INTÉRPRETE LATIN PARA LA IMPLEMENTACIÓN DE MODELOS EMERGENTES DE DATOS

JULIÁN RICARDO CAMARGO GONZÁLEZ

YULIANA MAYERLY SOTO CARREÑO

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICO-MECÁNICA

ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

BUCARAMANGA

2011

INTÉRPRETE LATIN PARA LA IMPLEMENTACIÓN DE MODELOS EMERGENTES DE DATOS

JULIÁN RICARDO CAMARGO GONZÁLEZ

YULIANA MAYERLY SOTO CARREÑO

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

2011

DEDICATORIA

A Dios, por la vida y la salud.
A mis padres y hermanas por su amor y respaldo.
A Yuliana, por trabajar a mi lado.
A los verdaderos amigos.

JULIÁN

DEDICATORIA

A Dios, por Ayudarme a alcanzar todos mis logros.
A mis padres, por su esfuerzo y respaldo incondicional.
A mi compañero Julián, por su colaboración.
Y a todas las personas que de una u otra forma hicieron esto posible.

YULIANA

AGRADECIMIENTOS

Los autores expresan sus agradecimientos a:

DIOS. Por la salud, sabiduría, entendimiento y la paciencia para afrontar las dificultades presentadas durante la realización de este proyecto.

UNIVERSIDAD INDUSTRIAL DE SANTANDER. Por acogernos durante estos años que duró la carrera, brindando calidad y servicio al alcance de nosotros.

JAIME OCTAVÍO ALBARRACÍN FERREIRA. El director, que siempre estuvo pendiente del desarrollado de este proyecto.

EMIRO MUÑOZ JEREZ. Un gran codirector que colaboro activamente, brindándonos dedicación y tiempo para obtener este gran logro.

FAMILIARES. Por que han sido quienes han visto el esfuerzo, y por que gracias a ellos es posible culminar este trabajo.

Muchisimas gracias a todas las personas que de cualquier forma contribuyeron con su apoyo incondicional. A todos nuestros más sinceros agradeciemientos.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

TITULO:

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

AUTORES:

JULIÁN RICARDO CAMARGO GONZÁLEZ**

YULIANA MAYERLY SOTO CARREÑO**

PALABRAS CLAVES

Sistemas Emergentes, Transacciones, Desarrollo de Intérpretes y Compiladores, Modelos de datos, Objetos.

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 debe haber algún tipo de conexión que sea pertinente.

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 valido decir que un sistema es un conjunto de clases que interactúan y comparten un método común. Con estos conceptos el profesor Albarracín expone en su tesis doctoral un Modelo de Datos Emergente (MODE) que básicamente refuta las teorías en las que mencionan objetos como entes agrupados al azar, y en acuerdo con el nuevo paradigma de sistemas, comparte que en las organizaciones tienen que estar unidas la parte funcional y la estructural. Además, presenta este modelo superando los objetos y proponiendo un lenguaje para describir y administrar sistemas. Dicho lenguaje ha sido denominado LATIN (Language Toward Integration) puesto que la integración de objetos es el fundamento de los sistemas.

Para esto, era necesario definir dicho lenguaje, así que el profesor Emiro Muñoz en su trabajo de maestría realizó la definición y especificación del intérprete LATIN, en el marco de los sistemas.

---

*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:

JULIÁN RICARDO CAMARGO GONZÁLEZ**

YULIANA MAYERLY SOTO CARREÑO**

KEY WORDS

Emergering Systems, Transactions, Develop of interpreters and compilers, Data Models, objects.

DESCRIPTION

Observing reality as objects implies that any element can affect another and does not belong to the same set, but in reality does not happen that way must necessarily be some kind of connection that is relevant.

An organization is part of reality, this means that belongs to a system. "A system is a set of interrelated objects so that each affects the structural and functional to the whole." With these concepts the prfofessor Albarracín out in his doctoral thesis Emerging Data Model (MODE), which basically refutes theories that mentioned objects as entities grouped at random. In addition, this model presents objects and proposing overcoming a language for describing and managing systems. This language has been called LATIN (Language Toward Integration) as the integration of objects is the foundation of the systems.

For this, it was necessary to define the language, so the teacher Emiro Muñoz master at work made the definition and specification of the interpreter LATIN, in the context of the systems.

_____

*Working Grade. Mode: Investigation.

**Physical Mechanical Engineering Faculty, the Systems Engineering School.

Director: Dr. Jaime Octavio Albarracín Ferreira.

INTRODUCCIÓN

En el estudio realizado por el profesor Albarracín en su tesis doctoral, muestra las falencias de los procesos en las organizaciones, ya que muchas de estas trabajan con modelos que emplean una estructura digital prefabricada. Pero en ocasiones estos procedimientos son tan fragmentarios que pueden llegan a ser de difícil compresión para el usuario.

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 llevo 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 esta describiendo. Entre tanto, el profesor Emiro Muñoz define este lenguaje en su trabajo de maestría tal y como lo propone el MODE (Tesis doctoral del profesor Albarracín).

De esta manera surge el "**INTERPRETE LATÍN PARA LA IMPLEMENTACIÓN DE MODELOS EMERGENTES DE DATOS"** con el propósito de plasmar este lenguaje y sustentar la elaboración de la gramática de la escritura de LATIN, de igual forma 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[2].

---

[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.
[2]MUÑOZ JEREZ EMIRO. "INTERPRETE LATÍN PARA LA IMPLEMENTACION DE UN MODELO DE DATOS REORIENTADO A SISTEMAS (MODRES)", Trabajo de Maestría. Universidad Industrial de Santander.

Básicamente este Lenguaje Interpreta sistemas y ejecuta transacciones como compras y se obtiene un resultado detallado dependiendo de las operaciones que se realicen. Esto se hace gracias a la descripción desarrollada en Java y SQL que son los mecanismos que se utilizaron para llevar a cabo este intérprete.

1. PRESENTACIÓN DEL PROYECTO

1.1Definición 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 este 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 por 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 uno 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 esto la idea de la construcción de un intérprete 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. Este lenguaje será denominado LATIN (Language Toward Integration).

1.1.2 Objetivos Específicos

1. Implementar la gramática para la escritura del LATIN. Esta gramática ha sido especificada en el trabajo de investigación "Interprete LATIN para la implementación de un modelo de datos emergente (MODE)".

2. Implementar los analizadores léxico y sintáctico, los cuales permitirán reconocer y ejecutar sistemas.

3. Implementar la conversión de los sistemas descritos en tablas SQL.

4. Implementar la ejecución de transacciones mediante programas Java y SQL.

5. Implementar la conversión de las respuestas del sistema en consultas mediante programas Java y SQL.

6. Diseñar e implementar una interfaz gráfica como editor de LATIN.

1.3 Justificación

Actualmente en las diferentes empresas donde se cuenta con un producto software, se tiene un esquema de sistemas que al consultar cualquier información, muestra datos sueltos que solo los desarrolladores pueden interpretar, es decir, que los resultados que se presentan no es más que la impresión de unos campos que corresponden a una tabla determinada sumados a la apreciación del experto.

Pensando precisamente en disminuir el trabajo del usuario y aumentar la eficiencia del sistema, el MODE (Tesis Doctoral Profesor Albarracín) descarta cualquier software robusto y antiguo e innova con la creación de un modelo que se aproxima al comportamiento, estructura y funcionalidad del mismo.

Partiendo de la idea que hace falta un lenguaje que reconozca dicho modelo, el profesor Emiro Muñoz en su trabajo de maestría elabora este lenguaje, y es el trabajo del INTERPRETE LATÍN PARA LA IMPLEMENTACIÓN DE MODELOS EMERGENTES DE DATOS quien materializa los trabajos desarrollados por los profesores ya mencionados.

1.4 Alcances y Limitaciones

La idea principal del Intérprete LATIN para la implementación de modelos emergentes de datos es mostrar mediante un ejemplo la funcionalidad básica del sistema. Partiendo de la definición como entrada, se tiene un conjunto de partes o elementos organizados y relacionados que interactúan entre sí para lograr un objetivo, y obteniendo como salida la información procesada por el software.

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 ejecutar de forma ordenada para encontrar información en cualquier instancia de dichos procesos.

Además su eficiente mecanismo de analizar el léxico y sintáctico, facilitará la corrección temprana de errores, y evitará que se avance en caso que se muestren mensajes de alerta.

Gracias a los botones 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 del intérprete, cuenta con el de ayuda que explica las instrucciones de uso y funcionalidad de cada botón, de tal manera que simplifique el trabajo del usuario.

Toda la información dentro del software se encuentra en ingles, ya que es un proyecto que pretende mostrarse en el exterior.

2. MARCO TEÓRICO

2.1 Introducción al Intérprete LATIN

2.1.1 Estructura

2.1.1.1 Valores Literales

- ***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: Parámetro_21, _variable6, nombre_clase, elemento_tabla

- ***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: 203, 0.8, 12, .87

- ***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: 2009-09-05, 1973-02-27

- ***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: 13:11:11, 09:25:48

- ***Valores Null:*** Este valor significa que "no existen datos". Solo se debe escribir null en minúscula, no se permite la combinación de mayúsculas y minúsculas.

2.1.1.2 Caracteres permitidos

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

CARACTER	SIGNIFICADO
+	Signo más
-	Signo menos
*	Multiplicación
/	División
%	Resto de la 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)
|	Compuerta lógica or
\	antieslas

Tabla 1: Caracteres permitidos

2.1.1.3 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.

PALABRAS RESERVADAS INTÉRPRETE LATIN			
all	numeric	habendo	Time
autoincrement	primary	in	unique
count	set	ADDO	values
decimal	sum	Integer	Varchar
double	MUTO	into	ex
exists	and	key	bool
usque	any	like	If
indicator	avg	method	Then
is	between	not	else
min	Iubere per	null	while
max	cascade	or	do
DELEO	check	on	true
disctinct	class	precision	false
drop	date	real	add
SubSystem	default	References	Alter
escape	float	ELIGO	some
byte	foreign	smallint	show
Lego per	table	char	extent
extents	System	SuperSystem	

Tabla 2: Palabras reservada

2.1.1.4 Comentarios

Estos son aceptados desde una secuencia '/*' hasta la máxima secuencia '*/'. La secuencia de cierre no necesita estar en la misma línea, lo que permite tener comentarios que abarquen múltiples líneas. También se permiten comentarios por línea tipo "//"

2.1.1.5 Sintaxis Iníciales

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

- Definición de sistemas, clases y métodos.
- Manipulación y consulta de clases.

2.1.1.6 Definición de Sentencias ODL

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

- Definición de Sistema

system nombre_sistema [herencia] [cuerpo_sistema];
herencia **:** /* vacio */
| **extent** nombre_sistema_padre
cuerpo_sistema **:**
{
[class**1**, class **2**,…class **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 **extent** nombre_sistema_padre.

Los nombres nombre_sistema y nombre_sistema_padre pueden ser literales de tipo variable.

- Definición de Clases

definicion_class :/* vacio */
|**class** nombre_class
[herencia]
{
[listado_atributos]

}
[definición_class]

herencia : /* vacio */
|**extent** nombre_sistema_padre

listado_atributos : /* vacio */
| [elemento_class]
| [listado_atributo], [elemento_class]

elemento_class : [def_columna]
| [def_interrelacion]

def_columna :
[columna][tipo_dato][opc_def_columna]

tipo_dato **: varchar**
| 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
| default ‘nombre’

	| 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_class **| foreign key (**[listado_columna]) **| references** nombre_class **(**[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] **|** [entre_predicado] **|** [entre_predicado] **|** [like_predicado] **|** [en_predicado] **|** [entre_predicado] **|** [todo_o_algun_predicado] **|** [prueba_existencia]
comparacion_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]**)**
opt_all_distinct	**:| all** **|distinct**
atom	: parametro **|** numero **|** &variable **|** ‘ nombre ‘ **|** ‘ literal_hora ‘ **|** ‘ literal_fecha ‘
comparación	**:=** **|<>** **|<** **|>** **|<=** **|>=**
escalar_exp	: [escalar_exp] + [escalar_exp] **|** [escalar_exp] - [escalar_exp] **|** [escalar_exp] * [escalar_exp] **|** [escalar_exp] / [escalar_exp] **|** [escalar_exp] % [escalar_exp] **| +** [escalar_exp] **| -** [escalar_exp]

| [atom]
| nombre_class **.** [nombre_class]
| [function_ref]
| **(**[escalar_exp]**)**

function_ref : [ammsc] **(*)**
: [ammsc] **(distinct** nombre_class**)**
: [ammsc] **(all** [escalar_exp]**)**
: [ammsc] **([**escalar_exp]**)**

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

Como se puede observar 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 creada en otro sistema no se deben colocar los atributos. En su caso se debe escribir:

class nombre_class {}

En llegado caso de que haya la necesidad de agregar atributos, esto se deberá hacer en la clase creada en el padre.

Como observación general, se deberá tener en cuenta de no definir atributos con nombres de otros atributos ya definidos con anterioridad, esto con el objetivo de no generar errores, aunque si se presentase, el intérprete retornara un mensaje de error indicando que ya existe el atributo que esta intentando crear.

2.1.1.7 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

addo_statement : **drop class** nombre_clase , opc_clase**….;**

2.1.2 Analizador Léxico

Es un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos. Estos tokens sirven para una posterior etapa del proceso de traducción, siendo la entrada para el analizador sintáctico (en inglés parser).

Las funciones que realiza este analizador son:

- Leer los caracteres de entrada del programa fuente.
- Generar la secuencia de componentes léxicos.
- Eliminar comentarios, espacios en blanco y todo lo que este excluido de la sintaxis del lenguaje.
- Reconocer identificadores, palabras reservadas del lenguaje, y tratarlos correctamente con respecto a la tabla de símbolos (solo en los casos que debe de tratar con dicha tabla).
- Avisar de errores léxicos. Por ejemplo, si @ no pertenece al lenguaje, avisar de un error.

Para la creación del analizador léxico se utiliza la herramienta **Lex**, el cual es un programa que permite generar analizadores léxicos, de forma sencilla, gratis y fácil acceso en internet.

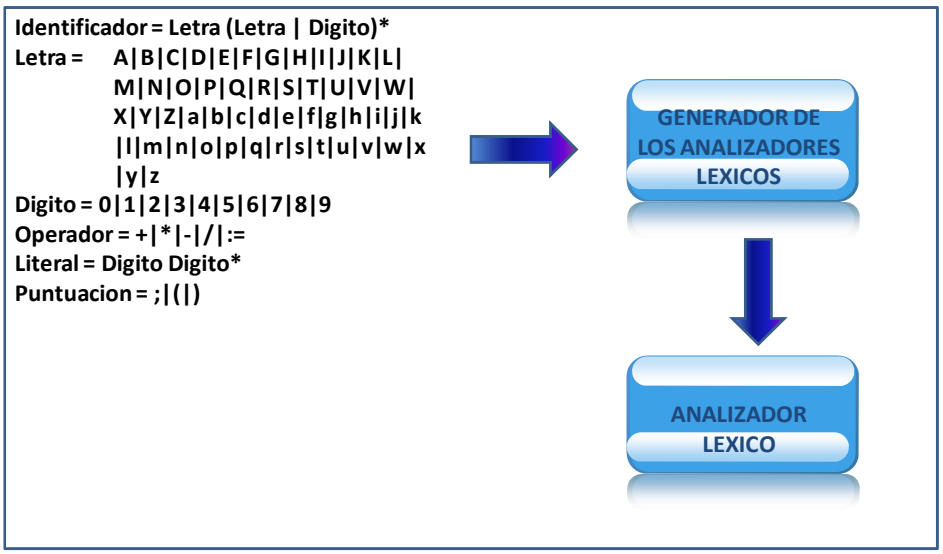

Figura 1: Analizador Léxico

2.1.3 Analizador Sintáctico

El análisis sintáctico o parser se encarga de revisar el texto de entrada, suministrado por el analizador léxico, en base a una gramática dada. En caso de que el texto corresponda a una sentencia valida, suministrará el árbol sintáctico que la reconoce. 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.

A continuación se presenta un ejemplo, con las producciones de una gramática libre de contexto, que describen expresiones aritméticas simples en notación BNF.

La expresión **"7+5*2"** generaría el siguiente árbol de derivación sintáctico.

