CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN DE COMPUTACIÓN

LIDA/REC Lenguaje Visual para Bases de Datos

Tesis que presenta el:

Ing. Adriana Hernández Montoya

para obtener el grado de

Maestra en Ciencias

en la especialidad de

Ingeniería Eléctrica

Opción: *Computación*

Director de la tesis

Dr. Sergio V. Chapa Vergara

México, D.F. ***Agosto de 2005***

Índice general

Agradecimientos.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo financiero que me otorgó, sin el cual me hubiera sido muy difícil cursar la maestría.

Agradezco muy especialmente a mi asesor, por su trato tan agradable, por todos los conocimientos que compartió conmigo y por su apoyo durante el desarrollo de mi proyecto de tesis. También a los revisores de mi tesis, el Dr. Jorge Buenabad Chávez. y al Dr. Manuel González Hernández, por el tiempo dedicado para enriquecer mi tesis.

A todos los profesores del CINVESTAV-IPN por transmitirme sus conocimientos, a Sofi por su amabilidad y atenciones, y a todo el personal administrativo.

Agradezco a mi hijo, por ser mi motivo de superación en esta vida y por toda la alegría que me ha dado desde que llegó a mi vida. A él dedico esta tesis, con todo mi amor.

Agradezco con todo el corazón a mis padres, por estar siempre conmigo, por su paciencia, apoyo y sobre todo por su amor durante toda mi vida. A mi hermana, hermano y cuñado por su apoyo y cariño. En especial quiero agradecerles a todos ellos por sus atenciones, cuidados y cariño que han dado a mi hijo a lo largo de su vida, y especialmente en el transcurso de mis estudios. Aprovecho para recordarles lo mucho que los quiero.

A mis sobrinos que llenan de alegría mi vida con sus ocurrencias y cariño; los quiero mucho y aunque ya no los tenga tan cerca de mí, siempre contarán conmigo.

Agradezco a Lalo, por ser como un ángel que siempre está pendiente de mí; por preocuparse de mis problemas y disfrutar con mis alegrías; por su apoyo y enseñanza durante la maestría, por animarme cuando estaba apunto de rendirme; pero sobre todo por su paciencia y cariño.

A Lore, por ser una gran amiga, por todos los momentos tan agradables que pasamos y que siempre tengo presentes; por su cariño, consejos y toda su ayuda.

A Isaí, por su amistad, por tenerme tanta confianza y por todas las veces que me hizo reír. También agradezco a Tere, Anahí, Paco, Alfredo, Luis, Lalo P., Enrique y Jaime; de cada uno tengo un detalle que me hará recordarlos por siempre.

Mi último agradecimiento, es el más importante y es para Diosito, por que sin él cuidándome y guiando mis pasos, no hubiera logrado alcanzar mis metas. Por todo lo que me ha dado a lo largo de mi vida y porque día a día me demuestra su amor.

RESUMEN

Las Bases de Datos (BD) son de gran importancia para cualquier empresa. El manejo adecuado y eficiente de eśtas se ha vuelto una necesidad primordial. Una de las actividades del proceso de gestión de las BD, es la recuperación de la información por medio de consultas; las cuales deben ser bien formuladas para obtener los resultados esperados. Por lo que es de gran ayuda contar con lenguajes que nos faciliten estas actividades. El Lenguaje Iconográfico para el Desarrollo de Aplicaciones (LIDA) es un lenguaje visual basado en flujogramas con el cual se pueden realizar consultas a una BD sin contar con demasiados conocimientos en el área. La creación de las sentencias SQL son sustituidas por un diagrama de flujo asociado a la consulta.

Actualmente existen dos implementaciones: LIDA y LIDAWEB. La primera cuenta con una característica muy importante que es generar un código intermedio y la segunda permite el acceso remoto a la BD. Actualmente no se cuenta con un sistema que contenga ambas características. Lo que motivó a crear una nueva versión.

En este proyecto se adicionó la propiedad de generar código en lenguaje REC (Regular Expression Compiler); esto modificó la forma de ejecución para obtener un mejor rendimiento y poder desarrollar cualquier tipo de aplicación. El código intermedio REC, se utiliza para ejecución en “mildware”; esto es, que la ejecución de las consultas se pueda delegar al servidor y dejar para el cliente únicamente la ejecución de la interfaz visual. El código intermedio se puede ejecutar en paralelo, si se cuenta con los medios tecnológicos necesarios. Esto se hace para optimizar la ejecución.

Con la implantación de LIDA/REC se obtuvo la ventaja de contar con una interfaz visual para el programador y con la generación de código, es posible desarrollar aplicaciones más complejas, dependiendo de los operadores o funciones que sean configuradas en el compilador de REC .

INTRODUCCION

Las bases de datos (BD) surgieron para mejorar la calidad de las prestaciones de los sistemas informáticos y aumentar su rendimiento. Una BD es un conjunto de datos interrelacionados y estructurados de acuerdo con un modelo y un modelo de datos es un conjunto de conceptos que permiten describir la estructura de una BD [3].

Por otro lado, los lenguajes visuales nacen a principio de la década de los 90's, por el natural desarrollo de la tecnología gráfica en computación. Estos usan una notación principalmente gráfica para codificar un algoritmo, a diferencia de las otras propuestas de programación, en donde, los programas son descritos por una sucesión de líneas de código.

Una clase importante de lenguajes visuales son los de consultas a BD. Estos fueron creados con el objetivo de proporcionar al usuario un ambiente gráfico para el fácil manejo de la BD utilizando formas, diagramas e iconos. En específico una herramienta de búsqueda mediante flujogramas reduce los problemas en el planteamiento de consultas, permitiendo al usuario construirlas sin tener demasiado conocimiento de SQL(Structured Query Language) .

Una implementación de un esquema para consultas basado en flujogramas es LIDA (Lenguaje Iconográfico para el Desarrollo de Aplicaciones) realizado por El Dr. Sergio V. Chapa Vergara en el año de 1991. El objetivo de LIDA es el de tener un nivel de representación visual con íconos ; y la generación automática de código para la solución de los problemas planteados. La segunda implementación fue el proyecto de LIDAWEB, el cual fue desarrollado en JAVA, con el objetivo de tener nuevas ventajas para interoperabilidad en conexión con base de datos remotas e interfaz JDBC; sobre la base CDBB-500 del proyecto Micro-500 (base de datos de microbiología).

La necesidad, como usuario final, de contar con una herramienta visual con la cual se puedan realizar consultas a una BD de una manera sencilla, desarrollar aplicaciones en donde se incluyan procesos y generar un código intermedio con el cual se facilite el uso del paralelismo, de acuerdo al enfoque de lenguaje de flujo de datos de LIDA. El Objetivo de este trabajo es la implementación de una nueva versión de LIDA, basada en un código intermedio denominado REC (Regular Expression Compiler).

Descripción del Sistema

El objetivo de este sistema es otorgar al usuario mediante una interfaz gráfica, las herramientas necesarias para construir consultas a una BD de forma sencilla y con estas desarrollar aplicaciones. Que dichas aplicaciones sean ejecutadas en midlware para optimizar el tiempo de ejecución.

El sistema fue desarrollado una parte con el lenguaje Java y la otra con lenguaje C. Se utilizó la herramienta REC, que es un compilador de expresiones regulares; el cual fue utilizado para la generación del código intermedio. Fue probado con la BD CDBB-500 perteneciente al proyecto MICRO-500; este sistema consite de una BD, que contiene información acerca de la colección de cultivos microbianos.

Organización de la tesis

El Capítulo 1 trata sobre los Lenguajes Visuales y algunos ejemplos de estos. En el capítulo 2, se describe LIDA como un lenguaje de consultas a la BD y para desarrollar aplicaciones. Asimismo, se describe una nueva versión de LIDA denomina LIDAWEB, la cual fue desarrollada en JAVA, con el objetivo de tener nuevas ventajas para interoperabilidad en conexión con BD remotas e interfaz JDBC y cuenta con las bondades de los sistemas visuales de búsqueda a BD, tales como un ambiente visual amigable y de fácil manejo. Se trata ampliamente el caso de estudio que es la BD CDBB-500 del proyecto Micro-500.

El capítulo 3, es para dar a conocer detalladamente el lenguaje REC, ya que el código intermedio generado por LIDA/REC es en REC. Primeramente se describe la sintaxis y semántica para la construcción del código en el lenguaje REC. Explicando detalladamente, como es que nuestro sistema genera un programa en REC y de que manera fue configurado el compilador de REC; en el cual se puede especificar los procesos para hacer las consultas a la BD.

En el capítulo 4, se describe el nuevo sistema, su arquitectura general y cada uno de sus dos módulos que lo componen; detallando como fueron implementados. En el capítulo 5, se muestra como debe operarse y algunos ejemplos sobre el caso de estudio. Para terminar, se dan a conocer las conclusiones a las que se llegó con la implementación del sistema y los trabajos futuros.

Alcances

Con la nueva versión ahora se genera código en lenguaje REC. Tomando en cuenta un trabajo en “mildware”, la ejecución se reparte entre cliente y servidor. Esto se puede lograr traduciendo el flujograma a un código de cadena intermedio; para ser posteriormente compilado por REC.

Capítulo 1

Lenguajes Visuales

Los lenguajes visuales ayudan a especificar el diseño de sistemas de software mediante modelos de representación. Existen productos comerciales que hacen más accesible el trabajo de diseño, ya que utilizan como base de aplicación los modelos de especificación propuestos por los lenguajes visuales. Un ejemplo de su campo de acción es la manipulación de bases de datos. En la primera sección del capítulo se menciona lo que es la comunicación visual, la importancia de las técnicas de construcción de iconos y lo que es un lenguaje icónico. Posteriormente, se describe lo que son los lenguajes visuales, es decir: su definición, su clasificación y su especificación. Finalmente se muestra, un panorama de los lenguajes visuales aplicados a la manipulación de las bases de datos.

1.1. Comunicación Visual.

El uso de imágenes ha resultado de gran beneficio y los lenguajes visuales, cada día ganan terreno como sistemas de software que ofrecen facilidades hombre-máquina. Por otro lado, se tienen lenguajes que generan gramáticas libres de contexto, que integran elementos visuales a sus conjuntos de símbolos terminales y conceptos de posición en el espacio [1], que permiten la elaboración de sistemas visuales con un grado de interacción más fuerte.

1.1.1. La simiótica y la dimensionalidad tecnológica

Uno de los principales problemas de los lenguajes visuales; es su alto contenido semántico. La significación puede ser presentada en un nivel primario y natural, secundario o convencional e intrínseca o contenido. El conocer el significado de los signos en su forma y significado se lleva al cabo con el uso de la simiótica. La simiótica es el estudio de la

ciencia de los signos.

Las imágenes serán mucho más interesantes y fáciles de memorizar si los signos que son entendidos por muchos son usados en ilustraciones grá ficas. Existen tres tipos diferentes de signos: el Icónico, el Indexado y el Simbólico. La diferencia entre estos tres signos es la velocidad para captar el mensaje que llevan dentro [4]:

- Icónico: Es una imagen que contiene información representada mediante gráficos. Por ejemplo, el dibujo de la silueta de un hombre o una mujer en la puerta de los baños.

- Indexado: Este tipo de signo lleva guardado dentro de sí un mensaje expresado indirectamente. Un ejemplo es una ventana por la cual sale una gran cantidad de humo.

- Simbólico: Los signos simbólicos son aquellos en los que se guarda información o mensajes en los que se expresa una idea clara pero en la que además se agrega algún elemento simbólico (o subliminal) para expresar alguna idea extra que refuerce lo que se busca manifestar. Por ejemplo, en el caso de la estatua de la libertad en Estados Unidos, el libro representa las leyes de su constitución y la antorcha representa la guía para conducir a la nación.

A continuación se presentan ocho propiedades ; las cuales pueden ser dimensionadas, para obtener una métrica cnológica, con base en el estudio resultante de la Simiótica.

1. Reconocimiento rápido y claro.

2. Memorización rápida y fácil.

3. Identificación con la imagen.

4. Identificación con el ambiente.

5. Identificación de elementos.

6. Facilidad para ignorar mensajes innecesarios.

7. Universalidad en la simbología usada

8. Concentración ante medios ruidosos.

1.1.2. Técnicas para el estudio y construcción de Iconos.

En la comunicación icónica puede describirse como el mensaje es transferido entre el diseñador y el usuario. Un icono puede verse primero por su forma perceptible (sintaxis), segundo por su relación entre su forma y qué significa (semántica) y tercero por su uso (pragmática) [5].

Si se habla de un lenguaje formal, éste está compuesto por tres componentes principales: (1) una sintaxis que define la notación en la que representa la especificación; (2) una semántica que ayuda a definir un universo de objetos que se usarán para describir el sistema y (3) un conjunto de relaciones que definen las reglas que indican qué objetos satisfacen adecuadamente la especificación [6].

Sintaxis

El dominio sintáctico de un lenguaje de especificación formal a menudo se basa en una sintaxis derivada de la notación estándar de la teoría de conjuntos y del cálculo de predicados. Aunque la sintaxis normalmente es simbólica, también se usan iconos (por ejemplo, símbolos gráficos como cuadros, flechas y círculos) si no resultan ambiguos [6].

A partir de iconos simples o atómicos se pueden construir iconos compuestos. Existen técnicas diferentes para formar iconos compuestos como son:

- Combinación.
  - Superposición: Un icono se pone en lo alto de otro.
  - Conjunción: Es la unión de dos símbolos gráficos, estos se ponen juntos para expresar un significado más complejo.
  - Concatenación: Es la unión de dos o más símbolos, que pueden ser el mismo símbolo o diferente.
  - Yuxtaposición. Son dos símbolos gráficos básicos que se ponen en una relación espacial de conjunto el uno al otro para formar un nuevo símbolo.
- Transformación. Una transformación toma un icono inicial y lo transforma, por ejemplo, el icono se puede estirar, se puede cambiar de color, etcétera.
- Derivación.Una derivación es cuando un icono complejo se compone de dos o más iconos elementales y uno de estos no tiene ningún uso fuera del compuesto. Por ejemplo el símbolo de no fumar.
- Herencia. Es donde una forma básica se usa y uno de sus elementos gráficos como el espesor de línea, dirección o color se modifica para expresar un nuevo elemento.

- Duplicación. Consiste en duplicar el icono. Por ejemplo, una imagen de un disco flexible significa guardar, una imagen con varios discos flexibles significa guardar todos los archivos activos.

La semántica de iconos. Esta puede estar basada sobre su dominio del mundo real, por lo que si hablamos de iconos de computadora, implica que debe estar basado sobre ese dominio. Algunos lenguajes permiten especificar el comportamiento de un sistema, el desarrollo de la sintaxis y la semántica es la especificación de sus estados y transiciones [6].

1.1.3. Lenguaje Visual Icónico.

La comunicación visual icónica es una tendencia creciente dentro de los sistemas computacionales. Para lograr este tipo de comunicación es necesario la definición de un lenguaje visual o un lenguaje visual icónico que otorgue los elementos visuales (iconos) y las reglas necesarias para la construcción de sentencias, no se puede olvidar que un icono puede ser utilizado al nivel de palabra y/o al nivel de sentencia [7].

Un lenguaje visual es aquel en el cual se construye un programa utlizando elementos textuales y gráficos para crear expresiones más claras.

El lenguaje icónico ideal es aquel que no requiere aprenderse y su uso sea intuitivo, que sea más realista al anticipar alguna explicación al usuario de ser necesario, minimizando esta necesidad [8]. Un sistema computacional amigable motivará al usuario a través de su uso a interrogar los iconos, para aprender (o confirmar) su aprendizaje; esta explicación viene en el nivel más bajo de la forma icónica. En la interfaz de usuario se pueden presentar los iconos con una explicación breve de su significado o función a fin de minimizar el posible error al ser utilizado.

1.2. Lenguajes y Programación Visuales.

Los lenguajes de texto, como por ejemplo C, COBOL, se han visto rebasados en cuanto a las expectativas de utilización [4]. El hecho son los nuevos servicios, los cuales deben permitir al usuario (desarrollador), escribir el menor número de líneas de código, y que agilice el tiempo de implantación.

Uno de los tópicos que se manejan actualmente es la programación visual, donde se entiende como programación visual el uso de expresiones visuales tales como dibujos, gráficas e íconos dentro del proceso de programación, estableciendo formas de comunicación entre éstas, para conducir al usuario dentro de un marco de trabajo más identificado con su manera de pensar y relacionar, reduciendo el tiempo y esfuerzo dedicado a la programación tradicional.

1.2.1. Enfoques de la Programación Visual.

Existen dos perspectivas de ver la programación visual que son :

- Ambientes de Programación Visual (VPE, por sus siglas en inglés)
- Lenguajes de Programación Visual (VPL, por sus siglas en inglés) [9]

Lo que hace la diferencia entre estas dos perspectivas es que los VPE sólo proporcionan al usuario medios visuales o gráficos para la creación de sistemas, pero la gramática con la cual trabajan es textual. Los VPL en cambio, combinan un ambiente de desarrollo basado en símbolos visuales, incorporando la posición espacial de los objetos. Cuentan con una gramática visual en la que por lo menos uno de sus símbolos terminales es un objeto gráfico.

El uso de símbolos gráficos como parte del proceso de comunicación visual tiene dos representaciones, la parte lógica y la parte f ísica; partes que deben ser interpretadas para dar forma al lenguaje de aplicación. La parte lógica se refiere a las funciones, relaciones, restricciones y comunicaciones ejecutadas por el símbolo y la parte física al diseño de la imagen. Existen también criterios de evaluación de los lenguajes de programación visual, siendo estos: su naturaleza visual, funcionalidad, facilidad de comprensión, paradigma soportado y escalabilidad [10].

Los lenguajes visuales se han convertido en elementos de aplicación popular, debido a la facilidad de uso e implementación, todo favorecido por los avances logrados en el hardware. La forma amigable en la que se pueden elaborar las interfaces al usuario, genera una alta aceptación entre los usuarios y los propios desarrolladores de sistemas de software.

1.2.2. Definición de un lenguaje visual.

Un lenguaje visual es un conjunto de diagramas u objetos gráficos con los que se pueden componer sentencias válidas en este lenguaje, donde un diagrama es una colección de símbolos en un espacio de dos o tres dimensiones [10] y cuya característica principal es que al menos en el conjunto de símbolos terminales se define un objeto gráfico [9].

Un lenguaje visual es una estructura formal que consta de:

- Un conjunto de elementos gráficos, en este caso primitivas como: botones, texto, zonas de dibujo, imágenes, etc. Estas son construidas en una interfaz visual.
- Un conjunto de reglas que nos guían en la combinación del conjunto anterior.
- Una métrica para evaluar las construcciones dentro del lenguaje visual.

Los lenguajes visuales se han extendido desde medios de comunicación en general hasta sistemas de definición de tareas o sistemas mismos.

El objetivo de los lenguajes visuales, es mejorar la comunicación entre el usuario y la computadora mediante el uso de los recursos computacionales y un análisis bien definido del dominio de aplicación del usuario, y la incorporación de diferentes áreas del conocimiento.

1.2.3. Clasificación de los lenguajes visuales.

El lenguaje natural vs. formal, ha sido el medio que ha permitido una vía de comunicación entre entidades similares o diferentes. El lenguaje natural es la facultad que tienen los humanos para comunicarse, entenderse y tomar decisiones.

En el contexto de interacción hombre-máquina y dentro de teorías formales; para el establecimiento de un lenguaje formal, lo primordial es : un alfabeto y reglas de construcción sintácticas.

En teoría de computación, la jerarquía de Chomsky, ha sido la principal forma de clasificación de los lenguajes de computación: regulares, libres de contexto, sensibles al contexto, sin restricciones. Cada lenguaje, es definido por un conjunto de reglas que determina su gramática.

Una de las ventajas de clasificar a los lenguajes en función de su gramática, es que estas clasificaciones proporcionan detalles con respecto a las estructuras de cadenas que pueden aparecer en dichos lenguajes.

Una gramática es un alfabeto, es decir, una colección de símbolos no terminales, terminales, un símbolo inicial y un conjunto finito de reglas de reescritura.

Esta definición podría ser también la definición de una gramática estructurada por frases, ya que esta basada en la composición de cadenas en términos de $frases$, donde cada frase está representada por un símbolo no terminal.

Para clasificar los lenguajes visuales se deben considerar propiedades específicas que permitan colocarlos a manera de clases, considerando dos puntos sobresalientes en la clasificación: la expresividad y la dificultad que implica el procesamiento de la información. Otro punto que debe ser tomado en cuenta para determinar la clasificación de los lenguajes de programación visual son sus propiedades formales.

Margaret M. Bunett y Marla J. Baker [**?**] presentan un sistema de clasificación para los lenguajes de programación visual considerando las principales áreas de contribución que permitan localizar información precisa del tema. Enseguida se muestra la clasificación.

