin

1. Botón Nuevo: limpiar el editor principal del interprete (11) para poder si se desea iniciar una nueva instrucción.
2. Botón Abrir: permite abrir documento de texto que para este caso contengan descripciones de sistemas.
3. Botón Guardar: almacena una instrucción descrita en el editor principal para ser posteriormente usado, si es el caso.
4. Botón Cortar: Es el encargado de cortar lo que el usuario seleccione. Este desaparece la información de su ubicación inicial.
5. Botón Copiar: Este botón copia la información seleccionada. A diferencia del anterior este solo lo copia pero deja el contenido intacto.
6. Botón Pegar: Después de haber copiado o cortado el texto, al dar clic en el este pega el contenido en el espacio establecido por el usuario.
7. Botón análisis léxico-sintáctico: realiza estos dos análisis y si es correcto crea el sistema descrito o ejecuta el método invocado.
8. Botón iniciar Factura: Este botón invoca el módulo de compra que inicialmente está inactivo hasta la correcta creación del sistema ejemplo; *ejemplo factura*, con sus respectivos subsistemas.
9. Botón Ayuda: invoca una herramienta web en el que presta una aclaración del uso del intérprete.
10. Botón borrar supersistema: permite eliminar un supersistema. Pero en algunos casos se debe completar la operación primero cerrando el programa segundo iniciarlo nuevamente y tercero invocando este botón para así borrar el supersistema.
11. Editor principal del intérprete: se describen las diferentes instrucciones que se van a realizar.
12. Interfaz de resultados: en él se le comunica al usuario el estado de sus diferentes análisis.

5.1.1 Uso de la interfaz factura Compra

5.1.1.1 Compra Contado

Figura 35. Manual de Compra ha contado

Los botones 1, 2, 3, 4,5, no son de uso exclusivo de cada una de los tipos de compra, sino son opciones para el modulo compra, que contiene a estas dos operaciones.

1. Botón historial: muestra el historial de las transacciones realizadas.
2. Botón Base de Datos: muestra la base de datos.
3. Botón Reiniciar Base de Datos: esta operación se debe realizar dado que se encuentra una cantidad de dinero limitada para realizar compras a contado, y llegado el caso un usuario puede agotar este recurso y se haría imposible realizar una nueva compra.
4. Botón Cambiar de Usuario: despliega la interfaz de login.
5. Botón ayuda: invoca una herramienta web en el que presta una aclaración del uso del intérprete.

6. Pestaña compra ha contado: permite interactuar con la factura compra a contado.
7. Pestaña compra crédito: permite interactuar con la factura compra crédito.
8. Factura No: es un número aleatorio de cuatro dígitos que funciona como el número de la factura que se queda como registro de la transacción hecha por la empresa.
9. Fecha: en él se permite escoger la fecha en la cual se realizó la compra (solo se puede escoger hasta un año atrás del día que se registre la transacción y tampoco se permite escoger fechas al futuro).
10. Proveedor: se registra el nombre del proveedor al que se le hizo la compra.
11. No documento: es el número de factura suministrado por el proveedor a la empresa, según compra la realizada.
12. Usuario: es el que previamente se logeado como usuario (este usuario para poder ser modificado debe *cambiar usuario*).
13. Cuenta Banco: para este caso se manejan dos tipos de cuentas, una que es del banco NACIONAL (3849) con un saldo inicial de $5000 (dólares) y otra del banco EXTERIOR (7000) con un saldo inicial de $500 (dólares).
14. Cantidad: registra la cantidad de mercancía a comprar. Hace una validación implícita de los datos ingresados. Reconociendo únicamente valores enteros positivos.
15. Mercancía: para este ejemplo en particular manejamos 4 tipos de mercancías, que se escogen mediante una lista desplegable.
16. V/uni: registra el valor unitario de la mercancía a comprar. Realiza una validación de la cantidad ingresada sea correcta y una comprobación de que el precio registrado también pertenezca al rango aceptado (enteros positivos). Se verifica el valor total de las unidades a comprar para comprobar si se dispone de suficiente cantidad de dinero en el banco para realizar la compra.
17. Valor: es un campo autogenerado por la multiplicación de cantidad y V/uni.
18. Total: es la suma de la columna valor. Se hace una comprobación de que este valor ingresado por el usuario sea tanto en el rango (enteros positivos) como en el valor correcto.
19. Comprar: Ejecuta la transacción después de haber realizado la validación de todos los campos donde no pueden haber campos nulos ni erróneos.

5.1.1.2 Compra A Crédito

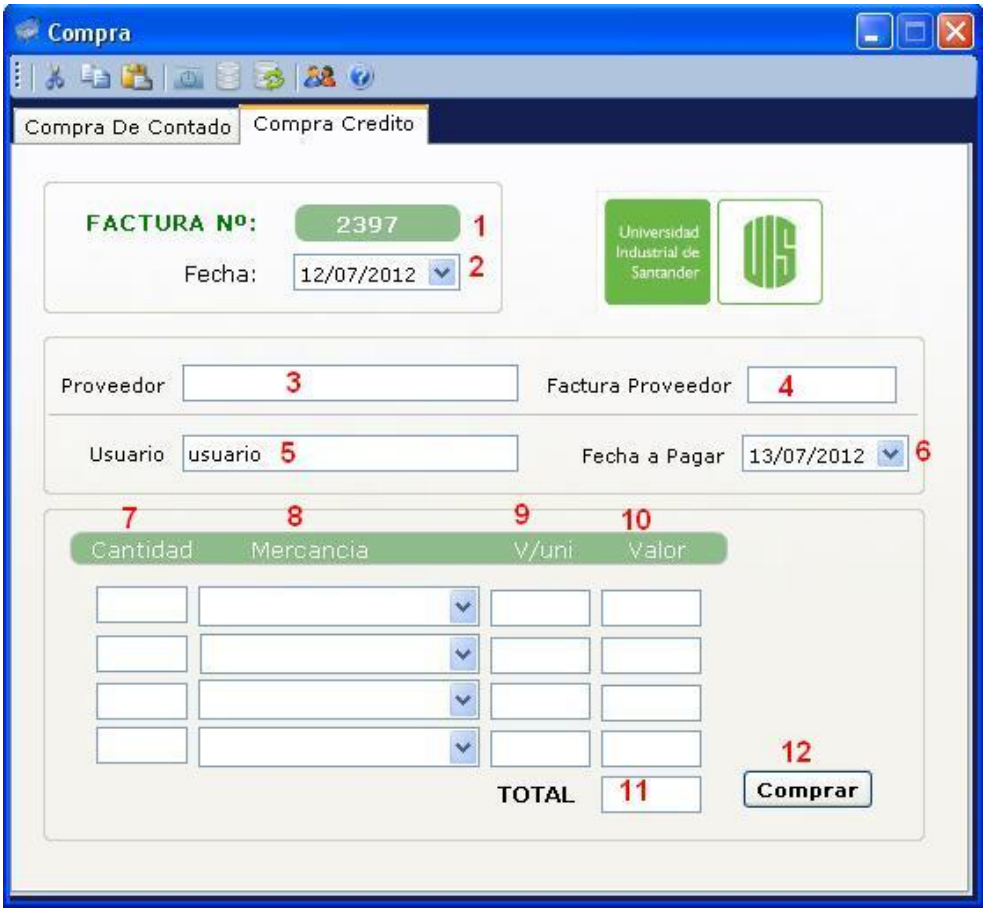

Figura 36. Manual de Compra crédito

1. Factura No: es un número aleatorio de cuatro dígitos que funciona como el número de la factura que se queda como registro de la transacción hecha por la empresa.
2. Fecha: en él se permite escoger la fecha en la cual se realizó la compra (solo se puede escoger hasta un año atrás del día que se registre la transacción y tampoco se permite escoger fechas al futuro).
3. Proveedor: para este caso, el proveedor debe de estar en la base de datos, entonces se realiza una validación, si llegado al caso el proveedor no existe, se nos desplegara la interfaz de nuevo proveedor, para allí poderlo registrar (ver figura 37).
4. No documento: es el número de factura suministrado por el proveedor a la empresa, según compra realizada.

5. Usuario: es el que previamente se logeado como usuario (este usuario para poder ser modificado debe cambiar usuario).
6. Fecha a Pagar: se selecciona la fecha en la cual se debe de realizar el pago de la factura (está garantizado fechas futuras).
7. Cantidad: registra la cantidad de mercancía a comprar. Hace una validación implícita de los datos ingresados. Reconociendo únicamente valores enteros positivos.
8. Mercancía: para este ejemplo en particular manejamos 4 tipos de mercancías, que se escogen mediante una lista desplegable.
9. V/uni: registra el valor unitario de la mercancía a comprar. Realiza una validación de la cantidad ingresada sea correcta y una comprobación de que el precio registrado también pertenezca al rango aceptado (enteros positivos). Se verifica el valor total de las unidades a comprar para comprobar si se dispone de suficiente cantidad de dinero en el banco para realizar la compra.
10. Valor: es un campo autogenerado por la multiplicación de cantidad y V/uni.
11. Total: es la suma de la columna valor. Se hace una comprobación de que este valor ingresado por el usuario sea tanto en el rango (enteros positivos) como en el valor correcto.
12. Comprar: Ejecuta la transacción después de haber realizado la validación de todos los campos donde no pueden haber campos nulos ni erróneos.

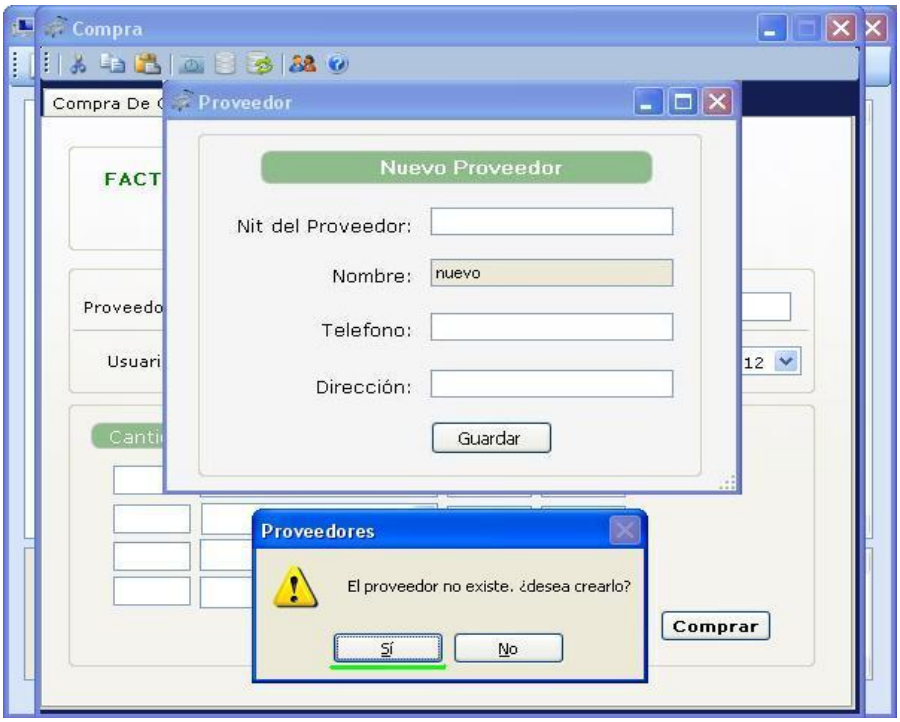

Figura 37. Manual Interfaz de nuevo proveedor

5.1.1.3 Historial De Transacciones

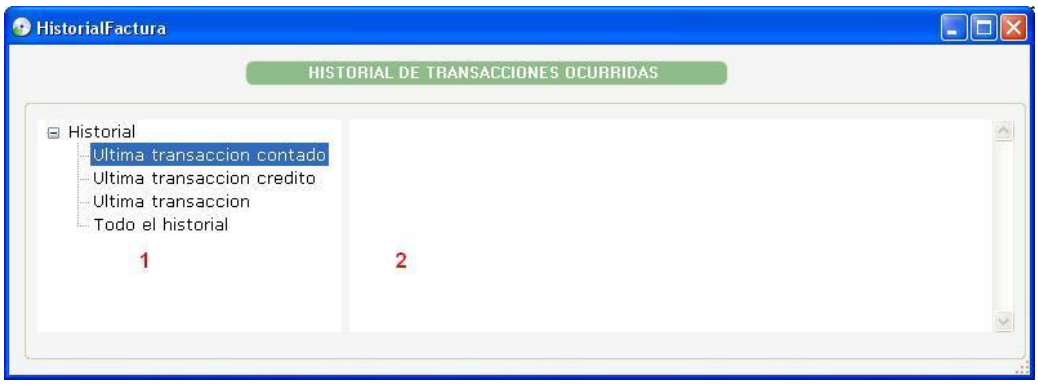

Figura 38. Manual de historial de transacciones.