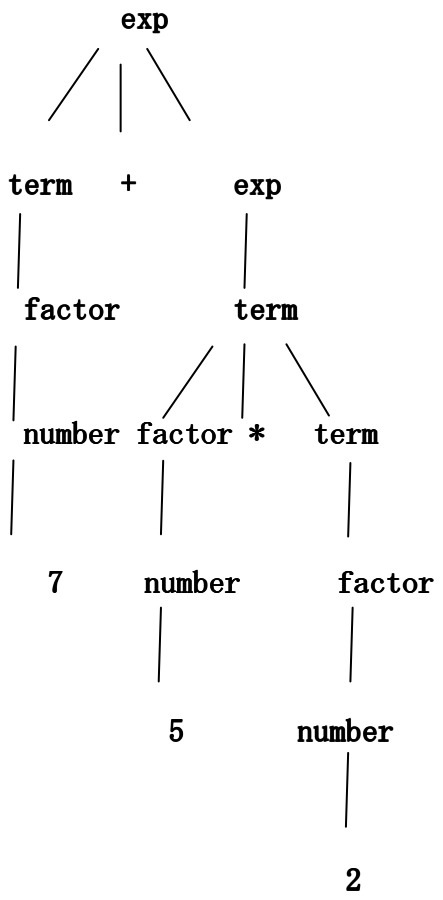

Figura 2: Árbol Sintáctico de la expresión

El analizador sintáctico puede ser descendente y ascendente:

Analizador Sintáctico Descendente: Parte del axioma inicial, y va efectuando derivaciones a izquierda hasta obtener la secuencia de derivaciones que reconoce a la sentencia. Pueden ser:

1. Con retroceso
2. Con funciones recursivas
3. De Gramáticas LL(1)

Analizador Sintáctico Ascendente: Parte de la sentencia de entrada, y va aplicando derivaciones inversas (desde el consecuente hasta el antecedente), hasta llegar al axioma inicial. Pueden ser:

4. Con retroceso
5. De Gramáticas LR(1)

Para generar el analizador sintáctico se utilizó 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 JAVA.
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.

2.1.4 Definición de la Estructura de Datos

Para una correcta interacción de cada uno de los componentes del Intérprete, se hace necesaria la creación de varios archivos de datos que permitan almacenar la información correspondiente a los Supersistemas, Subsistemas y Sistemas creados.

Por otro lado, el Intérprete genera un archivo temporal que es usado al momento de la creación de las clases que componen cada uno de los Supersistemas, Subsistemas y Sistemas. Este archivo deberá llamarse **sistemas.oql**.

La forma en la que el Intérprete crea estos archivos, se conoce como traducción dirigida por sintaxis, esto consiste en ir haciendo otras tareas al mismo tiempo en que se realiza el análisis sintáctico.

La gramática definida para el analizador sintáctico esta 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, cuyo valor asociado puede ser el nombre de un atributo, clase. 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

2.1.5 Archivos de Sistemas

Los archivos de sistemas cuya extensión en este proyecto es .latin, constituyen el medio de interacción entre el Intérprete y el sistema de Bases de Datos.

Los archivos de sistema se van generando al mismo tiempo que se hace el análisis sintáctico y se crea uno por cada sistema de acuerdo a la estructura que se muestra a continuación.

```
[Clase]
Nombres de clases propias del sistema
Nombres de clases heredadas del sistema padre
[method]
Nombre del método
(Parametros heredados del método del sistema padre, parametros del método
de la clase)
{
Instrucciones SQL heredadas del método del sistema padre
Instrucciones SQL del método propio de la clase
}
```

Figura 3: Estructura de los Archivos del Sistema

Los archivos de sistema, son ficheros planos que contienen:

- La palabra reservada **class** encerrada entre corchetes rectos ([ ]), para indicar que a continuación vienen palabras que corresponden a nombres de clases.
- Los nombres de las clases propias del sistema, que serán declaradas en la interfaz del Intérprete al momento de definir las clases junto con sus atributos e interrelaciones y el método de cada una de las clases a crear.
- Si la clase a la cual pertenece el archivo es derivada de otro sistema anteriormente declarado, se copian después de los
- Nombre de las clases del sistema hijo, los nombres de cada una de las clases del sistema padre.
- La palabra reservada **method** encerrada entre corchetes rectos ([ ]), para indicar que a continuación viene la especificación del método perteneciente a la clase por la cual se esta creando el archivo .LTS.

- El nombre del método que se especifico al momento de definir la estructura del sistema en la interfaz del Intérprete, seguido de un paréntesis de apertura **"("** el cual indica que en la misma línea, a continuación de este siguen los parámetros del método.
- Si la clase a la cual pertenece el archivo es derivada o hija de otro sistema, se copian los nombres de los parámetros del método de su padre.
- En la misma línea y separados por una coma **(,)** se copian los nombres de los parámetros del método de la clase, a la cual pertenece el archivo .LTS, seguidos en la misma línea de un paréntesis de cierre **")",** el cual indica que ha finalizado la definición de todos los parámetros del método.
- Una llave de apertura **({),** la cual indica que a continuación vienen las instrucciones SQL correspondientes al método.
- Las instrucciones SQL del método, tal y como se escriben al momento de definir la estructura de la clase en la interfaz del Intérprete.
- Las instrucciones SQL del método del sistema padre, cuando el sistema a la cual pertenece el archivo posee herencia.
- Una llave de cierre **(})**, para indicar que han finalizado las instrucciones OQL del método

A continuación se muestran las expresiones regulares que corresponden a la definición de un método en particular según la gramática desarrollada para el analizador sintáctico.

Defincion_metodo	*::= METHODUS NAME IPAREN* *paramet_list DPAREN* *cuerpo_metodo* *;*
paramet_list	*::=addo_paramet* | *paramet_list COMA addo_paramet* *;*
insert_paramet	*::=* | *NAME* *NULLX* *;*
cuerpo_metodo	*::= ILLAVE* *definicion_funciones*

DLLAVE
;

2.1.6 Archivo para la Definición de Sistemas (Clases)

El proceso de creación del archivo es muy sencillo, esto es debido a que para la definición de las clases con sus atributos, el LATIN hace uso de la palabra reservada **"class"** en lugar del tradicional **CREATE CLASS** usado en OQL, seguido de dos llaves y entre ellas la definición de los campos o atributos cuya sintaxis es idéntica a la usada en el lenguaje OQL, por lo tanto para generar el archivo **sistemas.oql** solo necesario reemplazar la palabra **"class"** por la expresión **CREATE CLASS**, de igual forma reemplazar las llaves por paréntesis. La definición de los campos se copia exactamente igual desde la interfaz al archivo OQL generado.

Es de aclarar, que debido a que en un sistema se pueden definir campos de una clase existente en el sistema padre (si se ha definido la herencia), es necesario modificar esa clase ya creada con anterioridad agregándole los nuevos campos definidos en el correspondiente sistema hijo, haciendo uso ya no de las instrucciones **CREATE CLASS** para la creación de las **clases,** sino de las correspondientes instrucciones **ALTER CLASS** para la modificación de las **clases.**

2.1.7 Conexión del Intérprete con MySQL

Una vez definida la gramática del LATIN, y comprobado por medio de los analizadores léxico, sintáctico y semántico que se puede ***CREAR (CREATE SYSTEM)*** el sistema con sus clases y su método común, como lo propone el MODE, el siguiente paso es que el LATIN, permita separar por un lado lo estructural **(Entidades "clases")** y por otro lado lo funcional **(El método del sistema que es común a las clases).**

Lo estructural se obtiene gracias al manejador de bases de datos SQL (Structured Query Language) y como motor de almacenamiento InnoDB.

Una vez decidido el gestor de bases de datos y el motor de almacenamiento, se procede a la parte funcional que se realiza mediante la conexión de Java y MySQL, esto se logra gracias a MySQL Connector/J, que es la solución implementada por MySQL para proporcionar conectividad a las aplicaciones clientes desarrolladas en Java vía al JDBC (Java Data Base Connectivity).

2.2 Representación de la Jerarquía de Sistemas

2.2.1 Definición del Sistema

Un sistema (System) es el conjunto de entidades (las mismas clases de los modelos de objetos actuales), interrelacionadas de forma tal que cada una afecta la estructura y el comportamiento de todo el conjunto. Este conjunto encapsula un comportamiento común a todas las entidades contenidas.

Esta jerarquía esta representada por el súpersistema, el sistema y el subsistema, descrito 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.

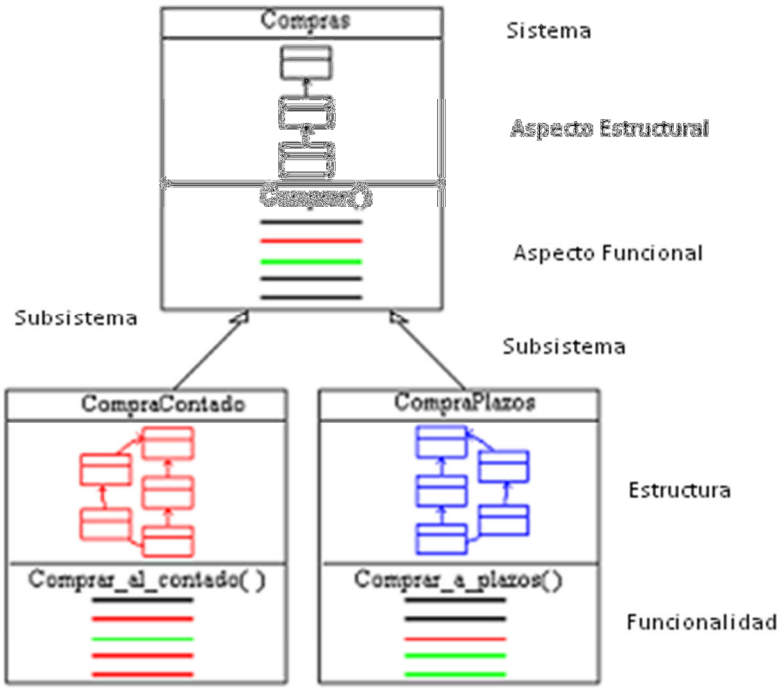

Figura 4: Definición del Sistema del Interprete LATIN

2.2.2 Codificación (Esqueleto) en Latín de la descripción de la jerarquía de sistemas

SuperSystem 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 Transaccion"**
Void Realizar_Transaccion (…)
}// Fin definicion de Super 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 Transaccion"** y completa **"Comprar"**
Void insertar_entrada_mercancia_en_diario ()
Void sumar_a_existencia_en_mercancia ()
Void sumar_entrada_mercancia_en_mayor ()
}// Fin definicion de sistema Compras