Sistema de Clasificación de los Lenguajes de Programación Visual (VPL :

1. Paradigmas
   a. Lenguajes concurrentes.
   b. Lenguajes basados en restricciones.
   c. Lenguajes de flujo de datos.
   d. Lenguajes basados en formas y hojas de cálculo.
   e. Lenguajes funcionales.
   f. Lenguajes imperativos.
   g. Lenguajes lógicos.
   h. Lenguajes multi-paradigma.
   i. Lenguajes orientados a objetos.
   j. Lenguajes de programación por demostración.
   k. Lenguajes basados en reglas.
2. Representaciones Visuales.
   a. Lenguajes diagramáticos.
   b. Lenguajes icónicos.
   c. Lenguajes basados en secuencias de imágnes estáticas.

Características de lenguaje :

1. Abstracción.
   a. Abstracción de datos.
   b. Abstracción procedural.
2. Flujo de control.
3. Estructuras y tipos de datos.
4. Documentación.
5. Manejo de eventos.
6. Manejo de excepciones.

Publicación de implementación del lenguaje :

1. Proposiciones computacionales.
2. Eficiencia.
3. Parseo.
4. Traductores (Interpretes y Compiladores)

Propósito del lenguaje :

1. Lenguajes de propósito general.
2. Lenguajes de bases de datos.
3. Lenguajes de procesamiento de imágenes.
4. Lenguajes de visualización científica.
5. Lenguajes de generación de interfaces de usuario.

Teoría de los VPLs.

1. Definición formal de VPLs.
2. Teoría de iconos.
3. Emitir lenguajes de diseño.
    a. Proporcionar diseño cognitivo e interfaces de usuario.
    b. Uso efectivo del estado real de la pantalla.
    c. Viveza.
    d. Alcance.
    e. Tipo de revisión y tipo de teoría.
    f. Proporcionar una representación visual (representación estática o animación).

Ejemplos de Lenguajes Visuales

Enseguida se listan unos ejemplos de Lenguajes Visuales que sirven para consultas a base de datos mediante flujo de datos:

- Data Flow Query Language for Relational Databases (DFQL) [12]. Es un lenguaje para hacer consultas a una BD relacional. Sustituyendo las sentencias SQL por una representación visual. Más adelante se extiende más sobre este lenguaje.
- Graphical Data Flow Language for Retrieval, Analysis, and Visualization of Scientific Databases [13]. Este leguaje también, se basa en la representación de flujos de datos para analizar una BD.
- Lenguaje Iconográfico para el Desarrollo de Aplicaciones (LIDA) [14] y LIDAWEB [15]. Estos lenguajes son especificamente para hacer consultas a una BD mediante flujogramas. En estos lenguajes se basa este proyecto.

1.2.4. Especificación de los lenguajes visuales.

La especificación de un lenguaje visual está dada de la siguiente manera: primero debe delimitarse el dominio de la aplicación, identificarse los elementos que forman parte de este dominio, definir las relaciones válidas dentro de este dominio y finalmente determinar el comportamiento de las relaciones.

Un lenguaje visual de acuerdo a su definición puede ser aplicado a un cierto número de representaciones, en donde su significado está dado por la relación que guardan los elementos visuales, por ejemplo, las expresiones matemáticas, la notación musical, los diagramas de flujo, etc.

Actualmente existen tres alternativas principales para la especificación de lenguajes visuales: la gramática, la lógica y la algebraica.

Especificación Gramatical.

La alternativa gramatical está basada en formalismos utilizados en la especificación de lenguajes de cadenas. Una forma representativa es la gramática de grafos. Una ventaja de su formalismo es mejor entendida, debido a que se presenta mediante nodos y arcos etiquetados, una desventaja de este formalismo es que requiere de la fase de análisis inicial "léxica". , para reconocer las relaciones entre los símbolos terminales que son importantes y que deben ser conservados como arcos en el grafo inicial.

Especificación Lógica.

La alternativa lógica, utiliza lógica matemática de primer orden u otras formas de lógica matemática que frecuentemente proveen de inteligencia artificial. Está basada en alternativas que son usualmente lógicas espaciales que axiomatizan las diferentes relaciones posibles entre objetos. Una de las ventajas de esta opción es que el mismo formalismo puede ser usado para especificar la sintaxis y semántica de un diagrama.

Especificación Algebraica.

Una especificación algebraica consiste de una composición de funciones que construyen imágenes complejas desde elementos simples de la imagen. La idea principal de esta opción consiste en mapear el dominio a ser definido, en estructuras de tipo de datos abstractos y definir los tipos de funciones y operación de predicados en estas estructuras para especificar las operaciones en el dominio de aplicación.

1.3. Lenguajes Visuales para consulta de bases de datos.

Una de las aplicaciones en la que estan presentes los lenguajes visuales, es en la consulta a bases de datos, previo a describir dichas aplicaciones se describirá el lenguaje de consultas a bases de datos más usado en la actualidad, es decir SQL. Además se hablará de algunos lenguajes visuales, cuyo objetivo es proveer las herramientas para realizar consultas a un base de datos sin necesidad de ser un usuario experto en la materia.

1.3.1. Lenguaje Estructurado de Consultas (SQL).

El lenguaje de consulta estructurado (SQL) se ha establecido como el lenguaje estándar de bases de datos relacionales. Hay muchas versiones de SQL, pero la versión original se

desarrolló en el laboratorio de investigación de San José, de IBM (actualmente es el Centro de investigación de Aldemán, Almaden)[16]. Este lenguaje, originalmente denominado Sequel, se implementó como parte del proyecto System R, a principios de 1970. El lenguaje Sequel ha evolucionado desde entonces y su nombre ha cambiado a *Structured Query Language o Lenguaje Estructurado de Consultas.* Actualmente numerosos productos son compatibles con el lenguaje SQL.

El lenguaje comprende una versión basada en sus diversos componentes que fueron determinados como estándar por CADASYL y son los siguientes :

- **Lenguaje de definición de datos** (LDD). El LDD de SQL proporciona órdenes para la definición de esquemas de relación, borrado de relaciones, creación de índices y modificación de esquemas de relación.
- **Lenguaje interactivo de manipulación de datos** (LMD). El LMD de SQL incluye un lenguaje de consultas, basado tanto en álgebra relacional como en el cálculo relacional de tuplas. Incluye también ordenes para insertar, borrar y modificar tuplas de la base de datos.
- **LMD incorporad**o. La forma incorporada de SQL se diseño para el uso sin lenguajes de programación de próposito general, tales como PL/I, Cobol y C.
- **Definición de vistas**. El lenguaje de definición de datos de SQL incluye órdenes para la definición de vistas.
- **Autorización**. El LDD de SQL incluye órdenes para la especificación de los derechos de acceso a relaciones y vistas.
- **Integridad**. El LDD de SQL incluye órdenes para la especificación de las ligaduras de integridad que deben satisfacer los datos almacenados en la base de datos. Las actualizaciones que violen las ligaduras de integridad se rechazan.
- **Control de transacciones**. SQL incluye órdenes para la especificación del comienzo y final de transacciones. Varias implementaciones permiten también bloqueo explícito de los datos para el control de la concurrencia.

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

SQL se construye con bloques de tres sentencias según el ejemplo siguiente :

SELECT Empleados.Nombre FROM Empleados WHERE Empleados.Cargo = "Electricista"AND sueldo = 10 000

Considerando este ejemplo se puede suponer que la creación de consultas usando SQL es sencilla porque al analizar la sentencia anterior se puede observar que se están seleccionando los nombres de los empleados, donde el cargo que tienen es el de Electricista y ganan entre 10 mil y 15 mil pesos. Dichos nombres se van a obtener de una tabla denominada Empleados.

Desafortunadamente la administración de la información que contienen las bases de datos, por ejemplo la de los bancos, requiere de una gran cantidad de consultas para saber, por ejemplo, el total de depósitos y el monto promedio de los mismos en un día ó efectuar los cálculos de la nómina, por lo tanto la complejidad de las sentencias sql aumenta demasiado.

Actualmente en la literatura se encuentran propuestas de lenguajes visuales para la realización de consultas a bases de datos, en las siguientes secciones se van a describir dichas investigaciones y aplicaciones.

1.3.2. Consultas dinámicas para exploración de información.

La mayoría de los sistemas de BD necesitan del usuario para crear y formular consultas complejas, las cuales implican que el usuario esté familiarizado con la estructura lógica de la BD. Las consultas sobre una BD se expresan, generalmente, en lenguajes de consulta de alto nivel (tales como SQL, QUEL y Query-by-Example). Esto funciona bien para algunas aplicaciones, pero no es una forma completamente satisfactoria de encontrar datos. Para usuarios "ingenuos" (inexpertos) esos sistemas son difíciles de usar y entender, y necesitan un largo período de entrenamiento.

Claramente, existe una necesidad de métodos poderosos, rápidos y fáciles de usar para la recuperación de BD. La manipulación directa ha demostrado ser exitosa para otras aplicaciones tales como editores de desplegado, hojas de cálculo, sistemas de diseño asistido por computadora, sistemas de desarrollo asistido por computadora, juegos de computadora y ambientes gráficos para sistemas operativos, por ejemplo Apple Macintosh.

Las representaciones gráficas de los esquemas de bases de datos como son los diagramas entidad-vínculo[1], son comunmente usados para soportar el diseño de las bases de datos.

[1]El proyecto de tesis de Teresa Villegas Casas; tiene como objetivo un ambiente visual basado en el modelo entidad-vínculo-extendido, para el diseño de BD. "Diseño asistido por computadora EVE-Mac. "

Las representaciones gráficas pueden ser utilizadas para facilitar la creación o formulación de consultas.

Un esquema de base de datos puede ser definido por un conjunto de entidades y relaciones de tipo descriptor. Cada entidad contiene el nombre de la entidad y los nombres de los atributos. Cada relación contiene el nombre que describe a la relación y los nombres de los atributos. El modelo puede ser una mejora para adaptar la representación gráfica de la definición y construcción de consultas el cual incluye la posición de una entidad y una relación en forma de iconos en la pantalla, la descripción de los atributos seleccionados y la especificación de las condiciones. A continuación se describe brevemente unos lenguaje de consulta por flujo de datos.

1.4. Lenguajes de Flujos de Datos.

1.4.1. Data Flow Query Language, DFQL

Un sistema manejador de bases de datos relacionales (RDBMS) es un producto de software que las estructuras de datos estan acorde con el modelo relacional de datos y permite manipular datos usando algebra relacional. Hay dos lenguajes de consulta altamente utilizados para los sistemas manejadores de bases de datos. Estos son el Lenguaje Estructurado de Datos (SQL) y Consulta por Ejemplo (QBE).

Aunque estos lenguajes son poderosos, ambos tienen desventajas referentes a la facilidad de empleo, especialmente al expresar la cuantificación universal y al especificar consultas jerarquizadas complejas. En orden para eliminar esos problemas, Data Flow Query Language (DFQL) fue propuesto. DFQL ofrece una interfaz de usuario gráfica facil de utilizar en el modelo relacional basado en diagramas de flujo de datos, mientras mantiene la fortaleza de SQL y QBE.

1.4.2. LIDA/LIDAWEB

El Lenguaje Iconográfico para el Desarrollo de Aplicaciones fue creado en 1991 por el Dr. Sergio V. Chapa V. Este lenguaje, esta basado en la representación visual de flujo de datos de información, mediante la cual se hacen consultas a BD y desarrollan aplicaciones. LIDA forma parte de un conjunto de proyectos en el campo de lenguajes y ambientes visuales que incluyen desde el diseño de bases de datos hasta la presentación visual de la información. Tales proyectos se están realizando en la sección de computacion del Depto. de Ingenería Elécrica del CINVESTAV. En el siguiente capítulo se describe más detalladamente LIDA y las modificaciones a las cuales ha sido sometida.

1.5. Conclusión.

La clase de lenguajes visuales basado en el paradigma de flujo de datos, ha mostrado ciertas ventajas por su interfaz hombre-máquina y su utilidad para : análisis de datos y desempeño de consultas.

Entre las ventajas que justifican sus implementaciones se tienen las siguientes :

- Como lenguaje de flujo de datos, puede ser llevado fácilmente al dominio de aplicación con operadores de alto nivel.
- Es un enfoque natural dentro del contexto de "queries dinámicos". A través del diagrama y dentro de íconos es posible cambiar instancias algebraicas y parámetros.
- Es un paradigma adecuado para análisis de optimización de consultas.
- Es un modelo de lenguaje paralelo.

Capítulo 2

Naturaleza Computacional de LIDA

Este capítulo, tiene por objetivo describir los lenguajes visuales de flujo de datos se ñalados en el capítulo anterior.

2.1. Lenguaje Iconográfico para el desarrollo de aplicaciones (LIDA)

Este lenguaje, fue creado por el Dr. Sergio V. Chapa Vergara en 1991[14] y forma parte de un conjunto de proyectos de lenguajes y ambientes visuales que incluye desde el diseño de BD hasta la presentación visual de la información.

Este lenguaje, básicamente funciona mediante flujogramas. Estos flujogramas constan de una serie de íconos interconectados. Los íconos se encargan de procesar la información. Los resultados obtenidos del procesamiento son envíados a otros íconos. Por medio, de las conexiones entre íconos, es como se representa como fluye la información.

Las principales carcterísticas de LIDA son:

- Su ambiente de programación es visual
- Se basa en el modelo relacional de datos.
- Su enfoque es funcional de lenguaje de flujo de datos.
- Tiene capacidad de programación en paralelo.

2.1.1. El Flujograma

La sintaxis de LIDA, se basa en los íconos interconectados y la semántica de los íconos es operacional; esto es, que a cada ícono se le asocia un operador de algebra relacional,

un operador con funciones agregadas a los atributos del objeto de relación, elementos funcionales con expresiones aritméticas o simbólicas. Los operadores, también pueden tener expresiones booleanas para decidir rutas de flujo de información.

Un ícono tiene como entrada objetos de relación y como salida otro objeto de relación. En el caso de los íconos que contienen funciones; arrojarán valores. Los íconos están constituidos por una parte física (la imagen) y una parte lógica (el significado). Los íconos están clasificados de la siguiente manera:

- Iconos terminales .- reciben una línea de conexión.
- Iconos que representan operadores binarios .- reciben dos relaciones y arrojan una salida.
- Iconos Unarios.- reciben una relación y dan como resultado otra relación.
- Iconos que aceptan una relación y bajo una condición establecida (expresión booleana) dividen las trayectorias de flujo en una verdadera y otra falsa.
- Iconos funciones .- aceptan en su entrada una relación y arrojan un valor.

Se muestra un ejemplo de un flujograma en la figura 2.1 que representa el planteamiento un problema que corresponde a una consulta a la BD y se muestra enseguida.

Problema. Dadas las relaciones y los esquemas de relación de $PARTE$ (#P, PNOMB, COLOR, PESO, CIUDAD) y $ORDEN$ (#S, #P, CANTIDAD). Se quiere obtener el número de partes (#P) cuyo peso sea mayor de 18 kg. o bien, sean suministradas por S2.

Se puede apreciar en el flujograma que la relación que la relación $ORDEN$ y $PARTE$, fluyen simultáneamente para entrar a dos íconos de selección. Estos seleccionan las entradas que cumplen con las condiciones de peso y proveedor. Después viene la operación de unión, se proyecta el atributo P. El resultado es arrojado en la relación Q5.

2.1.2. El Código Intermedio

El código intermedio es un código fuente que es generado para fungir de intermediario entre interfaz de usuario y la etapa de ejecución. Esto es que una aplicación se encargará de generar el código para que más adelante este sea ejecutado por un compilador. En lenguaje intermedio podría ser ISBL, por ser fácil de escribir y entender.

Enseguida se tiene un ejemplo de una cadena ISBL. Esta cadena, es el resultado del flujograma de la figura **??**.

ZZZ2! = R1 * R4 R6 = ZZZ2! . R5

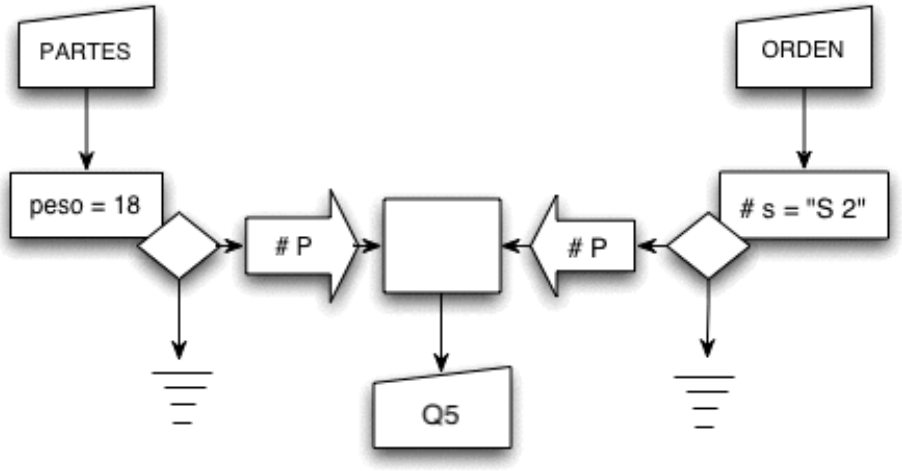


Figura 2.1: Ejemplo de flujograma

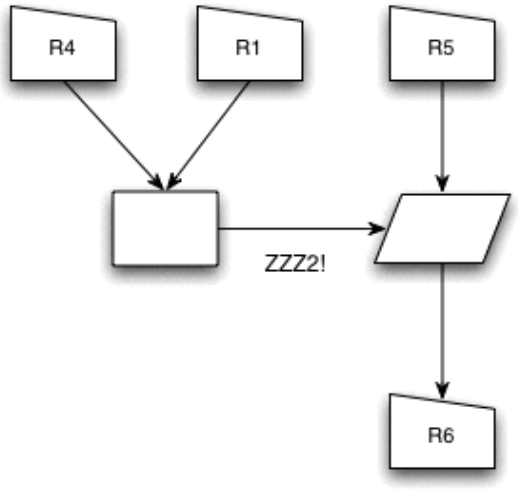


Figura 2.2: Flujograma en LIDA

2.1.3. Arquitectura General del Sistema LIDA

. Con el fin de tener un lenguaje con las características mencionadas, fue necesario integrar un sistema con la capacidad de definir y acceder a los esquemas de relación de

una base de datos con una estructura de relaciones.

De esta manera, el sistema se conformó con una máquina de base de datos con los siguientes módulos principales:

El módulo Descriptor de archivos funciona como base de metadatos. En él se encuentran definidos los esquemas de relación y son el mecanismo de acceso a la base de datos. El descriptor de archivos sirve de conexión entre el modulo conceptual y el modelo físico.

El módulo capturador sirve para construir la base de datos. El subsistema toma información del descriptor para definir el ambiente gráfico de pantallas y restricciones de integridad de los datos.

Por otro lado, tenemos que el sistema del lenguaje LIDA se constituye de un editor de gráficos de flujogramas el cual es traducido a una gráfica que verifica la sintaxis consultando el descriptor de archivos para la traducción de los objetos de relación. Posteriormente, se traduce a un código intermedio que es el ISBL el cual es un lenguaje algebraico relacional extendido. El lenguaje ISBL se compila y se ejecuta obteniendo las relaciones de la base de datos. El sistema tiene la capacidad de dejar código el cual se puede ejecutar en paralelo, y en el flujograma se permite una especificación en paralelo. Lo anterior se muestra en la figura 2.3

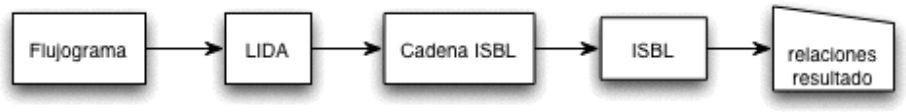


Figura 2.3: Etapas de ejecución de un flujograma

El compilador del lenguaje ISBL trabaja de la siguiente manera : comieza por leer el código ISBL, hace el análisis sintáctico, si no existieron errores se procede a la etapa de ejecución para producir los resultados. En la figura 2.4 se puede apreciar.

2.2. LIDAWEB : LIDA para un Ambiente de Interoperabilidad

Con base a lo anterior se planteo el proyecto de LIDAWEB [1] con el objetivo de tener nuevas ventajas para interoperabilidad en conexiones con bases de datos remotas e interfaz

[1]El proyecto LIDAWEB fue en parte apoyado por CONACYT a través del proyecto REDII. El resultado fue la tesis de licenciatura. “Interoperabilidad en Base de Datos a través del Sistema LIDAWEB. ” , elaborada por la Lic. Nancy Soto Godínez.

Figura 2.4: Diagrama de flujo para la interpretación del código ISBL.

ODBC o JDBC. En especial, como caso de estudio la base CDBB-500 del proyecto de Microbiología.

Por otra parte, se cuenta con una gran flexibilidad en la conexión a bases de datos tanto locales como remotas. Los cambios mínimos que se pueden hacer en el código, permiten realizar diferentes tipos de conexiones:

- Tipo 1. A bases de datos locales.
- Tipo 2. A bases de datos remotas.
- Tipo 3. A bases de datos remotas a través de *firewalls*

En el primer tipo de conexión la máquina en la que LIDAWEB corre es la misma que administra la base de datos.