1. Una lista con las opciones disponibles para el historial.
2. Presenta la información de la transacción seleccionada.

5.1.1.4 Base De Datos

Figura 39. Manual Base de Datos

1. Una lista con las opciones de las clases con las que cuenta el sistema.
2. Presenta la información que contiene cada clase.

5.1.1.5 Login Usuario

Figura 40. Manual login Usuario

1. Usuario: ingreso del alias o usuario registrado.
2. Contraseña: correspondiente al alias o usuario registrado.
3. Entrar: validación de los datos suministrados en usuario y contraseña. Da acceso a la interfaz de factura.
4. Alias o Usuario: para la creación de un nuevo usuario siendo este su identificador.
5. Nombre: correspondiente al alias de usuario.
6. Contraseña: campo de tipo char.
7. Confirmar Contraseña: verificación de la contraseña sea la correcta.
8. Registrar: Valida si los datos suministrados son correctos, si lo son realiza el registro, de lo contrario da notificación en donde se encuentra el error.

5.2 EJEMPLO SINTAXIS DEL LATIN CON COMENTARIOS

```
 1.   Supersystem SUPERIOR
 2.   {
 3.   class mayor
 4.         {
 5.         cuenta                  varchar(5) not null primary key,
 6.         nombre_cuenta     varchar(30) not null
 7.         }
 8.   class diario
 9.         {
10.         fecha       date not null,
11.         hora        time not null
12.         }
13.   };
```

linea 1: declaración del supersistema, esta creación es única, no puede existir más de un supersistema declarado. Su sintaxis es: *Supersystem* **nombre_supersistema** (La primera ‘s’ debe ser siempre en mayuscula)**.**

línea 2 -13: cuerpo del supersistema, contenido en dos corchetes y finalizado por un punto y coma. Dentro de este se encuentra la declaración de las clases.

Línea 3 y 8: declaración de clases, su sintaxis es: class nombre_clase.

Línea 4-7 y 9-12: Cuerpo de clase: inicia y finaliza con un corchete, en él está contenida su descripción.

```
1. System MEDIO extend SUPERIOR
```

```
2. {
3. class mercancia
4.       {
5.       codigo                        varchar(5) not null primary key,
6.       nombre                  varchar(30) not null,
7.       foreign key(cuenta_c)         references mayor(cuenta)
8.       }
9. class diario
10.           {
11.           cajena2                       varchar(5),
12.           foreign key(cajena2)          references
   mercancia(codigo)
13.           }
14.      method para_MEDIO (cod_,nombre_)
15.           {
16.           }
17.      };
```

Línea 1: declaración del Sistema, esta creación es única (No puede existir otro sistema llamado de igual forma) seguido del nombre del supersistema en el cual está contenido. Su sintaxis es: Sy*stem* **nombre_sistema** *extend* **nombre_supersistema** (La primera ‘s’ debe ser siempre en mayuscula)**.**

Línea 2-17: cuerpo del sistema, contenido en dos corchetes y finalizado por un punto y coma. Dentro de este se encuentra la declaración de las clases y un método asociado a dichas clases.

Línea 3 y 9: declaración de clases, su sintaxis es: class nombre_clase.

Línea 4-8 y 10-13: Cuerpo de clase: inicia y finaliza con un corchete, en él está contenida su descripción.

Línea 14: declaración del método, su sintaxis es: method nombre_metodo **(**parámetros_metodo**)**

Línea 15-16: cuerpo método, inicia y finaliza con corchetes, dentro de ellos va contenido las operaciones diseñadas para el sistema (actualizar, insertar, borrar las clases).

```
1. Subsystem INFERIOR extend MEDIO
2. {
3. class bancos
4.       {
5.       nit_bco          varchar(14) not null primary key,
6.       nom_banco        varchar(30) not null
```

```
7.          }
8.  class cuentas
9.          {
10.         cuenta_ban          varchar(15) not null primary key,
11.         cheque_gir          smallint not null
12.         }
13. class diario
14.         {
15.         cajena9             varchar(15),
16.         foreign key(cajena9)  references cuentas(cuenta_ban)
17.         }
18. method comprar(ch_gir__,cuenta_ban__ )
19.         {
20.         }
21. };
```

Línea 1: declaración del Subsistema, esta creación es única (No puede existir otro subsistema llamado de igual forma) seguido del nombre del sistema en el cual está contenido. Su sintaxis es: Sy*stem* **nombre_subsistema** *extend* **nombre_sistema** (La primera 's' debe ser siempre en mayuscula)**.**

Línea 2-21: cuerpo del sistema, contenido en dos corchetes y finalizado por un punto y coma. Dentro de este se encuentra la declaración de las clases y un método asociado a dichas clases.

Línea 3, 8 y 13: declaración de clases, su sintaxis es: class nombre_clase.

Línea 4-7 y 9-12 y 14-17: Cuerpo de clase: inicia y finaliza con un corchete, en él está contenida su descripción.

Línea 18: declaración del método, su sintaxis es: method nombre_metodo **(**parámetros_metodo**)**

Línea 19-20: cuerpo método, inicia y finaliza con corchetes, dentro de ellos va contenido las operaciones diseñadas para el sistema (actualizar, insertar, borrar las clases).

4. CONCLUSIONES

- Este proyecto es un primer paso, a la solución de la desintegración de los procesos, problemática planteada por el profesor Jaime Albarracín. Entendiendo que se requiere avanzar en la investigación y mejoramiento de las herramientas tecnológicas, para poder abarcar de una manera más amplia el modelo emergente de datos “MODE”.

- La parte estructural del interprete (el manejo de las clases) con un motor de bases de datos orientado a objetos como lo es db4o, en la actualidad presenta una incompatibilidad con los requisitos del interprete LATIN, este motor no cumple con unos de las funcionalidades básicas de este tipo de interprete, como lo es el manejo de objetos dinámicos.

- La propuesta de LATIN enfocada a sistemas, crea una gran expectativa con respecto al uso de los lenguajes anteriores, ya que el principal enfoque de dichos lenguajes está orientado a estructuras o cuando mucho a objetos. En este proyecto, al contrario, se ilustra el funcionamiento más adecuado que tienen las organizaciones.

- El modelo relacional no ha podido ser superado por ningún otro modelo en la actualidad. Sin embargo el modelo de objetos es un intento que ha quedado enfrascado puesto que no ha sido aceptado por la industria. A pesar de lo anterior, un modelo de bases de datos diferente ha sido propuesto sobre los cimientos tanto del modelo relacional como del modelo de objetos. En este sentido, nuestro proyecto de grado es una contribución para el desarrollo del lenguaje requerido para la implementación del modelo en cuestión.

5. RECOMENDACIONES

- Se recomienda trabajar sobre una interfaz gráfica, que permita visualizar: la ejecución de las transacciones, consulta al sistema y que maneje internamente la creación del sistema y el trabajo de los analizadores, para que sea de mayor utilidad observar el avance de este proyecto y los futuros.

- A los aspirantes a continuar con la mejora de este proyecto, y en realizar nuevas versiones de este, realizar un estudio previo, serio y concienzudo de cada uno de los componentes que se interactúan en este interprete. Realizar pruebas menores, que den luz sobre la posibilidad de ejecución, y finalización de lo emprendido.

- El análisis, diseño e implementación de un motor de almacenamiento propio (para este trabajo de grado se utilizó el SQLite) que permita aumentar el nivel estructural y funcional del modelo entidad – relación, a un modelo más acorde al planteado en la tesis del profesor Albarracín, que entre sus características contemple la direccionalidad entre las entidades padre–hijo.

- El desarrollo de los tipos (tipo compra, tipo ventas etc.) en especial el tipo compra para el LATIN, es fundamental para facilitar el completo uso del interprete.

- Se recomienda implementar en trabajos futuros otro gestor de bases de datos que, facilite el desarrollo y abarque más posibilidades no tenidas en cuenta en este proyecto y de igual manera se obtenga la integración de todos los componentes.

6. BIBLIOGRAFÍA

TEXTOS BIBLIOGRÁFICOS

- ALFONSECA MORENO MANUEL, ECHEANDIA MARINA DE LA CRUZ, PULIDO CAÑABATE ESTRELLA. Compiladores e Intérpretes. Prentice – Hall. ISBN: 84-205-5031-0; 2006. En este libro se encuentra todo lo concerniente a la teoría de los Compiladores e intérpretes.

- C. LAUDEN KENNETH. Construcción de Compiladores. Editorial THOMSON. ISBN: 970-686-299-4; 2001. En este libro se encuentra información pertinente al diseño y construcción de un compilador.

- CATTELL, BARRY. The Object Data Standard: ODMG 3.0. MORGAN KAUFMANN PUBLISHER. ISBN: 1-55860-647-4; 2000. Este libro define el Standard ODMG, con el cual se pueden implementar sistemas manejadores de bases de datos orientados a objetos y objeto – relacional.

- CUEVA LOVELLE JUAN MANUEL. Lenguajes Gramáticas y Autómatas. Universidad de Oviedo – España, 2001, Segunda Edición. En este libro se encuentra información sobre los lenguajes gramáticas y autómatas.

- GALVEZ ROJAS SERGIO/MATA MIGUEL. Java a Tope: Traductores y Compiladores. Universidad de Málaga – España, 2005. En este libro se encuentra información sobre Compiladores e Intérpretes.

- GRANT ALLEN, MIKE OWENS. The Definitive Guide to SQLite – EE.UU, 2010, Segunda edicion. Em este libro se encuentra un compendi importante de informacion, sobre este motor de base de datos relacional.

- LLANO DÍAZ EMILIANO. Análisis y Diseño de Compiladores - México, Junio de 2002. En este libro se encuentra información de cómo diseñar compiladores.

- LÓPEZ MELLADO ERNESTO. Curso Propedéutico de Computacion, Lenguajes Formales – Guadalajara - México, Junio de 2005. En este resumen se encuentra información sobre los lenguajes Formales.

- MORAL SERAFÍN. Modelos de Computación. Departamento de Ciencias de Computacion – España, 2002. En este libro se encuentra información sobre los modelos de computación, autómatas finitos y sus gramáticas.

- PRESSMAN S. ROGER. Ingeniería del Software, un enfoque práctico. Quinta Edición. MacGraw-Hill. ISBN: 84-481- 3214-9; 2002. En este libro se encuentra información sobre las diferentes metodologías de desarrollo de software.

- SCHILDT HERBERT. C# Manual de Referencia. MacGraw- Hill. ISBN: 84-481-3712-4; 2003. Este libro muestra información en lo referente al lenguaje de programación C# y la teoría de objetos.

- STEFAN EDLICH, JIM PATERSON. The Definitive Guide to db4o – EE.UU, 2006. En este libro está consignado, una cantidad importante de información de este motor de bases de datos orientado a objetos.

- WELEY ADDISON / DIAZ DE SANTOS. Sistemas de Bases de Datos Orientados a Objetos. ISBN: 0-201-65356-7; 1995. En este libro se encuentra la información que corresponde al diseño y construcción de sistemas de bases de datos orientados a objetos.

PROYECTO DE GRADO.

JULIÁN RICARDO CAMARGO GONZÁLEZ, YULIANA MAYERLY SOTO CARREÑO. “INTÉRPRETE LATIN PARA LA IMPLEMENTACIÓN DE MODELOS EMERGENTES DE DATOS.”, Proyecto de grado. Universidad Industrial de Santander. En esta tesis se especifican los detalles de la gramática del intérprete LATÍN.

TESIS DOCTORAL

ALBARRACIN FERREIRA JAIME OCTAVIO. “Integración de datos y procesos en las organizaciones mediante un Modelo de Datos Reorientado a Objetos”, Tesis Doctoral. Universidad de Oviedo. España. 2006. En esta tesis se encuentra toda la información correspondiente al Modelo de Datos Reorientados a Objetos “MODRO”.

DOCUMENTACION DISPONIBLE EN INTERNET

- **http://www.invemar.org.co/redcostera1/invemar/docs/6318MetodologiaProyectosLSI.pdf.** Sitio Web que muestra información referente a las diferentes metodologías de desarrollo de software en ingeniería del software.