SubSystem 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 definicion de subsistema CompraPlazos
```

2.2.3 Transacciones Ejemplo

2.2.3.1 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 6. 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.

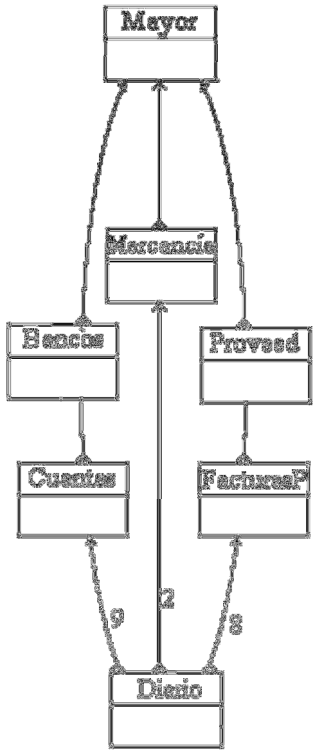

Figura 5: Tipo Compras

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.

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 un INSERT en la Bitácora o libro Diario para cada uno de los siguientes asientos contables, generando cada asiento una tupla.

Código Cuenta	*Nombre cuenta*	*Débito*	*Crédito*
300	*Existencias comerciales*	*2400*	
572	*Bancos*		*2400*

Se ve así que el "Diario" (Figura 6) 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	311297119	kra. 7 #4-80	5000	572

Mayor

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

Figura 6: 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 6 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 (UPDATE) por la transacción y las segundas los registros a ser insertados (INSERT) 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 es el resultado de la consulta hecha en LATIN para la siguiente operación:

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.

2.2.3.2 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***	***Débito***	***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 a continuación en la figura 7:

Diario

Tra	*Fecha*	*Hora*	*Nro_docu*	*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	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	←
7000				

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	
572	proveedores	u		u2+2400

Figura 7: Actualización de tablas de la base de datos en la compra a crédito.

Pero además, la instancia de esta transacción 0 crea la necesidad de ocurrir una instancia de la otra transacción, sea la 8 el "*Pago de facturas a proveedores nacionales de mercancía"*, en algún momento posterior. Así, cuando llegue el día de pagar, por ejemplo el 15 de enero de 2006, se ejecutará entonces la transacción 8 complementaria, instanciada con los siguientes asientos:

Codigo Cuenta	***Nombre Cuenta***	***Débito***	***Crédito***
400	*Proveedores*	*2400*	
572	*Bancos*		*2400*

El "Diario" es una tabla del MODE sobre la que los programas sólo pueden ejecutar inserciones, no pueden modificar registros o eliminar registros porque esto sería como alterar la historia de la organización. Por lo tanto, se puede observar en el "Diario" de la figura 8 la acumulación de transacciones. Entonces coexisten cuatro registros en esta entidad, dos de la transacción 0 y dos de la transacción 8. Los dos primeros son semejantes a los de la transacción 1. Los dos siguientes son los asientos 3 y 4, uno débito y otro crédito, cada uno por €2400 que es el valor a pagar.

La transacción 8, como se ve en el "Diario" en la figura 8, se moverá por las rutas 8 y 9, ya que existe un valor no nulo 78 (número de factura), para la clave foránea 8 (FK8) en el asiento 3, y un valor no nulo 3849 (número de la cuenta del cheque) para la clave foránea 9 (FK9) en el asiento 4. Además, el programa correspondiente a la transacción 8 accederá a las tablas "Facturas", "Proveedores", "Cuentas", "Bancos" y "Mayor" del MODE modificándolas como se ve a continuación:

Diario

Tra	***Fecha***	***Hora***	***Nro_docu***	***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	Cu	Empresa	Fulano	2	Cr	2400	200
8	20060115	14:30	12345	Fa	Empresa	Fulano	3	Db	2400	200
8	20060115	14:30	10263	Ch	Aceros	Fulano	4	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
Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	*78*	Nulls ←
Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	*3849* ←

FacturasP

Nro_fac	*Valor_fac*	*Fecha_fac*	*Nit_prov*
78	2400	20051215	91300 →
7000			

Proveedores

nit_prov	*nombre*	*teléfono*	*dirección*	*saldo*	*cuenta_c*
91300	Aceros	2348789	K 32 #3-01	4600	400

Cuentas

cuenta_b	*ch_gir*	*v_gir*	*ch_con*	*v_con*	*saldo*	*nit_bco*
3849	1	2400	0	0	50000	90318

Bancos

nit_bco	*nombre*	*teléfono*	*dirección*	*saldo*	*cuenta_c*
90318	Nacional	2479853	cl. 20#3-10	50000	572

Mayor

cuenta	*nombre_cuenta*	*saldo*	*Débitos*	*créditos*
572	Bancos	y		y2+2400
400	proveedores	u	u1+2400	

Figura 8: Actualización de tablas en el pago de compra a plazos.

Los valores x_i (para "*Existencias comerciales*"), y_i (para "*Bancos*") y u_i (para "*proveedores*") en el archivo "Mayor", se refieren a los créditos o a los débitos acumulados del mes con los cuales se calculará al terminar el período contable (el mes) los nuevos saldos x, y, u de cada cuenta.

3. METODOLOGÍA

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

En el siguiente gráfico se resumen las etapas de elaboración del proyecto, obteniendo como resultado el intérprete.

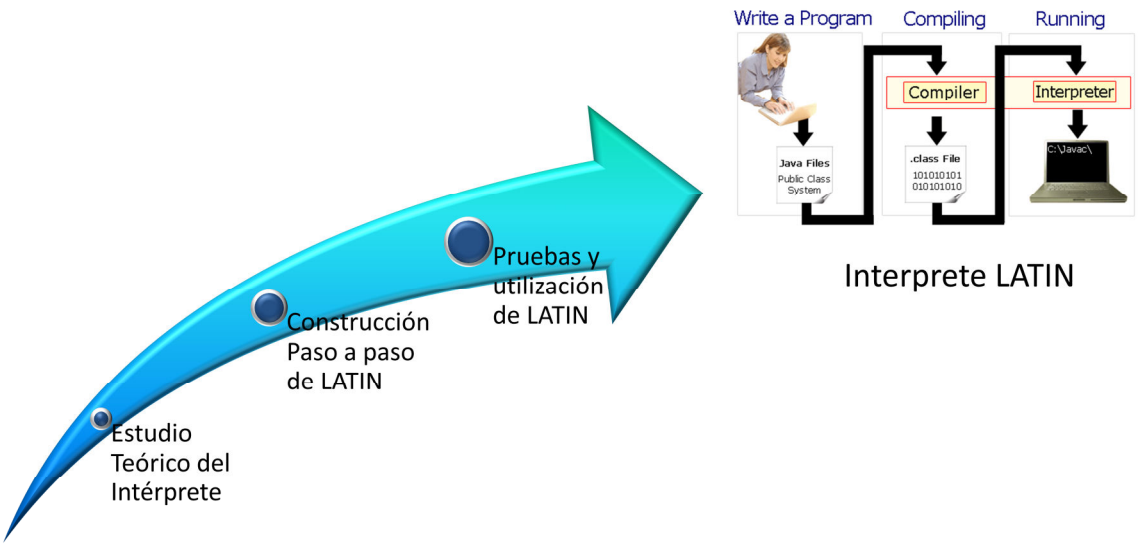

Figura 9: Principales etapas de la elaboración de LATIN

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.
- En muchos casos los usuarios no tienen una idea establecida de lo que van a utilizar, pero tampoco quieren emplear un sistema que requiera de una gran preparación, pensando precisamente en esto, LATIN lo puede comprender fácilmente cualquier persona que posea o no conocimientos de software.
- 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.

3.2 Diagramas de UML

3.2.1 Diagrama de Caso de Uso

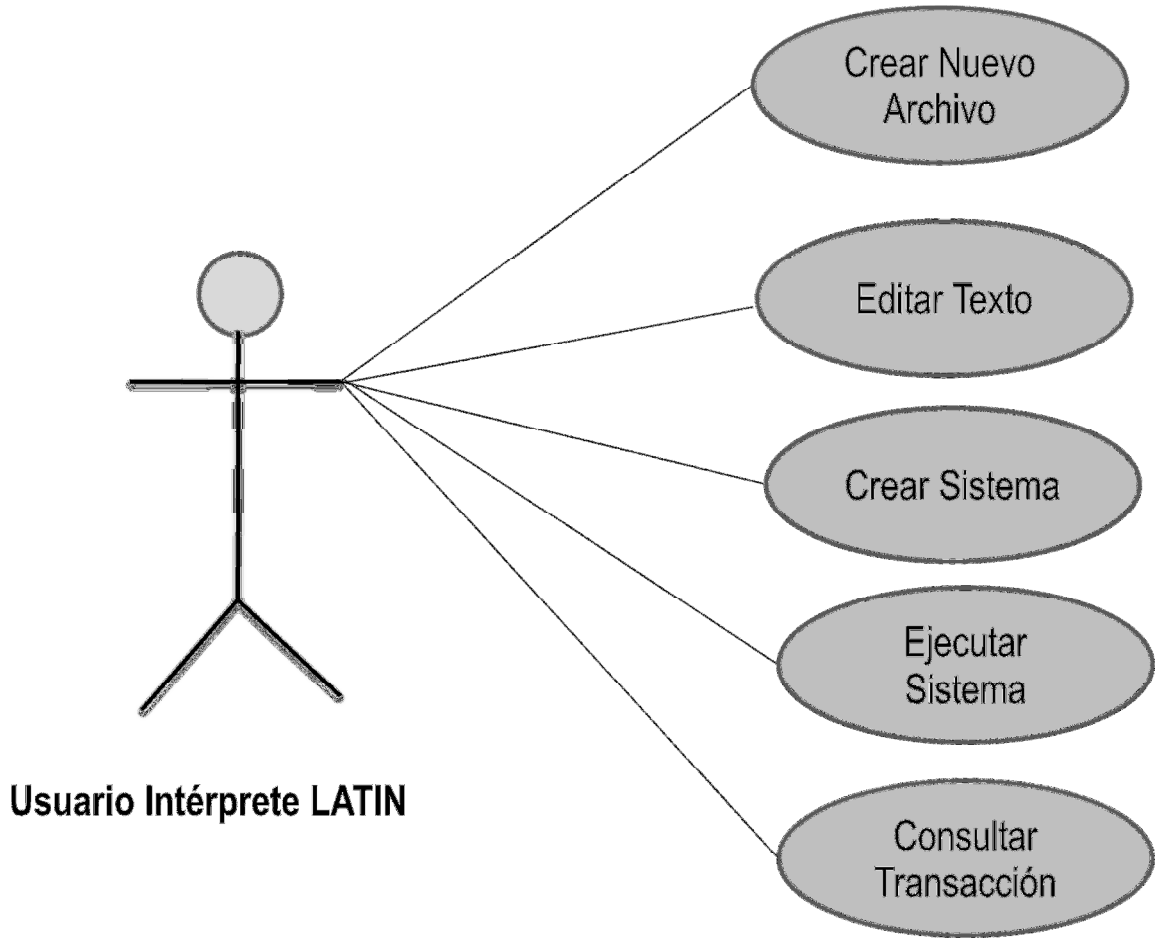

Figura 10: Diagrama Casos de Uso Intérprete LATIN

3.2.1.1 Descripción del Caso de Uso

A continuación se presenta la explicación detallada de las acciones realizadas por el usuario del Intérprete LATIN:

USUARIO	
Caso de Uso	**Descripción**
Crear Nuevo Archivo	Permite al usuario obtener una pantalla en la cual va poder crear un nuevo trabajo las veces que lo requiera.
Editar Texto	El usuario puede escribir, borrar y realizar los cambios que necesite durante este caso.
Crear Sistema	Aquí tendrá la posibilidad de iniciar el trabajo primordial, analizar el contenido del texto y visualizar la estructura jerárquica del intérprete.
Correr Sistema	Al indicar este caso el usuario se traslada a visualizar los formatos ejemplos diseñados para ilustrar el funcionamiento del intérprete.
Consultar Transacción	Se obtiene de manera inmediata las operaciones digitadas anteriormente y el resultado de dicho proceso.

Tabla 3: Descripción Caso de Uso

3.2.2 Diagrama de Clases

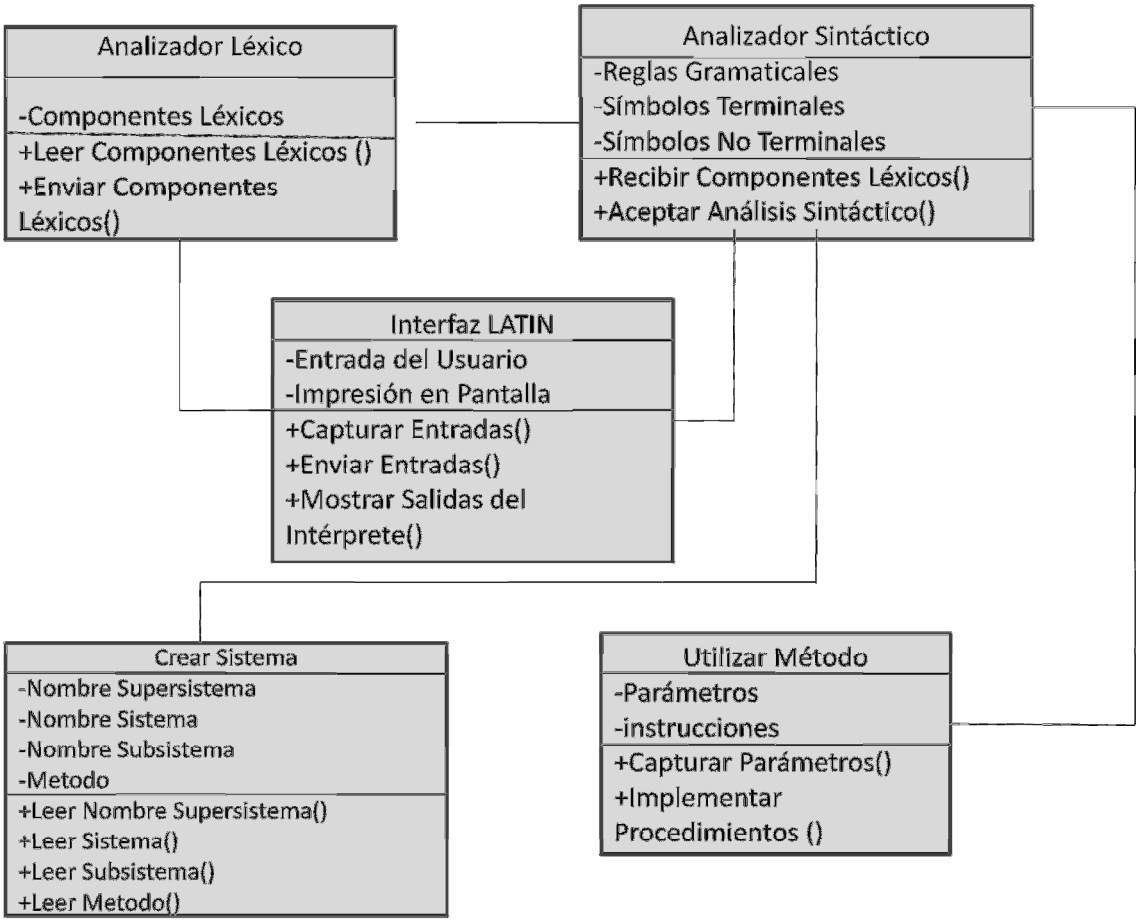

Figura 11: Diagrama de Clases

3.3 Requisitos del Sistema

3.3.1 Requisitos de Hardware

- Procesador Intel Dual Core o superior.
- 512 MB de memoria RAM o superior.
- 1 GB de espacio libre en disco o superior.

3.3.2 Requisitos de Software

Para la Ejecución del Intérprete LATIN para la implementación de modelos emergentes de datos se necesita lo siguiente:

- Servidor de Bases de Datos MySQL 5.0 (con el motor de almacenamiento InnoDB como predeterminado)

- Java SE Runtime Environment (JRE) versión 6
- Sistema Operativo: Windows de 32 bits (XP, 7).

4. DESARROLLO DEL INTERPRETE LATIN PARA LA IMPLEMENTACIÓN DE MODELOS EMERGENTE DE DATOS

4.1 Presentación de la Interfaz

A continuación se presenta la vista general de la interfaz, lista para ser utilizada (el funcionamiento de los botones será explicado con detalle en el capítulo 5 de este libro).

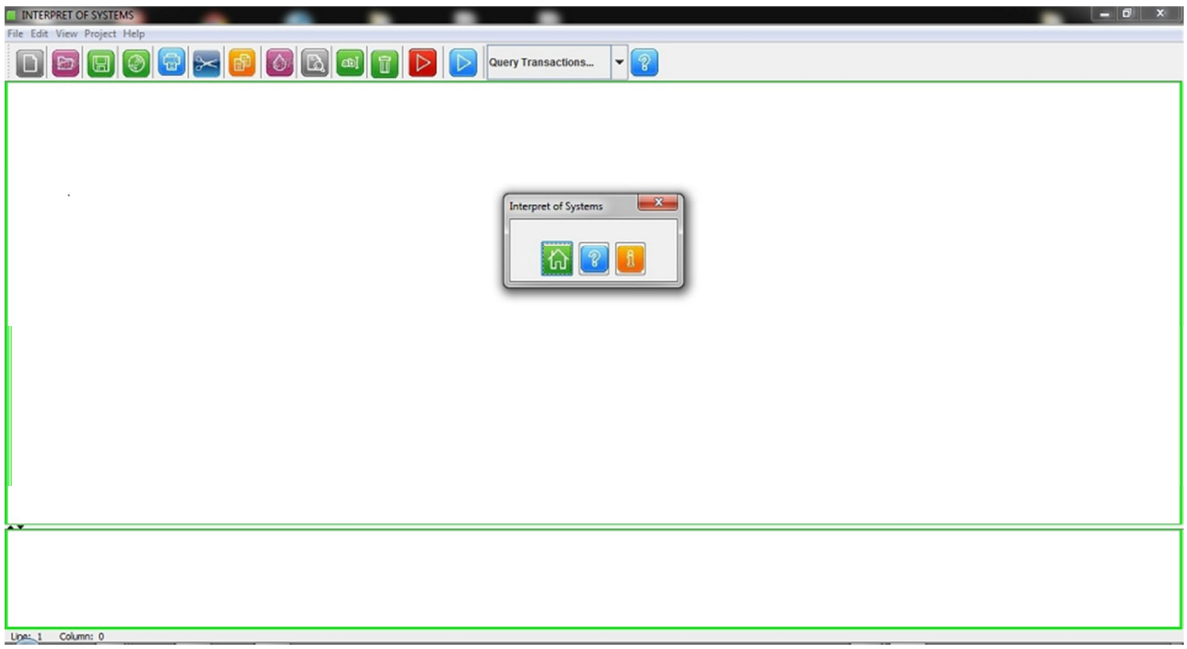

Figura 12: Interfaz Grafica de LATIN

4.2 Esquema de los Analizadores

4.2.1 Analizador Léxico

```
import java_cup.runtime.*;
```

```
import javax.swing.*;
import javax.swing.text.Caret;
import java.awt.event.*;
import java.awt.*;
import java.io.*;


%%
/* -----------------Opciones y Declaraciones----------------- */
/*
  El nombre de la Clase que JFlex creara sera Lexer
  El codigo se escribira en el archivo Lexer.java
*/
%class Lexer
/*
 El número actual de líneas puede ser accedido por la variable yyline
 y el número actual de columnas con la variable yycolumn
*/
%line
%column
   /* Compatibilidad con el generador de analizadores sintactico CUP */
%cup


/*Declaraciones  */
%{
```

```
   /* Para crear un nuevo java_cup.runtime.Symbol con información sobre el actual   token, los token
no tendrian valor en este caso. */

 private Symbol symbol(int type) {

return new Symbol(type, yyline, yycolumn);

   }

   /* También se crea un nuevo java_cup.runtime.Symbol con información sobre el actual token, pero este
objecto tiene un valor. */

private Symbol symbol(int type, Object value) {

return new Symbol(type, yyline, yycolumn, value);

   }

%}

/*Declaraciones Macro

  Estas declaraciones son expresiones regulares que serán usadas después en las Reglas Léxicas

*/

 /* Comentarios */

Comentario = {ComentarioTradicional} | {FinLineaComentario} | {DocumentacionComentario}

InputCharacter = [^\r\n]

ComentarioTradicional   = "/*" [^*] ~"*/" | "/*" "*"+ "/"

FinLineaComentario     = "//" {InputCharacter}* {LineTerminator}

DocumentacionComentario = "/**" {ContenidoComentario} "*"+ "/"

ContenidoComentario      = ( [^*] | \*+ [^/*] )*

  /* Final de una línea \r , \n , o  \r\n. */

LineTerminator = \r|\n|\r\n

  /* Espacio en Blanco. */

WhiteSpace     = {LineTerminator} | [ \t\f]
```

```
/* Números  */

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


/* Campo Fecha */

//DATE

/*Una fecha. El rango soportado es de '1000-01-01' a '9999-12-31'. MySQL muestra valores DATE en formato
'YYYY-MM-DD', pero permite asignar valores a columnas DATE usando cadenas de caracteres o números.  */

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


/* Campo Hora */

//TIME

/*Una hora. El rango es de '-838:59:59' a '838:59:59'. MySQL muestra los valores TIME en formato
'HH:MM:SS', pero permite asignar valores a columnas TIME usando números o cadenas de caracteres. */

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

/* Variable */

dec_variable = [A-Za-z_][A-Za-z_0-9]*

/* Comparación */

comparar =          "="
                        | "<>"
                        | "<"
                        | ">"
```

```
                    | "<="
                    | ">="

%%
/* ------------------------Reglas Léxicas---------------------- */

<YYINITIAL> {

   ";"                      {  return symbol(sym.PUNTOYCOMA); }
   "+"                      { return symbol(sym.MAS); }
   "-"                       {  return symbol(sym.MENOS); }
   "*"                       {  return symbol(sym.POR); }
   "/"                       {  return symbol(sym.DIVISION); }
   "("                       {  return symbol(sym.IPAREN); }
   ")"                       {  return symbol(sym.DPAREN); }
   "{"                      {  return symbol(sym.ILLAVE); }
   "}"                      {  return symbol(sym.DLLAVE); }
   ","                      {  return symbol(sym.COMA); }
   "."                      {  return symbol(sym.PUNTO); }
   ":"                      {  return symbol(sym.DOSPUNTO); }
   "'"                      {  return symbol(sym.COMILLA); }
   "&"                       {  return symbol(sym.AMPERSAND); }