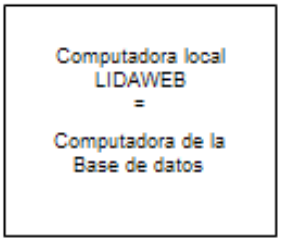


Figura 2.5: Corre en la misma máquina.

En el segundo caso, LIDAWEB reside en una computadora local y se conecta a una remota en donde se encuentra la base de datos.

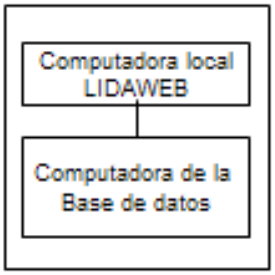


Figura 2.6: Se conecta una máquina remota.

Finalmente, el tercer tipo (utilizado en nuestro caso de estudio) es el que permite una conexión remota con un mayor grado de seguridad ya que la computadora intermedia es la encargada de verificar que el usuario que intenta conectarse a la base de datos es un usuario autorizado.

Con la posibilidad de las diversas conexiones de LIDAWEB, podemos hacer adaptaciones a distintos manejadores y por ende a diversas bases de datos. Para la implementación de LIDAWEB se realizaron los ajustes necesarios para manejar la base de datos CDBB-500 la cual fue implementada en el manejador PrimeBase en cuyo caso LIDAWEB accede a un servidor de aplicaciones que a su vez otorga autorización para accesar al servidor de bases de datos relacionales. En la figura 2.8 podemos apreciar más detalladamente como se hace la conexión a la BD.

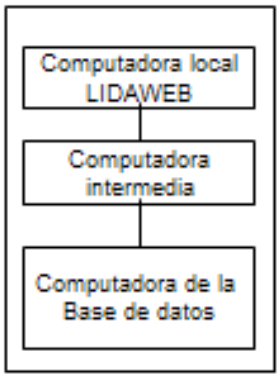


Figura 2.7: Se conecta con una máquina intermedia

2.2.1. Interoperabilidad con la base de datos.

JavaSoft desarrolló un API (Interfaz de Aplicación de Programa) para acceso a bases de datos llamada JDBC. Como parte de este proceso, se establecieron tres objetivos:

- JDBC es un API a nivel de SQL, es decir, JDBC permite construir expresiones SQL en el programa y los resultados son obtenidos en variables de Java.
- JDBC toma en cuenta la experiencia de los APIs para bases de datos existentes tales como el ODBC para así proveer rápido desarrollo de soluciones para manejadores de bases de datos que soporten los viejos protocolos.
- JDBC es simple.

Cuando se realiza un programa o aplicación en Java enfocada a bases de datos, la única información específica al driver que JDBC requiere es el URL de la base de datos, misma que puede ser obtenida en tiempo de ejecución. Usando el URL de la base de datos, el nombre de usuario y la contraseña, el programa o la aplicación realizan una petición a java.sql.Connection desde el DriveManager. Este busca a través de todas las implementaciones java.sql.Driver conocidas hasta encontrar la que conecta con el URL que se proporcionó. En caso de no encontrar la adecuada, este lanza una excepción. Una vez que el Driver reconoce la URL, crea una conexión con la base de datos utilizando el nombre de usuario y la contraseña especificados. El DriverManager pasa entonces al objeto Connection a la aplicación o programa.

El proceso de conexión es el que se muestra a continuación:

Connection con = DriverManager.getConnection(url, uid, password);

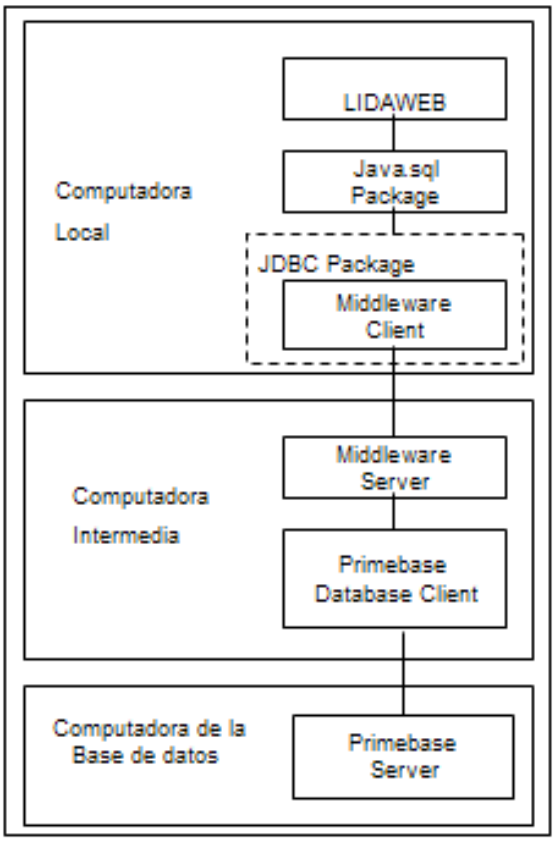


Figura 2.8: Conexión a la BD.

El DriverManager tiene una lista de clases que implementan la interfaz java.sql.Driver y hay varias maneras en las que la aplicación carga las implementaciones del Driver específico para el manejador de bases de datos que va a utilizar, una de ellas es cargando la clase dinámicamente usando

Class.forName("DriverImplementationClass");

Esta alternativa es muy conveniente ya que no requiere de código para los nombre de las clases ni de compilar cada vez que la base de datos o el driver cambian. Una vez que ha cargado el driver, este automáticamente se registra a si mismo usando el DriverManager.

Como se puede observar, para establecer una conexión con una base de datos, JDBC usa una clase:

ja*va.sql.DriverManager*. Su principal función es mantener una lista de implementaciones de drivers y establecer la correspondencia entre el Driver adecuado al URL solicitado.

Provee métodos para registrar o dar de baja drivers, así como dar una lista de los que estan disponibles.

Además hace uso de dos interfaces:

java.sql.Driver; responde a peticiones de conexión del DriverManager y provee información acerca de la implementación en cuestión.

java.sql.Connection: es usada para enviar sentencias SQL a la base de datos y para manejar ya sea el cometido o abandono de dichas expresiones.

2.3. Conclusión.

Después de casi 14 años que han aparecido los lenguajes visuales de una forma extensiva. Del artículo de revisión de lenguajes visuales de consulta para BD [12], podemos observar que LIDA es un lenguaje que ofrece varias ventajas.

Si consideramos los resultados que se tienen en implementaciones anteriores y nuevas consideraciones de contribución, entonces surgen las propuestas nuevas, relativas a este trabajo :

- La implementación de LIDA con la generación de un nuevo código intermedio REC.
- Nuevas facilidades para una ejecución en un mildware.
- Posibilidades para una ejecución en un ambiente paralelo.

Capítulo 3

Lenguaje REC

En este capítulo se describe el Compilador de Expresiones Regulares que por sus siglas en inglés será llamado de ahora en adelante REC (Regular Expression Compiler) [1]; como se escribe un programa en REC y como funciona su compilador.

En la sección 3.1 se describe el funcionamiento del compilador destacando las funciones principales contenidas en su biblioteca. También, se pone especial atención en explicar de que manera puede ser configurado para casos particulares. Se detalla la sintaxis y semántica de un programa REC y se dan algunos ejemplos útiles para el control de un programa. En la sección 3.2 se trata de la versión de REC que fue configurada para trabajar con BD y de esta forma servir para el sistema LIDAWEB

3.1. Descripción de REC configurable

REC consta de una biblioteca implementada en lenguaje C; para la compilación y ejecución de programas escritos en REC; su naturaleza configurable, permite que tales funciones sean adaptadas para diferentes propósitos.

REC es una herramienta que permite escribir programas mediante un código especial; en el cual se pueden describir operadores, predicados, comentarios y cualquier tipo de sentencia, como se haría en cualquier lenguaje de programación. Este código es interpretado y ejecutado por el compilador REC, el cual después de compilar un programa lo coloca en un arreglo de apuntadores a la función que los ejecuta; de esta forma llevar acabo la ejecución del programa.

El funcionamiento descrito anteriormente se realiza por medio de las funciones *rec_c(*

[1]La propuesta de REC aparece en 1968 en el Acta Mexicana de Ciencia y Tecnología vol. II, No.1 p.p. 33-43, (enero-abril) de 1968, con el título "A Convert Compiler of REC for the PDP-8 " , por Harold V. McIntosh.

) y *rec_x( )*, las cuales se encargan de la compilación y la ejecución respectivamente; tales funciones están Incluidas en la librería de REC y es suficiente con invocarlas para lograr que trabaje el compilador.

Figura 3.1: Diagrama de flujo del funcionamiento

La figura 3.1, es un diagrama donde se muestra el flujo del funcionamiento de REC. Podemos apreciar que; como anteriormente se mencionó; la función *rec_c( )* se encarga de la compilación del código fuente. Entonces *rec_c( )* realiza la lectura del código y el análisis sintáctico. Después si no se encontraron errores en la compilación; *rec_x( )* realiza la ejecución de las operaciones para producir los resultados.

La función *rec_c( )* se invoca de la siguiente manera:

*int rec_c (char *source, Inst *prog, int plen, struct fptbl *table)*

Sus principales argumentos son:

- El programa fuente (*char *source*). Es un apuntador al programa escrito en REC el cual debe compilarse.
- Un arreglo de apuntadores donde será colocado el programa ya compilado *(Inst *prog*).
- Una tabla de predicados (*struct fptbl *table*) que contiene, para cada código ASCII un apuntador a la función de compilación, un apuntador a la función que ejecuta las operaciones correspondientes y un apuntador a una cadena de caracteres que sirve para poner comentarios, mismos que pueden ser desplegados durante la edición del programa.

Esta función retorna un 1 si no hay errores en la compilación y -1 si su terminación fue ocasionada por un error. Este valor debe ser usado para controlar la ejecución. La función que se encarga de la ejecución de un programa REC es *rec_x ( )* y esta debe ser llamada solamente en caso de que el valor devuelto por la función de compilación sea 1; porque una compilación con error será causa de problemas en la ejecución.

La función *rec_x(Inst *prog)* recibe como parámetro el arreglo de apuntadores donde fue colocado el programa compilado; por medio de estos apuntadores se invocan a las funciones de ejecución.

En la biblioteca de REC también están incluidas las funciones necesarias para compilar los símbolos de control (mediante estos símbolos se controla la ejecución del programa) esenciales para la estructura de REC y funciones para compilar y ejecutar predicados con estructuras más complejas.

Un elemento importante en la biblioteca de REC, es la tabla de predicados; la cual puede ser modificada de acuerdo a las necesidades de cada aplicación [2]. En esta tabla se especifica el símbolo ASCII, la función de compilación y la función de ejecución asociadas a este. Esto es, que por cada símbolo ASCII que es leído del código fuente de REC, el compilador, mediante un apuntador, localiza la función que lo compila y la función que lo ejecuta. Por ejemplo tenemos el siguiente código REC :

(@ x ;)

[2]Posteriormente, al trabajo publicado en 1968 según la nota 1, fueron desarrolladas algunas versiones de REC como aplicaciones. REC/Mat, matricial; REC/V, Visual; REC/M, Markov, etc.

Enseguida vamos a ver como el compilador encuentra las funciones de compilación y de ejecución para el caracter leído del código fuente. En este caso el caracter es @. La entrada para el caracter @ en dicha tabla es la siguiente línea:

r_pred1, r_call, "@ - llama a la subrutina ",

Donde *r_pred*1 es una función para compilar un operador simple, r_call es la función definida por el programador para un caracter ASCII en particular, en esta función se especifican los procedimientos a ser ejecutados; enseguida tenemos la cadena de caracteres delimitada por doble comilla, la cual nos proporciona información acerca del propósito de la función. Se tiene una línea con los tres parámetros como en el ejemplo anterior, para cada caracter ASCII en la tabla de predicados.

Para explicar mejor lo anterior, se puede decir, que según el orden (en la tabla de predicados) en que se encuentre la función de compilación, la función de ejecución y los comentarios; es como se asignan a los caracteres del código ASCII. Cuando el compilador encuentra un caracter en el código fuente REC, será localizado en la tabla de predicados y ejecutará las funciones correspondientes a ese caracter.

Si se quiere adecuar el compilador de REC para un caso particular únicamente se necesita proporcionar la tabla de predicados e implementar las funciones con las operaciones correspondientes para cada predicado, según las necesidades para cada aplicación.

3.1.1. Sintaxis y Semántica de REC

REC es un lenguaje compacto que posee una estructura simple de control [18]; la cual, consiste de cuatro símbolos de control que son: paréntesis abierto "(", paréntesis cerrado ")", punto y coma ";" y dos puntos ":". También, símbolos que denotan operaciones; algunos pueden ser tratados como predicados ya que pueden ser evaluados como falso o verdadero. Mediante un predicado se puede controlar el camino a seguir; cuando un predicado es falso la secuencia tomará un nuevo camino. Esto permite al usuario describir lo que desea que un programa realice.

Por medio de esta herramienta se puede escribir fácilmente un programa, ya que posee una sintaxis simple, corta y fácil de recordar. El programador podrá tener a la mano las instrucciones que son más utilizadas dependiendo de la aplicación para la que fue configurado REC.

Para describir la forma en que debe estructurarse un programa en REC, se comenzará por mencionar el funcionamiento de los símbolos básicos de REC, mediante los cuales se lleva el control del programa.

- Los paréntesis delimitan subrutinas y causan que la expresión tome valor falso, lo-

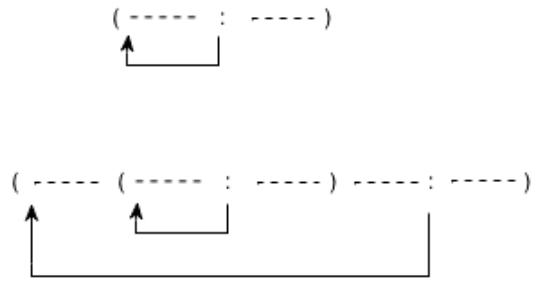


Figura 3.2: Comportamiento de los dos puntos

Figura 3.3: Comportamiento del punto y coma

grando un salto al siguiente segmento de la expresión como sucede con un falso predicado.

- Los dos puntos (:) ejecuta y regresa al principio de la expresión (donde abre el paréntesis). Como se muestra en la figura 3.2.
- El punto y coma (;) ejecuta y continua a la derecha de la expresión (sale de los paréntesis con valor verdadero) vemos un ejemplo en la figura 3.3.

Como en todos los lenguajes de programación se deben seguir ciertas reglas en el momento de la escritura de un programa. A continuación se muestra la sintaxis para crear un programa REC :

```
Prog : : = Expr  | "{" {Prog  char} Prog "}"

Expr : : = "("{ {Prog  |  pred} ( " : "  |  " ; " ){ Prog | pred} ")"
```

Donde *Prog* y *Expr* representan un programa REC y una expresión, respectivamente, *char* representa un caracter ASCII y *pred* representa un predicado (usualmente uno o dos caracteres, pero en algunos casos serán más).

Para que se comprenda mejor cómo crear una sentencia de control en REC, se muestra el siguiente ejemplo donde P es un predicado (condición) y W, W_1 y W_2 son operadores, los cuales siempre son evaluados como verdaderos:

Expresión REC	*Sentencia de control*
()	false
(;)	true
(P_1; P_2; . . . P_n;)	P_1 or P_2 or . . . P_n
(P_1P_2 . . . P_n;)	P_1 and P_2 and . . . P_n
(P W_1;W_2;)	if P then W_1 else W_2
(P W: ;)	while P do W
(WP : ;)	repeat W until P
(P_1 W_1; P_2 W_2;P_3 W_3; W_4;)	if P_1 then W_1
	else if P_2 then W_2
	else if P_3 then W_3
	else W_4

Como se puede apreciar en el ejemplo anterior, los paréntesis están delimitando subrutinas y cuando ponemos un punto y coma; el compilador ejecuta lo que esté dentro del paréntesis y enseguida sale. Cuando el compilador encuentra un predicado (P) falso, el compilador salta a la derecha y continua con el siguiente. Con la explicación anterior, se puede comprender mejor el *or*; primeramente se evalua P_1 y si se encuentra que es un predicado falso, entonces, se sigue evaluando el siguiente que es P_2, si este es verdadero, se ejecuta y sale del paréntesis; por lo que se ejecuta únicamente un predicado.

Para el caso del *and*, se ejecutan todos los predicados verdaderos, ya que el punto y coma, está colocado al final de la expresión y cuando el compilador llega hasta el punto y coma, es cuando sale de la subrutina. Se puede crear una sentencia if mediante la evaluación de P, si es un predicado verdadero entonces se ejecuta el operador W_1; si P es falso entonces el compilador ira a ejecutar W_2.

Para el *while* primero se evalua P y si este es verdadero se ejecuta W, mediante los dos puntos se regresa al principio de la expresión entonces se comienza de nuevo a evaluar P y ejecutar W; mientras P sea verdadero seguirá repeitiendose el ciclo. En el caso del *repeat* sólo cambiará el orden se ejecutara W repitiéndose varias veces hasta que P sea falso.

3.2. REC para Base de Datos

Una Base de Datos (BD) es una colección de datos almacenados en un soporte informático de acceso directo [18]. Para poder acceder a los datos es necesario un Sistema Gestor de Bases de Datos (SGBD).

El concepto de SGBD (en inglés Data Base Management System DBMS), aparece a principios de los 7oťas por CODASYL; como un sistema de "software " que permite llevar acabo gestión de datos en una BD. [3]

En principio, de la definición de BD como un conjunto de datos, los cuales se encuentran todos interrelacionados. Existe una estructura de almacenamiento que permite navegar a través de toda la base.

Por otro lado, un SGBD es un conjunto coordinado de programas, procedimientos, lenguajes, etc. que suministra, tanto a los usuarios no informáticos como a los analistas, programadores o al administrador, los medios necesarios para describir, recuperar y manipular los datos almacenados en la BD, manteniendo su integridad, confidencialidad y seguridad [18].

Las dos principales funciones del manejador de BD. Primero, corresponde al enfoque de solución de problemas como indagar o la actividad de recuperar, que reacciona al previo almacenamiento de los datos, en adecuadas estructuras de almacenamiento como mapeo de entidades o relaciones, que se codifican en el mundo real.

La segunda función corresponde a la actualización, la cual incluye el almacenamiento original de los datos, su repetida modificación debido a sus cambios frecuentes, y finalmente el borrado.

Existen varios gestores de BD entre los cuales tenemos Oracle, Microsoft SQL Server, Borland Interbase entre otros; el software que hemos citado es comercial y dentro del software libre tenemos MySQL, gestor usado en la web (combinado con php y apache) y PostgreSQL, que es el gestor que trataremos.

3.2.1. PostgreSQL

Antes de comenzar a hablar de PostgreSQL, es necesario conocer el Lenguaje Estructurado para Consultas, el cual por sus siglas en inglés es llamado SQL. SQL es un lenguaje relacional, versátil y poderoso para realizar consultas a una BD [3]. Debido a la diversidad de lenguajes y de BD existentes, la manera de comunicar entre unos y otras sería realmente complicada. Por lo que se han creado estándares que permiten realizar las operaciones básicas de una forma universal. SQL permite trabajar con cualquier tipo de lenguaje (C, PHP, etc) en combinación con cualquier tipo de BD (PostgreSQL, SQL Server, MySQL,

[3] CODASYL DBTG. Reporte publicado en ACM SIGFIDET, Noviembre de 1971.

etc.).
PostgreSQL, es un sofisticado SGBD objeto-relacional. Un SGBD relacional, permite al usuario almacenar partes de información relacionada, en estructuras de datos de dos dimensiones, llamadas tablas. Estos datos pueden consistir de algunos tipos definidos, como números enteros, números de punto flotante, cadenas de caracteres y timestamps. Los datos insertados en una tabla son organizados usando un sistema de celdas, formado por filas y columnas [3].

El aspecto objeto-relacional de PostgreSQL adiciona numerosas mejoras a el modelo relacional de datos convencional. Este incluye soporte para arreglos (multiples valores en una sola columna), herencia (relaciones de padre-hijo entre tablas), y funciones (métodos invocados por sentencias SQL). Para el desarrollo avanzado, PostgreSQL incluso soporta la extensibilidad de sus tipos de datos y lenguajes procedurales. Debido a este concepto objeto-relacional, las tablas son aveces llamadas clases, mientras las filas y columnas pueden ser referidas como instancias-objeto y atrfibutos-objeto, respectivamente [3].

Open Data Base Conectivity (ODBC) es un API (interfaz) de conectividad entre aplicaciones de BD cliente y servidor. Mediante el API ODBC se lleva acabo la comunicación remota desde un cliente a la BD de PostgreSQL; se utiliza la entrada directa de comandos SQL para introducir y recuperar los datos de la BD; por ejemplo, se pueden crear tablas, llenar dichas tablas con datos y manejar estos datos por medio de sentencias SQL que son mandadas a través de la interfaz ODBC.