- **http://www.*sqlite*.org.** Documentación y versiones disponibles de SQLite

7. ANEXOS

ANEXO A. CÓDIGO NECESARIO PARA PARA GENERAR EL ANALIZADOR SINTACTICO DEL INTERPRETE LATIN

```
using TUVienna.CS_CUP.Runtime;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.IO;
using System.CodeDom.Compiler;
using System.Linq;
using System.Data.SQLite;
using System.Windows.Forms;


parser code {:

/*public static void Main(string[] args)  {
             new parser(new Yylex(System.Console.In)).parse();
       }*/
//guardar el nombre de la clase
public String nombre_clase;
//public String tipo_dato;
public String aux_clase;
public String nombre_sistema;
public int super_sistema=0;
public int ejecutar_metodo=0;
public StreamWriter guardar_metodo;
public String nombre_metodo;
public int correcto=1;
public String nombre_padre;
//para controlar si el sistema existe o no en la herencia
public int sistema=0;
public StreamWriter sw ;
public StreamWriter sw1 ;
public StreamWriter sw2 ;
public String ruta="";
public String ruta_padre="";
public String funciones;
```

```
public int alter=0;
public int herencia_subsistema=0;
public String nombre_supersistema;
public String nombre_metodo_padre;
public int subsistema=0;
public int No_leer_conexion=0;
public String lista="";
public String especificacion="";
public String nombre_clase2;
public String nombre_atributo;
public String numero1;
public String constructor;
public String aux_metodo;
public String aux_muto;
public String aux_eligo;
public String aux_deleo;
public String aux_atributo;
public String aux_subquery;
public String esc_exp1;
public String esc_exp2;
public String actualizador;
public String sub_busqueda;
public SQLiteConnection conexion;
public String nombre_global;
public String[] temp_clases=new String[50];
public String[] temp_nombre_clases=new String[50];
public String[] metodos_actuales=new String[50];
public String[] sistemas_actuales=new String[50];
public String[] clases_alteradas=new String[50];
public int cont=0;

public override int error_sym(){syntax_error(cur_token); return 1;}

public void report_error(String message, Object info)
 {/* Crea un StringBuilder llamado 'm' con la cadena 'Error'. */

    StringBuilder m = new StringBuilder("Error");
    done_parsing();
    /* Verifica si la informacion pasada es del tipo
       java_cup.runtime.Symbol. */

  if(info.GetType()==typeof(TUVienna.CS_CUP.Runtime.Symbol))
     {
      /* Declara un TUVienna.CS_CUP.runtime.Symbol 's'
         con la informacion en la info del objeto. */
```

```
      TUVienna.CS_CUP.Runtime.Symbol s = ((TUVienna.CS_CUP.Runtime.Symbol)
      info);
          /* Verifica si el numero de lineas en la entrada es >= cero */
       if (s.left >= 0)
              {
               /* Agrega al final del StringBuffer un mensaje de error
                   con el numero de la linea en la entrada. */
                m.Append(" en linea "+(s.left+1));
               /* Verifica si el numero de columna en la entrada es >=
                 cero.   */
               if (s.right >= 0)
               /* Agrega al final del StringBuffer un mensaje de error
                  con el numero del caracter en la entrada. */
              m.Append(", caracter "+(s.right+1));
              throw new System.Exception(m.ToString());
             }
      }
 }
/* Cambia el metodo report_fatal que reporta un error fatal mostrando el
   numero de la linea y la columna donde el error ocurrio en la entrada
   como la razon por la que el error fatal ocurrio dentro del metodo.*/
  public void report_fatal_error(String message, Object info)
   {report_error(message, info);
   }
  public void syntax_error(Symbol cur_token)
   {/* Este metodo es llamado por el parser tan pronto como un error
       de sintaxis es detectado*/
    report_error("Error de Sintaxis", cur_token);
   }
  public void extraer_supersistema()
  {string contenido = File.ReadAllText("conexion.txt");
   string conexion="";
   for (int i = 0; i <= contenido.Length;i++ )
        { if(contenido[i]=='\r')
              {break;
              }
          conexion = conexion + contenido[i];
        }
   nombre_supersistema=conexion;
  }
public void extraer_nombremetodo()
{ string contenido = File.ReadAllText(ruta_padre+"//"+"metodo.txt");
  string conexion="";
  for (int i = 0; i <= contenido.Length;i++ )
        { if(contenido[i]=='\r')
              {break;
```

```
              }
          conexion = conexion + contenido[i];
        }
  nombre_metodo_padre=conexion;
}
public void extraer_metodos_almacenados()
{int aux_cont=0;
 if(File.Exists(nombre_supersistema+"//"+"nombre_metodos.txt"))
  {string contenido =    File.ReadAllText(nombre_supersistema+"//"
   +"nombre_metodos.txt");
   string conexion="";
   for (int i = 0; i < contenido.Length;i++ )
        { if(contenido[i]=='\r'||contenido[i]=='\n')
              {if(conexion=="")
                 {aux_cont--;
                 }
                aux_cont++;
               conexion="";
              }
           else{conexion = conexion + contenido[i];
              }
          metodos_actuales[aux_cont]=conexion;
        }
  }
}
public void extraer_sistemas_almacenados()
{int aux_cont=0;
 if(File.Exists(nombre_supersistema+"//"+"sistemas.txt"))
   {string contenido =   File.ReadAllText(nombre_supersistema+"//"+
    "sistemas.txt");
    string conexion="";
    for (int i = 0; i < contenido.Length;i++ )
        { if(contenido[i]=='\r'||contenido[i]=='\n')
              {if(conexion=="")
                 {aux_cont--;
                 }
                aux_cont++;
               conexion="";
              }
           else{conexion = conexion + contenido[i];
              }
           sistemas_actuales[aux_cont]=conexion;
        }
  }
}
:}
```

```
/* ------------Declaracion de Terminales y No Terminales ----------- */

/*
Terminales (tokens regresados por el scanner).
Los terminales que no tienen valor son llamados primeros
y los que si tienen son  llamados posteriormente.
*/

terminal  AMPERSAND, MAS, MENOS, POR, DIVISION, IPAREN,
DPAREN, ILLAVE, DLLAVE, COMA;
terminal  PUNTO, DOSPUNTOS, ALL, AND, ANY, BETWEEN, BY,
VARCHAR, CHECK, CLASS, EXTEND;
terminal   COMPARACION, COMILLA, DATE, DECIMAL, DEFAULT, DISTINCT, DOUBLE,CHAR;
terminal   ESCAPE, EXISTS, FLOAT, FOREIGN,
FROM, GROUP, HAVING,SYSTEM,BYTE,BOOL,NULL;
terminal  IN, INTEGER, INTO, IS, KEY,
LIKE, METHOD, NOT;
terminal  NUMERIC, OR, PRECISION, PRIMARY, REAL,
REFERENCES, SET, SMALLINT,MUTO;
terminal  SOME, TIME, UNIQUE, VALUES,
PUNTOYCOMA, SHOW,AVG,MIN,MAX,SUM,COUNT,DELEO,EX,ADDO;
terminal   Object  NUMERO, NAME, HORA, FECHA, DROP,ELIGO;
terminal SUBSYSTEM, SUPERSYSTEM;


/*
  No terminales usados en la gramatica.
  Los Terminales que no tienen un valor entero están en la lista.
  Un tipo objeto significa que puede ser de cualquier tipo,es decir,
  no pertenece a un conjunto específico.
  Por lo tanto, podría ser un Integer o un String o lo que sea.
*/

non terminal Object  inicial, volver, definicion_sistema, herencia,
cuerpo_sistema, definicion_clase, listado_atributos;
non terminal Object  elemento_clase, def_columna, tipo_dato,
opc_def_columna, listado_opc_columna, def_interrelacion;
non terminal Object  listado_columna, condicion_busqueda,
predicado, comparacion_predicado, like_predicado, opt_escape;
non terminal Object  prueba_para_null, en_predicado, atom_commalist,
todo_o_algun_predicado, any_all_some, prueba_existencia, subquery;
non terminal Object  opt_all_distinct, atom, comparacion,
escalar_exp, function_ref, ammsc, usar_metodo, parametro_lista;
non terminal Object  addo_parametro, cuerpo_metodo, aux_definicion_funciones,
definicion_funciones, deleo_statement_searched, opt_ex_clause;
```

```
non terminal Object  ex_clause, addo_statement, opt_column_commalist,
value_or_query_spec, query_spec, selection, escalar_exp_commalist;
non terminal Object  addo_atom_commalist, addo_atom, muto_statement_searched,
assignment_commalist, assignment,column_ref,table,parameter,parameter_ref;
non terminal Object
consulta_clases,eligo_statement,drop_statement,vacio,columna,definicion_metodo;
 non terminal
entre_predicado,table_exp,opt_group_by_clause,opt_having_clause,column_ref_comma
list;
 non terminal
from_clause,table_ref_commalist,table_ref,cuerpo_supersistema,parametro_lista_me
todo,addo_parametro1,definicion_metodo_usar;
non terminal
cuerpo_metodo_usar,definicion_funciones_usar,addo_statement_usar,value_or_query_
spec_usar,addo_atom_commalist_usar,addo_atom_usar,atom_usar;

/* -------------Precedencia y asociacion ----------- */
/*
La línea Inferior siempre tendrá mayor precedencia que la línea anterior,
es decir, que POR y DIVISION debería tener una mayor precedencia que MAS,
MENOS
*/
precedence left MAS, MENOS;
precedence left POR, DIVISION;
precedence left AND, OR;


/* ----------------------------  Gramatica  -------------------- */

/* Producciones para gramatica :: SYDL */

inicial ::=  definicion_sistema  volver
           | consulta_clases PUNTOYCOMA
           | usar_metodo
           | SHOW NAME PUNTOYCOMA;

volver ::= vacio
         |inicial ;

definicion_sistema ::= SUPERSYSTEM {:my_parser.No_leer_conexion=1;:}
NAME:nombre_sistema  cuerpo_supersistema PUNTOYCOMA
{:my_parser.nombre_sistema=nombre_sistema.ToString();
  //verifica si ya hay creado un supersistema
  if(!File.Exists("conexion.txt"))
    {//verifica si el nombre de la base de datos ya fue creado
     if(!File.Exists(nombre_sistema.ToString()+".sqlite"))
```

```
{// Creamos la conexion a la BD.
 // El Data Source contiene la ruta del archivo de la BD
 my_parser.conexion =new SQLiteConnection("Data     Source="
 +nombre_sistema.ToString()+".sqlite;Version=3;New=True
 ;Compress=True;");
 my_parser.conexion.Open();
 System.IO.Directory.CreateDirectory(nombre_sistema.ToString());
 my_parser.ruta=nombre_sistema.ToString()+"/";
 // crea un archivo txt con el nombre del supersistema
 StreamWriter guardar_conexion = new
 StreamWriter("conexion.txt");
 guardar_conexion.WriteLine(my_parser.nombre_sistema);
 guardar_conexion.Close();
 //variable usada para validar el camino como supersistema
 my_parser.super_sistema=0;
 if(my_parser.super_sistema==0)
   {string contenido1 = File.ReadAllText("conexion.txt");
    string conexion="";
    for(int i = 0; i <= contenido1.Length;i++ )
       {if(contenido1[i]=='\r')
           {break;
           }
          conexion = conexion + contenido1[i];
       }
    my_parser.nombre_supersistema=conexion;
    my_parser.conexion =new SQLiteConnection("Data Source="
   +conexion.ToString()+".sqlite;Version=3;New=True
   ;Compress=True;");
    my_parser.conexion.Open();
    SQLiteCommand cmd = new SQLiteCommand("PRAGMA FOREIGN_KEYS  =
   1;", my_parser.conexion);
    cmd.ExecuteNonQuery();
   for(int i=0;i<my_parser.cont;i++)
       {cmd = new SQLiteCommand(my_parser.temp_clases[i],
        my_parser.conexion);
        cmd.ExecuteNonQuery();
        my_parser.sw = new  StreamWriter(my_parser.ruta+my_parser.temp_
        nombre_clases[i]+".txt");
        my_parser.sw.WriteLine(my_parser.temp_clases[i]);
        my_parser.sw.Close();
        my_parser.nombre_atributo="";
        //usado para verificar las tablas que se crearon
        if(File.Exists("tablas.txt"))
          {string contenido=File.ReadAllText("tablas.txt");
           my_parser.sw1 = new StreamWriter("tablas.txt");
           my_parser.sw1.WriteLine(contenido);
```

```
                    my_parser.sw1.Write(my_parser.temp_nombre_clases[i]);
                    my_parser.sw1.Close();
                   }
                 else{my_parser.sw1 = new StreamWriter("tablas.txt");
                      my_parser.sw1.Write(my_parser.temp_nombre_clases[i]);
                      my_parser.sw1.Close();
                     }
                }//end for
            my_parser.conexion.Close();
            cmd.Dispose();
           }
         }
       else{my_parser.ruta=nombre_sistema.ToString()+"/";
            MessageBox.Show("El nombre del suspersistema ya fue creado", "",
            MessageBoxButtons.OK);
             my_parser.super_sistema=1;
           }
       }
     else{//variable usada para validar la existencia correcta del suspersitema
          my_parser.super_sistema=1;
          //variable que guarda la direccion de
          my_parser.ruta=nombre_sistema.ToString()+"/";
          MessageBox.Show("El Supersystem ya fue creado", "",