   /* Palabras Reservervadas */
   "all"                    {  return symbol(sym.ALL); }
```

```
"avg"                 {  return symbol(sym.AMMSC, new String(yytext())); }
"min"                 {  return symbol(sym.AMMSC, new String(yytext())); }
"max"                 {  return symbol(sym.AMMSC, new String(yytext())); }
"sum"                 {  return symbol(sym.AMMSC, new String(yytext())); }
"count"               {  return symbol(sym.AMMSC, new String(yytext())); }
"and"                 {  return symbol(sym.AND); }
"any"                 {  return symbol(sym.ANY); }
"auto_increment"      {  return symbol(sym.AUTOINCREMENT); }
"between"             { return symbol(sym.BETWEEN); }
"by"                  {   return symbol(sym.BY); }
"varchar"             {  return symbol(sym.VARCHAR); }
"cascade"             {  return symbol(sym.CASCADE); }
"check"               {  return symbol(sym.CHECK); }
"System"              {  return symbol(sym.SYSTEM); }
"SuperSystem"         {  return symbol(sym.SUPERSYSTEM); }
"SubSystem"           {  return symbol(sym.SUBSYSTEM); }
"date"                {  return symbol(sym.DATE); }
"decimal"             {  return symbol(sym.DECIMAL); }
"default"             {  return symbol(sym.DEFAULT); }
"delete"              {  return symbol(sym.DELETE); }
"distinct"            {  return symbol(sym.DISTINCT); }
"double"              {  return symbol(sym.DOUBLE); }
"drop"                {  return symbol(sym.DROP); }
"class"               {  return symbol(sym.CLASS); }
"escape"              {  return symbol(sym.ESCAPE); }
```

```
"exists"            { return symbol(sym.EXISTS); }
"extend"            { return symbol(sym.EXTEND); }
"float"             { return symbol(sym.FLOAT); }
"foreign"           { return symbol(sym.FOREIGN); }
"from"              { return symbol(sym.FROM); }
"group"             { return symbol(sym.GROUP); }
"having"            { return symbol(sym.HAVING); }
"in"                { return symbol(sym.IN); }
"indicator"         { return symbol(sym.INDICATOR); }
"insert"            { return symbol(sym.INSERT); }
"integer"           { return symbol(sym.INTEGER); }
"into"              { return symbol(sym.INTO); }
"is"                { return symbol(sym.IS); }
"key"               { return symbol(sym.KEY); }
"like"              { return symbol(sym.LIKE); }
"method"            { return symbol(sym.METHOD); }
"not"               { return symbol(sym.NOT); }
"null"              { return symbol(sym.NULLX); }
"numeric"           { return symbol(sym.NUMERIC); }
"on"                { return symbol(sym.ON); }
"or"                { return symbol(sym.OR); }
"precision"         { return symbol(sym.PRECISION); }
"primary"           { return symbol(sym.PRIMARY); }
"real"              { return symbol(sym.REAL); }
"references"        { return symbol(sym.REFERENCES); }
```

```
  "select"                  {  return symbol(sym.SELECT); }
  "set"                     {  return symbol(sym.SET); }
  "smallint"                {  return symbol(sym.SMALLINT); }
  "some"                    {  return symbol(sym.SOME); }
  "show"                    {  return symbol(sym.SHOW); }
  "table"                   {  return symbol(sym.TABLE); }
  "time"                    {  return symbol(sym.TIME); }
  "unique"                  {  return symbol(sym.UNIQUE); }
  "update"                  {  return symbol(sym.UPDATE); }
  "values"                  {  return symbol(sym.VALUES); }
  "where"                   { return symbol(sym.WHERE); }

   /* Si encuentra un numero, retorna el token INTNUM y el valor en el string yytext */
   {dec_numero}      { return symbol(sym.INTNUM, new String(yytext())); }
  /* Si encuentra un identificador retorna el token NAME     */
   {dec_variable}    { return symbol(sym.NAME, new String(yytext()));}

/* Si encuentra un identificador retorna el token FECHA     */
   {campo_fecha}    { return symbol(sym.FECHA, new String(yytext()));}
/* Si encuentra un identificador retorna el token HORA     */
   {campo_hora}    { return symbol(sym.HORA, new String(yytext()));}
/*Si encuentra un signo de comparacion retorna el token COMPARACION */
{comparar}        { return symbol(sym.COMPARACION, new String(yytext())); }
  /* No hace nada si encuetra espacio en blanco */
   {WhiteSpace}      { /* una vez encontrado, no hace nada */ }
```

```
/*Comentarios */

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

}

/* Si encuentra un caracter extraño regresa un error. */

[^]                   { Interfaz.correcto = false;


Interfaz.areaTexto2.setText(Interfaz.areaTexto2.getText()+"Error : Syntactic error \n");


Interfaz.areaTexto2.setText(Interfaz.areaTexto2.getText()+"Error : Unrecognized character < "+yytext()+"
>");

                      throw new Error("Unrecognized character < "+yytext()+" >"); }
```

4.2.2 Analizador Sintáctico

```
/* ----------------------Declaraciones Preliminares---------------------*/


/* Importar la java_cup.runtime.* y las clases necesarias para manipulación de archivos  */

import java_cup.runtime.*;

import java.lang.*;

import java.io.*;

import javax.swing.*;

//import javax.swing.text.Caret;

//import java.awt.event.*;

//import java.awt.*;

import javax.swing.text.*;

import java.awt.Color;


//  En el siguiente segmento de codigo encerrado en "parser code {:  :}" se cambian
```

```
// los metodos de reportes de error y se agregan nuevos metodos para ser usados en la generacion de
archivos*/


parser code {:

            /*Variable en la cual se almacena el nombre de la Clase,

            necesaria para la creacion de archivos de clases*/

            public String NombreClase = new String();

          /*Variable en la cual se almacena el nombre de la clase padre cuando una de  ellas posee
herencia*/

            public String ClasePadre = new String();

            /*Variable booleana que me indica si una clase posee herencia*/

            public boolean TienePadre = false;

            /*Variable en la cual se almacena el nombre de la entidad*/

            public String NombreEntidad = new String();

/* Cambia el metodo report_error que muestra la línea y la columna donde ocurrió el error en la entrada
como también la razón por medio

            del String 'message'. */

            public void report_error(String message, Object info) {

/* Crea un StringBuffer llamado 'm' con la cadena 'Error'. */

      StringBuffer m = new StringBuffer("Error");

/* Verifica si la información pasada es del tipo java_cup.runtime.Symbol. */

if (info instanceof java_cup.runtime.Symbol) {

/* Declara un objeto java_cup.runtime.Symbol 's' con la información en la info del objeto. */

java_cup.runtime.Symbol s = ((java_cup.runtime.Symbol) info);

/* Verifica si el numero de líneas en la entrada es >= cero */

if (s.left >= 0) {

 /* Agrega al final del StringBuffer un mensaje de error con el numero de la línea en la entrada. */

m.append(" in line "+(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 de la columna en la entrada. */

m.append(", column "+(s.right+1));

}

      }

 /* Agrega al final del StringBuffer un mensaje de error creado en este metodo. */

m.append(" : "+message+"\n");

/* Imprimie el contenido del StringBuffer 'm', que contiene un mensaje de error.  En AreaTexto2*/

Interfaz.correcto = false;

try {

       StyleConstants.setForeground(Interfaz.colorTexto, Color.RED);

           Interfaz.areaTexto2.getStyledDocument().insertString(Interfaz.areaTexto2.getStyledDocument().get
           Length(),m.toString()+"\n", Interfaz.colorTexto);

                       }

catch (Exception ex_) {}

      //Interfaz.areaTexto2.append(m.toString()+"\n");

                       }

/** Cambia el metodo report_fatal que reporta un error fatal mostrando el numero de la línea y la columna
donde el error ocurrió en la entrada como la razón por la que el error fatal ocurrió 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("Syntax error", cur_token);

}

public void unrecovered_syntax_error(Symbol cur_token) {
```

```
/* Este metodo es llamado por el parser si no es posible recuperarse de un error de sintaxis */

report_fatal_error("Can't continue the analysis", null);

        }

/**Metodo usado para la creacion y escritura del archivo SQL generado por LATIN que contiene las
consultas SQL equivalentes para cada una de las entidades*/

public void escribir_sql(String cadena) {

        char comilla = 34;

        try     {

                DataOutputStream archivo_sql = null;

                archivo_sql = new DataOutputStream( new
FileOutputStream("data/"+Interfaz.nombre_basedatos+"/entidades.sql",true));

                archivo_sql.writeBytes(cadena);

                archivo_sql.close();

              }

        catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/

        report_error("File not found: "+comilla+"entidades.sql"+comilla,    null);

        }

        catch (IOException ioe) {/* Error al escribir */

        report_error("can not write to file: "+comilla+"entidades.sql"+comilla, null);

        }

    }


/** Método usado para la creación de los archivos de clase, que contienen los nombres de las entidades y
los métodos de cada una de las clases declaradas por el usuario nom_archivo: Nombre del archivo de clase
donde se va a escribir cadena: Es la cadena de texto que va a escribirse en el archivo tipo: Indica si se va a
escribir el nombre de una entidad (e) o se va a escribir un metodo (m) */

public void escribir_clase(String nom_archivo, String cadena, char tipo) {

String linea = new String(); boolean escribirTitulo = true; char comilla = 34;

        try {
```

```
                BufferedReader archivo_clase = null;
                archivo_clase = new BufferedReader(new InputStreamReader(new
FileInputStream("data/"+Interfaz.nombre_basedatos+"/"+nom_archivo+".ltc")));
                if (tipo == 'e') {
                    linea = archivo_clase.readLine();
                    while (linea != null) {
                        if (linea.equals("[class]")){
                        escribirTitulo = false;
                        break;
                        }
                linea = archivo_clase.readLine();
                }
        }
            if (tipo == 'm') {
                        linea = archivo_clase.readLine();
                        while (linea != null) {
                            if (linea.equals("[method]")){
                                escribirTitulo = false;
                                break;
                                }
                        linea = archivo_clase.readLine();
                            }
                        }
                        archivo_clase.close();
                }
        catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
                report_error("File not found: "+comilla+nom_archivo+".ltc"+comilla, null);
```

```
                }
        catch(IOException ioe) {/* Error al leer*/
        report_error(": "+comilla+nom_archivo+".ltc"+comilla, null);
                }
        try {
                DataOutputStream archivo_clase = null;
                archivo_clase = new DataOutputStream( new
FileOutputStream("data/"+Interfaz.nombre_basedatos+"/"+nom_archivo+".ltc", true));

                if (escribirTitulo && tipo == 'e')
                archivo_clase.writeBytes("[class]\n");
                if (escribirTitulo && tipo == 'm')
                archivo_clase.writeBytes("\n[method]\n");
                archivo_clase.writeBytes(cadena);
                archivo_clase.close();
                }
        catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
                report_error("File not found: "+comilla+nom_archivo+".ltc"+comilla, null);
                }
        catch(IOException ioe) {/* Error al escribir */
        report_error("can't write to file: "+comilla+nom_archivo+".ltc"+comilla, null);
                }
        }
/**Método usado para copiar las instrucciones SQL del método de la clase padre al método de la clase Hijo
en su respectivo archivo de clase*/
public void escribir_herencia(String Padre, String Hijo) {
String linea = new String(); char comilla = 34;
try  {
```

```
	BufferedReader archivo_padre = null;

	archivo_padre = new BufferedReader(new InputStreamReader(new
FileInputStream("data/"+Interfaz.nombre_basedatos+"/"+Padre+".ltc")));

try {

	DataOutputStream archivo_hijo = null;

	archivo_hijo = new DataOutputStream( new
FileOutputStream("data/"+Interfaz.nombre_basedatos+"/"+Hijo+".ltc", true));

	linea = archivo_padre.readLine();

	while(linea != null) {

		if (linea.equals("{")) {

		linea = archivo_padre.readLine();

		while(linea.equals("}")==false) {

		archivo_hijo.writeBytes(linea+"\n");

		linea = archivo_padre.readLine();

		}

	}

linea = archivo_padre.readLine();

}

archivo_hijo.close();

}

catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/

report_error("File doesn't exist: "+comilla+Hijo+".ltc"+comilla, null);

	}

catch(IOException ioe) {/* Error al escribir */

report_error("can't write to file: "+comilla+Hijo+".ltc"+comilla, null);

}

archivo_padre.close();

}
```

```
catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
report_error("File doesn't exist: "+comilla+Padre+".ltc"+comilla, null);
}
catch(IOException ioe) {/* Error al leer */
report_error("can't write to file: "+comilla+Padre+".ltc"+comilla, null);
        }
}
/**Retorna un string que contiene los parámetros del método de la clase padre, para poder ser escritos en el método de la clase hijo*/
public String parametros_padre(String ClasePadre) {
String linea = new String(); char comilla = 34;
        try {
                BufferedReader archivo_padre = null;
                archivo_padre = new BufferedReader(new InputStreamReader(new
FileInputStream("data/"+Interfaz.nombre_basedatos+"/"+ClasePadre+".ltc")));
                linea = archivo_padre.readLine();
                while(linea != null){
                    if(linea.equals("[method]")) {
                        linea = archivo_padre.readLine();
                        archivo_padre.close();int us_comas = 0;
                        for ( int i=0; i<linea.length()-1; i++)   {
                                if ( linea.substring(i,i+1).compareTo(",") == 0)
                                        us_comas++;}
                                        if(us_comas != 0 )
                                        return linea.substring(linea.indexOf('(')+1,linea.indexOf(')'))+", ";
                                else
                                return linea.substring(linea.indexOf('(')+1,linea.indexOf(')'));
                                }
```

```
                    linea=archivo_padre.readLine();
                    }
                    archivo_padre.close();
                }
catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
report_error("File doesn't exist: "+comilla+ClasePadre+".ltc"+comilla, null);
 }
catch(IOException ioe) {/* Error al leer */
report_error("can't write to file: "+comilla+ClasePadre+".ltc"+comilla, null);
}
return "";
}
/**Este metodo copia el nombre de todas las entidades de la clase padre  al archivo de la clase hijo*/
public String entidades_padre(String ClasePadre, String ClaseHijo) {
String linea = new String(); char comilla = 34;
        try {
            BufferedReader archivo_padre = null;
            archivo_padre = new BufferedReader(new InputStreamReader(new
FileInputStream("data/"+Interfaz.nombre_basedatos+"/"+ClasePadre+".ltc")));
           DataOutputStream archivo_hijo = null;
        archivo_hijo = new DataOutputStream( new
FileOutputStream("data/"+Interfaz.nombre_basedatos+"/"+ClaseHijo+".ltc", true));
        linea = archivo_padre.readLine();
        while(linea.equals("[method]") == false)
        {
            if(linea.equals("[class]") == false)
                archivo_hijo.writeBytes(linea+"\n");
            linea = archivo_padre.readLine();
```

```
            }

            archivo_padre.close();

            archivo_hijo.close();

}

catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/

report_error("Can't open one of the following files: "+comilla+ClasePadre+".ltc"+comilla+" o
"+comilla+ClaseHijo+".ltc"+comilla, null);

}

catch(IOException ioe) {/*Error al leer*/

report_error("Can't write/read one of the following files: "+comilla+ClasePadre+".ltc"+comilla+" o
"+comilla+ClaseHijo+".ltc"+comilla, null);

}

return "";

}

/** Comprueba que la entidad: "NombreEntidad"  esta creada en la clase: "ClasePadre", en cuyo caso
retorna true en caso contrario retorna false */

public boolean existe_entidad(String ClasePadre, String NombreEntidad) {

 String linea = new String(); char comilla = 34;

             try{

                        BufferedReader archivo_padre = new BufferedReader(new InputStreamReader(new
FileInputStream("data/"+Interfaz.nombre_basedatos+"/"+ClasePadre+".ltc")));

            linea = archivo_padre.readLine();

            while(linea != null){

            if(linea.equals(NombreEntidad)){

                archivo_padre.close();

                return true;