El objetivo principal que movió a realizar la configuración del compilador REC, fue lograr una conexión remota por vía ODBC, a una BD desarrollada en PostgreSQL y realizar consultas a esta, mediante peticiones en lenguaje SQL. En la figura 3.4 podemos visualizar como funciona el compilador de REC y una BD en PostgreSQL.

Como podemos ver en el cliente se encuentra el sistema visual, mediante el cual se edita el flujograma y se traduce a código REC. Este código es compilado en el servidor y este se encarga de ejecutar las consultas a la BD.

3.2.2. Flujograma, REC y Base de Datos.

El compilador de REC fue configurado para interactuar con una BD, como ya se explicó anteriormente. De manera que, mediante el flujograma se obtiene el código REC. Este Código expresa la consulta que deseamos realizar a la BD.

El ejemplo de la figura 3.5 es un flujograma que expresa consultas a una BD. El código REC equivalente a ese flujograma es el siguiente:

((U(# AB;)C;)D;)

donde:

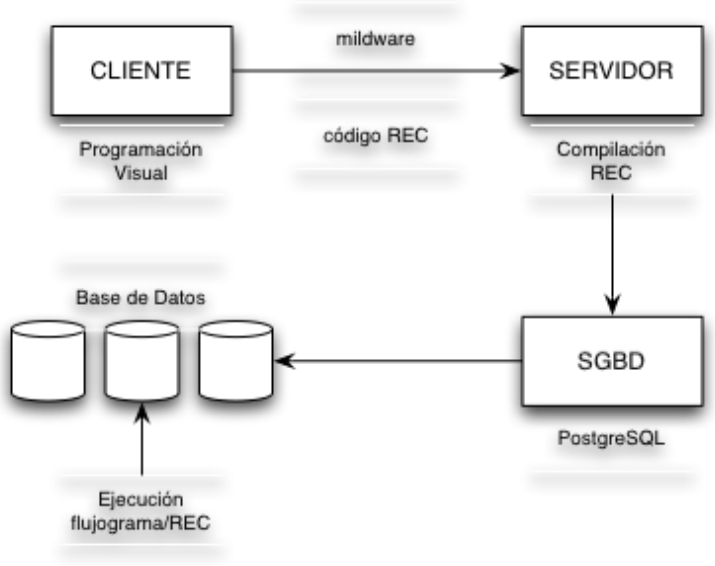


Figura 3.4: Funcionamiento de REC con PostgreSQL.

U	es el operador UNION del álgebra relacional.
#	es el operador JUNTA NATURAL del álgebra relacional.
A, B, C y D	son relaciones.

Si apreciamos el flujograma 3.5; podemos observar que entre las relaciones A y B se está expresando una JUNTA NATURAL. Entre la relación C y lo que resultó de la operación relacional anterior, se hace una UNIÓN y el resultado es colocado en D.

Por medio de el ejemplo anterior; podemos ver la forma de representar consultas, operaciones sobre esas consultas y como fluye la información en el flujograma.

3.2.3. REC para LIDAWEB

Anteriormente, se describió la sintaxis y semántica de un programa fuente escrito en REC, ya con esta base es más sencillo comprender como se escribe un programa REC mediante el cual se necesita realizar consultas a BD. Básicamente, se utiliza una notación de postfijo (el operador va después de los operandos) y se utilizan los símbolos de control (paréntesis, punto y coma y dos puntos); a continuación tenemos un ejemplo de un programa REC para BD.

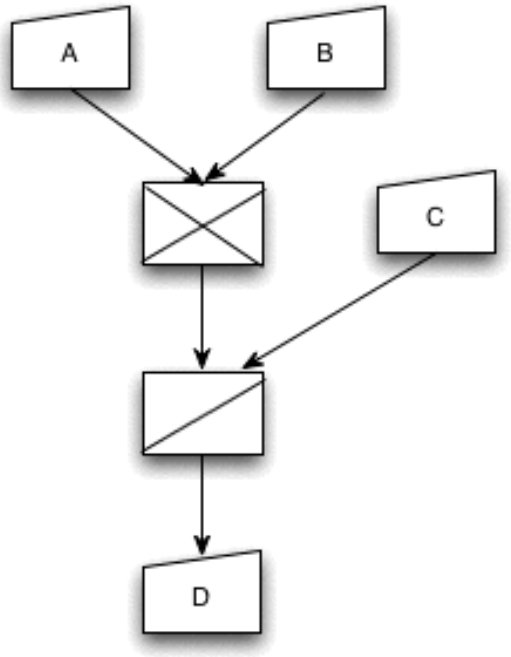


Figura 3.5: Consulta expresada mediante un flujograma

```
({("campo""tabla""condición"&;)}{("campo""tabla"&;)}{("*""tabla"&;)})
```

Como podemos apreciar en la estructura anterior las expresiones que definen consultas están delimitadas por paréntesis y llaves; las cuales logran que se pueda realizar cada subrutina por separado y entregar un resultado independiente para cada una. Los caracteres entre comillas, definen los nombres del campo, tabla y condición que son necesarios para conformar una sentencia en SQL. El símbolo & representa un Select, el cual es un comando de SQL que permite mediante la especificación de ciertos parámetros (tabla, campo, etc.) hacer una consulta a la BD

Cuando el compilador encuentra una expresión con un & y antecedida por mínimo dos cadenas de caracteres delimitadas por comillas y seguido de un punto y coma; será interpretado como una sentencia en SQL, compuesta de un Select con los datos de nombre de tabla, campo y condición en algunos casos. Enseguida tenemos un ejemplo de código fuente REC:

```
({("coleccion""ACRONIMOCAT" &;)}{("*""AISLA"&;)}{("numero""ACRONI"
     "numero < 629"&;)})
```

Podemos ver en este código, la escritura del programa se hace facilmente ya que se sigue una sintaxis simple y fácil de recordar. También vemos como es posible utilizar un comodín u omitir algún dato (campo o condición) para definir una consulta.

En este ejemplo se especifica una consulta al campo "colección " de la tabla ACRONIMOCAT. Paralelamente a esta consulta se especifica la consulta a todos los campos de la tabla AISLA y de la misma forma es especificada la consulta al campo "numero " de la tabla ACRONI, donde "numero " es menor que 629. Como puede verse el código permite una especificación en paralelo.

El código anterior es interpretado por el compilador RECBD y transformado a sentencias SQL que quedan de la siguiente manera:

```
select  coleccion  from  ACRONIMAT
select  *          from  AISLA
select  numero     from  ACRONI       where  numero  629
```

Las sentencias SQL anteriores, son enviadas a la BD desde una función escrita en lenguaje C; la cual forma parte del compilador RECBD para ser ejecutadas y entonces obtendremos los resultados de dichas consultas. La forma en que fue implementada está versión de REC para LIDAWEB; es explicada en la sección de implementación del sistema.

3.3. Conclusion.

Un sistema visual generador de código intermedio es una herramienta que nos da muchas ventajas entre las cuales se pueden citar las siguientes :

- Se puede ejecutar el código en otra máquina. Puede ser en una máquina que se encuentre entre el cliente y el servidor (mildware).
- Se puede paralelizar el código; esto es que, una parte del código se ejecute en un procesador y otra parte en otro, dividiendo el código según los procesadores con que se cuente; de esta forma se ejecutan los procesos al mismo tiempo.

Capítulo 4

Diseño e Implementación de LIDA/REC

Principalmente esta implementación surge de la necesidad de mejorar el funcionamiento de LIDAWEB. Esto se llevó acabo, complementando el sistema existente. El sistema fue implementado en Lenguaje Java y otra parte en lenguaje C.

Debido a la implantación en lenguaje intermedio REC; la nueva versión de LIDAWEB será llamada ahora LIDA/REC. Se describe el diseño y como fue implementada. Explicando cada módulo que lo compone. Se muestra como fue configurado el compilador de REC, para trabajar con BD en PostgreSQL y de esta forma funcionar para LIDAWEB.

4.1. Descripción del Sistema

Este sistema, consta básicamente de dos módulos que son: el sistema visual LIDAWEB (Editor) y el compilador REC (intérprete); los cuales ya fueron explicados anteriormente; ahora se describirá como funcionan en conjunto.

El principal cambio que tuvo el sistema LIDAWEB; fue la adición de la propiedad de generar código intermedio. En general; LIDA/REC genera un código en lenguaje REC; este código es interpretado y ejecutado por el compilador REC. En la figura 4.1 se muestra el funcionamiento.

Enseguida se describe el funcionamiento de LIDA/REC, más detalladamente. Por medio del editor LIDAWEB, es creado el flujograma. Este flujograma representa las consultas, las cuales deseamos realizar a la BD. LIDA/REC transforma el flujograma a código intermedio en lenguaje REC y lo guarda en un archivo de texto. Con la generación del código, termina el funcionamiento de LIDA/REC (interfaz visual).

Entonces comienza el compilador REC. El compilador, tiene la facultad de leer el

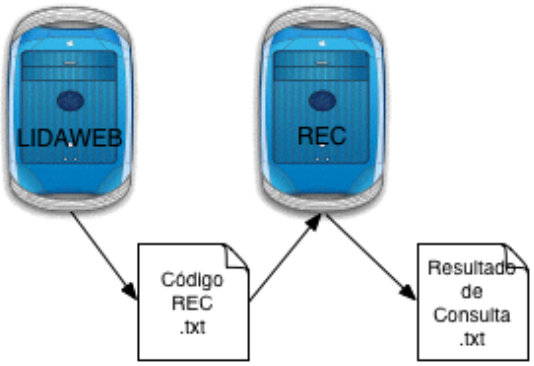


Figura 4.1: Panorama General del Sistema LIDA/REC

archivo de texto, donde fue guardado el código REC y después interpretarlo y ejecutarlo. Los resultados son colocados en otro archivo de texto.

Como vemos, el funcionamiento de LIDA/REC se divide en dos partes; que pueden operar por separado; esto es que LIDAWEB, puede ser operado en una máquina cliente y REC en otra máquina que para mayores ventajas deberá ser un servidor. Esto se hace, para delegar al servidor la tarea de interactuar con la BD y que la máquina que cotiene a LIDAWEB sólo se ocupe de interactuar con el usuario.

El módulo de REC, fue incorporado para utilizar un código intermedio entre la interfaz visual (usuario) y la BD. Esto se aprecia en la figura 4.2. En dicha figura se ve la diferencia entre la versión anterior de LIDAWEB y la que ha sido implementada.

En la versión anterior, por medio de LIDAWEB se hacían las consultas a la BD. En la nueva versión, se encuentra REC de intermediario entre la BD y LIDAWEB. De tal manera que LIDAWEB ahora sólo funciona como editor del flujograma y generador de código REC y ahora las consultas a la BD se hacen por medio de REC.

Entonces enseguida, se explicará detalladamente cada módulo por separado. Se comienza por LIDAWEB la interfaz visual con la que interactúa el usuario.

4.2. LIDAWEB (Editor)

4.2.1. Descripción del Módulo

Como vemos la nueva versión de LIDAWEB ya cuenta con la propiedad de generar código intermedio. Para lograrlo fueron sustituidas las funciónes de traducir el flujograma

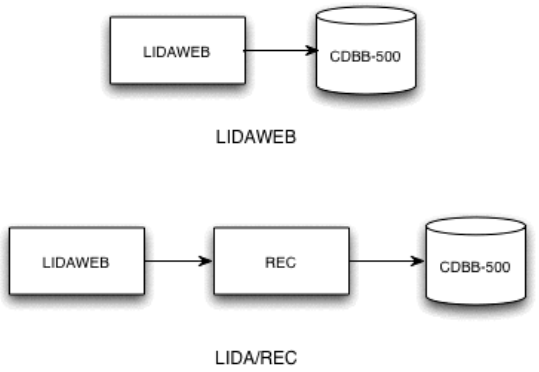


Figura 4.2: Diferencia entre las versiones de LIDAWEB

en sentencias SQL (que eran envíadas por medio de JDBC a la BD). En su lugar se implementaron funciones, que traducen el flujograma en código REC.

Por lo que, la función de este módulo es permitir la creación del flujograma y la traducción de este a lenguaje REC. Podemos apreciarlo mejor en la figura 4.3.

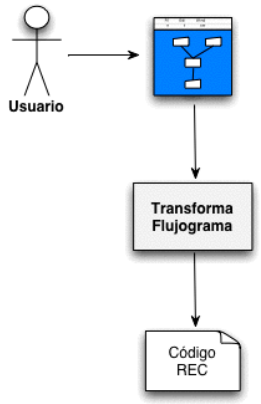


Figura 4.3: funcionamiento de LIDAWEB

El usuario interactúa con este editor, a través del mouse y el teclado para crear dicho flujograma. En el flujograma se especifican las consultas a la BD. Este flujograma se transforma a código REC.

Una de las principales ventajas de laprogramación visual, basada en el paradigma

de diagramas de flujo de datos, es que la edición es dirigida por sintaxis. El editor de flujogramas no permite hacer conexiones incorrectas entre íconos, de tal forma que ayuda a prevenir los errores de lógica y sintaxis.

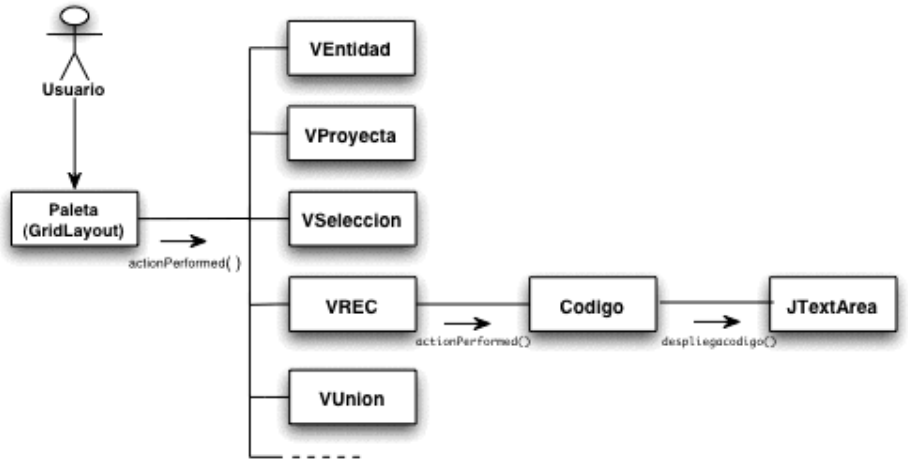


Figura 4.4: Eventos internos para obtener el código REC

En la figura 4.4, podemos ver el comportamiento interno del sistema para obtener el código REC. Es necesario presionar el botón del ícono que tiene por nombre Código REC. Este botón se encuentra dentro del GridLayout de la clase Paleta. Por medio del método $actionPerformed()$ es instanciada la clase $VREC$. Esta clase ya ha recibido antes el código REC relacionado al flujograma. Después al oprimir el botón que dice código; será instanciada la clase *Código* mediante el método $actionPerformed()$. Por último por medio de un JTextArea se despliega en pantalla el código resultante.

4.2.2. Arquitectura del Módulo

Para mostrar el diseño del sistema, es necesario utilizar un lenguaje de modelado. El lenguaje de modelado es la notación (principalmente gráfica) de que se valen los métodos para expresar los diseños . UML (Unified Modeling Language); es un lenguaje unificado de modelado es el sucesor de la oleada de métodos de análisis y diseño orientado a objetos; que surgió a finales de la década de 1980 y principios de la siguiente [22].

UML define una notación y un metamodelo. La notación es el material gráfico, que se ve en los modelos; es la sintaxis del lenguaje de modelado. Un metamodelo es un diagrama, usualmente un diagrama de clases, que defina la notación [21].

En general; UML consta de una notación (esquemática en su mayor parte) con que se construyen sistemas, por medio de conceptos orientados a objetos [21].

4.2.3. Diagrama de Clases del Sistema

El diagrama de clases del diseño, describe gráficamente las especificaciones de las clases de software y de las interfaces (las de *Java*, por ejemplo) [21]. Normalmente contiene la siguiente información: clases, asociaciones, atributos (información sobre los tipos de los atributos), interfaces, métodos, dependencias, etc.

El primer paso, en la elaboración de este diagrama de clases; consiste en identificar las clases que intervienen en la solución del software. El siguiente paso, consiste en dibujar un diagrama de clases para estas clases e incluir en el modelo conceptual los atributos ya identificados. Para finalizar, se identifican e incluyen los métodos al diagrama de clases.

Para la implementación de la nueva versión de LIDAWEB, se modificaron y adicionaron algunas clases. Estos cambios fueron, con el fin de mejorar el sistema existente. En la figura anterior; podemos apreciar el diagrama de clases general. En el diagrama, se muestran todas las clases que componen el sistema y como están relacionadas. También se resaltan, aquellas clases que fueron modificadas y las que fueron adicionadas.

En general; *Lida* es la clase principal. La clase *Paleta*; contiene a los botones, encargados de crear los Iconos (ventanas) y añadirlos al espacio donde se editan los flujogramas (lienzo); para esto existe una clase llamada *Lienzo*. La clase *LaConsulta* es la que muestra la instrucción SQL asociada al flujograma.

En la figura que se encuentra en la siguiente página; se muestra el diseño en UML del sistema. Sistema que fue desarrollado en *Java*. Después se describe detalladamente cada nueva clase; explicando como y para que fue implementada.

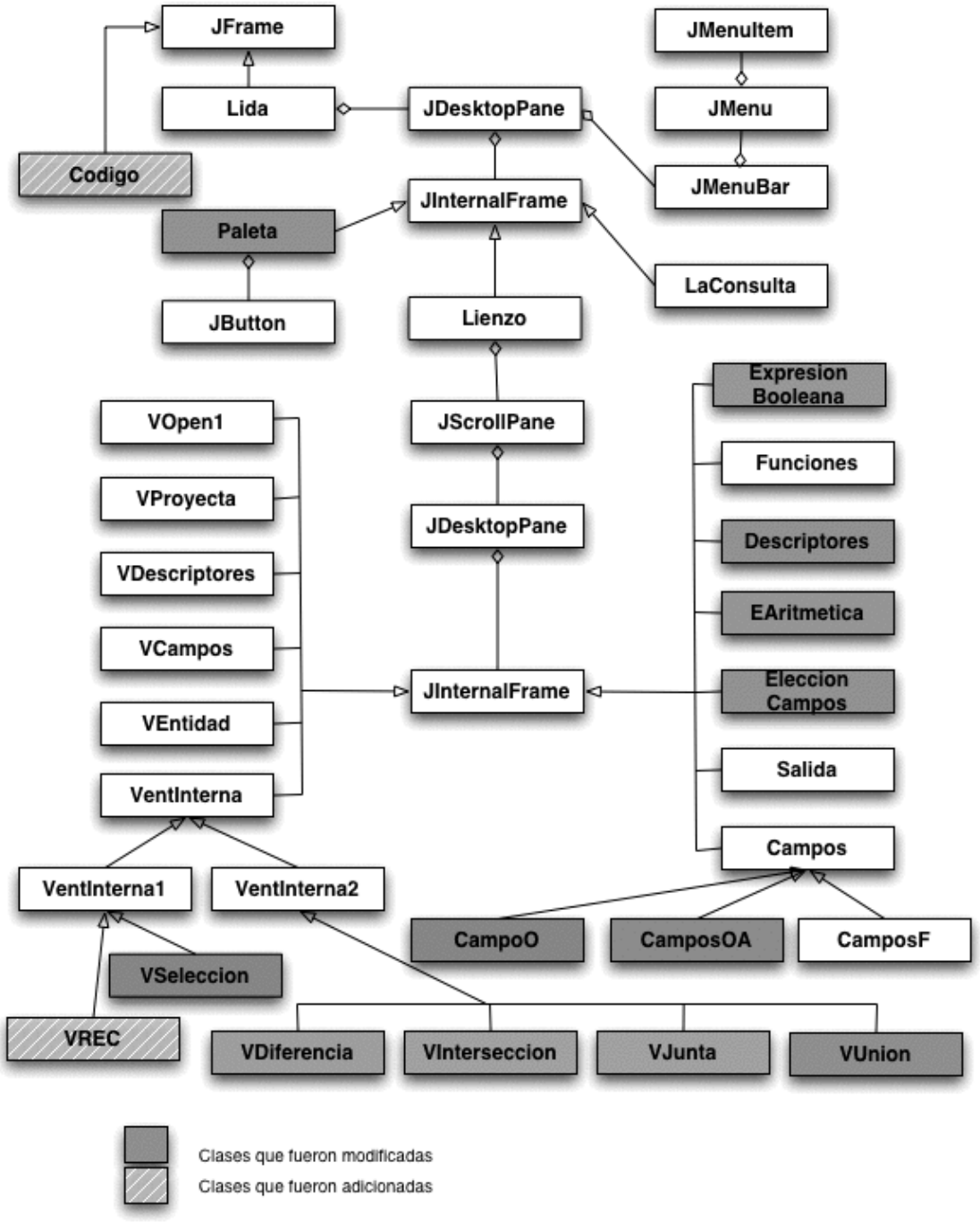
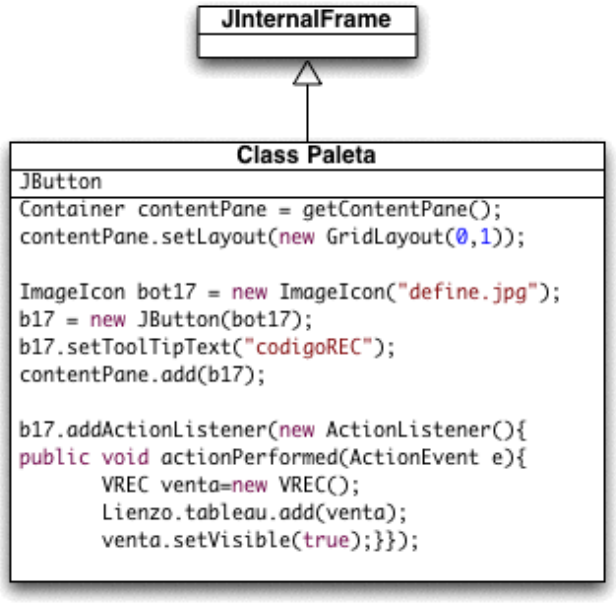


Diagrama de Clases del Sistema.