          MessageBoxButtons.OK);
         }
     my_parser.cont=0;
     my_parser.No_leer_conexion=0;
:};

definicion_sistema ::= SYSTEM  NAME:nombre_sistema
{:my_parser.herencia_subsistema=0;
  my_parser.nombre_sistema=nombre_sistema.ToString();
  if(File.Exists("conexion.txt"))
    {//contiene texto del archivo con el nombre del supersistema
     string contenido = File.ReadAllText("conexion.txt");
     //contiene el nombre del suspersistema
     string conexion="";
     for(int i = 0; i <= contenido.Length;i++ )
        {if(contenido[i]=='\r')
           {break;
           }
         conexion = conexion + contenido[i];
        }
     my_parser.ruta=conexion+"//"+nombre_sistema.ToString();
     my_parser.ruta=conexion+"//"+nombre_sistema.ToString()+"/";
     my_parser.ruta_padre=conexion;
```

```
     my_parser.ruta_padre=conexion;
     my_parser.super_sistema=0;
     my_parser.nombre_supersistema=conexion;
    }
:}     herencia cuerpo_sistema PUNTOYCOMA
{:string contenido;
  my_parser.nombre_sistema=nombre_sistema.ToString();
  if(File.Exists("conexion.txt"))
    {//contiene texto del archivo con el nombre del supersistema
     contenido = File.ReadAllText("conexion.txt");
     //contiene el nombre del suspersistema
     string conexion="";
     for(int i = 0; i <= contenido.Length;i++ )
         {if(contenido[i]=='\r')
            {break;
            }
          conexion = conexion + contenido[i];
         }
     my_parser.ruta=conexion+"//"+nombre_sistema.ToString();
     if(Directory.Exists(my_parser.ruta))
       {MessageBox.Show("El Sistema YA EXISTE", "", MessageBoxButtons.OK);
        my_parser.sistema=1;
        my_parser.subsistema=1;
       }
     else{System.IO.Directory.CreateDirectory(my_parser.ruta);
          //ruta donde se almacena la estructura del sistema
          my_parser.ruta=conexion+"//"+nombre_sistema.ToString()+"/";
          my_parser.ruta_padre=conexion;
          my_parser.super_sistema=0;
          my_parser.nombre_supersistema=conexion;
          my_parser.conexion =new SQLiteConnection("Data
          Source="+conexion.ToString()+".sqlite;Version=3;New=True;
          Compress=True;");
          my_parser.conexion.Open();
          SQLiteCommand cmd = new SQLiteCommand("PRAGMA FOREIGN_KEYS  = 1;",
          my_parser.conexion);
          cmd.ExecuteNonQuery();

          for(int i=0;i<my_parser.cont;i++)
             {cmd = new SQLiteCommand(my_parser.temp_clases[i],
              my_parser.conexion);
              cmd.ExecuteNonQuery();
              my_parser.sw = new StreamWriter(my_parser.ruta+my_parser.
              temp_nombre_clases[i]+".txt");
              my_parser.sw.WriteLine(my_parser.temp_clases[i]);
              my_parser.sw.Close();
```

```
    my_parser.nombre_atributo="";
    //usado para verificar las tablas que se crearon
    if(File.Exists("tablas.txt"))
      {contenido=File.ReadAllText("tablas.txt");
       my_parser.sw1 = new StreamWriter("tablas.txt");
       my_parser.sw1.WriteLine(contenido);
       my_parser.sw1.Write(my_parser.temp_nombre_clases[i]
       +my_parser.clases_alteradas[i]);
       my_parser.sw1.Close();
      }
    else{my_parser.sw1 = new StreamWriter("tablas.txt");
         my_parser.sw1.Write(my_parser.temp_nombre_clases[i]+
         my_parser.clases_alteradas[i]);
          my_parser.sw1.Close();
        }
    }//end for
my_parser.conexion.Close();
cmd.Dispose();
String direcccion="";
my_parser.extraer_supersistema();
if(my_parser.subsistema==0)
  {if(my_parser.ejecutar_metodo==0)
     {if(my_parser.herencia_subsistema==1)
        {direcccion=my_parser.nombre_supersistema+"//"+my_parser
         .nombre_sistema+".txt";
        }
      else{direcccion =my_parser.ruta+my_parser.nombre_metodo+".txt";
          }
      StreamWriter msave = new StreamWriter(direcccion);
      if(File.Exists(my_parser.ruta_padre+"//"+my_parser.
         nombre_metodo+".txt"))
        {contenido=File.ReadAllText(my_parser.ruta_padre+"//"
         +my_parser.nombre_metodo+".txt");
         string nuevo_metodo="";
         char comparar=')';
         for(int i=0;i<contenido.Length;i++)
            {if(contenido[i]==comparar)
               {nuevo_metodo=nuevo_metodo+","+my_parser.lista;
                comparar='?';
               }
             if(contenido[i]=='}')
               {nuevo_metodo=nuevo_metodo+"\r\n"+my_parser.funciones;
               }
             nuevo_metodo=nuevo_metodo+contenido[i];
            }
         msave.WriteLine(nuevo_metodo);
```

```
                    }
                else{msave.WriteLine(my_parser.aux_metodo);
                    }
                msave.Close();
                StreamWriter guardar_metodo = new StreamWriter(my_parser.ruta
                +"//"+"metodo.txt");
                guardar_metodo.WriteLine(my_parser.nombre_metodo);
                guardar_metodo.Close();
               }
            //usado para guardar los nombre metodos que se crearon
           if(File.Exists(my_parser.ruta_padre+"//nombre_metodos.txt"))
             {contenido=File.ReadAllText(my_parser.ruta_padre+
              "//nombre_metodos.txt");
              my_parser.sw1 = new StreamWriter(my_parser.ruta_padre
              +"//nombre_metodos.txt");
              my_parser.sw1.WriteLine(contenido);
              my_parser.sw1.Write(my_parser.nombre_metodo);
              my_parser.sw1.Close();
             }
            else{my_parser.sw1 = new StreamWriter(my_parser.ruta_padre
                +"//nombre_metodos.txt");
                my_parser.sw1.Write(my_parser.nombre_metodo);
                my_parser.sw1.Close();
               }
            }
          }
     my_parser.ruta=conexion+"//"+nombre_sistema.ToString()+"/";
     my_parser.ruta_padre=conexion;
     my_parser.super_sistema=0;
    }
else{
     MessageBox.Show("El suspersistema NO HA SIDO CREADO", "",
     MessageBoxButtons.OK);
     my_parser.super_sistema=1;
    }
//usado para guardar los sistemas que se crearon
if(File.Exists(my_parser.ruta_padre+"//sistemas.txt"))
  {contenido=File.ReadAllText(my_parser.ruta_padre+"//sistemas.txt");
   my_parser.sw1 = new StreamWriter(my_parser.ruta_padre+"//sistemas.txt");
   my_parser.sw1.WriteLine(contenido);
   my_parser.sw1.Write(my_parser.nombre_sistema);
   my_parser.sw1.Close();
  }
else{my_parser.sw1 = new StreamWriter(my_parser.ruta_padre+"//sistemas.txt");
     my_parser.sw1.Write(my_parser.nombre_sistema);
     my_parser.sw1.Close();
```

```
    }
my_parser.cont=0;
:} ;

definicion_sistema ::= SUBSYSTEM NAME:nombre_sistema herencia:herencia
{:
my_parser.herencia_subsistema=1;
my_parser.sistema=0;
my_parser.nombre_sistema=nombre_sistema.ToString();
if(File.Exists("conexion.txt"))
  {string contenido = File.ReadAllText("conexion.txt");
   string conexion="";
   for(int i = 0; i <= contenido.Length;i++ )
      {if(contenido[i]=='\r')
         {break;
         }
       conexion = conexion + contenido[i];
      }
   my_parser.ruta=conexion+"//"+herencia.ToString()+"//"+nombre_sistema
  .ToString();
  my_parser.ruta=conexion+"//"+herencia.ToString()+"//"+nombre_sistema
  .ToString()+"/";
  my_parser.ruta_padre=conexion+"//"+herencia.ToString();
  if(Directory.Exists(conexion+"//"+herencia.ToString()))
    {if(!File.Exists(conexion+"//"+nombre_sistema.ToString()+".txt"))
       {my_parser.ruta=conexion+"//"+herencia.ToString()+"//"+nombre_sistema
        .ToString()+"/";
        my_parser.ruta_padre=conexion+"//"+herencia.ToString();
        my_parser.herencia_subsistema=1;
       my_parser.super_sistema=0;
       my_parser.nombre_supersistema=conexion;
       }
    }
  }
:}  cuerpo_sistema  PUNTOYCOMA
{:
my_parser.herencia_subsistema=1;
my_parser.sistema=0;
my_parser.nombre_sistema=nombre_sistema.ToString();
string contenido;
if(File.Exists("conexion.txt"))
  {contenido = File.ReadAllText("conexion.txt");
   string conexion="";
   for(int i = 0; i <= contenido.Length;i++ )
      {if(contenido[i]=='\r')
         {break;
```

```
      }
     conexion = conexion + contenido[i];
    }
my_parser.ruta=conexion+"//"+herencia.ToString()+"//"+nombre_sistema
.ToString();
if(Directory.Exists(conexion+"//"+herencia.ToString()))
  {if(!File.Exists(conexion+"//"+nombre_sistema.ToString()+".txt"))
     {System.IO.Directory.CreateDirectory(my_parser.ruta);
      my_parser.ruta=conexion+"//"+herencia.ToString()+"//"+nombre_sistema
      .ToString()+"/";
      my_parser.ruta_padre=conexion+"//"+herencia.ToString();
      my_parser.herencia_subsistema=1;
      my_parser.super_sistema=0;
      my_parser.nombre_supersistema=conexion;
      my_parser.conexion =new SQLiteConnection("Data Source="+conexion.
      ToString()+".sqlite;Version=3;New=True;Compress=True;");
      my_parser.conexion.Open();
      SQLiteCommand cmd = new SQLiteCommand("PRAGMA FOREIGN_KEYS  = 1;",
      my_parser.conexion);
      cmd.ExecuteNonQuery();
      for(int i=0;i<my_parser.cont;i++)
        {cmd = new SQLiteCommand(my_parser.temp_clases[i],
         my_parser.conexion);
         cmd.ExecuteNonQuery();
         my_parser.sw = new StreamWriter(my_parser.ruta
         +my_parser.temp_nombre_clases[i]+".txt");
         my_parser.sw.WriteLine(my_parser.temp_clases[i]);
         my_parser.sw.Close();
         my_parser.nombre_atributo="";
         //usado para verificar las tablas que se crearon
         if(File.Exists("tablas.txt"))
           {contenido=File.ReadAllText("tablas.txt");
            my_parser.sw1 = new StreamWriter("tablas.txt");
            my_parser.sw1.WriteLine(contenido);
            my_parser.sw1.Write(my_parser.temp_nombre_clases[i]+
            my_parser.clases_alteradas[i]);
            my_parser.sw1.Close();
           }
         else{my_parser.sw1 = new StreamWriter("tablas.txt");
              my_parser.sw1.Write(my_parser.temp_nombre_clases[i]
              +my_parser.clases_alteradas[i]);
              my_parser.sw1.Close();
             }
         }//end for
      my_parser.conexion.Close();
      cmd.Dispose();
```

```
String direcccion="";
my_parser.extraer_supersistema();
if(my_parser.subsistema==0)
  {if(my_parser.ejecutar_metodo==0)
     {if(my_parser.herencia_subsistema==1)
        {direcccion=my_parser.nombre_supersistema+"//"
         +my_parser.nombre_sistema+".txt";
        }
      else{direcccion =my_parser.ruta+my_parser.nombre_metodo+".txt";
           if(File.Exists("tablas.txt"))
             {contenido=File.ReadAllText("tablas.txt");
              my_parser.sw1 = new StreamWriter("tablas.txt");
              my_parser.sw1.WriteLine(contenido);
              my_parser.sw1.Write(my_parser.nombre_clase);
              my_parser.sw1.Close();
             }
           else{my_parser.sw1 = new StreamWriter("tablas.txt");
                my_parser.sw1.Write(my_parser.nombre_clase);
                my_parser.sw1.Close();
               }
           }
       StreamWriter msave = new StreamWriter(direcccion);
       if(File.Exists(my_parser.ruta_padre+"//"+my_parser
          .nombre_metodo+".txt"))
         {contenido=File.ReadAllText(my_parser.ruta_padre+"//"
          +my_parser.nombre_metodo+".txt");
          string nuevo_metodo="";
          char comparar=')';
          for(int i=0;i<contenido.Length;i++)
            {if(contenido[i]==comparar)
               {nuevo_metodo=nuevo_metodo+","+my_parser.lista;
                comparar='?';
               }
             if(contenido[i]=='}')
               {nuevo_metodo=nuevo_metodo+"\r\n"+my_parser.funciones;
               }
             nuevo_metodo=nuevo_metodo+contenido[i];
            }
          msave.WriteLine(nuevo_metodo);
         }
  else{msave.WriteLine(my_parser.aux_metodo);
      }
  msave.Close();
  StreamWriter guardar_metodo=new StreamWriter(my_parser.ruta+"//"
  +"metodo.txt");
  guardar_metodo.WriteLine(my_parser.nombre_metodo);
```

```
          guardar_metodo.Close();
         //usado para guardar los nombre metodos que se crearon
         if(File.Exists(conexion+"//nombre_metodos.txt"))
           {contenido=File.ReadAllText(conexion+"//nombre_metodos.txt");
            my_parser.sw1 = new StreamWriter(conexion+"//nombre_metodos.txt");
            my_parser.sw1.WriteLine(contenido);
            my_parser.sw1.Write(my_parser.nombre_sistema);
            my_parser.sw1.Close();
           }
        else{my_parser.sw1 = new StreamWriter(conexion+"//nombre_metodos.txt");