                        }

            linea = archivo_padre.readLine();
```

```
            }
            archivo_padre.close();
}
catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
report_error("File doesn't exist: "+comilla+ClasePadre+".ltc"+comilla, null);
}
catch(IOException ioe) {/*Error al leer*/
report_error("Can't write to file: "+comilla+ClasePadre+".ltc"+comilla, null);
}
return false;
}
/**Modifica los métodos de  una clase padre cuando en una entidad hija se agregan
Nuevos campos a sus entidades*/
public void actualizar_metodo(String NombreClase, String NombreEntidad) {
String linea = new String(); char comilla = 34;
//Paso todo el contenido del archivo de clase, a un archivo temporal
            try{
                        BufferedReader archivo_clase = new BufferedReader(new InputStreamReader(new
FileInputStream("data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".ltc")));
                        DataOutputStream archivo_temporal = new DataOutputStream( new
FileOutputStream("data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".tmp"));
                        linea = archivo_clase.readLine();
                        while(linea != null){
                        archivo_temporal.writeBytes(linea+"\n");
                        linea = archivo_clase.readLine();
                        }
                        archivo_temporal.close();
                        archivo_clase.close();
```

```
                    }

catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/

report_error("Can't open one of the following files: "+comilla+NombreClase+".ltc"+comilla+" o
"+comilla+NombreClase+".tmp"+comilla, null);

}

catch(IOException ioe) {/*Error al leer*/

report_error("Can't write/read one of the following files: "+comilla+NombreClase+".ltc"+comilla+" o
"+comilla+NombreClase+".tmp"+comilla, null);

}

//Ahora reescribo todo el archivo de clase, haciendo las modificaciones al método si encuentra una
sentencia INSERT INTO NombreEntidad

        try{

                BufferedReader arch_temporal = new BufferedReader(new InputStreamReader(new
FileInputStream("data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".tmp")));

                DataOutputStream arch_clase = new DataOutputStream( new
FileOutputStream("data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".ltc"));

                linea = arch_temporal.readLine();

                while(linea != null){

                        if(linea.startsWith("  INSERT INTO "+NombreEntidad)){

                                linea = linea.substring(0, linea.lastIndexOf(')'))+", NULL);";

                                arch_clase.writeBytes(linea+"\n");

                                }

                        else

                                arch_clase.writeBytes(linea+"\n");

                                linea = arch_temporal.readLine();

                        }

        arch_clase.close();

        arch_temporal.close();

}
```

```
catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/

report_error("Can't open one of the following files: "+comilla+NombreClase+".ltc"+comilla+" o
"+comilla+NombreClase+".tmp"+comilla, null);

}

catch(IOException ioe) {/*Error al leer*/

report_error("Can't write/read one of the following files: "+comilla+NombreClase+".ltc"+comilla+" o
"+comilla+NombreClase+".tmp"+comilla, null);

}

//Por ultimo borro el archivo temporal

File temporal = new File("data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".tmp");

temporal.delete();

}

:};

/* El bloque de código encerrado en "init with {: :}" se ejecuta justo antes de que el parser empiece a hacer
el analisis*/

init with{:

/*Verifico que la carpeta "data" donde se guardan los archivos de clase este creada,

si no es así, entonces se crea antes de empezar el análisis */

File carpeta = new File("data");

        if(carpeta.isDirectory() == false)

                carpeta.mkdir();


                carpeta = new File("data/"+Interfaz.nombre_basedatos);

                if(carpeta.isDirectory() == false)

                        carpeta.mkdir();


/*La siguiente porción de código es usada para crear un nuevo archivo entidades.sql justo antes de empezar
el analisis y escribir un encabezado. Si ya existe un archivo llamado entidades.sql se borra todo su
contenido*/
```

```
                                char comilla = 34;
try {
        DataOutputStream archivo_sql = null;
        archivo_sql = new DataOutputStream( new
FileOutputStream("data/"+Interfaz.nombre_basedatos+"/entidades.sql"));
        archivo_sql.writeBytes("/*"+
        "Archivo SQL..... generado por software MODE v1.1\n"+
"Este archivo contiene las consultas SQL equivalentes a cada una de las entidades\n"+
"Programado Por:\n"+
        "     \n"+
        "     * Yuliana Mayerly Soto\n"+
        "     * Julián Ricardo Camargo\n"+
        "     \n"+
        "     Universidad Industrial de Santander\n"+
        "Nombre del archivo: entidades.sql  "+
        "*/\n");
        archivo_sql.close();
        }
catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
report_error("Can't open to file: "+comilla+"entidades.sql"+comilla, null);}
catch (IOException ioe) { /* Error al escribir */
report_error("can't write to file: "+comilla+"entidades.sql"+comilla, null);
}
:};
/* ------------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, DOSPUNTO, ALL, AND, ANY, BETWEEN, BY, VARCHAR, CASCADE, CHECK, SYSTEM;

terminal        COMPARACION, COMILLA, DATE, DECIMAL, DEFAULT, DELETE, DISTINCT, DOUBLE;

terminal        CLASS, ESCAPE, EXISTS, EXTEND, FLOAT, FOREIGN, FROM, GROUP, HAVING;

terminal        IN, INDICATOR, INSERT, INTEGER, INTO, IS, KEY, LIKE, METHOD, NOT, NULLX;

terminal        NUMERIC, ON, OR, PRECISION, PRIMARY, REAL, REFERENCES, SELECT, SET, SMALLINT;

terminal        SOME, TIME, UNIQUE, UPDATE, VALUES, WHERE, PUNTOYCOMA, SHOW,
                AUTOINCREMENT;

terminal        Object AMMSC, INTNUM, NAME, HORA, FECHA, DROP, TABLE, SUPERSYSTEM, SUBSYSTEM;

/* No terminales usados en la gramática.  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 especifico. Por lo
tanto,  podría ser un Integer o un String o lo que sea. */

non terminal Object     select_statement, consulta_entidades, update_statement_searched;

non terminal Object     assignment, assignment_commalist, insert_atom;

non terminal Object     insert_atom_commalist, query_spec, values_or_query_spec,
                        opt_column_commalist;

non terminal Object     insert_statement, opt_where_clause, delete_statement_searched;

non terminal Object     definicion_funciones, cuerpo_metodo, insert_paramet, paramet_lista,
                        definicion_metodo;

non terminal Object     parameter, identificador, columna, tipo_dato, column_ref, table;

non terminal Object     function_ref, parameter_ref, atom, escalar_exp, opt_having_clause,
                        column_ref_commalist;

non terminal Object     opt_group_by_clause, where_clause, table_ref, table_ref_commalist,
                        from_clause, table_exp;

non terminal Object     escalar_exp_commalist, selection, opt_all_distinct, subquery, prueba_existencia,
                        any_all_some;

non terminal Object     todo_o_algun_predicado, atom_commalist, en_predicado, prueba_para_null,
                        opt_escape;

non terminal Object     like_predicado, entre_predicado, comparacion_predicado, predicado,
                        condicion_busqueda;
```

```
non terminal Object     listado_columna, def_interrelacion, listado_opc_columna, opc_def_columna,
                        def_columna;

non terminal Object     elemento_tabla, listado_atributo, definicion_entidad, cuerpo_clase,
                        definicion_clase;

non terminal Object     usar_metodo, inicial, herencia, identificador_copy, seguir;

non terminal Object     insertar_parametro, parametro_lista, aux_definicion_funciones, drop_statement;


/* -------------Precedencia y asociación ----------- */

/*  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;

/* ----------------------------  Gramática  --------------------- */

/* Producciones para gramatica :: ODL */

inicial             ::=     definicion_clase PUNTOYCOMA seguir

                            {: Interfaz.def_clase = true; :}

                            |

                            consulta_entidades PUNTOYCOMA

                            {: Interfaz.consulta_sql = true; :}

                            |

                            METHOD usar_metodo PUNTOYCOMA

                            {: Interfaz.usar_metodo = true; :}

                            |

                            SHOW NAME PUNTOYCOMA  // Ver Grafico

                            ;

seguir              ::=      |

                            inicial
```

```
                        ;
usar_metodo     ::=     NAME IPAREN parametro_lista DPAREN
                        ;
parametro_lista  ::=    insertar_parametro
                        |
                        parametro_lista COMA insertar_parametro
                        ;
insertar_parametro ::=  NAME
                        |
                        INTNUM
                        |
                        NULLX
                        |
                        HORA
                        |
                        FECHA
                        ;
/* Definicion de una Clase */

definicion_clase  ::=   SYSTEM NAME:nombre_clase
                        {:
                        parser.NombreClase = nombre_clase.toString();
                        /*A continuacion se borran los archivos de clase con el mismo nombre */
                        File archivo_clase;
                        archivo_clase = new
                        File("data/"+Interfaz.nombre_basedatos+"/"+parser.NombreClase+".ltc");
                                if(archivo_clase.exists())
```

```
                                        if(archivo_clase.delete()){

                                                char comilla = 34;


        Interfaz.areaTexto2.setText(Interfaz.areaTexto2.getText()+"System:
"+comilla+parser.NombreClase+comilla+" has been modified\n");

                                        }

                                        archivo_clase.createNewFile();

                                        :}

                                        herencia:padre

                                        {: parser.ClasePadre = padre.toString(); :}


                                        cuerpo_clase

                                        ;

definicion_clase  ::=       SUPERSYSTEM NAME:nombre_clase

                            {:

                            parser.NombreClase = nombre_clase.toString();

                            /*A continuacion se borran los archivos de clase con el mismo nombre */

                            File archivo_clase;

                            archivo_clase = new
                            File("data/"+Interfaz.nombre_basedatos+"/"+parser.NombreClase+".ltc");

                            if(archivo_clase.exists())

                                    if(archivo_clase.delete()){

                                            char comilla = 34;


        Interfaz.areaTexto2.setText(Interfaz.areaTexto2.getText()+"superSystem:
"+comilla+parser.NombreClase+comilla+" has been modified\n");

                                            }

                            archivo_clase.createNewFile();

                            :}
```

```
                        cuerpo_clase

                        ;

definicion_clase  ::=   SUBSYSTEM NAME:nombre_clase

                        {:

                        parser.NombreClase = nombre_clase.toString();

                        /*A continuacion se borran los archivos de clase con el mismo nombre */

                        File archivo_clase;

                        archivo_clase = new
                        File("data/"+Interfaz.nombre_basedatos+"/"+parser.NombreClase+".ltc");

                        if(archivo_clase.exists())

                                if(archivo_clase.delete()){

                                char comilla = 34;


        Interfaz.areaTexto2.setText(Interfaz.areaTexto2.getText()+"subSystem:
"+comilla+parser.NombreClase+comilla+" has been modified\n");

                                }

                        archivo_clase.createNewFile();

                        :}

                herencia:padre

                {: parser.ClasePadre = padre.toString(); :}

                cuerpo_clase;

herencia::=             {: RESULT = new String(); parser.TienePadre = false; :}

                        |

                        EXTEND identificador:padre

                        {: RESULT = new String(padre.toString()); parser.TienePadre = true; :}

                        ;

cuerpo_clase    ::=     ILLAVE

                        definicion_entidad
```

```
                    {:
                    if (parser.TienePadre)
                     parser.entidades_padre(parser.ClasePadre, parser.NombreClase); // Modificado
                    :}
                    definicion_metodo
                    DLLAVE
                    ;
definicion_entidad  ::=     |
                            CLASS NAME:nombre_entidad
                            {:      parser.NombreEntidad = nombre_entidad.toString();
                            parser.escribir_sql("\n\n/*Entidad "+parser.NombreEntidad+" from to
                            the class "+parser.NombreClase+" */\n");
                                    if (parser.TienePadre &&
                                    parser.existe_entidad(parser.ClasePadre,
                                    parser.NombreEntidad))
                                    parser.escribir_sql("ALTER TABLE "+ parser.NombreEntidad + "
                                    \n");
                                    else {
                                    parser.escribir_sql("CREATE TABLE "+ parser.NombreEntidad +
                                    "(\n" );
                               parser.escribir_clase(parser.NombreClase, parser.NombreEntidad
                               +"\n",'e')}
                            :}
                            ILLAVE listado_atributo:list_atrib DLLAVE
                            {:      if (parser.TienePadre &&
                            parser.existe_entidad(parser.ClasePadre, parser.NombreEntidad))
                            parser.escribir_sql(list_atrib.toString()+"\n;" );
                            else
                            parser.escribir_sql(list_atrib.toString()+"\n);" );
```

```
                    :}

                    definicion_entidad

                    ;

listado_atributo    ::= elemento_tabla:elem_tabla

                    {:      if (parser.TienePadre &&
                    parser.existe_entidad(parser.ClasePadre, parser.NombreEntidad))

                    RESULT = new String(" ADD "+elem_tabla.toString());

                    else

                    RESULT = new String(" "+elem_tabla.toString());

                    :}

                    |

                    listado_atributo:list_atrib COMA elemento_tabla:elem_tabla

                    {:      if (parser.TienePadre &&
                    parser.existe_entidad(parser.ClasePadre, parser.NombreEntidad))

                    RESULT = new String(list_atrib.toString()+",\n ADD
                    "+elem_tabla.toString());

                    else

                    RESULT = new String(list_atrib.toString()+",\n "+elem_tabla.toString());


                    :}

                    ;

elemento_tabla  ::= def_columna:def_col

                    {: RESULT = new String(def_col.toString()); :}

                    |

                    def_interrelacion:def_inter

                    {:

                    RESULT = new String(def_inter.toString());
```

```
                    if(parser.TienePadre &&
                    parser.existe_entidad(parser.ClasePadre,
                    parser.NombreEntidad))

                    parser.actualizar_metodo(parser.ClasePadre,parser.NombreEnti
dad);

:}

;

def_columna      ::=    columna:col tipo_dato:tip_dat opc_def_columna:opc_def_col

{: RESULT = new String(col.toString()+"
"+tip_dat.toString()+opc_def_col.toString()); :}

;

opc_def_columna  ::=    {: RESULT = new String(); :}

|

opc_def_columna:opc_def_col listado_opc_columna:list_opc_col

{: RESULT = new String(opc_def_col.toString()+" "+list_opc_col.toString());
:}

;


listado_opc_columna  ::=    NOT NULLX

{: RESULT = new String("NOT NULL"); :}

|

NOT NULLX UNIQUE

{: RESULT = new String("NOT NULL UNIQUE"); :}

|

NOT NULLX PRIMARY KEY

{: RESULT = new String("NOT NULL PRIMARY KEY"); :}

|

NOT NULLX AUTOINCREMENT PRIMARY KEY

{: RESULT = new String("NOT NULL AUTO_INCREMENT PRIMARY KEY"); :}
```

```
                    |
                    DEFAULT INTNUM:numero
                    {: RESULT = new String("DEFAULT "+numero.toString()); :}
                    |
                    DEFAULT NULLX
                    {: RESULT = new String("DEFAULT NULL"); :}
                    |
                    DEFAULT NAME:nombre
                    {: RESULT = new String("DEFAULT "+nombre.toString()); :}
                    |
                    DEFAULT COMILLA NAME:nombre COMILLA
                    {: RESULT = new String("DEFAULT '"+nombre.toString()+"'"); :}
                    |
                    CHECK IPAREN condicion_busqueda:cond_busq DPAREN
                    {: RESULT = new String("CHECK("+cond_busq.toString()+")"); :}
                    ;
def_interrelacion ::=   FOREIGN KEY IPAREN listado_columna:list_col DPAREN REFERENCES
                    table:nombre
                    {: RESULT = new String("FOREIGN KEY("+list_col.toString()+") REFERENCES
                    "+nombre.toString()); :}
                    |
                    FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN REFERENCES
                    table:nombre IPAREN listado_columna:list_col2 DPAREN
                    {: RESULT = new String("FOREIGN KEY("+list_col1.toString()+")
                    REFERENCES "+nombre.toString()+"("+list_col2.toString()+")"); :}
                    |
                    FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN REFERENCES
                    table:nombre IPAREN listado_columna:list_col2 DPAREN ON DELETE
                    CASCADE
```

{: RESULT = new String("FOREIGN KEY("+list_col1.toString()+") REFERENCES "+nombre.toString()+"("+list_col2.toString()+") ON DELETE CASCADE"); :}

|

FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN REFERENCES table:nombre IPAREN listado_columna:list_col2 DPAREN ON UPDATE CASCADE

{: RESULT = new String("FOREIGN KEY("+list_col1.toString()+") REFERENCES "+nombre.toString()+"("+list_col2.toString()+") ON UPDATE CASCADE"); :}

|

FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN REFERENCES table:nombre IPAREN listado_columna:list_col2 DPAREN ON DELETE CASCADE ON UPDATE CASCADE

{: RESULT = new String("FOREIGN KEY("+list_col1.toString()+") REFERENCES "+nombre.toString()+"("+list_col2.toString()+") ON DELETE CASCADE ON UPDATE CASCADE"); :}

|

FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN REFERENCES table:nombre IPAREN listado_columna:list_col2 DPAREN ON UPDATE CASCADE ON DELETE CASCADE

{: RESULT = new String("FOREIGN KEY("+list_col1.toString()+") REFERENCES "+nombre.toString()+"("+list_col2.toString()+") ON UPDATE CASCADE ON DELETE CASCADE"); :}

;

listado_columna ::= columna:col

{: RESULT = new String(col.toString()); :}

|

listado_columna:list_col COMA columna:col

{: RESULT = new String(list_col.toString()+", "+col.toString()); :}

;

condicion_busqueda ::= condicion_busqueda:cond_busq1 OR condicion_busqueda:cond_busq2