4.2.4. La Clase Paleta

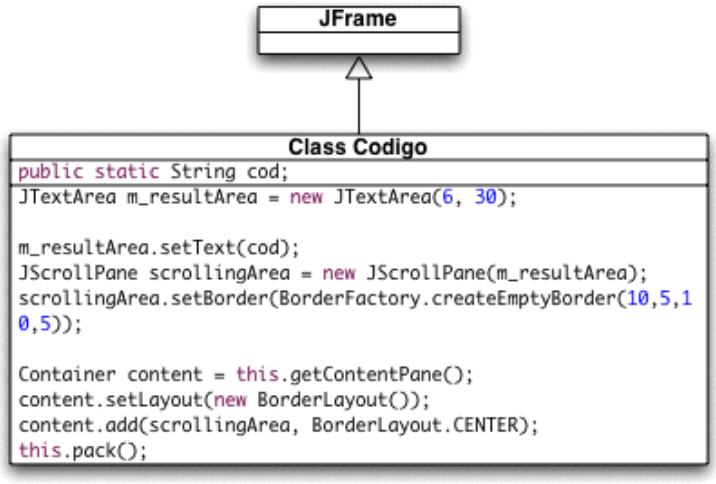


La clase *Paleta* se muestra en figura anterior; la cual extiende de la clase $JInternalFrame$. Esta clase; por medio del método *getContentPane*() crea un *GridLayout* que contiene los botones ($JButton$). Estos botones reciben una imagen como parámetro mediante el método $ImageIcon()$. El código restante es para definir las acciones del botón. En este caso; la acción de cada botón es hacer una instancia de una clase. En el código podemos ver que se está haciendo una instancia de la clase $VREC$. Todo ese código es por cada uno de los botones que componen la paleta.

En LIDAWEB existe una clase por cada boton. Cuando el usuario presiona uno de los botones es creada una instancia de la clase que repesenta gráficamente y la añade al espacio donde se edita el flujograma (lienzo).

La clase *Paleta* fue modificada para añadir un boton más. Este botón se añadió en la parte de abajo de la paleta con el nombre de REC. Por medio de este botón se hace una instancia de la clase $VREC$ la cual se tratara más adelante.

4.2.5. La Clase Codigo

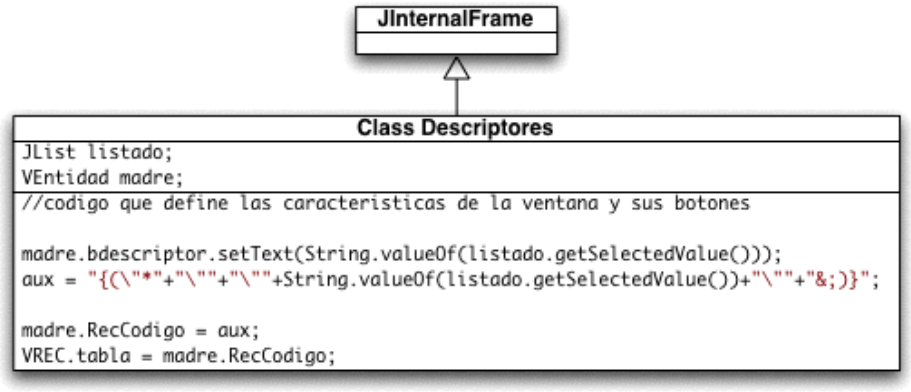


La clase *Codigo* se muestra en la figura anterior. Esta extiende la la clase $JFrame$ y utiliza una variable pública y estática ($String\ cod$); donde guarda una cadena de caracteres que corresponde al código REC.

Con el método $setText(cod)$ inicializa el área de texto y se envía el contenido de *cod*. Con $JScrollPane$ se inicializa el scroll para la ventana y con $setBorder()$ los bordes para esta. En el código restante se toma el contenido del texto y se centra en la ventana. Con esto queda definida la ventana, donde se desplegará el texto contendido en la variable *cod*.

Esta clase fue adicionada, para recibir el código REC. Este código se recibe en una variable *String*. Después el código es desplegado en la pantalla, mediante una caja de texto.

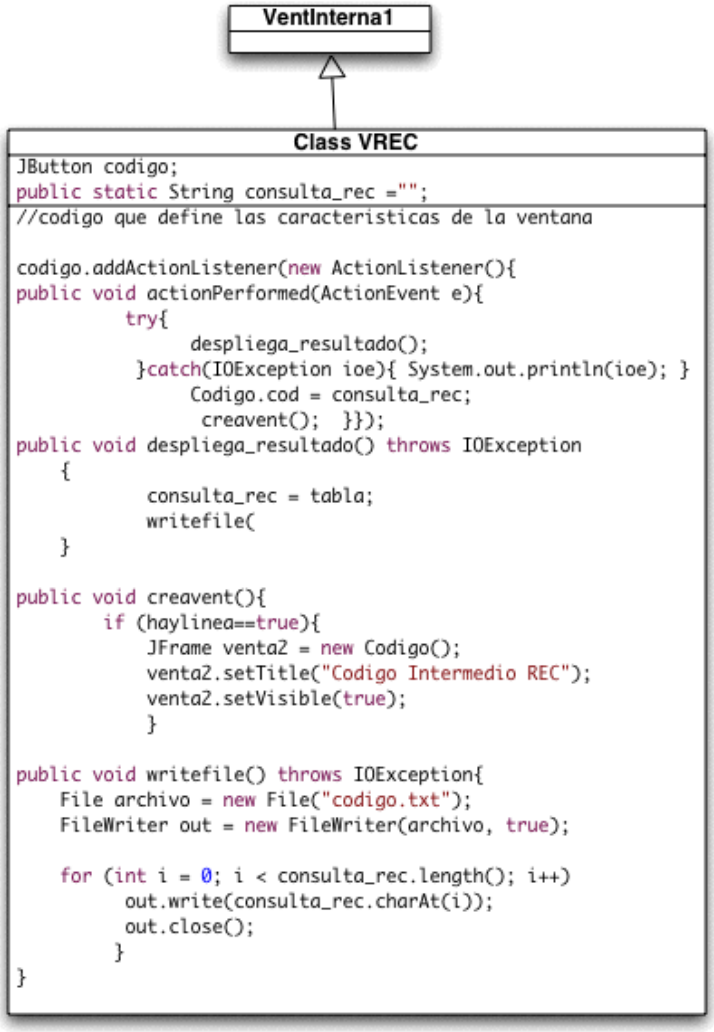
4.2.6. La Clase Descriptores



La clase *Descriptores* en la figura anterior; muestra el código que fue modificado para crear la sentencia en código REC. Esta clase es instanciada por la clase $VEntidad$; después de selecionar el ícono (ventana) Entidad. Entonces; en la pantalla se muestra, una lista de relaciones (entidades) de las cuales se debe elegir una. Después el ícono guarda el nombre de dicha entidad.

Podemos apreciar en el código; que con el método $getSelectedValue()$; se obtiene la entidad que fue elegida de una lista. En la variable *aux*; se guarda la sentencia *select* en código REC. Esta sentencia se creo con la entidad que fue elegida. Después, se envía a la variable *tabla* de la clase $VREC$. Dicha clase, es la encargada de recibir el código REC.

4.2.7. La Clase VREC



La clase *VREC* fue adicionada y se muestra en la figura anterior. En dicha figura, se muestra el código más relevante. Por cuestiones de espacio se omitió la parte de definición de la ventana (ícono).

Esta clase, es la que se invoca al oprimir en la paleta de íconos el botón de código REC. El ícono se despliega en la pantalla y se conecta al final del flujograma.

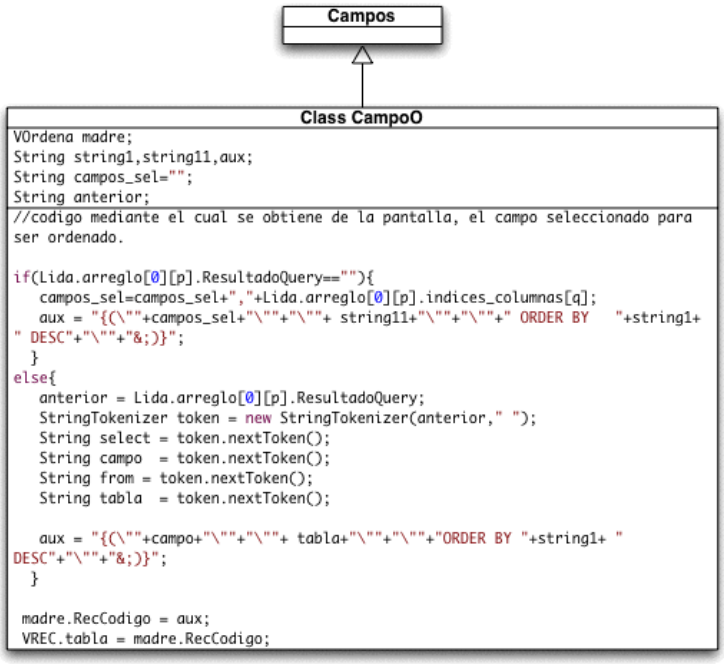
Dentro de la ventana se incluye un botón. Este botón se llama *codigo* y por medio del método *addActionListener*() podemos especificar que tiene que hacer este botón. Las acciones son las siguientes: se invoca un método llamado *despliega_resultado*(), se invoca a *creavent*() y también se asigna el contenido de la variable *consulta_rec* (aquí se guarda

la consulta) a la variable *cod* de la clase *Codigo*.

En el método *despliega_resultado*(); se asigna el contenido de *tabla* a *consulta_rec* y se llama al método *writefile*(). En este método; se guarda en un archivo de texto el contenido de *consulta_rec*.

Por medio del método *creavent*(); se crea una instancia de la clase Codigo. Esta clase como ya se explicó anteriormente crea una ventana de texto. En esa ventana se despliega el código REC.

4.2.8. La Clase CampoO

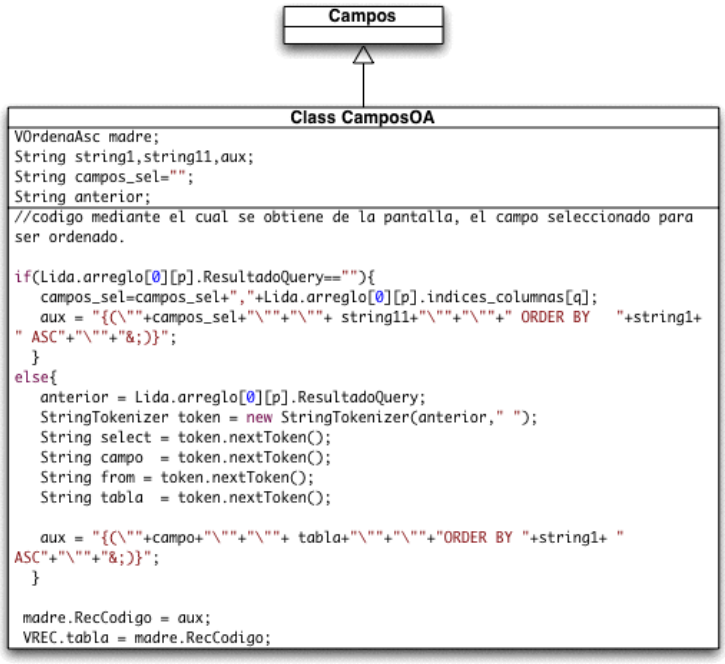


Esta clase extiende de la clase Campos. Por medio de esta clase se ordena de forma descendente el campo especificado. En la figura anterior se puede apreciar el código que fue anexado, para crear la consulta en REC. Esta consulta especifica el ordenamiento.

La bifurcación del código especifica, si hay guardada una consulta previa en la variable $Lida.arreglo[0][P].ResultadoQuery$. Si no hay una consulta anterior entonces; se procede a armar la consulta a partir de la variable *campos_cel* que recolecta mediante un ciclo for todos los campos que han sido seleccionados; con la variable $string11$, la cual contiene el nombre de la tabla; que se captó de la clase VDescriptores; y por último la variable $string1$, que contiene el campo que se debe de ordenar de manera descendente; y este fue captado por la clase $VOrdena$.

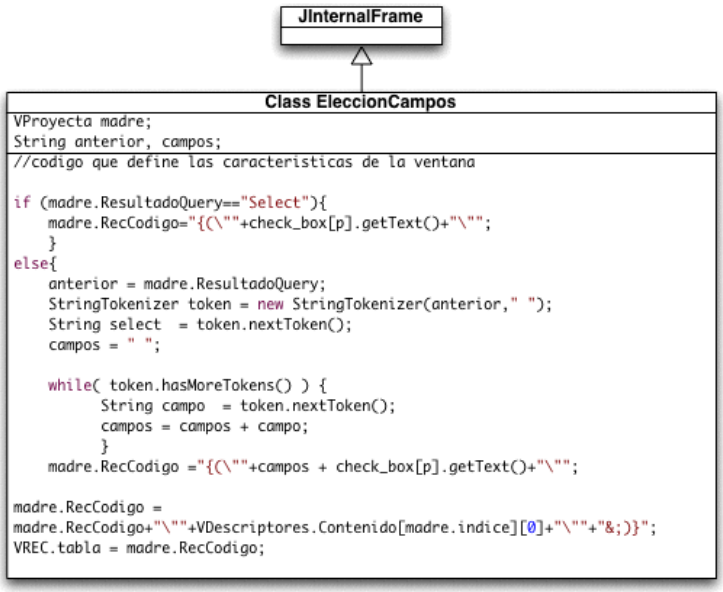
En el caso contrario, de que si haya una consulta previa en la variable $Lida.arreglo[0][P].ResultadoQuery$; entonces, es necesario descomponer esa consulta; por medio de un $StringTokenizer$. Después de haber obtenido las variables de *campo* y *tabla*; se crea la consulta. Por último se manda esta consulta a la variable *tabla* de la clase $VREC$.

4.2.9. La Clase CamposOA



Esta clase extiende de la clase Campos. Esta clase es la encargada de ordenar el campo especificado de forma ascendente. Si apreciamos el código de la figura anterior; podemos ver que es muy parecido al de la clase CampoO (anterior). Sólo cambia la clase madre que en esta caso es la clase $VOrdenaAsc$. Esta clase se encarga de obtener el nombre del campo a ser ordenado. El otro cambio fue; en la creación de la consulta se cambio la parte del $ORDERBY$; ahora se pone ASC.

4.2.10. La Clase EleccionCampos



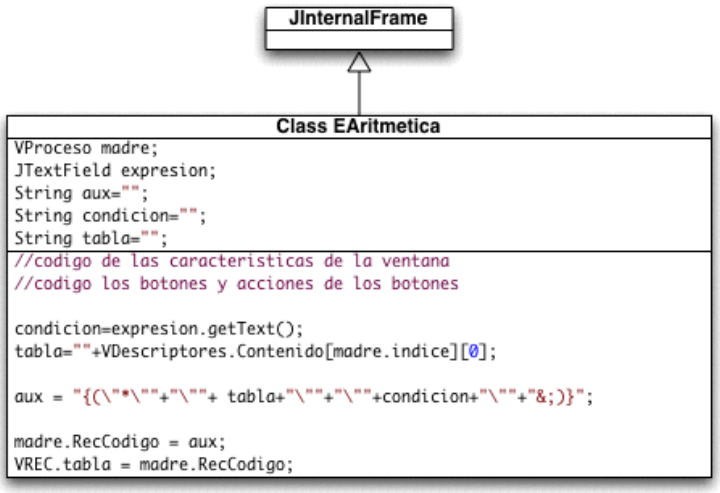
Esta clase muestra su código en la figura anterior. La sección de código que vemos, es la que fue modificada. Esta clase es instanciada por la clase $VProyecta$. El ícono VProyecta sirve para elegir campos. Los campos que nosotros queremos para crear nuestra consulta.

En el código se muestra una condición. Esta condición es para diferenciar, si es el primer campo que se ha elegido, ya hay alguno o varios antes. En el if se compara; si la variable $madre.ResultadoQuery$ es igual a "Select"; quiere decir que es el primer campo que se va a seleccionar. Entonces la consulta se crea con un solo campo y se guarda en la variable $madre.RecCodigo$.

Después para la otra opción; donde la variable $madre.ResultadoQuery$ es diferente. Esto quiere decir que en la variable aparte de la palabra Select, ya lleva dentro un campo que ya fue elegido anteriormente. Entonces lo que se necesita es quitar el Select de la variable. De esta forma sólo debe de quedar el nombre del campo.

Lo anterior se logra mediante un $StringTokenizer$. Mediante este se quita el Select de la variable y sólo se deja el campo. También se van aumentando en la variable, todos los campos que van siendo elegidos. Se crea la consulta en código REC y es guardada en la variable $madre.RecCodigo$. Para finalizar se manda a la variable $tabla$ de la clase $VREC$.

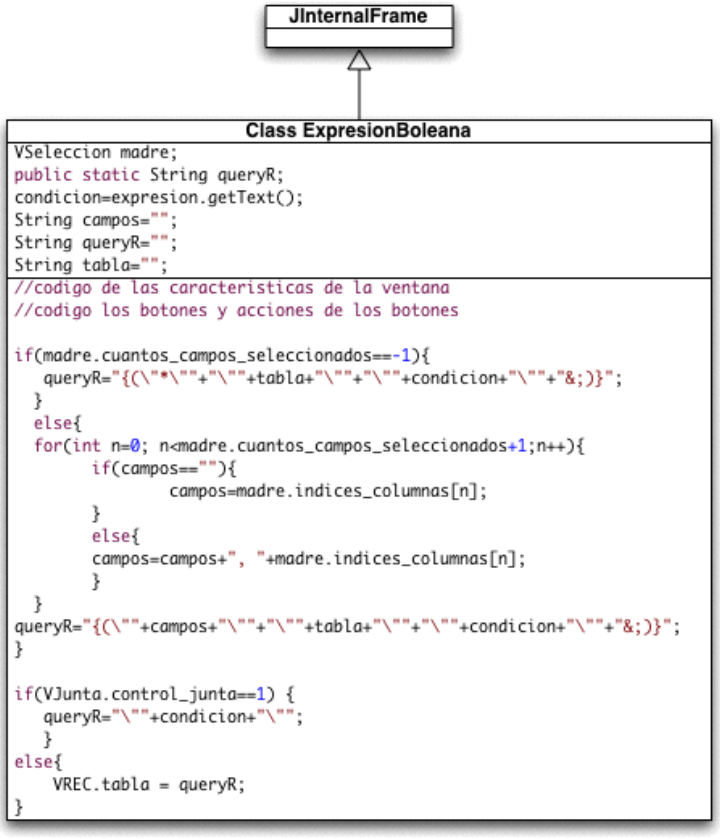
4.2.11. La Clase EAritmetica



Esta clase extiende de la clase $JInternalFrame$ y es instanciada por la clase $VProceso$. Ya que esta clase sirve para definir una expresión aritmética para un proceso. En la figura anterior se muestra el código que fue agregado para crear la consulta en REC.

Esta clase, despliega una ventana que contiene botones y un espacio para escribir una expresión aritmética. Como podemos apreciar en el código; la expresión se obtiene de un campo de texto y es guardada en la variable *condicion*. El nombre de la tabla o relación se obtiene de la clase VDescriptores. Con las dos variables anteriores se crea la sentencia en código REC. Esta sentencia es guardada en la variable *aux*. Para finalizar se envía la consulta a la variable *tabla* de la clase $VREC$.