             my_parser.sw1.Write(my_parser.nombre_sistema);
             my_parser.sw1.Close();
            }
         }//fin comparacion ejecutar metodo
       }//fin comparacion subsistema=0
     }//fin comparacion existencia subsistema
   else{ MessageBox.Show("El subsistema YA EXISTE", "", MessageBoxButtons.OK);
          my_parser.sistema=1;
         my_parser.subsistema=1;
       }
   }
  else{ MessageBox.Show("El sistema NO EXISTE", "", MessageBoxButtons.OK);
        my_parser.sistema=1;
      }
  }
else{MessageBox.Show("El suspersistema NO HA SIDO CREADO", "",
     MessageBoxButtons.OK);
     my_parser.super_sistema=1;
    }
my_parser.cont=0;
:};

herencia::=EXTEND NAME:nombre_sistema_padre
{: RESULT=nombre_sistema_padre.ToString();
   my_parser.nombre_padre=nombre_sistema_padre.ToString();
:} ;

cuerpo_supersistema::= ILLAVE definicion_clase DLLAVE;

cuerpo_sistema::= ILLAVE definicion_clase definicion_metodo DLLAVE;

definicion_clase::= {:RESULT="";:}
                   |CLASS NAME:nombre_clase
{:my_parser.nombre_clase = nombre_clase.ToString();
  //sentencias para guardar lo escrito en  text 1
  my_parser.alter=0;
```

```
  my_parser.aux_clase="CREATE TABLE " + nombre_clase.ToString()+" (";
  if(File.Exists(my_parser.ruta_padre+"//"+nombre_clase+".txt"))
    {my_parser.alter=1;
    }
:}  ILLAVE listado_atributos:atributos DLLAVE
{:
//extrae la conexion
if(File.Exists(my_parser.ruta_padre+"//"+nombre_clase+".txt"))
  {my_parser.aux_clase="ALTER TABLE "+nombre_clase.ToString()+" ADD
   "+atributos.ToString();
   my_parser.extraer_metodos_almacenados();
   my_parser.extraer_sistemas_almacenados();
   int temp_cont=-1;
   int temp2_cont=0;
   string recorrer=my_parser.nombre_supersistema;
   //busca en todos los metodos y añade en la insercion un nuevo campo
   do{if(temp_cont!=-1)
        {recorrer=my_parser.nombre_supersistema+"//"+my_parser.
         sistemas_actuales[temp_cont];
        }
      do{if(File.Exists(recorrer+"//"+my_parser.metodos_actuales
           [temp2_cont]+".txt"))
          {TextReader   linea = new StreamReader(recorrer+"//"+my_parser.
           metodos_actuales[temp2_cont]+".txt");
           string contenido = linea.ReadLine();
           string temporal = "";
           while(contenido != null)
                {if(contenido.StartsWith("addo into " + nombre_clase))
                   {contenido = contenido.Substring(0, contenido.IndexOf(')'))
                     + ",'');";
                    temporal = temporal + contenido+"\r\n";
                   }
                 else{temporal = temporal + contenido+"\r\n";
                     }
                 contenido = linea.ReadLine();
                }
           linea.Close();
           StreamWriter sw = new StreamWriter(recorrer+"//"+my_parser.
           metodos_actuales[temp2_cont]+".txt");
           sw.WriteLine(temporal);
           sw.Close();
          }
         temp2_cont++;
        }while(my_parser.metodos_actuales[temp2_cont]!=null);
       temp2_cont=0;
       temp_cont++;
```

```
      }while(my_parser.sistemas_actuales[temp_cont]!=null);
  }
else{my_parser.aux_clase=my_parser.aux_clase+atributos.ToString()+");";
    }
my_parser.temp_clases[my_parser.cont]=my_parser.aux_clase;
my_parser.clases_alteradas[my_parser.cont]=my_parser.especificacion;
my_parser.temp_nombre_clases[my_parser.cont]=nombre_clase.ToString();
my_parser.cont++;
my_parser.especificacion="";
my_parser.nombre_atributo="";
:} definicion_clase ;

listado_atributos::={:RESULT="";:}
                   |elemento_clase:clase {:RESULT=clase.ToString();:}
                   |listado_atributos:atributos COMA  elemento_clase:clase
                   {:RESULT=atributos.ToString()+","+clase.ToString();
                     if(File.Exists(my_parser.ruta_padre+"//"+
                        my_parser.nombre_clase+".txt"))
                       {RESULT=atributos.ToString()+" "+clase.ToString();
                       }
                      else{RESULT=atributos.ToString()+","+clase.ToString();
                          }
                   :};

elemento_clase::=def_columna:columna         {:RESULT=columna.ToString();:}
                |def_interrelacion:def_int   {:RESULT=def_int.ToString();:};

def_columna::= columna:nom tipo_dato:dato opc_def_columna:def_col
              {:RESULT=nom.ToString()+dato.ToString()+def_col.ToString(); :};

columna::= NAME:nombre_atributo  {:RESULT=nombre_atributo.ToString()+" ";  :};

tipo_dato::=VARCHAR:tipo IPAREN NUMERO:numero DPAREN
            {: RESULT="varchar ("+numero.ToString()+")";:}
            |NUMERIC:numero           {: RESULT="numeric ";:}
            |NUMERIC IPAREN NUMERO:numero DPAREN
            {: RESULT="numeric ("+numero.ToString()+")";:}
            |NUMERIC IPAREN NUMERO:numero COMA NUMERO:numero1 DPAREN
            {:RESULT="numeric ("+numero.ToString()+","+numero1.ToString()+")";:}
            |DECIMAL:numero           {: RESULT="decimal ";:}
            |DECIMAL IPAREN NUMERO:numero DPAREN
            {: RESULT="decimal ("+numero.ToString()+")";:}
            |DECIMAL IPAREN NUMERO:numero COMA NUMERO:numero1 DPAREN
            {:RESULT="decimal ("+numero.ToString()+","+numero1.ToString()+")";:}
            |INTEGER:numero           {: RESULT="integer ";:}
            |SMALLINT:numero          {: RESULT="smallint ";:}
```

```
            |FLOAT:numero             {: RESULT="float ";:}
            |FLOAT IPAREN NUMERO:numero DPAREN
            {: RESULT="float ("+numero.ToString()+")";:}
            |REAL:numero              {: RESULT="real ";:}
            |CHAR:numero              {: RESULT="char ";:}
            |DOUBLE PRECISION         {: RESULT="double precision ";:}
            |DATE:numero              {: RESULT="date ";:}
            |TIME:numero              {: RESULT="time ";:}
            |BYTE:numero              {: RESULT="byte ";:}
            |BOOL:numero              {: RESULT="bool";:}  ;

opc_def_columna::= opc_def_columna:opc  listado_opc_columna:columna
                   {: RESULT=opc.ToString()+columna.ToString();:}
                  |{:RESULT="";:}     ;

listado_opc_columna::= NOT NULL              {:RESULT="not null";:}
                       |NOT NULL UNIQUE      {:RESULT="not null unique";:}
                       |NOT NULL PRIMARY KEY {:RESULT="not null primary key";:}
                       |DEFAULT NUMERO:numero
                        {:RESULT="default "+numero.ToString();:}
                       |DEFAULT NULL         {:RESULT="default null";:}
                       |DEFAULT NAME:nombre
                        {:RESULT="default "+nombre.ToString();:}
                       |CHECK IPAREN condicion_busqueda:busqueda DPAREN
                        {:RESULT="check ("+busqueda.ToString()+")";:}
                       |REFERENCES NAME:nombre_clase
                         {:RESULT="references "+nombre_clase.ToString();:}
                       |REFERENCES NAME:nombre_clase IPAREN
                        listado_columna:columna DPAREN
                        {:RESULT="references "+nombre_clase.ToString()
                          +"("+columna.ToString()+")";:}   ;

def_interrelacion::= UNIQUE IPAREN listado_columna:columna DPAREN
                     {:RESULT="unique ("+columna.ToString()+") ";:}
                     |PRIMARY KEY IPAREN listado_columna:columna DPAREN
                     {:RESULT="primary key ("+columna.ToString()+") ";:}
                     |FOREIGN KEY IPAREN listado_columna:columna DPAREN
                      REFERENCES NAME:nombre_clase
                      {:if(my_parser.alter==0)
                         {RESULT="foreign key ("+columna.ToString()+") "+"
                           references "+nombre_clase.ToString();
                          }
                        else{RESULT="references "+nombre_clase.ToString();
                             my_parser.especificacion=" alterado con referencia
                             a "+nombre_clase.ToString();
                            }
```

```
                        my_parser.alter=0;
                       :}
                      |FOREIGN KEY IPAREN listado_columna:columna DPAREN
                       REFERENCES NAME:nombre_clase IPAREN
                       listado_columna:columna1 DPAREN
                       {:if(my_parser.alter==0)
                           {RESULT="foreign key ("+columna.ToString()+") "+"
                            references "+nombre_clase.ToString()+"
                            ("+columna1.ToString()+" )";
                           }
                         else{RESULT=" references "+nombre_clase.ToString()+"
                             ("+columna1.ToString()+" )";
                             my_parser.especificacion=" alterado con referencia a
                             "+nombre_clase.ToString();
                             }
                         my_parser.alter=0;
                       :}
                      |CHECK IPAREN condicion_busqueda:busqueda DPAREN
                       {:RESULT="chek ("+busqueda.ToString()+") ";:}  ;

listado_columna::= NAME:nombre      {:RESULT=nombre.ToString();:}
                  |listado_columna:columna COMA NAME:nombre
                   {:RESULT=columna.ToString()+","+nombre.ToString();:} ;

condicion_busqueda::=condicion_busqueda:condicion OR
                      condicion_busqueda:condicion1
                      {:RESULT=condicion.ToString()+" or
                       "+condicion1.ToString();:}
                     |condicion_busqueda:condicion AND
                      condicion_busqueda:condicion1
                      {:RESULT=condicion.ToString()+" and
                        "+condicion1.ToString();:}
                     |NOT condicion_busqueda:condicion
                     {:RESULT=" not "+condicion.ToString();:}
                     |IPAREN condicion_busqueda:condicion DPAREN
                     {:RESULT="("+condicion.ToString()+") ";:}
                     |predicado:predi
                     {:RESULT=predi.ToString();:} ;

predicado::=  comparacion_predicado:predicado  {:RESULT=predicado.ToString();:}
             |like_predicado:predicado         {:RESULT=predicado.ToString();:}
             |entre_predicado:predicado        {:RESULT=predicado.ToString();:}
             |en_predicado:predicado           {:RESULT=predicado.ToString();:}
             |prueba_para_null:predicado       {:RESULT=predicado.ToString();:}
             |todo_o_algun_predicado:predicado {:RESULT=predicado.ToString();:}
             |prueba_existencia:predicado      {:RESULT=predicado.ToString();:};
```

```
comparacion_predicado::=escalar_exp:esc_exp1 COMPARACION:comp
                        escalar_exp:esc_exp2

                         {:RESULT=esc_exp1.ToString()+comp.ToString()+
                           esc_exp2.ToString();:}
                        |escalar_exp:esc_exp COMPARACION:comp  subquery:subcons
                         {:RESULT=esc_exp.ToString()+comp.ToString()+
                           subcons.ToString();:} ;

entre_predicado::= escalar_exp:esc_exp1 NOT BETWEEN escalar_exp:esc_exp2 AND
                   escalar_exp:esc_exp3
                   {:RESULT=esc_exp1.ToString()+" not between
                     "+esc_exp2.ToString()+" and "+esc_exp3.ToString();:}
                   |escalar_exp:esc_exp1 BETWEEN escalar_exp:esc_exp2 AND
                    escalar_exp:esc_exp3
                    {:RESULT=esc_exp1.ToString()+" between
                      "+esc_exp2.ToString()+" and "+esc_exp3.ToString();:};

like_predicado::= escalar_exp:esc_exp NOT LIKE atom:atom opt_escape:opt
                   {:RESULT=esc_exp.ToString()+" not like "+ atom.ToString()+" "
                    +opt.ToString();:}
                  |escalar_exp:esc_exp LIKE atom:atom opt_escape:opt
                  {:RESULT=esc_exp.ToString()+" like "+ atom.ToString()+"
                    "+opt.ToString();:} ;

opt_escape::= {:RESULT="";:}
             |ESCAPE atom:atom {:/*NO SABEMOS PA QUE SIRBE*/ RESULT="escape
                                 "+atom.ToString();:}  ;

prueba_para_null::= column_ref:refe IS NOT NULL
                     {:RESULT=refe.ToString()+" is not null";:}
                    |column_ref:refe IS NULL
                     {:RESULT=refe.ToString()+" is null";:};

column_ref::= table:nombre  {:RESULT=nombre.ToString();:}