```
                    {: RESULT = new String(cond_busq1.toString()+" OR
                    "+cond_busq2.toString()); :}

                    |

                    condicion_busqueda:cond_busq1 AND condicion_busqueda:cond_busq2

                    {: RESULT = new String(cond_busq1.toString()+" AND
                    "+cond_busq2.toString()); :}

                    |

                    NOT condicion_busqueda:cond_busq

                    {: RESULT = new String(" NOT "+cond_busq.toString()); :}

                    |

                    IPAREN condicion_busqueda:cond_busq DPAREN

                    {: RESULT = new String(" ("+cond_busq.toString()+")"); :}

                    |

                    predicado:pred

                    {: RESULT = new String(pred.toString()); :}

                    ;

predicado     ::=   comparacion_predicado:comp_pred

                    {: RESULT = new String(comp_pred.toString()); :}

                    |

                    entre_predicado:ent_pred

                    {: RESULT = new String(ent_pred.toString()); :}

                    |

                    like_predicado:like_pred

                    {: RESULT = new String(like_pred.toString()); :}

                    |

                    prueba_para_null:prueb_null

                    {: RESULT = new String(prueb_null.toString()); :}

                    |
```

```
                    en_predicado:en_pred

                    {: RESULT = new String(en_pred.toString()); :}

                    |

                    todo_o_algun_predicado:toa_pred

                    {: RESULT = new String(toa_pred.toString()); :}

                    |

                    prueba_existencia:exist

                    {: RESULT = new String(exist.toString()); :}

                    ;


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

                    {: RESULT = new String(esc_exp1.toString()+" "+comp.toString()+"
                    "+esc_exp2.toString()); :}

                    |

                    escalar_exp:esc_exp COMPARACION:comp subquery:subcons

                    {: RESULT = new String(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 = new String(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 = new String(esc_exp1.toString()+" BETWEEN
                    "+esc_exp2.toString()+" AND "+esc_exp3.toString()); :}

                    ;
```

```
like_predicado    ::=        escalar_exp:esc_exp NOT LIKE atom:atm opt_escape:opt_esc

                             {: RESULT = new String(esc_exp.toString()+" NOT LIKE
                             "+atm.toString()+opt_esc.toString()); :}

                             |

                             escalar_exp:esc_exp LIKE atom:atm opt_escape:opt_esc

                             {: RESULT = new String(esc_exp.toString()+" LIKE
                             "+atm.toString()+opt_esc.toString()); :}

                             ;


opt_escape ::=               {: RESULT = new String(); :}

                             |

                             ESCAPE atom:atm

                             {: RESULT = new String(" ESCAPE "+atm.toString()); :}

                             ;


prueba_para_nul ::=          column_ref:col_ref IS NOT NULLX

                             {: RESULT = new String(col_ref.toString()+ "IS NOT NULL "); :}

                             |

                             column_ref:col_ref IS NULLX

                             {: RESULT = new String(col_ref.toString()+ "IS NULL "); :}

                             ;
en_predicado      ::=        escalar_exp:esc_exp NOT IN IPAREN subquery:sub_cons DPAREN

                             {: RESULT = new String(esc_exp.toString()+" NOT IN(
                             "+sub_cons.toString()+")"); :}

                             |

                             escalar_exp:esc_exp IN IPAREN subquery:sub_cons DPAREN

                             {: RESULT = new String(esc_exp.toString()+" "+sub_cons.toString()+")"); :}
```

```
                                |

                                escalar_exp:esc_exp NOT IN IPAREN atom_commalist:atm_com DPAREN

                                {: RESULT = new String(esc_exp.toString()+" NOT IN(
                                "+atm_com.toString()+")"); :}

                                |

                                escalar_exp:esc_exp IN IPAREN atom_commalist:atm_com DPAREN

                                {: RESULT = new String(esc_exp.toString()+" IN(
                                "+atm_com.toString()+")"); :}

                                ;

atom_commalist ::=              atom:atm

                                {: RESULT = new String(atm.toString()); :}

                                |

                                atom_commalist:atm_com COMA atom:atm

                                {: RESULT = new String(atm_com.toString()+", "+atm.toString()); :}

                                ;

todo_o_algun_predicado::=       escalar_exp:esc_exp COMPARACION:comp  any_all_some:any_some
                                subquery:sub_cons

                                {: RESULT = new String(esc_exp.toString()+" "+comp.toString()+"
                                "+any_some.toString()+" "+sub_cons.toString()); :}

                                ;

any_all_some      ::=           ANY

                                {: RESULT = new String("ANY"); :}

                                |

                                ALL

                                {: RESULT = new String("ALL"); :}

                                |

                                SOME

                                {: RESULT = new String("SOME"); :}
```

```
                                ;

prueba_existencia ::=           EXISTS subquery:sub_cons
                                {: RESULT = new String(" EXISTS "+sub_cons.toString()); :}
                                ;
subquery ::=                    IPAREN SELECT opt_all_distinct:opt_all selection:sel table_exp:tab_exp
                                DPAREN
                                {: RESULT = new String("(SELECT "+opt_all.toString()+" "+sel.toString()+"
                                "+tab_exp.toString()+")"); :}
                                ;
opt_all_distinct   ::=          {: RESULT = new String(); :}
                                |
                                ALL
                                {: RESULT = new String("ALL"); :}
                                |
                                DISTINCT {: RESULT = new String("DISTINCT"); :}
                                ;

selection  ::=                  escalar_exp_commalist:esc_exp_list
                                {: RESULT = new String(esc_exp_list.toString()); :}
                                |
                                POR
                                {: RESULT = new String("*"); :}
                                ;
escalar_exp_commalist  ::=      escalar_exp:esc_exp
                                {: RESULT = new String(esc_exp.toString()); :}
                                |
                                escalar_exp_commalist:esc_exp_list COMA escalar_exp:esc_exp
```

```
                                {: RESULT = new String(esc_exp_list.toString()+",   "+esc_exp.toString()); :}

                                ;

table_exp ::=                   from_clause:from opt_where_clause:where opt_group_by_clause:group
                                opt_having_clause:having

                                {: RESULT = new String(from.toString()+" "+where.toString()+"
                                "+group.toString()+" "+having.toString()); :}

                                ;


from_clause ::=                 FROM table_ref_commalist:tab_ref

                                {: RESULT = new String(" FROM "+tab_ref.toString()); :}

                                ;


table_ref_commalist ::=         table_ref:tab_ref

                                {: RESULT = new String(tab_ref.toString()); :}

                                |

                                table_ref_commalist:tab_ref_list COMA table_ref:tab_ref

                                {: RESULT = new String(tab_ref_list.toString()+", "+tab_ref.toString()); :}

                                ;


table_ref ::=                   table:nombre

                                {: RESULT = new String(nombre.toString()); :}

                                ;


where_clause ::=                 WHERE condicion_busqueda:cond_busq

                                {: RESULT = new String(" WHERE "+cond_busq.toString()); :}

                                ;


opt_group_by_clause  ::=        {: RESULT = new String(); :}
```

```
                                |
                                GROUP BY column_ref_commalist:col_ref_list
                                {: RESULT = new String(" GROUP BY "+col_ref_list.toString()); :}
                                ;
column_ref_commalist  ::=       column_ref:col_ref
                                {: RESULT = new String(col_ref.toString()); :}
                                |
                                column_ref_commalist:col_ref_list COMA column_ref:col_ref
                                {: RESULT = new String(col_ref_list.toString()+", "+col_ref.toString()); :}
                                ;
opt_having_clause  ::=          {: RESULT = new String(); :}
                                |
                                HAVING condicion_busqueda:cond_busq
                                {: RESULT = new String(" HAVING "+cond_busq.toString()); :}
                                ;

escalar_exp        ::=          escalar_exp:esc_exp1 MAS escalar_exp:esc_exp2
                                {: RESULT = new String(esc_exp1.toString()+" + "+esc_exp2.toString()); :}
                                |
                                escalar_exp:esc_exp1 MENOS escalar_exp:esc_exp2
                                {: RESULT = new String(esc_exp1.toString()+" - "+esc_exp2.toString()); :}
                                |
                                escalar_exp:esc_exp1 POR escalar_exp:esc_exp2
                                {: RESULT = new String(esc_exp1.toString()+" * "+esc_exp2.toString()); :}
                                |
                                escalar_exp:esc_exp1 DIVISION escalar_exp:esc_exp2
                                {: RESULT = new String(esc_exp1.toString()+" / "+esc_exp2.toString()); :}
```

```
                    |
                    MAS escalar_exp:esc_exp
                    {: RESULT = new String(" + "+esc_exp.toString()); :}
                    |
                    MENOS escalar_exp:esc_exp
                    {: RESULT = new String(" - "+esc_exp.toString()); :}
                    |
                    atom:atm
                    {: RESULT = new String(atm.toString()); :}
                    |
                    column_ref:col_ref
                    {: RESULT = new String(col_ref.toString()); :}
                    |
                    function_ref:fun_ref
                    {: RESULT = new String(fun_ref.toString()); :}
                    |
                    IPAREN escalar_exp:esc_exp DPAREN
                    {: RESULT = new String("("+esc_exp.toString()+")"); :}
                    ;
atom    ::=         parameter_ref:param_ref
                    {: RESULT = new String(param_ref.toString()); :}
                    |
                    INTNUM:numero
                    {: RESULT = new String(numero.toString()); :}
                    |
                    AMPERSAND NAME:nombre
                    {: RESULT = new String("&"+nombre.toString()); :}
```

```
                                    |
                                    COMILLA NAME:nombre COMILLA
                                    {: RESULT = new String("'"+nombre.toString()+"'"); :}
                                    |
                                    COMILLA HORA:ora COMILLA
                                    {: RESULT = new String("'"+ora.toString()+"'"); :}
                                    |
                                    COMILLA FECHA:fech COMILLA
                                    {: RESULT = new String("'"+fech.toString()+"'"); :}
                                    ;

parameter_ref    ::=                parameter:param
                                    {: RESULT = new String(param.toString()); :}
                                    |
                                    parameter:param1 parameter:param2
                                    {: RESULT = new String(param1.toString()+" "+param2.toString()); :}
                                    |
                                    parameter:param1 INDICATOR parameter:param2
                                    {: RESULT = new String(param1.toString()+" INDICATOR
                                    "+param2.toString()); :}
                                    ;

function_ref ::=                    AMMSC:func IPAREN POR DPAREN
                                    {: RESULT = new String(func.toString()+"(*)"); :}
                                    |
                                    AMMSC:func IPAREN DISTINCT column_ref:col_ref DPAREN
                                    {: RESULT = new String(func.toString()+"(DISTINCT "+col_ref.toString()+"
                                    )"); :}
```

```
                    |
                    AMMSC:func IPAREN ALL escalar_exp:esc_exp DPAREN
                    {: RESULT = new String(func.toString()+"(ALL "+esc_exp.toString()+")"); :}
                    |
                    AMMSC:func IPAREN escalar_exp:esc_exp DPAREN
                    {: RESULT = new String(func.toString()+"("+esc_exp.toString()+")"); :}
                    ;

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

tipo_dato     ::=   VARCHAR IPAREN INTNUM:numero DPAREN
                    {: RESULT = new String("VARCHAR("+numero.toString()+")"); :}
                    |
                    NUMERIC
                    {: RESULT = new String("NUMERIC"); :}
```

```
|

NUMERIC IPAREN INTNUM:numero DPAREN

{: RESULT = new String("NUMERIC("+numero.toString()+")"); :}

|

NUMERIC IPAREN INTNUM:numero1 COMA INTNUM:numero2 DPAREN

{: RESULT = new
String("NUMERIC("+numero1.toString()+","+numero2.toString()+")"); :}

|

DECIMAL

{: RESULT = new String("DECIMAL"); :}

|

DECIMAL IPAREN INTNUM:numero DPAREN

{: RESULT = new String("DECIMAL("+numero.toString()+")"); :}

|

DECIMAL IPAREN INTNUM:numero1 COMA INTNUM:numero2 DPAREN

{: RESULT = new
String("DECIMAL("+numero1.toString()+","+numero2.toString()+")"); :}

|

INTEGER

{: RESULT = new String("INTEGER"); :}

|

SMALLINT

{: RESULT = new String("SMALLINT"); :}

|

FLOAT

{: RESULT = new String("FLOAT"); :}

|

FLOAT IPAREN INTNUM:numero DPAREN
```

```
                                {: RESULT = new String("FLOAT("+numero.toString()+")"); :}
                                |
                                REAL
                                {: RESULT = new String("REAL"); :}
                                |
                                DOUBLE PRECISION
                                {: RESULT = new String("DOUBLE PRECISION"); :}
                                |
                                DATE
                                {: RESULT = new String("DATE"); :}
                                |
                                TIME
                                {: RESULT = new String("TIME"); :}
                                ;
columna::=                      NAME:nombre
                                {: RESULT = new String(nombre.toString()); :}
                                ;
identificador       ::=         NAME:nombre identificador_copy:id
                                {: RESULT = new String(nombre.toString()+""+id.toString()); :}
                                ;
identificador_copy ::=          {: RESULT = new String(); :}
                                |
                                PUNTO NAME:nombre identificador_copy:id
                                {: RESULT = new String("."+nombre.toString()+""+id.toString()); :}
                                ;

Parameter ::=                   DOSPUNTO NAME:nombre
```

```
                                    {: RESULT = new String(":"+nombre.toString() ); :}

                                    ;

/* Fin Definicion de Atributos : Entidades - Inicio definicion de metodos  */


definicion_metodo ::=               METHOD NAME:nombre_metodo IPAREN

                                    {: parser.escribir_clase(parser.NombreClase,
                                    nombre_metodo.toString()+"(",'m');

                                    if (parser.TienePadre)

                                    parser.escribir_clase(parser.NombreClase,
                                    parser.parametros_padre(parser.ClasePadre),'m');

                                    :}

                                    paramet_lista:param_list DPAREN

                                    {: parser.escribir_clase(parser.NombreClase,
                                    param_list.toString()+")",'m'); :}

                                    cuerpo_metodo

                                    ;


paramet_lista     ::=               insert_paramet:ins_param

                                    {: RESULT = new String(ins_param.toString()); :}

                                    |

                                    paramet_lista:param_list COMA insert_paramet:ins_param

                                    {: RESULT = new String(param_list.toString()+", "+ins_param.toString()); :}

                                    ;

insert_paramet   ::=                {: RESULT = new String(); :}

                                    |

                                    NAME:nombre

                                    {: RESULT = new String(nombre.toString()); :}

                                    |
```

```
                                    NULLX
                                    {: RESULT = new String("NULL"); :}
                                    ;

cuerpo_metodo  ::=                  ILLAVE
                                    {: parser.escribir_clase(parser.NombreClase, " \n{\n",'m'); :}
                                    {: if (parser.TienePadre)

        parser.escribir_herencia(parser.ClasePadre, parser.NombreClase);

                                    :}
                                    definicion_funciones:def_func
                                    {: parser.escribir_clase(parser.NombreClase, def_func.toString(),'m'); :}

                                    DLLAVE
                                    {: parser.escribir_clase(parser.NombreClase, "}\n",'m'); :}
                                    ;
aux_definicion_funciones ::=        definicion_funciones:def_func
                                    {:RESULT = new String(def_func.toString()); :}
                                    ;
definicion_funciones ::=            {: RESULT = new String(); :}
                                    |
                                    delete_statement_searched:delete PUNTOYCOMA
                                    aux_definicion_funciones:aux_func
                                    {: RESULT = new String("+delete.toString()+";\n"+aux_func.toString()); :}
                                    |
                                    insert_statement:insert PUNTOYCOMA
                                    aux_definicion_funciones:aux_func
                                    {: RESULT = new String("  "+insert.toString()+";\n"+aux_func.toString()); :}
```