4.2.12. La Clase ExpresionBoleana



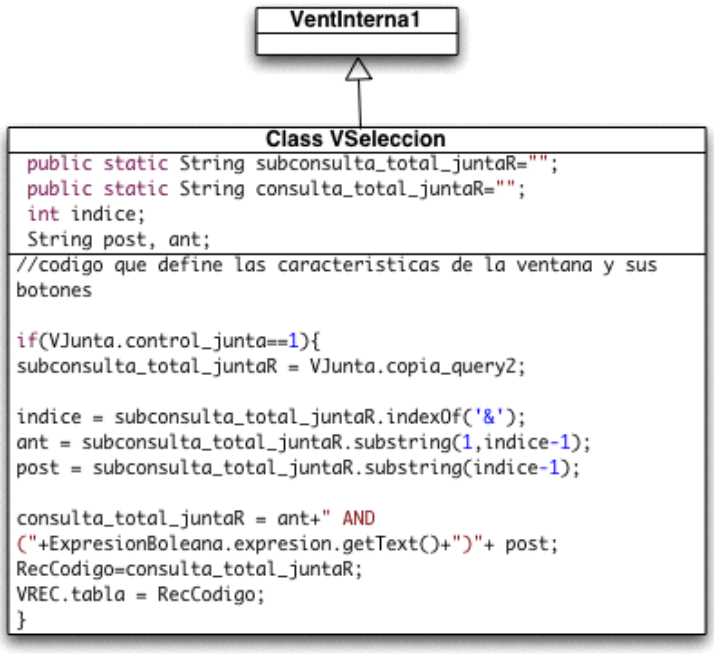
La clase *ExpresionBoleana* fue modificada y en la figura anterior se muestra sólo la parte de código que fue modificado. Esta Clase es instanciada por la clase $VSeleccion$; ya que en esta clase se especifica la expresión booleana o condición para la consulta.

Como podemos apreciar en el código se hacen varias bifurcaciones mediante la sentencia if. Esta sentencia primero evalúa; si el número de campos seleccionados desde la clase $VSeleccion$, es igual a -1. Entonces; se arma una sentencia *select* con nombre de tabla y condición; pero sin nombre de campo; en su lugar se pone un comodín. Toda esta consulta ya esta especificada en código REC.

Si el número de campos es diferente a -1 (*sentencia else*). Entonces se entra en un ciclo for. Este ciclo se hará tantas veces como el número de campos. En el cual; se asignará a la variable *campos* los nombres de los campos seleccionados. Si es más de un campo; se separa con una coma cada nombre. Termina el ciclo y enseguida se crea la consulta en código REC. Ahora si; es una sentencia *select* con el nombre o nombres de los campos, el nombre de la tabla y la condición.

Si el control de la clase $VJunta$ es igual a 1; la variable $queryR$ sólo deberá recibir la condición. Esto es; porque la sentencia *select* completa, será armada más adelante. En el otro caso de que el control de $VJunta$ sea diferente de 1; entonces la variable $queryR$ (contiene la consulta en código REC) es envíada a la variable *tabla* de la clase $VREC$.

4.2.13. La Clase VSeleccion



La clase $VSelección$ fue modificada. En la figura anterior; sólo se muestra el código que fue implementado. Como ya antes se mencionó; esta clase hace una instancia de la clase $ExpresionBoleana$. Esto es; para crear una ventana donde puede ser definida la expresión booleana. Esta expresión es la condición de una sentencia *select*.

En una consulta a la BD dentro de la estructura del *select*; puede utlizarse el operador *and*. El *and* se especifica dentro de la condición. Precisamente esa condición; es la que muestra el código. Vemos que; si se cumple la condición, que debe efectuarse un *and*; entonces se recibe la variable $VJunta.copia\_query2$ (consulta anterior). Esta variable fue captada por la clase $VJunta$ y es una consulta ya armada. Por lo que; por medio de los métodos $indexOf()$ y $substring()$; se logra separar y anexar en el lugar adecuado, la expresión que incluye el *and*.

Por último; se envía la consulta completa, por medio de la variable *RecCodigo* a la clase VREC.

4.2.14. La Clase VJunta

Esta clase, se encarga de hacer la operación relacional junta natural. En la figura anterior, se muestra el bloque de código que fue modificado. Se instancía esta clase, utilizando el ícono Junta.

En el código se aprecia lo siguiente: en la variable *query* se tiene una consulta anterior que consta de un Select y campos. Con el $StringTokenizer$ se descompone la consulta; ya que sólo se requiere tener los campos seleccionados. Al obtener los campos, se guardan en la variable *campos* y se crea la consulta en REC. Para finalizar se copia la consulta en la variabe $copia\_query2$ que será utilizada en otra clase más adelante y se envía a la variable *tabla* de la clase $VREC$.

4.2.15. La Clase VDiferencia

Esta clase extiende de la clase $VentInterna2$ y en la figura que se muestra en la hoja siguiente; se muestra el código que fue modificado. Esta clase responde al ícono que realiza la operación relacional diferencia.

En el código, se aprecian las condiciones que se deben respetar para la creación de la sentencia en REC. La condición principal $if((string1.compareTo(string2) == 0)(string1! = ""))$ especifica que deben de ser iguales los campos de las consultas a operar. Si esto ocurre entonces se procede a hacer la diferencia. Los if, que están dentro; como por ejemplo: $if(cad1! = nullcad1! = null(cad2 == null|cad2.equals("")))$ son para

distinguir si existe condición para la sentencia *Select*. Las opciones que hay son: condición sólo para la primera consulta, condición sólo para la segunda consulta, condición para las dos consultas y condición para ninguna de las consultas. Para cada una de las opciones anteriores es creada la sentencia en REC. Después la consulta en REC, es envíada a la variable *tabla* de la clase $VREC$.

La Clase VUnion

Esta clase realiza la operación relacional de unión. La clase es instanciada en el momento que se ocupa un ícono con el nombre de unión. En la figura anterior se muestra el código. Este código es el mismo que la clase $VDiferencia$. Unicamente en el momento de crear la consulta; se sustituye el símbolo de la diferencia (-) por el símbolo de la unión (U).

4.2.16. La Clase VInterseccion

La clase $VInterseccion$ extiende de la clase $VInterna2$. Se muestra el código modificado en la figura anterior. Por medio de esta clase se puede hacer la operación relacional intersección. Esto se logra al momento de utilizar el ícono de Intersección.

Al observar el código; se aprecia, que es muy parecido al de la clase anterior (VDiferencia). En este se ocupa también una condición $if(string1.compareTo(string2) == 0)$ que compara los campos. Si los campos son iguales se procede a realizar otras bifurcaciones. Dichas bifurcaciones; son para saber si existe condición para la sentencia Select; de igual forma que en la clase anterior.

Y las opciones que se presentan son: condicíon sólo para la primera sentencia, condición sólo para la segunda sentencia, condición para las dos sentencias y condición para ninguna sentencia Select. Después se procede a armar la consulta en código REC. Por último se envía la consulta REC a la variable *tabla* de la clase $VREC$.

4.3. REC (Intérprete)

En esta sección se explica, como se configuró el compilador para interpretar el código REC generado por LIDA/REC. El compilador se encarga de hacer la conexión a la BD de PostgreSQL. Después hace las consultas mediante peticiones en lenguaje SQL y las manda a la BD por vía ODBC. Las librerías de este compilador; están implementadas en lenguaje C. Enseguida se describe como fue implementado este módulo.

4.3.1. Descripción del Sistema

Primero se explicará más ampliamente como se realiza la compilación y ejecución del código fuente. En la figura 4.5 podemos apreciar que el funcionamiento del compilador está dividido en dos fases. Estas dos fases son: la fase de compilación y la fase de ejecución.

En la fase de compilación, se comienza por leer el código fuente. El código fuente se encuentra en un archivo de texto. Cada caracter del código leído, es localizado en la tabla de Predicados. Mediante esta localización se obtienen los apuntadores a sus funciones de compilación y ejecución. Enseguida son ejecutadas las funciónes de compilación y mediante estas funciones, se envían los apuntadores de las funciónes de ejecución al programa compilado.

La fase de ejecución se lleva acabo si no existieron errores en la fase anterior (compilación). Para la ejecución sólo se necesita un apuntador al programa compilado. Por medio de este apuntador localizamos las funciones que deben ser ejecutadas. Los resultados se obtienen mediante las operaciones que son ejecutadas en dichas funciones. Estos resultados se guardan en un archivo de texto.

Como ya se explicó, el compilador básicamente trabaja en dos fases. Cada fase es ejecutada por una función. Es decir, la función $rec\_c()$ se ocupa de la fase de compilación y la función $rec\_x()$ de la fase de ejecución. Entonces para utilizar el compilador REC; es necesario invocar a estas dos funciones y con eso es suficiente para que trabaje el compilador. Con esto explicado; se puede pasar a la descripción de como se realizó la implementación.

Para que el compilador REC funcionara para nuestro propósito; fue necesario implementar una función principal. En esta función principal $main()$; se realiza la conexión a la BD de PostgreSQL. Enseguida se lee el código fuente desde un archivo de texto. El código que fue leído se guarda en una cadena y se manda como argumento (entre otros) a la función de compilación $rec\_c()$. Después se hace una bifurcación para que, en el caso de que no exista ningún error después de la compilación, se ejecute la función de ejecución $rec\_x()$. En el otro caso de que si exista error; termina el procesamiento y manda un mensaje de error. En la figura 4.6 se muestra el diseño de la función principal.

En la figura 4.7; se puede apreciar una porción del código de la función principal;

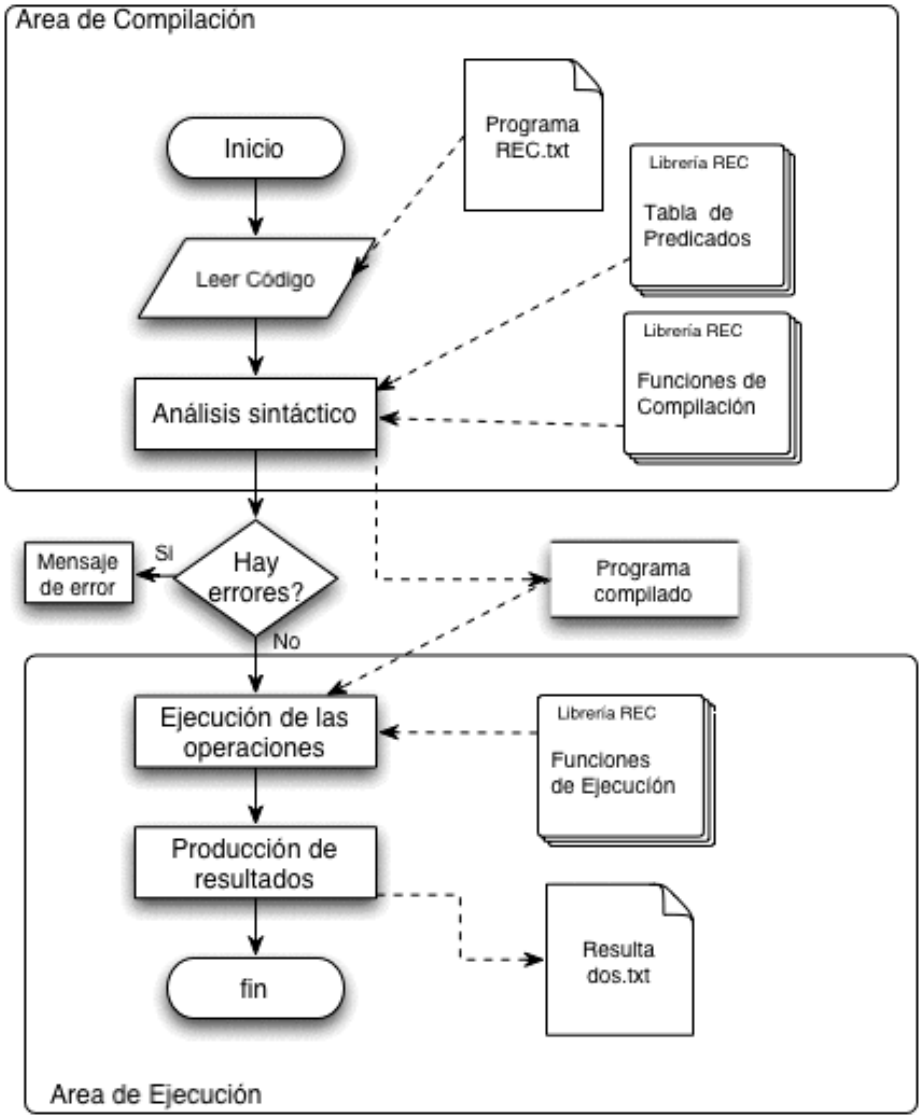


Figura 4.5: Panorama General del Sistema REC

en el cual se hace la conexión a la BD. Para hacer uso de las funciones de PostgreSQL se debe incluir la librería $libpq - fe.h$. Mediante la función $PQsetdbLogin()$se hace la conexión a la BD; especificando en esta función el host, el puerto, el password, etc. Con la función $PQexec(conn, "BEGIN")$ se inicia un bloque de transacciones y con la función

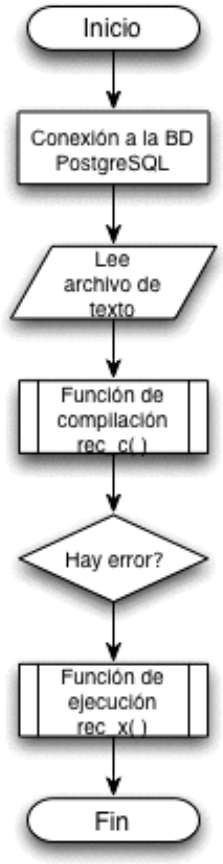


Figura 4.6: Diagrama de flujo de la función *main*()

PQexec(*conn*,"*COMMIT*") se finaliza el bloque. Entre ese bloque de transacciones, se encuentra el código que se encarga de invocar el compilador de REC. Se hace de esta forma, para que REC pueda hacer las consultas a la BD. Para finalizar, se cierra la conexión a la BD con la función PQfinish( ).

En la otra porción de código de la función principal (figura 4.8) se muestra, de que manera fueron incluidas las funciones *rec_c*() y *rec_x*() para utilizar el compilador REC. También se incluye la lectura de un archivo de texto; el cual contiene el código REC. El código leído se guarda en una cadena de caracteres.

La función *rec_c*() que está encargada de la compilación del código REC; recibe como parámetro la cadena de caracteres que contiene el código REC, un arreglo de apuntadores

```
RESUMEN: Conexión a una Base de Datos en PostgreSQL

#include <stdio.h>
#include <stdlib.h>
#include "libpq-fe.h"
PGconn   *conn;
PGresult *res;

void exit_nicely(PGconn *conn);

main(int argc, char **argv)
{
  char *pghost, *pgport, *pgoptions, *pgtty, *dbName, *login, *pwd, *anc, *chl;
  int nFields, i,j;

  pghost = "xserver00.cs.cinvestav.mx";
  pgport = "5432";
  pgoptions = NULL;
  pgtty = NULL;
  dbName = "microbd";
  login = "investigador1";
  pwd = "investigador1";

  conn = PQsetdbLogin(pghost, pgport, pgoptions, pgtty, dbName, login, pwd);
        if (PQstatus(conn) == CONNECTION_BAD)
        {
           fprintf(stderr, "Connection to database '%s' failed.\n", dbName);
           exit_nicely(conn);
        }
        res = PQexec(conn, "BEGIN");
            if (!res || PQresultStatus(res) != PGRES_COMMAND_OK)
        {
             fprintf(stderr, "BEGIN command failed: %s");
             PQclear(res);
             exit_nicely(conn);
        }

        // código para invocar al compilador REC

        res = PQexec(conn, "COMMIT");
        PQclear(res);

        PQfinish(conn);
 }
void exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
 }
```

Figura 4.7: Porción de código de la función *main*()

donde será colocado el programa compilado, la longitud de ese arreglo, y la tabla de

predicados. Después de haber ejecutado la compilación; y la función *rec_c*() regresa un valor de 1. Esto quiere decir, que no hay errores en la compilación; entonces procede la función de ejecución *rec_x*(), esta recibe como parámetro el arreglo de apuntadores donde fue colocado el programa compilado. Por medio de estos apuntadores se invocan las funciones de ejecución.

```
RESUMEN: Lectura de archivo de texto e invocación de las funciones principales de REC
ENTRADA: Archivo .txt con el código fuente REC

#include <ctype.h>
#include "rectblAdry.h"
#include "rec.h"

#define TAM 1000
#define PLEN 3000

FILE *in;
int i;
char cadena[TAM];
Inst prog[PLEN];
char *str;
struct fptbl dtbl[];

main(int argc, char **argv)
{
       in = fopen ("recprog.txt", "r");

        while ((cadena[i]=fgetc(in))  != ' ' && cadena[i] != '\t' && feof(in) == 0){
        i++;
      }

// código para iniciar un bloque de transacciones de PostgreSQL

        if (rec_c(cadena,prog,PLEN,dtbl)==1) rec_x(prog);
        else  printf("\nerror en compilador REC");

// código para finalizar el bloque de transacciones de PostgreSQL

fclose(in);
```

Figura 4.8: Porción de código de la función *main*()

4.3.2. Configuración de REC para Bases de Datos

Para la configuración del compilador de REC es necesario implementar nuevas funciones en lenguaje C que formarán parte de la librería de REC. Estas funciones se localizan

mediante la tabla de predicados y son invocadas por las funciones principales *rec_c*() y *rec_x*().

En la figura 4.9, se muestra el procedimiento que sigue la función *rec_c*() para realizar la compilación. Primero se lee el caracter ASCII de la cadena (programa en REC). Depediendo de que caracter sea, son localizadas las funciones. La localización de las funciones que corresponden a cada caracter se hace mediante la tabla de predicados. Esto se hace para cada uno de los caracteres de la cadena. Podemos ver en la figura; que para cada caracter se ejecuta su función de compilación y se guarda el apuntador a la función de ejecución. Entonces las funciones que son ejecutadas en esta etapa son *r_cstrp*(), *selectc*(), *r_lpar*(), etc.

En la otra etapa la función *rec_x*(); ya no necesita consultar a la tabla de predicados. Como ya se mencionó anteriormente; esta función recibe un arreglo de apuntadores a las funciones de ejecución. Estos apuntadores fueron obtenidos de la etapa anterior. Entonces se ejecutan las funciones de ejecución, *r_lstrp*(), *selectx*() y para el caracter (no se tiene función de ejecución.

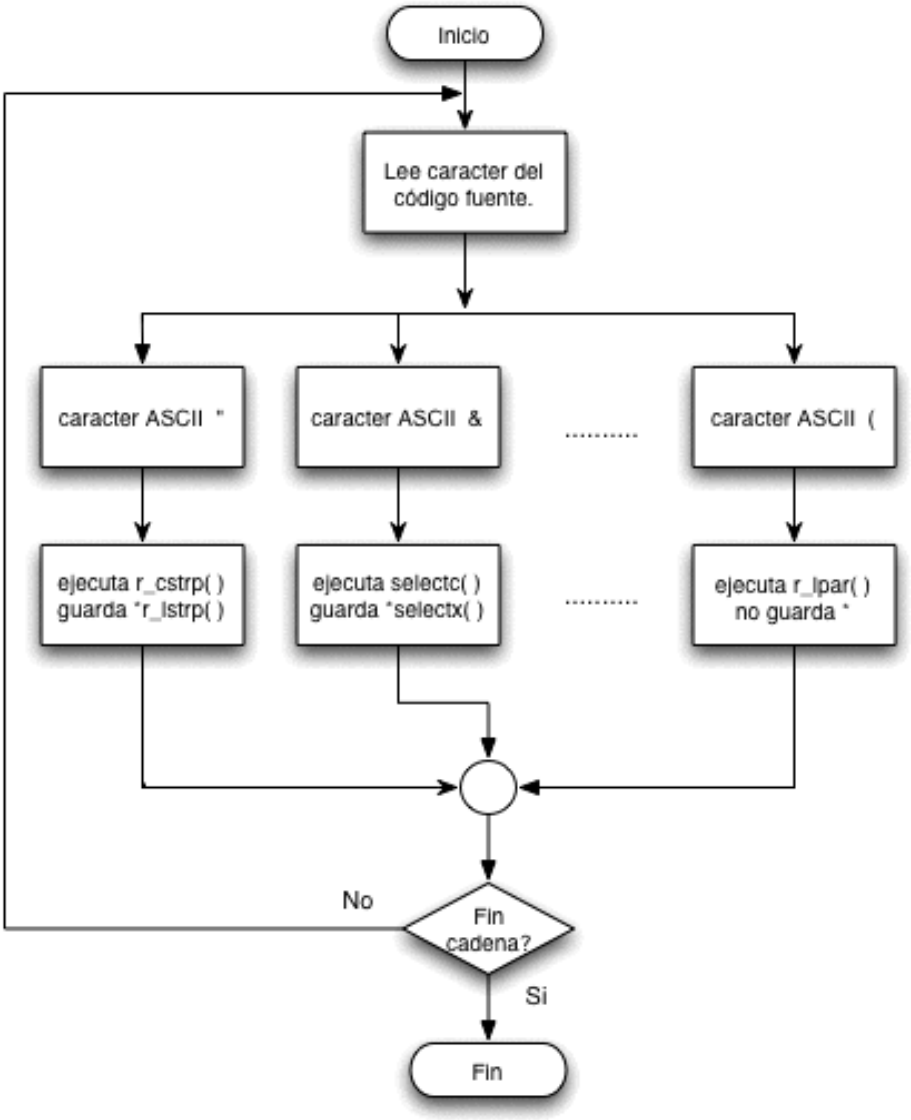
Anteriormente se habló de la tabla de predicados. En esta tabla se especifican los nombres de las funciones de compilación y ejecución para cada caracter ASCII. En esta tabla se declararon las nuevas funciones implementadas. Enseguida se muestra una pequeña parte de la estructura de la tabla de predicados.

```
struct fptbl dtbl [ ] = {

r_cstrp,   r_lstrp, "guarda cadena de caracteres entre comillas",
selectc,   selectx, "realiza un select en SQL a la BD",
r_lpar,    FALSE, "Inicia una expresión".,
.....      .......
.....      .......
};
```

La primer línea de este ejemplo; especifica, que cuando el copilador encuentre el caracter ("); se ejecutaran las siguientes funciones: la función *r_cstrp*() y la función *r_lstrp*(). La función *r_cstrp*() se encarga de compilar una cadena de caracteres que se encuentra delimitada por comillas. La función *r_lstrp*() guarda dicha cadena de caracteres en una variable para que sea después utilizada por la función que contiene al Select.