             |NAME:nombre1 PUNTO NAME:nombre2 PUNTO NAME:nombre3
              {:RESULT=nombre1.ToString()+"."+nombre2.ToString()+
                "."+nombre3.ToString();:};

table::= NAME:esc_exp1  {: RESULT=esc_exp1.ToString();  :}
        |NAME:esc_exp1 PUNTO NAME
         {:RESULT=esc_exp1.ToString()+"."+esc_exp1.ToString();:} ;

en_predicado::= escalar_exp:exp NOT IN IPAREN subquery:sub DPAREN
                {: RESULT=exp.ToString()+" not in ("+sub.ToString()+") ";  :}
```

```
                |escalar_exp:exp IN IPAREN subquery:sub DPAREN
                  {: RESULT=exp.ToString()+" in ("+sub.ToString()+") ";  :}
                |escalar_exp:exp NOT IN IPAREN atom_commalist:atom DPAREN
                  {: RESULT=exp.ToString()+" not in ("+atom.ToString()+") ";  :}
                |escalar_exp:exp IN IPAREN atom_commalist:atom DPAREN
                 {: RESULT=exp.ToString()+" in ("+atom.ToString()+") ";:} ;

atom_commalist::= atom:atom            {:RESULT=atom.ToString();:}
                  |atom_commalist:atom COMA atom:atom1
                   {:RESULT=atom.ToString()+","+atom1.ToString();:} ;

todo_o_algun_predicado::= escalar_exp:exp comparacion:comp any_all_some:any
                          subquery:sub
                           {:RESULT=exp.ToString()+comp.ToString()
                            +any.ToString()+" "+sub.ToString();:} ;

any_all_some::= ANY  {:RESULT="any";:}
               |ALL  {:RESULT="all";:}
               |SOME {:RESULT="some";:} ;

prueba_existencia::= EXISTS subquery:sub   {:RESULT="exist "+sub.ToString();:};

subquery  ::= IPAREN ELIGO opt_all_distinct:opt_all  selection:sel
              table_exp:tab_exp DPAREN
              {:if(my_parser.ejecutar_metodo==1)
                  {RESULT="(select"+opt_all.ToString()+" "+sel.ToString()+"
                     "+tab_exp.ToString()+")";
                   }
                 else{RESULT="(eligo"+opt_all.ToString()+" "+sel.ToString()+"
                      "+tab_exp.ToString()+")";
                      }
               :};

opt_all_distinct::=          {:RESULT="";:}
                   |ALL      {:RESULT=" all ";:}
                   |DISTINCT {:RESULT=" distinct ";:};

atom::=  parameter_ref:refe            {:RESULT=refe.ToString();:}
        |NUMERO:numero                  {:RESULT=numero.ToString();:}
        |AMPERSAND NAME:nombre          {:RESULT="&"+nombre.ToString();:}
        |COMILLA NAME:nombre   COMILLA {:RESULT="'"+nombre.ToString()+"'";:}
        |COMILLA FECHA:nombre  COMILLA {:RESULT="'"+nombre.ToString()+"'";:}
        |COMILLA HORA:nombre   COMILLA {:RESULT="'"+nombre.ToString()+"'";:}
        |AMPERSAND NUMERO:nombre       {:RESULT=nombre.ToString(); :}
        |COMILLA COMILLA               {:RESULT="''"; :}
        ;
```

```
comparacion::= COMPARACION:comp   {:RESULT=comp.ToString();:};

escalar_exp::= escalar_exp:exp MAS escalar_exp:exp1
               {:RESULT=exp.ToString()+" + "+exp1.ToString();:}
              |escalar_exp:exp MENOS escalar_exp:exp1
               {:RESULT=exp.ToString()+"-"+exp1.ToString();:}
              |escalar_exp:exp POR escalar_exp:exp1
               {:RESULT=exp.ToString()+"*"+exp1.ToString();:}
              |escalar_exp:exp DIVISION
               escalar_exp:exp1{:RESULT=exp.ToString()+"/"+exp1.ToString();:}
               /*|escalar_exp PORCENTAJE escalar_exp*/
              |MAS escalar_exp:exp           {:RESULT="+"+exp.ToString();:}
              |MENOS escalar_exp:exp         {:RESULT="-"+exp.ToString();:}
              |atom:atm                      {:RESULT=atm.ToString();:}
              |column_ref:refe               {:RESULT=refe.ToString();:}
               /*  |NAME:nombre_class PUNTO NAME:nombre_class*/
              |function_ref:refe             {:RESULT=refe.ToString();:}
              |IPAREN escalar_exp:exp DPAREN {:RESULT="("+exp.ToString()+")";:};

function_ref::= ammsc:am IPAREN POR DPAREN
                {:RESULT=am.ToString()+"(*)";:}
                |ammsc:am IPAREN DISTINCT column_ref:refe DPAREN
                {:RESULT=am.ToString()+"(distinct"+refe.ToString()+")";:}
                |ammsc:am IPAREN ALL escalar_exp:exp DPAREN
                {:RESULT=am.ToString()+"(all"+exp.ToString()+")";:}
                |ammsc:am IPAREN escalar_exp:exp DPAREN
                 {:RESULT=am.ToString()+"("+exp.ToString()+")";:} ;

ammsc::=  AVG   {:RESULT="avg";:}
         |MIN   {:RESULT="min";:}
         |MAX   {:RESULT="max";:}
         |SUM   {:RESULT="sum";:}
         |COUNT {:RESULT="count";:}  ;

definicion_metodo::= {:RESULT="";:}
                     |METHOD NAME:nombre_metodo
                      {:my_parser.nombre_metodo=nombre_metodo.ToString();:}
                      IPAREN parametro_lista:list DPAREN cuerpo_metodo:cuerpo
                      {:RESULT="method "+nombre_metodo.ToString()+"("
                        +list.ToString()+")"+cuerpo.ToString();
                        my_parser.aux_metodo="method
                        "+nombre_metodo.ToString()+"("+list.ToString()+
                        ")"+cuerpo.ToString();
                        my_parser.lista=list.ToString();
                      :};
```

```
parametro_lista::= addo_parametro:addo    {:RESULT=addo.ToString();:}
                  |parametro_lista:lista COMA addo_parametro:parametro
                  {:RESULT=lista.ToString()+","+parametro.ToString();:};

addo_parametro::=             {:RESULT="";:}
                 |NAME:nombre {:RESULT=nombre.ToString();:}
                 |NULL        {:RESULT="null";:};

cuerpo_metodo::= ILLAVE definicion_funciones:funciones DLLAVE
                 {:RESULT="\r\n{"+funciones.ToString()+"}"+"\n";
                   my_parser.funciones=funciones.ToString();
                 :};

aux_definicion_funciones::= definicion_funciones:funciones
                            {:RESULT=funciones.ToString();:};

definicion_funciones::={:RESULT="";:}
                       |deleo_statement_searched:deleo PUNTOYCOMA
                        aux_definicion_funciones:funciones
                        {:RESULT="\n"+deleo.ToString()+"; \n
                           "+funciones.ToString();
                          if(my_parser.ejecutar_metodo==1)
                           {string contenido = File.ReadAllText("conexion.txt");
                            string conexion="";
                            for(int i = 0; i <= contenido.Length;i++ )
                               {if(contenido[i]=='\r')
                                   {break;
                                   }
                                 conexion = conexion + contenido[i];
                                }
                            if(File.Exists(conexion+".sqlite"))
                              {// Creamos la conexion a la BD.
                               my_parser.conexion =
                                new SQLiteConnection("Data Source="+conexion+
                                ".sqlite;Version=3;New=True;Compress=True;");
                                my_parser.conexion.Open();
                                string creacion = deleo.ToString();
                                SQLiteCommand cmd = new SQLiteCommand("PRAGMA
                                FOREIGN_KEYS  = 1;",my_parser.conexion);
                                cmd.ExecuteNonQuery();
                                 cmd=new SQLiteCommand(creacion,
                                my_parser.conexion);
                                cmd.ExecuteNonQuery();
                                my_parser.conexion.Close();
                                cmd.Dispose();
                              }
```

```
     else{MessageBox.Show("El suspersistema NO HA
          SIDO CREADO", "", MessageBoxButtons.OK);
         }
     }:}
|addo_statement:addo PUNTOYCOMA
 aux_definicion_funciones:funciones
 {:RESULT="\n"+addo.ToString()+"; \n
   "+funciones.ToString();
  if(my_parser.ejecutar_metodo==1)
    {string contenido = File.ReadAllText("conexion.txt");
     string conexion="";
     for(int i = 0; i <= contenido.Length;i++ )
        {if(contenido[i]=='\r')
          {break;
          }
         conexion = conexion + contenido[i];
        }
      if(File.Exists(conexion+".sqlite"))
        {my_parser.conexion =new SQLiteConnection("Data
         Source="+conexion+".sqlite;Version=3;
         New=True;Compress=True;");
         my_parser.conexion.Open();
         string creacion = addo.ToString();
         SQLiteCommand cmd = new SQLiteCommand("PRAGMA
         FOREIGN_KEYS  = 1;",my_parser.conexion);
         cmd.ExecuteNonQuery();
         cmd = new SQLiteCommand
         (creacion,my_parser.conexion);
         cmd.ExecuteNonQuery();
         my_parser.conexion.Close();
         cmd.Dispose();
        }
     else{MessageBox.Show("El suspersistema
           NO HA SIDO CREADO", "", MessageBoxButtons.OK);
          }
    }
   :}
 |muto_statement_searched:muto PUNTOYCOMA
  aux_definicion_funciones:funciones
  {:RESULT="\n"+muto.ToString()+"; \n
    "+funciones.ToString();
    if(my_parser.ejecutar_metodo==1)
      {string contenido=File.ReadAllText("conexion.txt");
        string conexion="";
        for(int i = 0; i <= contenido.Length;i++ )
           {if(contenido[i]=='\r')
```

```
                                        {break;
                                        }
                                       conexion = conexion + contenido[i];
                                     }
                                   if(File.Exists(conexion+".sqlite"))
                                     {my_parser.conexion =new SQLiteConnection("Data
                                      Source="+conexion+".sqlite;Version=3;
                                      New=True;Compress=True;");
                                      my_parser.conexion.Open();
                                      string creacion = muto.ToString();
                                      SQLiteCommand cmd = new SQLiteCommand("PRAGMA
                                      FOREIGN_KEYS  = 1;",my_parser.conexion);
                                      cmd.ExecuteNonQuery();
                                      cmd = new SQLiteCommand
                                      (creacion,my_parser.conexion);
                                      cmd.ExecuteNonQuery();
                                      my_parser.conexion.Close();
                                      cmd.Dispose();
                                     }
                                   else{MessageBox.Show("El suspersistema NO
                                        HA SIDO CREADO", "", MessageBoxButtons.OK);
                                       }
                                  }
                                 :};

deleo_statement_searched::=DELEO FROM NAME:nombre_clase opt_ex_clause:clause
                           {:if(my_parser.ejecutar_metodo==1)
                               {RESULT="delete from
                                 "+nombre_clase.ToString()+clause.ToString();
                               }
                             else{RESULT="deleo from
                                  "+nombre_clase.ToString()+clause.ToString();
                                 }
                            :};

opt_ex_clause::={:RESULT="";:}
                |ex_clause:clause  {:RESULT=clause.ToString();:};

ex_clause::= EX  condicion_busqueda:busqueda
             {:if(my_parser.ejecutar_metodo==1)
                 {RESULT=" where "+busqueda.ToString();
                 }
               else{RESULT=" ex "+busqueda.ToString();
                   }
             :};
```

```
addo_statement::= ADDO INTO NAME:nombre opt_column_commalist:opt_col
                  value_or_query_spec:value
                  {:if(my_parser.ejecutar_metodo==1)
                      {RESULT="insert into "+nombre.ToString()+"
                       "+opt_col.ToString()+" "+value.ToString();
                       }
                     else{RESULT="addo into "+nombre.ToString()+"
                          "+opt_col.ToString()+" "+value.ToString();
                          }
                    :};

opt_column_commalist::={:RESULT="";:}
                       |IPAREN listado_columna:lista DPAREN
                        {:RESULT="("+lista.ToString()+")";:};

value_or_query_spec::= VALUES IPAREN addo_atom_commalist:addo DPAREN
                        {: RESULT="values ("+addo.ToString()+")";    :}