```
                                    |

                                    update_statement_searched:update PUNTOYCOMA
                                    aux_definicion_funciones:aux_func

                                    {: RESULT = new String("  "+update.toString()+";\n"+aux_func.toString());
                                    :}

                                    ;


delete_statement_searched ::=       DELETE FROM table:nombre opt_where_clause:where

                                    {: RESULT = new String("DELETE FROM "+nombre.toString()+"
                                    "+where.toString()); :}

                                    ;

opt_where_clause ::=                {: RESULT = new String(); :}

                                    |

                                    where_clause:where

                                    {: RESULT = new String(where.toString()); :}

                                    ;

insert_statement::=                 INSERT INTO table:nombre opt_column_commalist:opc_col
                        values_or_query_spec:values

                                    {: RESULT = new String("INSERT INTO "+nombre.toString()+"
                                    "+opc_col.toString()+" "+values.toString()); :}

                                    ;

opt_column_commalist ::=            {: RESULT = new String(); :}

                                    |

                                    IPAREN listado_columna:list_col DPAREN

                                    {: RESULT = new String("("+list_col.toString()+")"); :}

                                    ;

values_or_query_spec ::=            VALUES IPAREN insert_atom_commalist:ins_atm DPAREN

                                    {: RESULT = new String("VALUES ("+ins_atm.toString()+")"); :}

                                    |
```

```
                                query_spec:cons_spec
                                {: RESULT = new String(cons_spec.toString()); :}
                                ;
query_spec ::=                  SELECT opt_all_distinct:opc_all selection:sel table_exp:tab_exp
                                {: RESULT = new String("SELECT "+opc_all.toString()+" "+sel.toString()+"
                                "+tab_exp.toString()); :}
                                ;

insert_atom_commalist ::=       insert_atom:ins_atm
                                {: RESULT = new String(ins_atm.toString()); :}
                                |
                                insert_atom_commalist:ins_atm_list COMA insert_atom:ins_atm
                                {: RESULT = new String(ins_atm_list.toString()+", "+ins_atm.toString()); :}
                                ;
insert_atom ::=                 atom:atm
                                {: RESULT = new String(atm.toString()); :}
                                |
                                NULLX
                                {: RESULT = new String("NULL"); :}
                                ;
assignment_commalist ::=        assignment:asignacion
                                {: RESULT = new String(asignacion.toString()); :}
                                |
                                assignment_commalist:asign_com COMA assignment:asignacion
                                {: RESULT = new String(asign_com.toString()+", "+asignacion.toString()); :}
                                ;
assignment      ::=             columna:col COMPARACION:comp escalar_exp:esc_exp
```

```
                                {: RESULT = new String(col.toString()+" "+comp.toString()+"
                                "+esc_exp.toString()); :}

                                |

                                columna:col COMPARACION:comp NULLX

                                {: RESULT = new String(col.toString()+" "+comp.toString()+" NULL"); :}

                                |

                                columna:col COMPARACION:comp subquery:subcons
                                escalar_exp:esc_exp

                                {: RESULT = new String(col.toString()+" "+comp.toString()+"
                                "+subcons.toString()+" "+esc_exp.toString()); :}

                                ;


update_statement_searched ::=   UPDATE table:nombre SET assignment_commalist:asignacion
                                opt_where_clause:opc_where

                                {: RESULT = new String("UPDATE "+nombre.toString()+" SET
                                "+asignacion.toString()+" "+opc_where.toString()); :}

                                                                ;