La segunda línea es para el caracter (&). Las funciones a ejecutarse son: la función *selectc*() y la función *selectx*(). La función *selectc*(), compila una expresión que realiza un *Select* en SQL y la función *selectx*(), ejecuta ese Select tomando las cadenas de caracteres que fueron obtenidas con la función *r_lstrp*() para utilizarlas como los nombres de la tabla, el campo y la condición y armar con todo eso la sentencia SQL y mediante ODBC ejecutar



la consulta.

Acontinuación se describen las funciones que fueron modificadas y anexadas. La primera es la función r_lstrp( ); se modificó de manera que guarde la cadena delimitada por comillas, en una variable. El código se ve en la figura 4.10 y sirve para guardar una cadena de caracteres que se encuentra delimitada por doble comilla. En el código se puede ver

```
RESUMEN: función que lee una cadena string entre comillas y la guarda en una variable.

r_lstrp()
{
   int i=0,j;
   char *a;
   r_pbufp = (char *) (*r_pc++);
   r_pbufl = (int) (*r_pc++);

   for (j=0; j<20; j++)
        aux[j] = ' ';

   while (i < r_pbufl) {
        aux[i] = *r_pbufp;
        r_pbufp++;
        i++;
        }
   getflag(gflag);

   if (flag == 0)
       {
        strcpy(var1, aux);
        flag = 1;
         printf("variable 1 %s\n",var1);
        }
    else if (flag == 1)
        {
         strcpy(var2, aux);
         if (gflag == 1)
              flag = 2;
          else flag = 0;
            printf("variable 2 %s\n",var2);
         }
   else if (flag == 2)
         {
          strcpy(var3, aux);
          flag = 0;
          printf("variable 3 %s\n",var3);
          }
}
```

Figura 4.10: Código de la función r_lstrp( )

como se extrae toda la cadena del arreglo r_pbufp y se guarda en el arreglo aux. Después se va asignando a las variables $var1$, $var2$ y $var3$. Que más adelante van a servir para el campo, la tabla y la condición. Esto va a depender de cuantas cadenas de caracteres existan en la sentencia y esto se controla mediante la bandera $gflag$. Esta bandera se trae de la función $getflag()$ y esta nos indica si son tres variables las que componen al $Select$(campo, tabla y condición) y en caso contrario serían sólo dos (campo y tabla).

La otra función que se añadió a la tabla de predicados fue *selectc*(). Esta función se encarga de la compilación del caracter . Por medio de esta función, se coloca en el programa compilado, el apuntador a la función de ejecución. Podemos ver en el código de la figura 4.11; que se llama a la función *r_code*. Dicha función sirve para compilar un operador simple. Esta función, tiene como argumento a la variable $func$. Esta variable es un apuntador a la función de ejecución; que en este caso es la función *selectx*().

La variable $flagvar$ es una bandera que se enciende (toma valor de 1) en el momento en que se confirma que hay 6 comillas (tres pares); esto quiere decir que son tres variables. En el código se aprecia que con el ciclo (while) se hace un barrido y se van contando las comillas. Después se hace la bifurcación de que debe haber cuatro o seis comillas; si esto no sucede es porque hay un error en la sintaxis. Por último se verifica que si hay seis comillas la bandera toma valor de 1. Esta variable es envíada a la función r_lstrp( ) mediante la función $getflag$().

La función *selectx*() es la que se encarga de la ejecución del caracter &. En esta función se ocupan las variables que fueron obtenidas anteriormente por la función r_lstrp(). Para armar la sentencia en SQL y realizar la consulta a la BD. En la figura **??**; se muestra una porción de la función; en la cual se obtienen las variables por medio de la función $getVar$(). Después se procede, a crear la sentencia SQL con las variables obtenidas. Por medio del $sprintf$, se guarda en la variable *consulta*. Se abre un archivo de texto; en el que se guardarán los resultados de dicha consulta.

La otra parte de la función *selectx*(), que se muestra en la figura 4.13; se encarga de envíar la consulta (sentencia SQL) a la BD y guardar en un archivo el resultado de dicha consulta. Se puede apreciar en el código, que por medio de la función $recuperaConexion$(); se obtiene la conexión a la BD que se hizo desde un principio por medio de la función principal $main$(). Con $PQexec$() se envía la consulta a la BD, para ser ejecutada. Con el $PQdisplayTuples$(), se guarda el resultado de la consulta en un archivo de texto.

También fueron implementadas las funciones $getflag$(), $getVar$(), $getVar3$() y $recupera Conexion$(). De estas funciones ya se mencionó para que fueron utilizadas y por su simpleza no tiene caso explicar como fueron implementadas. Con esto se termina esta sección. En la siguiente sección, se muestran algunos ejemplos de consultas a la BD de PostgreSQL.

4.4. Conclusión

El sistema LIDA//REC es una contribución importante enel área de los lenguajes visuales; por su propiedad de generar código en lenguaje REC. Este código es simple y fácil de entender. Se puese ver que la configuración del compilador de REC, es sencilla y tiene varias ventajas al permitir introducir cualquier tipo de operador en la biblioteca de REC, para ser interpretado.

```
RESUMEN: función que compila un select (&). Cuenta el número de comillas que hay
antes del Select, y con esto determina si hay dos o tres variables.

selectc(func)
Inst func;
{

 int c, cont=0;
 int len=0;
 char aux[200];
 char *buffer;

 r_code(func);
 flagvar = 0;

 while ((c = *buffer--)  != -1 && c != ';' && c != '&' && c != '(') {
       aux[len] = c;
       len++;
         if(c == '"')
           cont++;
       }

 if(cont == 4 || cont == 6)
      {
         if( cont == 6)
             flagvar = 1;
       }
 else { r_errterm(8);}

 for(i=0;i<200;i++)
        aux[i] = ' ';

 }
```

Figura 4.11: Código de la función selectc( )

```
RESUMEN: función que ejecuta un select (&). Obtiene las variables que conformarán
la sentencia SQL. Parte I
int selectx() {

PGresult   *res;
PGconn     *conn1;
char campo[20], condicion[200];
char tabla[20], consulta[300];
int     fillAlign;
char *  fieldSep;
int     printHeader;
int     quiet;
char   archivo[20];
FILE *  fp;
printHeader  = 1;
fillAlign   = 1;
quiet = NULL;

 if (flagvar == 1)
    {
        getVar(campo,tabla);
        getVar3(condicion);
        sprintf(consulta, "DECLARE myportal CURSOR FOR select %s from %s where %s",
campo, tabla, condicion);
    }
 else
    {
        getVar(campo,tabla);
        sprintf(consulta, "DECLARE myportal CURSOR FOR select %s from %s", campo,
tabla);
    }
 sprintf(archivo,"resultado%d.txt",a);
 a++;
 fp = fopen(archivo,"w");
```

Figura 4.12: Porción de código de la función selectx( )

```
RESUMEN: función que ejecuta un select (&). Recupera la conexión a la BD. y envía
la consulta a la BD. el resultado lo coloca en un archivo de texto. Parte II
int selectx() {

 conn1 = recuperaConexion();
 res = PQexec(conn1, consulta);
    if (!res || PQresultStatus(res) != PGRES_COMMAND_OK)
        {
                fprintf(stderr, "SELECT failed: %s");
                PQclear(res);
                exit_nicely(conn1);
        }
        PQclear(res);

        res = PQexec(conn1, "FETCH ALL in myportal");
        if (!res || PQresultStatus(res) != PGRES_TUPLES_OK)
        {
                fprintf(stderr, "FETCH ALL command didn't return tuples
properly\n");
                PQclear(res);
                exit_nicely(conn1);
        }

    PQdisplayTuples(res, fp, fillAlign, fieldSep, printHeader, quiet);
    PQclear(res);


        for(i=0;i<300;i++)
        consulta[i] = ' ';

        res = PQexec(conn1, "CLOSE myportal");
        PQclear(res);
        return 0;
}
```

Figura 4.13: Ultima parte de código de la función selectx( )

Capítulo 5

Caso de Estudio y Ejemplos

En esta sección, se muestran algunos ejemplos de consultas a una BD. La BD que se utilizó como caso de estudio para el sistema LIDA/REC; fue la base CDBB-500 del proyecto Micro-500 el cual se tratará enseguida.

5.1. La Base de Datos CDBB-500

CDBB-500 es el número con el que se identifica a nivel internacional "La Colección de Cultivos Microbianos". Esta colección fue creada en el CINVESTAV IPN; con el objetivo de contar con un aservo de cultivos puros para su aplicación en la docencia e investigación. La Unidad de la Colección de Microorganismos del CINVESTAV mantiene aproximadamente 3,000 cultivos microbianos, provenientes de diferentes zonas geográficas del territorio nacional y del mundo e incluye varios grupos de cepas.

Las cepas que conforman esta colección de cultivos; son organizadas mediante sus características morfológicas, fisiológicas, genéticas, de aislamiento, de conservación, de modificación, de aplicación y de uso. Para organizar esos datos se creó el proyecto de base de datos CDBB-500 dirigido por la maestra Jovita Martínez y el Dr. Sergio V. Chapa Vergara. Su principal objetivo es organizar y explotar la BD para aplicaciones y estudios en microbiología.

En la figura 5.1; se muestra el modelo conceptual del proyecto de base de datos CDBB-500; mediante un diagrama entidad vínculo extendido. Este proyecto contiene datos taxonómicos, sinónimos, acrónimos, fuente de aislamiento, identificador, años de identificación y depósito, nombre de instituciones depositantes, datos geográficos, datos curatoriales y datos bibliográficos entre otros.

También se le adicionó, información concerniente a la aplicación industrial de cada una de las cepas. Esta información es: medios de cultivo, temperaturas de crecimiento y

parámetros fisicoquímicos y biológicos que intervienen en la consrvacieon de un estado viable y estable del acervo microbiano. Esta información es porque es una colección de interes biotecnológico.

Otra parte del proyecto lo constituye la información referente a la morfología macro y micro, la fisiología y los estudios de biología molecular. Así también la información acerca del depósito, el aislamiento, la identificación, la distribución y la custodia de las cepas. Todo esto resulta de importancia primordial para el manejo y control integral de las colecciones.

Esta BD de microbiología, fue el caso de estudio de el sistema implementado. El usuario interactúa con el Lenguaje Visual LIDA/REC para poder consultar a esta BD. Enseguida se describe como se hacen las consultas mediante la interfaz visual.

5.2. interfaz Visual

En esta sección, se explica el manejo del sistema LIDA/REC. Esto se hace, mostrando algunos ejemplos de consultas a la base de datos CDBB-500.

Como ya se mencionó anteriormente; el flujograma está compuesto por íconos interconectados, por los cuales fluye la información. Cada ícono representa un proceso; ya sea una sentencia para consulta a BD o una operación sobre dicha consulta. En la figura 5.2; podemos apreciar el editor de flujogramas que es toda la zona azul y en la parte izquierda del editor, se encuentra la paleta de íconos disponibles para la creación del flujograma.

En la figura 5.3 podemos ver la paleta de botones. En esta paleta de íconos se tienen disponibles los botones con los nombres de Entidad, Proyecta, Diferencia, Unión, Intersección, Selección, Proceso, Ordena Ascendente, Ordena Descendente, Función, Clustering, Junta Natural y Código REC. Este último fue agregado y se encarga de proporcionar un ícono (ventana) con el nombre de Código REC.

Este ícono, se encarga de generar el código en lenguaje REC (código asociado al flujograma). Dicho ícono debe conectarse al final del flujograma y automáticamente desplegará una ventana que contiene el código REC. Este código, aparte de ser desplegado en la pantalla; se guarda automáticamente en un archivo de texto. Esto puede apreciarse en la figura 5.4.

Con la representación de íconos interconectados, se describe una consulta a la BD. En la cual se puede especificar la entidad, el campo y la condición. Si se quiere hacer una consulta a la BD en la que se incluya un *Select*; como el de la sentencia SQL siguiente:

Select Proyecta $from$ Entidad *where* Condición.

Se comienza por seleccionar el ícono llamado *Entidad*. Con este ícono, como su nombre

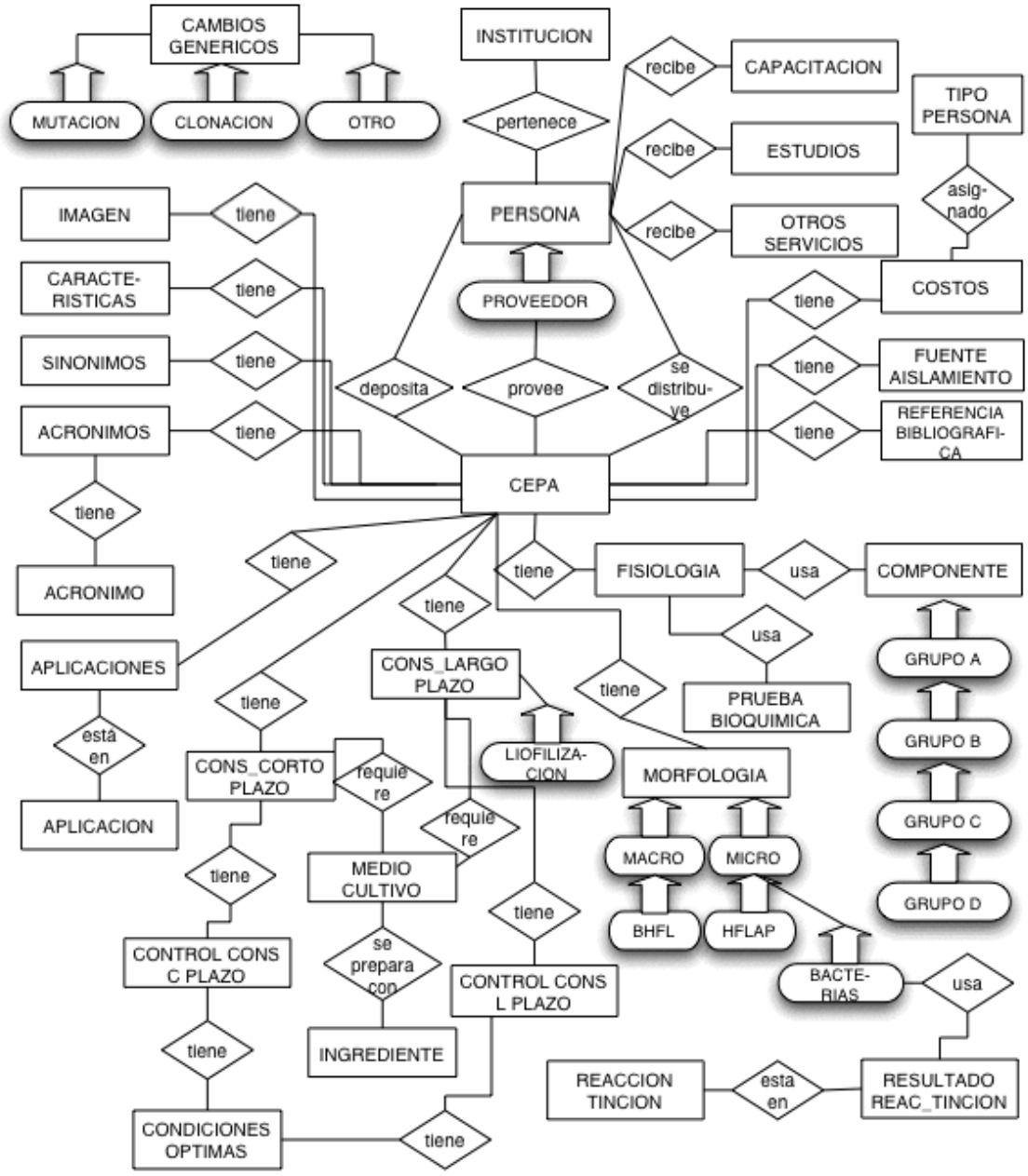


Figura 5.1: Modelo conceptual de la base de datos CDBB-500

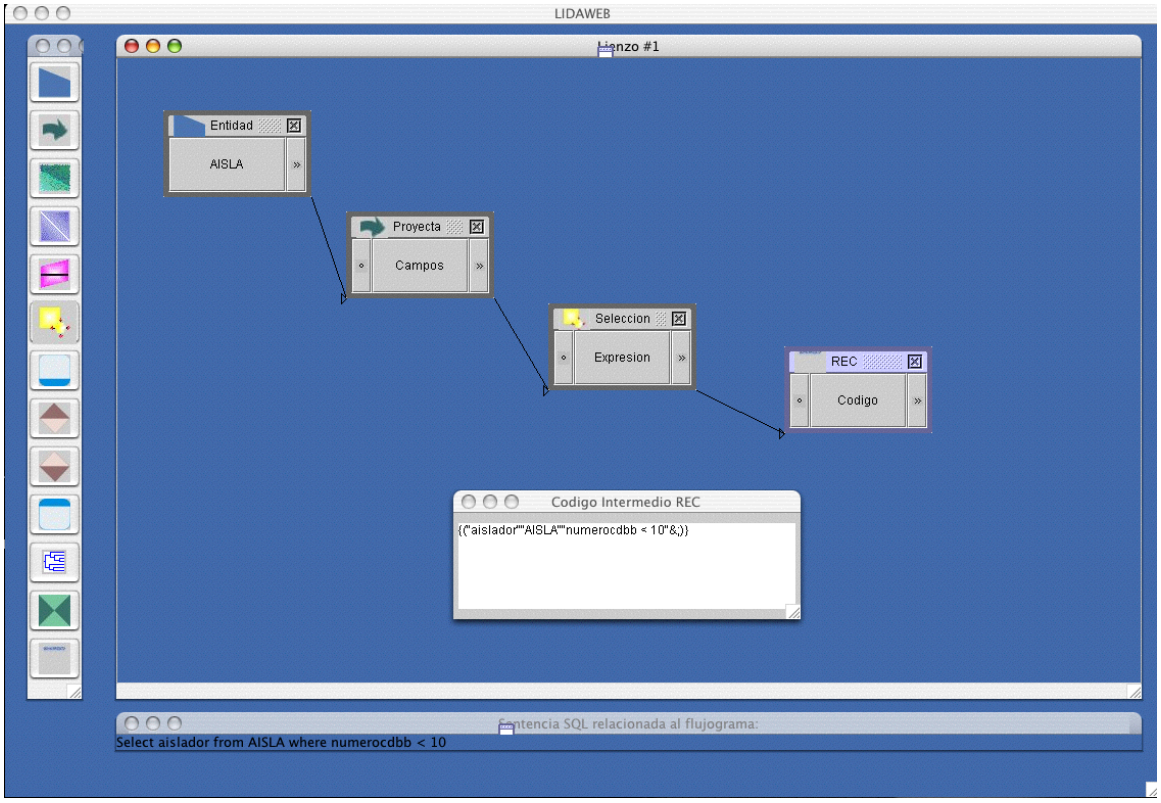


Figura 5.2: flujograma en LIDA/REC

lo dice, se especifica el nombre de la entidad o tabla. Enseguida se selecciona el ícono llamado *Proyecta* y se conecta al de *Entidad*. Con este ícono se especifica el nombre del campo. Enseguida se puede conectar el ícono llamado *Condición*. Al final, se conecta el ícono llamado *Código REC*; el cual despliega, el código REC equivalente a la consulta especificada por el flujograma.