                       |query_spec:query    {:  RESULT=query.ToString(); :};

query_spec::= ELIGO opt_all_distinct:opt_all selection:sele NAME:nombre
              table_exp:table
              {: RESULT="select "+opt_all.ToString()+" "+sele.ToString()+"
                 "+nombre.ToString()+" "+table.ToString();:};

table_exp::= from_clause:from opt_ex_clause:where opt_group_by_clause:group
             opt_having_clause:having
             {:RESULT=from.ToString()+" "+where.ToString()+"
               "+group.ToString()+" "+having.ToString();:} ;

from_clause::= FROM table_ref_commalist:tab_ref
               {:RESULT="from "+tab_ref.ToString();:} ;

table_ref_commalist::=table_ref:tab_ref  {:RESULT=tab_ref.ToString();:}
                      |table_ref_commalist:tab_ref_com COMA table_ref:tab_ref
                      {:RESULT=tab_ref_com.ToString()+","+tab_ref.ToString();:};

table_ref ::= table:nombre  {:RESULT=nombre.ToString();:} ;

opt_group_by_clause ::=  {:RESULT="";:}
                         |GROUP BY column_ref_commalist:col
                          {:RESULT="group by "+ col.ToString();:};

opt_having_clause::={:RESULT="";:}
                    |HAVING condicion_busqueda:cond
                     {:RESULT="having "+cond.ToString();:};
```

```
column_ref_commalist::= column_ref:col   {:RESULT=col.ToString();:}
                       |column_ref_commalist:col  COMA column_ref:col_ref
                        {: RESULT=col.ToString()+","+col_ref.ToString(); :};

selection::= escalar_exp_commalist:esc {: RESULT=esc.ToString(); :}
            |POR  {: RESULT="*"; :}  ;

escalar_exp_commalist::= escalar_exp:esc_exp  {:RESULT=esc_exp.ToString();:}
                    |escalar_exp_commalist:esc_exp_list COMA escalar_exp:esc_exp
                     {:RESULT=esc_exp_list.ToString()+","+esc_exp.ToString();:};

addo_atom_commalist::= addo_atom:addo{:RESULT=addo.ToString();:}
                      |addo_atom_commalist:addo_atom COMA addo_atom:addo
                       {:RESULT=addo_atom.ToString()+","+addo.ToString();:};

addo_atom::= atom:atm {:RESULT=atm.ToString();:}
            |NULL     {:RESULT="null";:};

muto_statement_searched::= MUTO NAME:nombre SET assignment_commalist:assig
                           opt_ex_clause:opt
                           {:if(my_parser.ejecutar_metodo==1)
                               {RESULT="update "+nombre.ToString()+" set
                                "+assig.ToString()+" "+opt.ToString();
                               }
                             else{RESULT="muto "+nombre.ToString()+" set
                                  "+assig.ToString()+" "+opt.ToString();
                                 }
                            :} ;

assignment_commalist::= assignment:asig   {:RESULT=asig.ToString();:}
                      |assignment_commalist:asig_com COMA assignment:asig
                       {:RESULT=asig_com.ToString()+","+ asig.ToString();:};

assignment::= NAME:nombre COMPARACION:comp escalar_exp:exp
             {:RESULT=nombre.ToString()+comp.ToString()+exp.ToString();:}
             |NAME:nombre COMPARACION:comp NULL
             {:RESULT=nombre.ToString()+comp.ToString()+" null";:}
             |NAME:nombre COMPARACION:comp subquery:sub escalar_exp:exp
             {:RESULT=nombre.ToString()+comp.ToString()+sub.ToString()
               +exp.ToString();:} ;

consulta_clases::= eligo_statement:eligo          {:RESULT=eligo.ToString();:}
                  |deleo_statement_searched:deleo{:RESULT=deleo.ToString();:}
                  |addo_statement:addo           {:RESULT=addo.ToString();:}
                  |muto_statement_searched:muto  {:RESULT=muto.ToString();:}
                  |drop_statement:drop           {:RESULT=drop.ToString();:};
```

```
eligo_statement::= ELIGO opt_all_distinct:opt
                   {:RESULT="select "+opt.ToString();:}
                   |selection:sele NAME:nombre
                     {:RESULT=sele.ToString()+" "+nombre.ToString();:};

drop_statement::= DROP CLASS  parametro_lista:lista PUNTOYCOMA
                  {:RESULT="drop table "+lista.ToString()+";";:};

parameter::= DOSPUNTOS NAME:nombre {:RESULT=":"+nombre.ToString();:};

parameter_ref::= parameter:parame{:RESULT=parame.ToString();:}
                |parameter:parame parameter:parame1{:RESULT=parame.ToString()+"
                 "+parame1.ToString();:}
                 /*|parameter INDICATOR parameter */ ;

vacio ::= /* nothing */{:RESULT=" ";:};

parametro_lista_metodo::=  addo_parametro1:addo      {:RESULT=addo.ToString();:}
                     |parametro_lista_metodo:para COMA addo_parametro1:addo
                      {:RESULT=para.ToString()+","+addo.ToString();:};

addo_parametro1::={:RESULT="";:}
                  |HORA:hora    {:RESULT=hora.ToString();:}
                  |FECHA:fecha  {:RESULT=fecha.ToString();:}
                  |NUMERO:numero{:RESULT=numero.ToString();:}
                  |NAME:nombre  {:RESULT=nombre.ToString();:}
                  |NULL         {:RESULT="null";:};

usar_metodo  ::=   NAME:nombre IPAREN parametro_lista_metodo:lista DPAREN
{:my_parser.extraer_supersistema();
  if(File.Exists(my_parser.nombre_supersistema+"//"+nombre.ToString()+".txt"))
    {my_parser.ejecutar_metodo=1;
     my_parser.sw1 = new StreamWriter("ejecutar.txt");
     my_parser.sw1.Write("la vida es una pasion");
     my_parser.sw1.Close();
    }
  else{my_parser.sw1 = new StreamWriter("Noejecutar.txt");
       my_parser.sw1.Write("la vida es una pasion");
       my_parser.sw1.Close();
      }
:} PUNTOYCOMA definicion_metodo_usar ;

definicion_metodo_usar::={:RESULT="";:}
                         |METHOD NAME:nombre_metodo
                          {:my_parser.nombre_metodo=nombre_metodo.ToString();:}
```

```
                        IPAREN parametro_lista_metodo:list DPAREN
                        cuerpo_metodo_usar:cuerpo  ;


cuerpo_metodo_usar::= ILLAVE definicion_funciones_usar:funciones DLLAVE
                       {:RESULT="\r\n{"+funciones.ToString()+"}"+"\n";
                         my_parser.funciones=funciones.ToString();:};

definicion_funciones_usar::={:RESULT="";:}
                     |deleo_statement_searched:deleo PUNTOYCOMA
                      aux_definicion_funciones:funciones
                      {:RESULT="\n"+deleo.ToString()+"; \n
                        "+funciones.ToString();
                        if(my_parser.ejecutar_metodo==1)
                          {string contenido = File.ReadAllText("conexion.txt");
                           string conexion="";
                           for(int i = 0; i <= contenido.Length;i++ )
                              {if(contenido[i]=='\r')
                                 {break;
                                 }
                               conexion = conexion + contenido[i];
                              }
                           if(File.Exists(conexion+".sqlite"))
                             {my_parser.conexion =new SQLiteConnection
                              ("Data Source="+conexion+".sqlite;
                              Version=3;New=True;Compress=True;");
                              my_parser.conexion.Open();
                              string creacion = deleo.ToString();
                              SQLiteCommand cmd = new SQLiteCommand("PRAGMA
                              FOREIGN_KEYS  = 1;", my_parser.conexion);
                              cmd.ExecuteNonQuery();
                              cmd=new SQLiteCommand(creacion,my_parser.conexion);
                              cmd.ExecuteNonQuery();
                              my_parser.conexion.Close();
                              cmd.Dispose();
                             }
                           else{MessageBox.Show("El suspersistema NO
                                HA SIDO CREADO", "", MessageBoxButtons.OK);
                               }
                          }
                       :}
                     |addo_statement_usar:addo PUNTOYCOMA
                      aux_definicion_funciones:funciones
                      {:RESULT="\n"+addo.ToString()+"; \n "+funciones.ToString();
                        if(my_parser.ejecutar_metodo==1)
                          {string contenido = File.ReadAllText("conexion.txt");
```

```
      string conexion="";
      for(int i = 0; i <= contenido.Length;i++ )
         {if(contenido[i]=='\r')
            {break;
            }
           conexion = conexion + contenido[i];
          }
       if(File.Exists(conexion+".sqlite"))
         {my_parser.conexion =new SQLiteConnection
          ("Data Source="+conexion+".sqlite;Version=3;
          New=True;Compress=True;");
          my_parser.conexion.Open();
          string creacion = addo.ToString();
          SQLiteCommand cmd = new SQLiteCommand("PRAGMA
          FOREIGN_KEYS  = 1;", my_parser.conexion);
          cmd.ExecuteNonQuery();
          cmd=new SQLiteCommand(creacion,my_parser.conexion);
          cmd.ExecuteNonQuery();
          my_parser.conexion.Close();
          cmd.Dispose();
         }
       else{MessageBox.Show("El suspersistema
            NO HA SIDO CREADO", "", MessageBoxButtons.OK);
           }
     }
 :}
|muto_statement_searched:muto PUNTOYCOMA
 aux_definicion_funciones:funciones
 {:RESULT="\n"+muto.ToString()+"; \n "+funciones.ToString();
   if(my_parser.ejecutar_metodo==1)
    {string contenido = File.ReadAllText("conexion.txt");
     string conexion="";
     for(int i = 0; i <= contenido.Length;i++ )
        {if(contenido[i]=='\r')
           {break;
           }
         conexion = conexion + contenido[i];
        }
      if(File.Exists(conexion+".sqlite"))
        {my_parser.conexion =new SQLiteConnection
         ("Data Source="+conexion+".sqlite;Version=3
         ;New=True;Compress=True;");
         my_parser.conexion.Open();
         string creacion = muto.ToString();
         SQLiteCommand cmd = new SQLiteCommand("PRAGMA
         FOREIGN_KEYS  = 1;", my_parser.conexion);
```

```
                                cmd.ExecuteNonQuery();
                                cmd=new SQLiteCommand(creacion,my_parser.conexion);
                                cmd.ExecuteNonQuery();
                                my_parser.conexion.Close();
                                cmd.Dispose();
                               }
                              else{MessageBox.Show("El suspersistema NO
                                   HA SIDO CREADO", "", MessageBoxButtons.OK);
                                  }
                             }
                            :};

addo_statement_usar::= ADDO INTO NAME:nombre opt_column_commalist:opt_col
                       value_or_query_spec_usar:value
                       {:if(my_parser.ejecutar_metodo==1)
                           {RESULT="insert into "+nombre.ToString()+"
                            "+opt_col.ToString()+" "+value.ToString();
                           }
                         else{RESULT="addo into "+nombre.ToString()+"
                              "+opt_col.ToString()+" "+value.ToString();
                             }
                       :};

value_or_query_spec_usar::= VALUES IPAREN addo_atom_commalist_usar:addo DPAREN
                            {:RESULT="values ("+addo.ToString()+")";:}
                           |query_spec:query{:RESULT=query.ToString();:};

addo_atom_commalist_usar::=addo_atom_usar:addo{:RESULT=addo.ToString();:}
                           |addo_atom_commalist_usar:addo_atom COMA
                            addo_atom_usar:addo
                            {:RESULT=addo_atom.ToString()+","+addo.ToString();:};

addo_atom_usar::= atom_usar:atm  {:RESULT=atm.ToString();:}
                 |NULL          {:RESULT="null";:};


atom_usar::= parameter_ref:refe           {:RESULT=refe.ToString();:}
            |NUMERO:numero                {:RESULT=numero.ToString();:}
            |AMPERSAND NAME:nombre        {:RESULT=nombre.ToString();:}
            |COMILLA NAME:nombre  COMILLA {:RESULT="'"+nombre.ToString()+"'";:}
            |COMILLA FECHA:nombre COMILLA {:RESULT="'"+nombre.ToString()+"'";:}
            |COMILLA HORA:nombre  COMILLA {:RESULT="'"+nombre.ToString()+"'";:}
            |AMPERSAND NUMERO:nombre      {:RESULT=nombre.ToString();:}
            |COMILLA COMILLA              {:RESULT="''";:};
```