/* Finaliza Definicion Metodos  -- OQL Definicion de Consulta de Entidades  */


consulta_entidades ::=          select_statement

                                {: Conexion.select = true;  :}

                                |

                                delete_statement_searched

                                |

                                insert_statement
                                |

                                update_statement_searched

                                |

                                drop_statement
```

;

select_statement::= SELECT opt_all_distinct selection table_exp

;

drop_statement ::= DROP TABLE paramet_lista

;

4.2.3 Prueba de los Analizadores y Creación del Sistema

Para obtener un resultado tanto de los analizadores como del sistema es necesario crearlo como lo indica la figura, así de manera automática, LATIN examina el contenido mostrando un mensaje exitoso para el análisis léxico, sintáctico y para la creación del sistema, en caso contrario un mensaje de error.

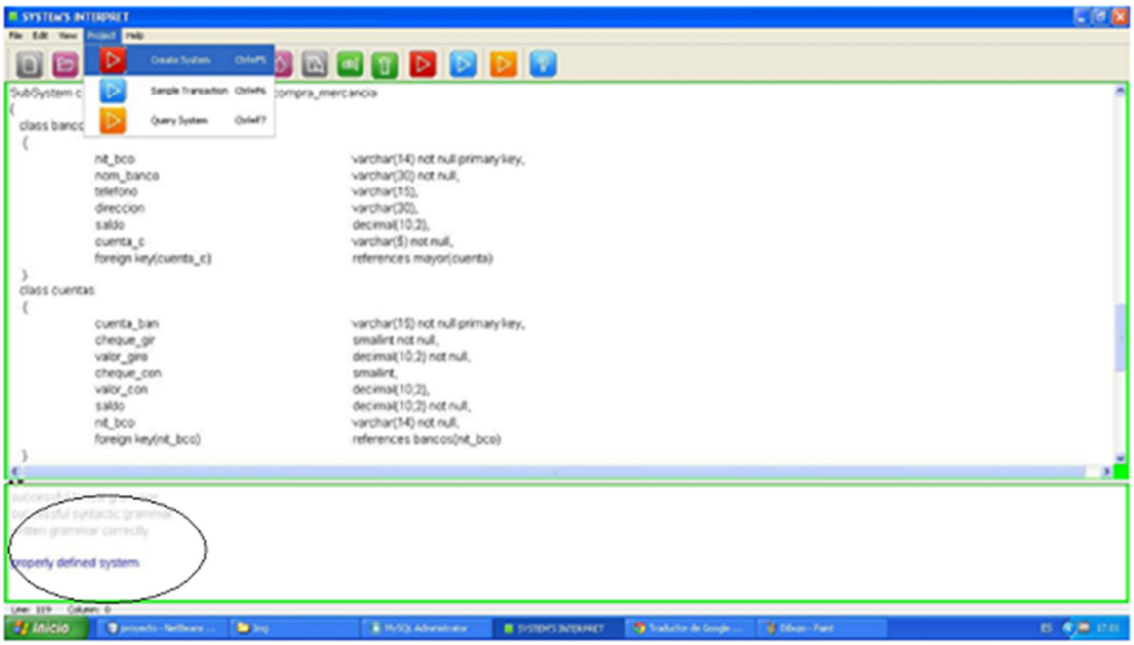

Figura 13: Prueba de los Analizadores

4.3 Transacción Ejemplo

4.3.1 Compra al Contado

Después de haber creado el sistema se obtiene esta ventana que abre el ejemplo de la compra de contado, cuyo formato es una factura tradicional.

En este tipo de compra la transacción se diligencia ingresando los datos y efectuando el registro de la compra, que esta a su vez se guarda automáticamente a la espera del botón de ejecución de consulta.

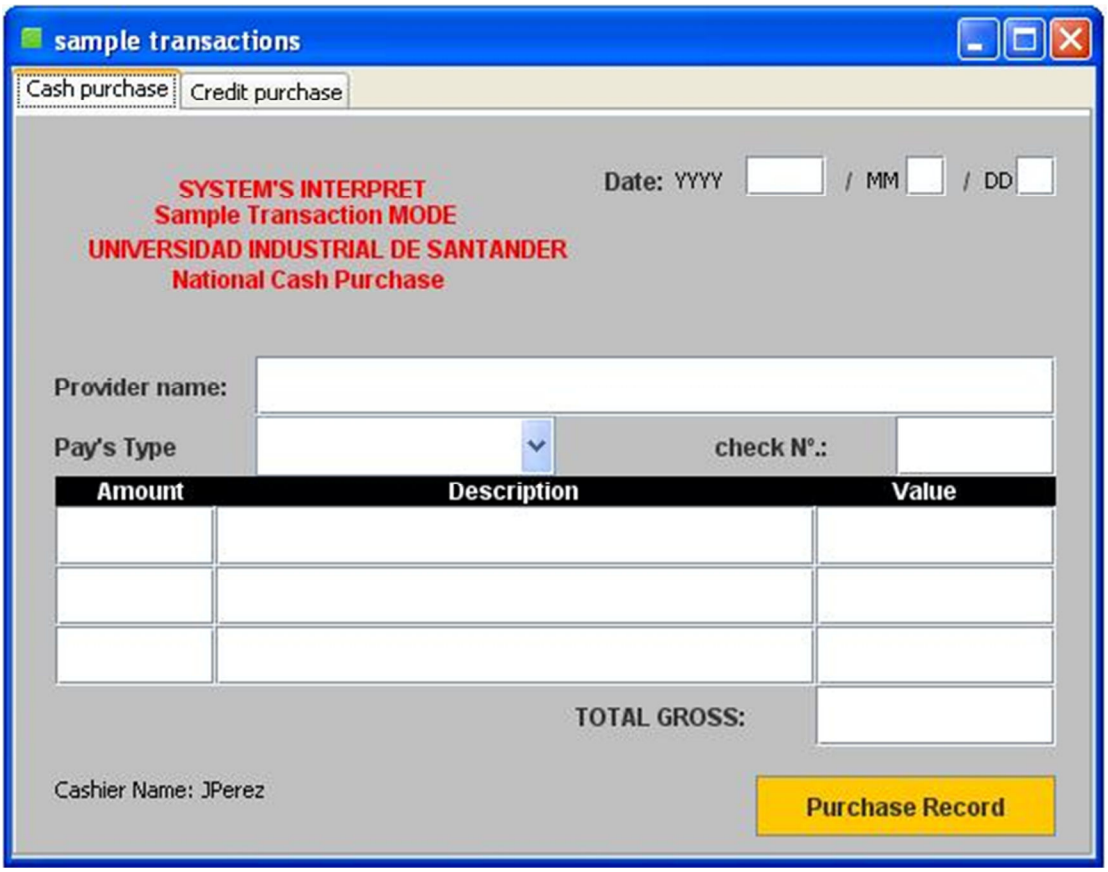

Figura 14: Factura compra al contado

4.3.1.1 Ejecución del método

Una vez ingresados los datos en la factura de compra al contado, es necesario hacer la ejecución del método que garantiza la realización de la debida transacción y el suministro

de la información a la base de datos del intérprete, esto se realiza al dar click en el botón “Purchase Record”.

A continuación se muestra el proceso y respuesta al ingresar el método al sistema.

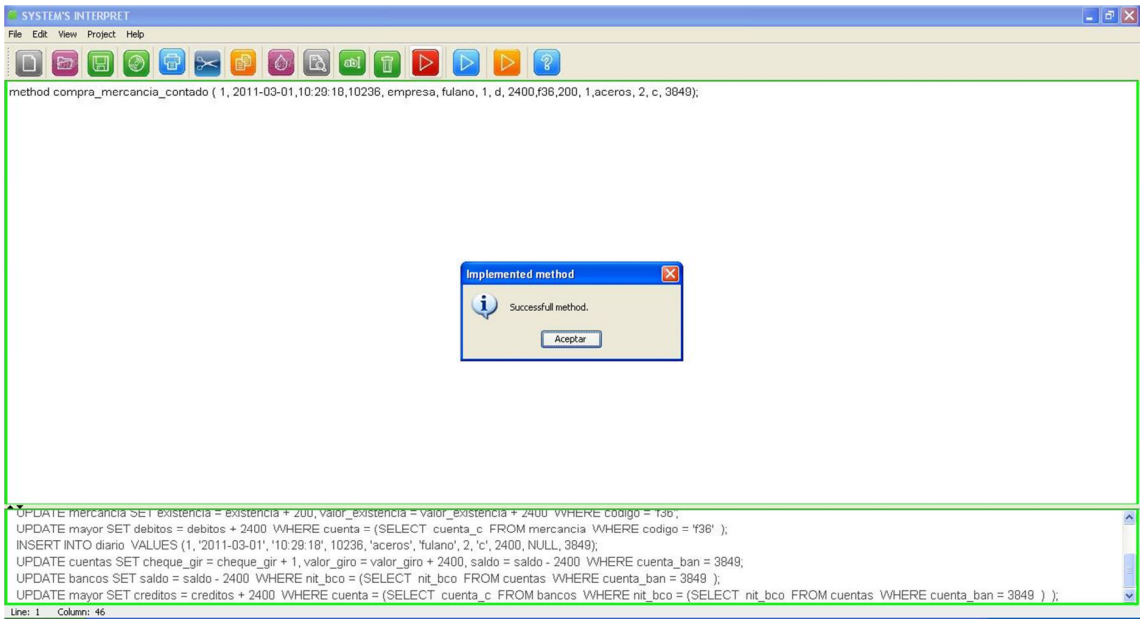

Figura 15: Implementación del Método

4.3.2 Compra a Plazos

Esta compra varía con respecto a la anterior y su diferencia radica en que no se paga inmediatamente como su nombre lo indica, si no que es a plazos acordados con el proveedor, por esta razón se afecta la cuenta de proveedores y no la de bancos.

Como se observó en la factura de la compra al contado, hay que diligenciar este formato con los datos del ejemplo explicado anteriormente y al finalizar registrar la compra en el botón purchase record.

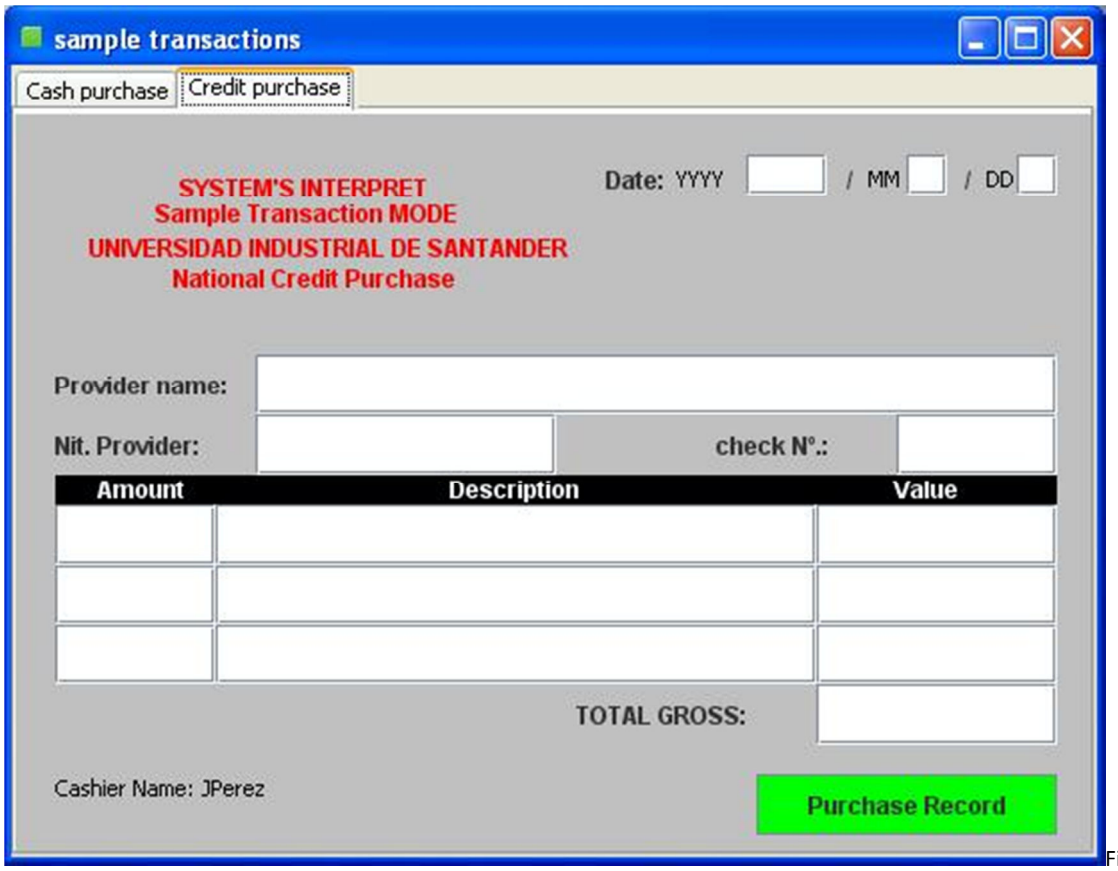

Figura 16: Factura compra a plazos

4.3.2.1 Ejecución del Método

En la ejecución del método para la compra a plazos se realiza igual que en la compra al contado (ver Figura 15), la única diferencia como se observa en la factura es que hay nuevos factores como los datos del proveedor que están involucrados.

4.3.3 Consulta Final

Una vez efectuada una compra sea al contado o a plazos, genera una información ubicada en la base de datos interna del intérprete, el usuario puede acceder a esta búsqueda consultando el sistema y este inmediatamente arroja dicho contenido.

A continuación se muestra el resultado de la consulta al contado y a plazos respectivamente.

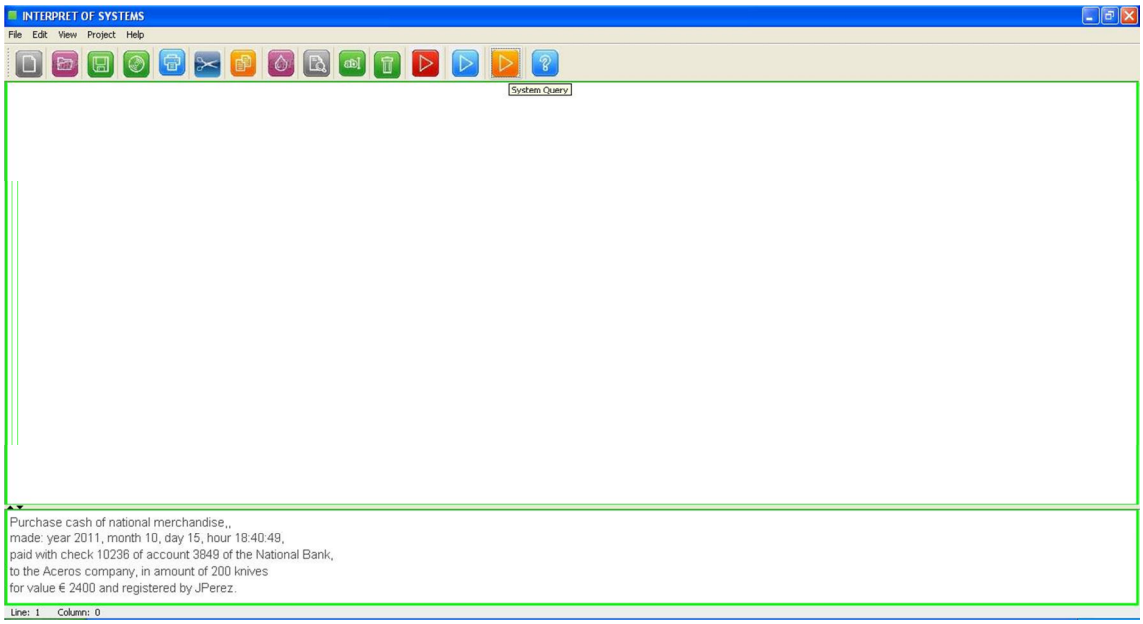

Figura 17: Consulta Final Compra al contado

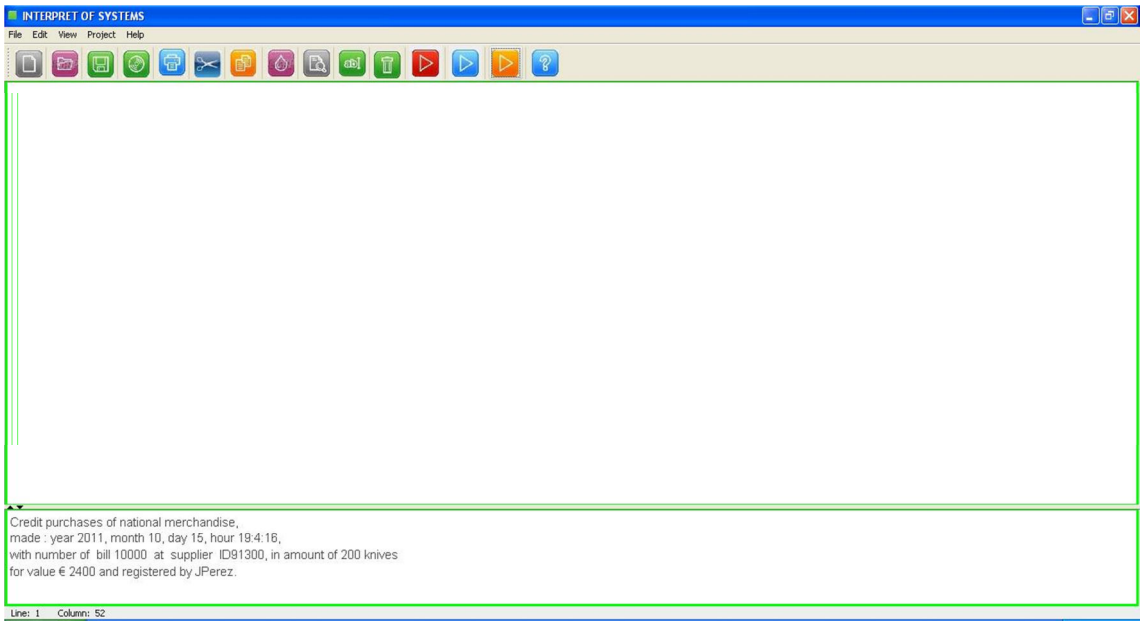

Figura 18: Consulta final compra a plazos

4.4 Resultado de las Pruebas

Para la implementación de las pruebas del intérprete en los sistemas operativos a explicar, se tuvieron en cuenta que para ambos corriera satisfactoriamente el contenido del software, además que cargaran adecuadamente las imágenes y no se presentaran complicaciones a la hora de probar en cualquier computador.

4.4.1 Pruebas para Windows XP

A continuación se presenta la imagen (ver barra de tareas) que muestra a LATIN trabajando adecuadamente en Windows XP.

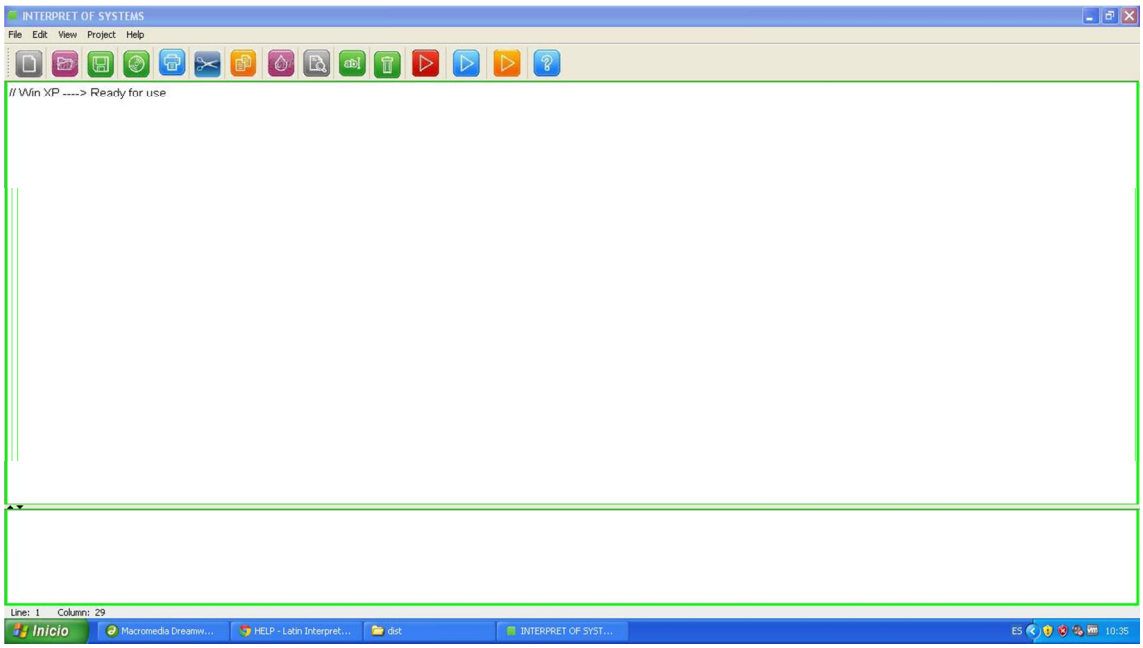

Figura 19: Prueba para Windows XP

4.4.2 Pruebas para Windows 7

De igual forma está la imagen de prueba para Windows 7 que LATIN corre satisfactoriamente.

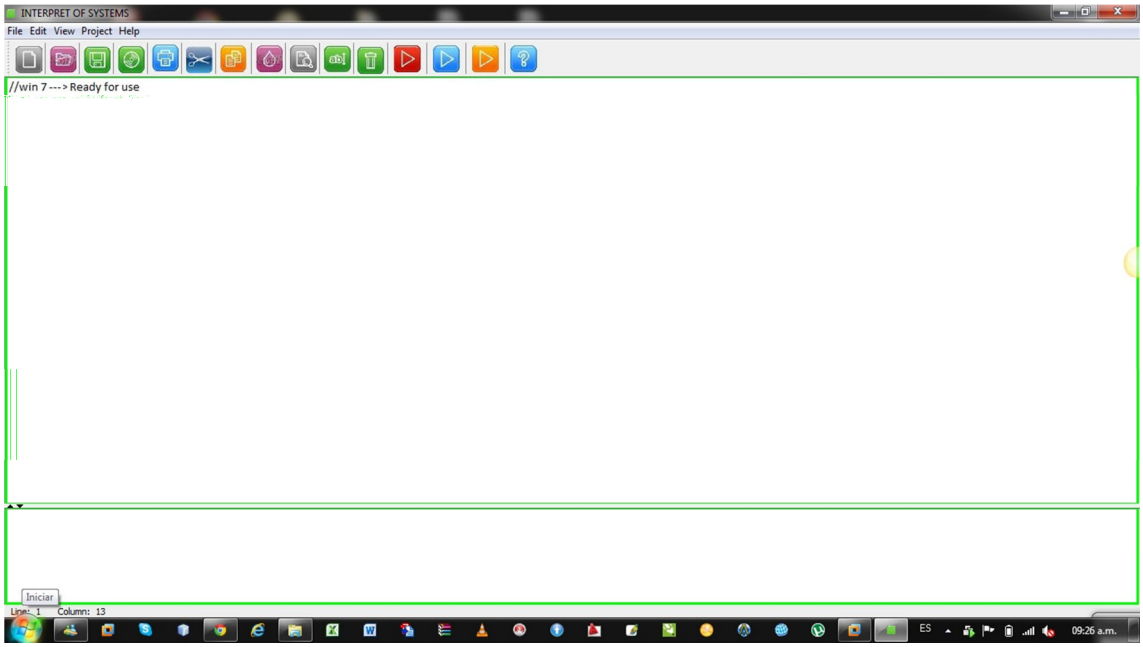

Figura 20: Prueba para Windows 7

5. MANUAL DE USUARIO

5.1 Instrucciones de uso de los elementos de la interfaz

A continuación se presentan los botones de izquiera a derecha tal y como están en la interfaz de LATIN (ver figura 12) y su debida funcionalidad dentro del intérprete:

 **New:** Este botón se encarga de la creación de un nuevo sistema. Al dar click se abre una pantalla en blanco como se mostró en la figura 12.

 **Open:** Es el botón de acceso a un sistema guardado dentro del intérprete, al dar click en el abre automáticamente un cuadro donde se puede buscar el archivo correspondiente.

 **Save:** Es el encargado de guardar los avances de un proyecto abierto, teniendo en cuenta que antes ha sido almacenado en una ubicación previa.

 **Save as:** Este botón guarda lo que se este digitando y lo ubica en una carpeta elegida por el usuario dentro del intérprete.

 **Print:** Dá la orden de imprimir el archivo que se encuentre abierto. Al dar click, abre unas opciones de ajuste (cuantas copias, impresión de todo el docuemento, por páginas) que el usuario puede modificar.

 **Cut:** Es el encargado de cortar lo que el usuario seleccione. Este desaparece la información de su ubicación inicial.

 **Copy:** Este botón copia la información seleccionada. A diferencia del anterior este solo lo copia pero deja el contenido intacto.

 **Paste:** Después de haber copiado o cortado el texto, al dar click en el este pega el contenido en el espacio establecido por el usuario.

 **Search:** Se encarga de la búsqueda de palabras, signos o de información dentro del sistema.

 **Fontsize:** Este botón es el encargado de modficar el tamaño de la letra del texto.

 **Delete:** Borra toda la información seleccionada por el usuario.

 **Create System:** Al dar click en este botón se crea el sistema (ver figura 13).

 **Sample Transaction:** Al oprimir este segundo botón de ejecución este abre automáticamente los formularios de la compra al contado y a plazos.

 **Query Of System:** Este botón muestra la consulta final del sistema tal y como se muestra en la figura 17 y 18 respectivamente.

 **Help:** Abre un archivo en html donde presenta el intérprete, la interfaz, las características e información de los desarrolladores.

 **About:** Muestra una ventana que presenta los autores, director y codirector de LATIN.

6. CONCLUSIONES

- El intérprete LATIN para la implementación de modelos emergentes de datos, permitió materializar la idea desarrollada por el proafesor Jaime Octavio Albarracín en su tesis doctoral, cuyas especificaciones de implementación están siendo adelantadas por el profesor Emiro Muñoz en su trabajo de maestría, obteniendo un software sencillo y fácil de utilizar que sin duda alguna ilustra lo planeado en las investigaciones mencionadas.

- La idea innovadora de tratar la verdadera dimensión de la realidad tanto en la naturaleza como dentro de las organizaciones, abre la necesidad de un lenguaje como el LATIN el cual promete y es germen en la búsqueda de soluciones para la desintegración actual de las empresas.

- 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.

- Si bien es cierto que con los lenguajes de la actualidad (tanto los de programación como los de datos) es posible desarrollar software poderoso y de buena calidad, tal software no puede ir más allá de los objetos. La realidad, no obstante, no es de objetos sino de sistemas, es decir de objetos interrelacionados, pero no cualquier objeto. El lenguaje latín cuya primera versión ofrecemos con este proyecto, asume esta verdadera dimensión de la realidad: Los sistemas.

7. RECOMENDACIONES

- Se sugiere a los estudiantes que deseen continuar trabajando con este lenguaje, hacer una documentación de la bibliografía mencionada para que haya mayor entendimiento en el momento de avanzar.

- 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.

- Debido al avance de nuevas tecnologías es necesario consultar primero las versiones de la plataforma de desarrollo, para no presentar inconvenientes.

- 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.

8. REFERENCIAS BIBLIOGRÁFICAS

- **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 Emergente de Datos "MODE".

- **BOOCH*, Grady*, JACOBSON, Ivar, RUMBAUGH, James. El Lenguaje Unificado de Modelado**, Segunda Edición. Se encuentra información acerca de UML y la orientación de los diferentes diagramas utilizados en ingeniería del software.

- **CATTELL, R.G.G, BARRY, DOUGLAS, BERLER MARK, EASTMAN, JEFF, JORDAN, DAVID, RUSELL, CRAIG, SCHADOW OLAF, STANIENDA TORSTEN, VELEZ, FERNANDO. The Object Data Standard: ODMG 3.0. ISBN 1-55860-647-4.** Este libro habla de los diferentes modelos de datos.

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

- **DAWSON, Christian W, Universidad de Valencia España. El Proyecto de Fin de Carrera de Ingeniería Informática**. Primera Edición. Se encuentra información para el alumno que realiza su proyecto final de carrera sepa cómo prepararlo desde el momento en que empieza a pensar en ello, con la búsqueda del tema y el director, hasta su presentación y defensa, pasando por todas las fases de realización.

- **JEREZ MUÑOZ EMIRO. "INTERPRETE LATÍN PARA LA IMPLEMENTACIÓN DE UN MODELO DE DATOS REORIENTADO A SISTEMAS (MODRES). De la tesis a la síntesis", Tesis de Maestría. Universidad Industrial de Santander.** En esta tesis se especifican los detalles de implementación del intérprete LATÍN

- **JOHNSON, STEVEN. Sistemas Emergentes. Editorial Turner. ISBN FCE: 968-16-7074-4; 2003.** En este libro se encuentra toda la información sobre Sistemas Emergentes y Modelos de datos.

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

- **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://latecladeescape.com/basico/compiladores-interpretes-y-maquinas-virtuales.html.** En este sitio se encuentra información acerca de compiladores e interpretes, características y diferencias.