ANEXO B. CÓDIGO NECESARIO PARA PARA GENERAR EL ANALIZADOR LÉXICO DEL INTERPRETE LATIN

```
using System.IO;
using System.Windows.Forms;
using System.Text;
using System.Drawing;
using System.Data;
using System.ComponentModel;
using System.Collections.Generic;
using System;
using TUVienna.CS_CUP.Runtime;
%%

%line
%char
%cup


%{ private Symbol symbol(int type)
    {
     return  new Symbol(type, yyline, yychar);
    }
   private Symbol symbol(int type, Object value)
    {
     return  new Symbol(type, yyline, yychar, value);
    }
%}


Comentario = {ComentarioTradicional} | {FinLineaComentario} |
{DocumentacionComentario}
InputCharacter = [^\r\n]
ComentarioTradicional   = "/*" [^*] ~"*/" | "/*" "*"+ "/"
FinLineaComentario     = "//" {InputCharacter}* {FinLinea}
DocumentacionComentario = "/**" {ContenidoComentario} "*"+ "/"
ContenidoComentario       = ( [^*] | \*+ [^/*] )*

FinLinea = \r|\n|\r\n

EspacioBlanco     = {FinLinea} | [ \t\f]

campo_fecha = [1-9][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9]

campo_hora = [0-2][0-9]:[0-5][0-9]:[0-5][0-9]

variable = [A-Za-z_][A-Za-z_0-9]*
NONLINE_WHITE_SPACE_CHAR=[\ \t\b\012]
```

```
WHITE_SPACE_CHAR=[\n\ \t\b\012]

STRING_TEXT=(\\\|[^\n\]|\\{WHITE_SPACE_CHAR}+\\)*

COMMENT_TEXT=([^/*\n]|[^*\n]/[^*\n]|[^/\n]*[^/\n]|*[^/\n]|/[^*\n])*
%%
  <YYINITIAL>   ";"               { return  symbol(sym.PUNTOYCOMA); }
  <YYINITIAL>   ","               { return  symbol(sym.COMA); }
  <YYINITIAL>   "+"               { return  symbol(sym.MAS); }
  <YYINITIAL>   "-"               { return  symbol(sym.MENOS); }
  <YYINITIAL>   "*"               { return  symbol(sym.POR); }
  <YYINITIAL>   "("               { return  symbol(sym.IPAREN); }
  <YYINITIAL>   ")"               { return  symbol(sym.DPAREN); }
  <YYINITIAL>   "{"               { return  symbol(sym.ILLAVE); }
  <YYINITIAL>   "}"               { return  symbol(sym.DLLAVE); }
  <YYINITIAL>   ":"               { return  symbol(sym.DOSPUNTOS); }
  <YYINITIAL>   "&"               { return  symbol(sym.AMPERSAND); }
  <YYINITIAL>   "/"               { return  symbol(sym.DIVISION); }
  <YYINITIAL>   "."               { return  symbol(sym.PUNTO); }
  <YYINITIAL>   "'"               { return  symbol(sym.COMILLA); }
  <YYINITIAL>   avg               { return  symbol(sym.AVG); }
  <YYINITIAL>   all               { return  symbol(sym.ALL); }
  <YYINITIAL>   count             { return  symbol(sym.COUNT,yytext());}
  <YYINITIAL>   decimal           { return  symbol(sym.DECIMAL); }
  <YYINITIAL>   double            { return  symbol(sym.DOUBLE); }
  <YYINITIAL>   exists            { return  symbol(sym.EXISTS); }
  <YYINITIAL>   deleo             { return  symbol(sym.DELEO);}
  <YYINITIAL>   is                { return  symbol(sym.IS); }
  <YYINITIAL>   min               { return  symbol(sym.MIN, yytext()); }
  <YYINITIAL>   max               { return  symbol(sym.MAX, yytext()); }
  <YYINITIAL>   numeric           { return  symbol(sym.NUMERIC); }
  <YYINITIAL>   primary           { return  symbol(sym.PRIMARY); }
  <YYINITIAL>   set               { return  symbol(sym.SET); }
  <YYINITIAL>   sum               { return  symbol(sym.SUM, yytext()); }
  <YYINITIAL>   muto              { return  symbol(sym.MUTO);}
  <YYINITIAL>   and               { return  symbol(sym.AND); }
  <YYINITIAL>   any               { return  symbol(sym.ANY); }
  <YYINITIAL>   between           { return  symbol(sym.BETWEEN); }
  <YYINITIAL>   by                { return  symbol(sym.BY); }
  <YYINITIAL>   check             { return  symbol(sym.CHECK); }
  <YYINITIAL>   class             { return  symbol(sym.CLASS); }
  <YYINITIAL>   date              { return  symbol(sym.DATE); }
  <YYINITIAL>   default           { return  symbol(sym.DEFAULT); }
  <YYINITIAL>   distinct          { return  symbol(sym.DISTINCT); }
  <YYINITIAL>   drop              { return  symbol(sym.DROP); }
  <YYINITIAL>   escape            { return  symbol(sym.ESCAPE); }
  <YYINITIAL>   float             { return  symbol(sym.FLOAT); }
  <YYINITIAL>   char              { return  symbol(sym.CHAR);}
  <YYINITIAL>   byte              { return  symbol(sym.BYTE);}
  <YYINITIAL>   foreign           { return  symbol(sym.FOREIGN); }
  <YYINITIAL>   in                { return  symbol(sym.IN); }
  <YYINITIAL>   addo              { return  symbol(sym.ADDO);}
  <YYINITIAL>   integer           { return  symbol(sym.INTEGER); }
```

```
  <YYINITIAL>   into              { return  symbol(sym.INTO); }
  <YYINITIAL>   key               { return  symbol(sym.KEY); }
  <YYINITIAL>   like              { return  symbol(sym.LIKE); }
  <YYINITIAL>   method            { return  symbol(sym.METHOD); }
  <YYINITIAL>   not               { return  symbol(sym.NOT); }
  <YYINITIAL>   null              { return  symbol(sym.NULL); }
  <YYINITIAL>   or                { return  symbol(sym.OR); }
  <YYINITIAL>   precision         { return  symbol(sym.PRECISION); }
  <YYINITIAL>   real              { return  symbol(sym.REAL); }
  <YYINITIAL>   references        { return  symbol(sym.REFERENCES); }
  <YYINITIAL>   eligo             { return  symbol(sym.ELIGO);}
  <YYINITIAL>   smallint          { return  symbol(sym.SMALLINT); }
  <YYINITIAL>   some              { return  symbol(sym.SOME); }
  <YYINITIAL>   show              { return  symbol(sym.SHOW); }
  <YYINITIAL>   time              { return  symbol(sym.TIME); }
  <YYINITIAL>   unique            { return  symbol(sym.UNIQUE); }
  <YYINITIAL>   values            { return  symbol(sym.VALUES); }
  <YYINITIAL>   varchar           { return  symbol(sym.VARCHAR); }
  <YYINITIAL>   ex                { return  symbol(sym.EX);}
  <YYINITIAL>   bool              { return  symbol(sym.BOOL);}
  <YYINITIAL>   extend            { return  symbol(sym.EXTEND);}
  <YYINITIAL>   class             { return  symbol(sym.CLASS);}
  <YYINITIAL>   System            { return  symbol(sym.SYSTEM);}
  <YYINITIAL>   Subsystem         { return  symbol(sym.SUBSYSTEM);}
  <YYINITIAL>   Supersystem       { return  symbol(sym.SUPERSYSTEM);}
  <YYINITIAL>   having            { return symbol(sym.HAVING);}
  <YYINITIAL>   from              { return  symbol(sym.FROM);}

 "0" { return  symbol(sym.NUMERO,yytext()); }
[1-9][0-9]* { return  symbol(sym.NUMERO,yytext()); }
[0-9]+"."[0-9]* { return  symbol(sym.NUMERO,yytext()); }
[1-9][0-9]*"."[0-9]* { return  symbol(sym.NUMERO,yytext()); }

"="  { return  symbol(sym.COMPARACION, yytext()); }
"<>" { return  symbol(sym.COMPARACION, yytext()); }
"<"  { return  symbol(sym.COMPARACION, yytext()); }
">"  { return  symbol(sym.COMPARACION, yytext()); }
"<=" { return  symbol(sym.COMPARACION, yytext()); }
">=" { return  symbol(sym.COMPARACION, yytext()); }

{variable}    { return  symbol(sym.NAME,yytext());}

{campo_fecha} { return  symbol(sym.FECHA,yytext());}

{campo_hora}  { return  symbol(sym.HORA,yytext());}

{Comentario}  { /* una vez encontrado, no hace nada */ break; }

"\r\n" {return  symbol(sym.NAME,yytext());}
'\n' {return  symbol(sym.NAME,yytext());}

{EspacioBlanco} { /* ignore white space. */break; }
. { System.Console.Error.WriteLine("Illegal character: "+yytext());break; }
```

ANEXO C. INTERACCIÓN DE DB4O CON UNA CLASE EXTERNA

Supongamos este es el contenido de tu archivo cs

```
using System;
namespace MySpace
{
    public class Person
    {
        public string Nombre { get; set; }
        public object Estado { get; set; }
                public string Saludar()
                {
                        return "hola mundo!";
                }
    }
}
```

Con este otro código lo utilizamos para compilar el contenido, instanciar e invocar a un método desde un programa consola normal.

```
Microsoft.CSharp.CSharpCodeProvider provider = new
Microsoft.CSharp.CSharpCodeProvider();
var op = new CompilerParameters();
// decidimos que el assembly se almacene en memoria
op.GenerateInMemory = true;
// compilamos
var result = provider.CompileAssemblyFromFile(op, @"D:\prueba.cs");
var assembly = result.CompiledAssembly;
//instanciamos la clase Person
var ty = assembly.GetType("MySpace.Person");
var inst = Activator.CreateInstance(ty);
//invokamos al metodo Saludar
var saludo = ty.GetMethod("Saludar").Invoke(inst, null);
Console.Write(saludo);
```