Explicado lo anterior, ahora se prosigue a mostrar algunos ejemplos realizados a la base de datos CDBB-500 de microbiología.

5.2.1. Ejemplo No. 1

La Consulta más sencilla que se puede realizar; es donde solamente se específica la tabla. Esto se hace cuando queremos seleccionar todos los registros de todos los campos que se encuentran en la tabla $ACRONI$. Lo anterior se expresa en un sentencia SQL de la forma que sigue:

Figura 5.3: paleta

Select * *from* ACRONI

Para crear la consulta en LIDA/REC; primero se crea un ícono *Entidad* y se selecciona la tabla *ACRONI*, de la lista de tablas del botón *Descriptores*. Después se crea el ícono *Codigo REC* y se conecta al de entidad. En la figura 5.5 se ve el flujograma en LIDA/REC y la ventana del código REC generado.

El código REC que se generó, también se guardó en un archivo (codigo.txt). Este Archivo es leído por el compilador de REC para su interpretación. Para ejecutar el compilador de REC se hace lo siguiente: dentro del directorio RECL se ejecuta *./mainC*. Esto se muestra en la figura 5.6.

Al ejecutar el compilador, este interpreta el código REC del archivo de texto y guarda el resultado de la consulta en un archivo de texto. En la figura 5.7, se muestra el resultado de la consulta a la tabla *ACRONI*.

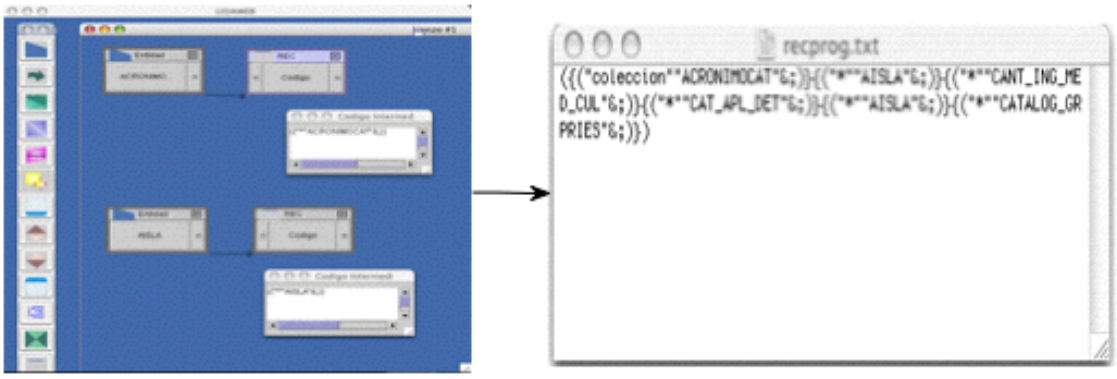


Figura 5.4: flujograma y código REC resultante

5.2.2. Ejemplo No. 2

En este ejemplo se desea seleccionar algunos de los campos que se encuentran en la tabla *ACRONI*. Lo anterior se expresa en un sentencia *SQL* de la forma que sigue:

Select numerocdbb, coleccion *from* ACRONI

Para crear la consulta en LIDA/REC; primero se crea un ícono *Entidad* y se selecciona la tabla *ACRONI*, de la lista de tablas del botón Descriptores. Enseguida se crea el ícono *Proyecta* y se seleccionan los campos: *numerocdbb* y *coleccion*. Después se crea el ícono *Codigo REC* y se conecta al de *Proyecta*. En la figura 5.8 se ve el flujograma en LIDA/REC y la ventana del código REC generado.

El código REC generado, se guardó en un archivo (codigo.txt). Este Archivo es leído por el compilador de REC para su interpretación. Se ejecuta el compilador de REC como se hizo anteriormente en el ejemplo 1. El resultado de la consulta es el archivo de texto que se ve en la figura 5.9.

5.2.3. Ejemplo No. 3

Con este ejemplo se muestra una consulta que contiene un *Select*, con una condición. Se desea seleccionar los campos *clavemedio* y *elemento* de la tabla *CANT_ING_MED_CUL*; donde *clavemedio* = 2. La sentencia *SQL* que expresa esta consulta es la siguiente:

Select clavemedio, elemento *from* CANT_ING_MED_CUL *where* clavemedio = 2

Para crear el flujograma se hace lo mismo que en los dos ejemplos anteriores; pero

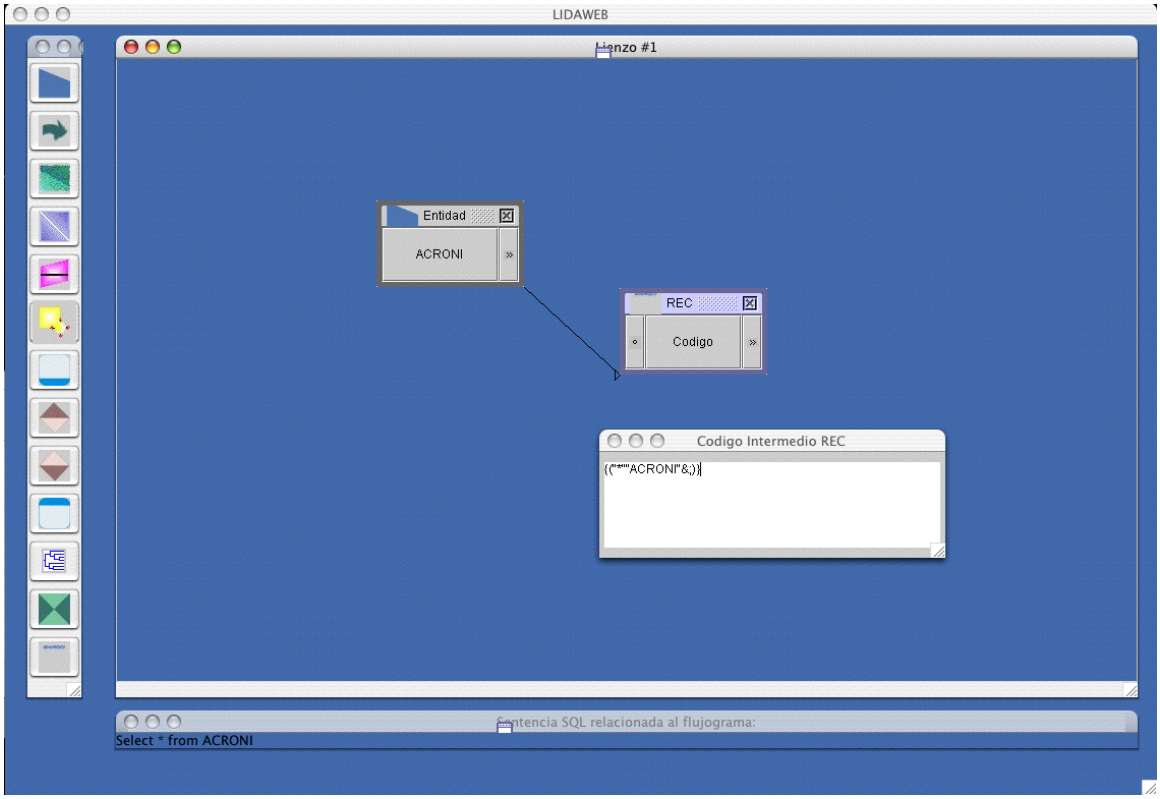


Figura 5.5: Ejemplo 1. Flujograma asociado a la consulta

además se conecta el ícono *selección*. En este ícono se permite escribir la condición mediante una área de texto. En la figura 5.10 se aprecia el flujograma y el código en REC que se generó.

Por último en la figura 5.11 se muestra el archivo de texto que contiene el resultado de la consulta.

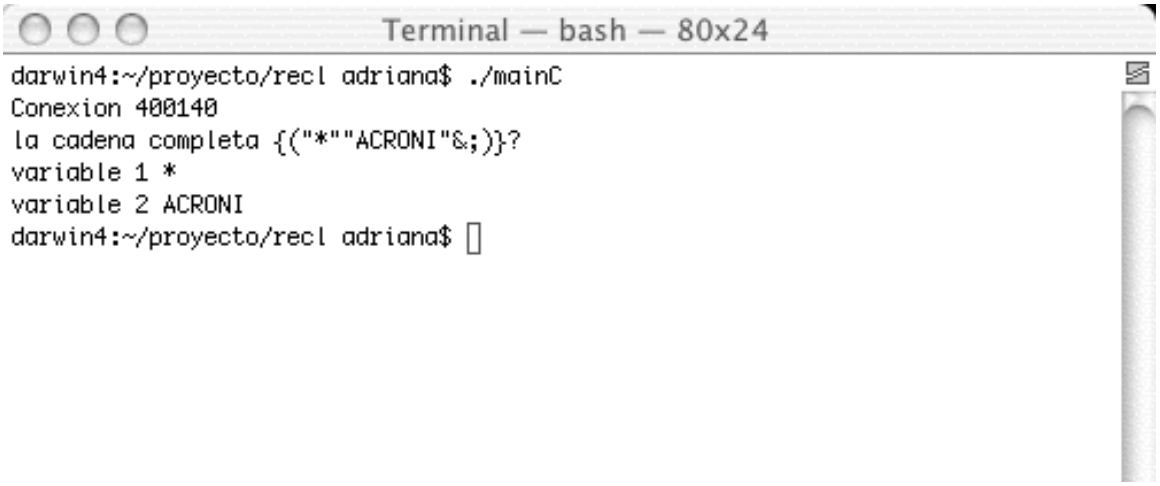


Figura 5.6: ejecutando el compilador REC en la terminal.

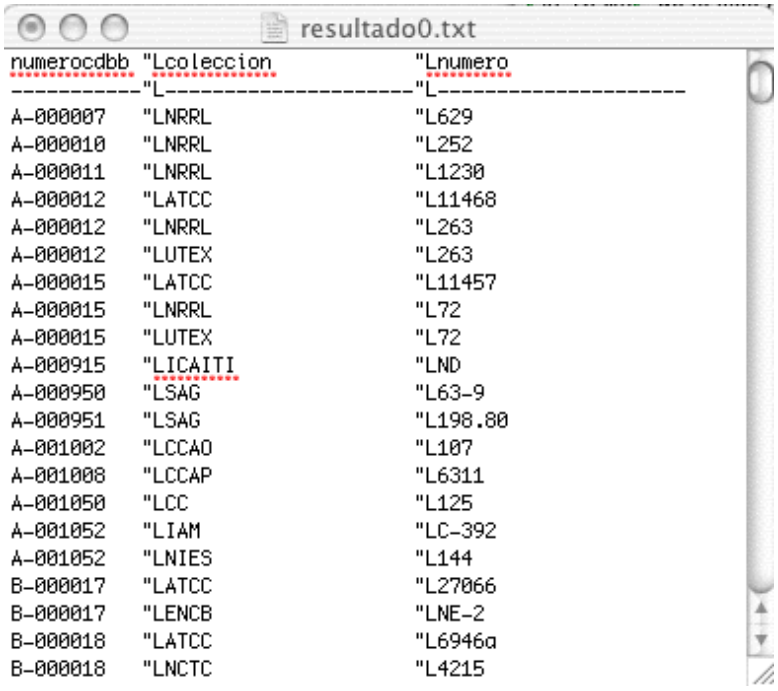


Figura 5.7: Ejemplo 1. Archivo que contiene el resultado de la consulta.

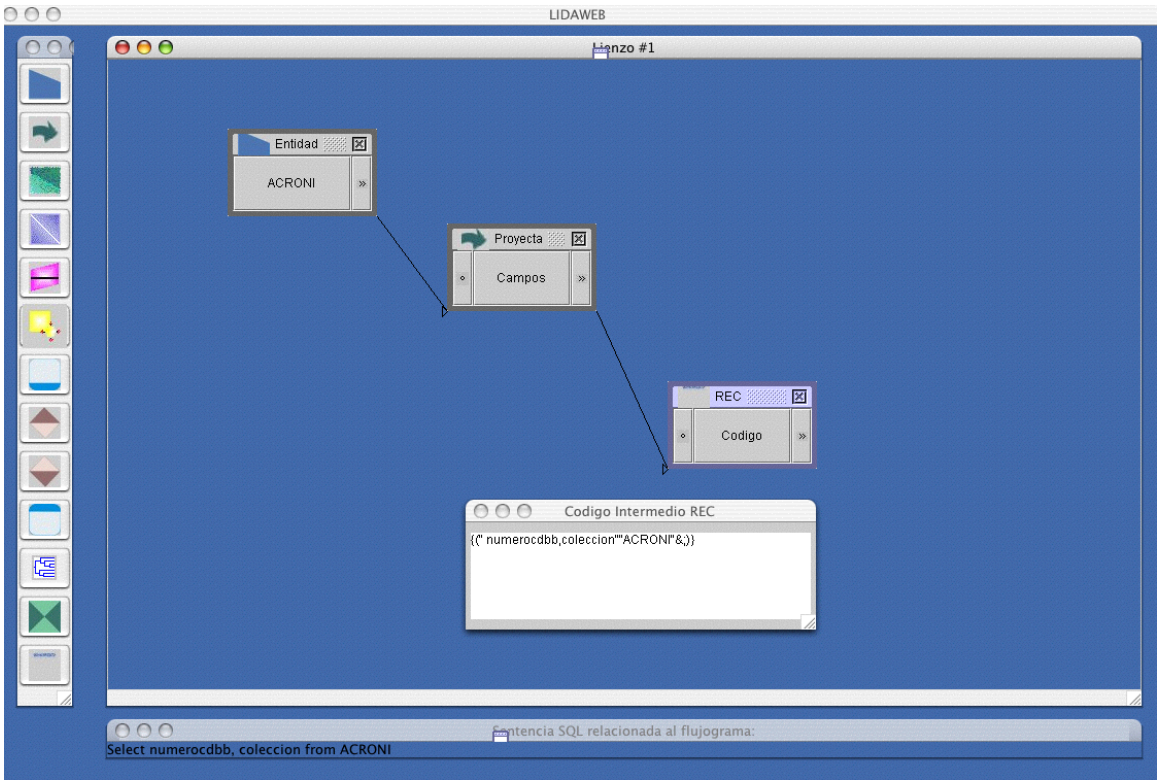


Figura 5.8: Ejemplo 2. Flujograma asociado a la consulta.

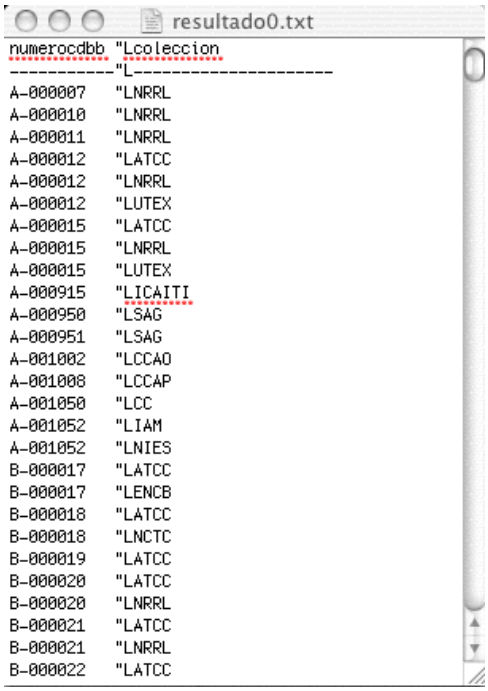


Figura 5.9: Ejemplo 2. Archivo que contiene el resultado de la consulta.

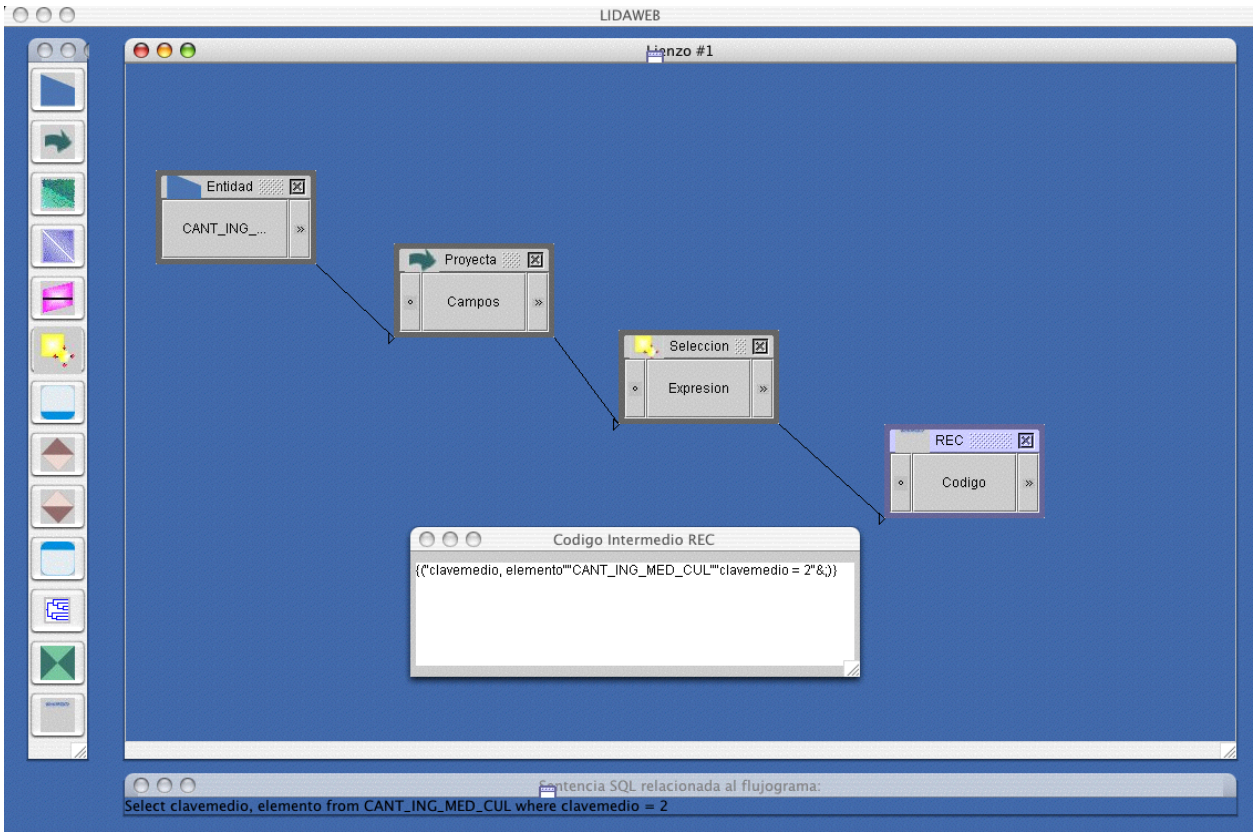


Figura 5.10: Ejemplo 3. Flujograma asociado a la consulta.

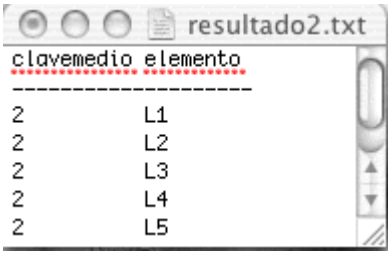


Figura 5.11: Ejemplo 3. Archivo que contiene el resultado de la consulta.

CONCLUSIONES

El objetivo de este proyecto fue implementar el sistema LIDA/REC. El cual esta conformado por dos módulos. El primer módulo es la interfaz visual; mediante la cual se crean los flujogramas. Dicho módulo fue implementado usando el lenguaje Java. El segundo módulo, está constituido por el compilador REC el cual fue configurado para interactuar con una BD en PostgreSQL. El compilador fue implementado en lenguaje C.

Los resultados obtenidos durante el desarrollo de el proyecto de LIDA/REC serán publicados en el 2nd International Conference on Electrical and Electronics Engineering and XI Conference on Electrical Engineering ICEEE-CIE 2005 a llevarse al cabo del 7 al 9 de Septiembre.

Con la implementación de LIDA/REC se obtuvo la ventaja de contar con una interfaz visual para el programador. Mediante esta interfaz que es un lenguaje visual basado en flujogramas, se susituye la programación de líneas de código por representaciones gráficas más comprensibles por el usuario. Esto evita errores de sintaxis y facilita el desarrollo del sistema.

La funcionalidad del sistema, está enfocada a la creación de consultas para BD; es decir se crea un flujograma con base a la representación gráfica de la consulta deseada.

Otra ventaja, es con base a la propiedad de generación de código REC. Este código se utiliza para ejecución en mildware; esto es, que la ejecución de las consultas se puede delegar al servidor y dejar para el cliente únicamente la ejecución de la interfaz visual. El código intermedio se puede ejecutar en paralelo, si se cuenta con los medios tecnológicos necesarios. Esto se hace para optimizar la ejecución.

Trabajo futuro

Se puede anexar a la configuración de REC, operadores aritméticos para que se puedan hacer operaciones con los resultados de las consultas, de tal forma que podría servir, para cualquier tipo de aplicaión.

También se puede hacer la división del código REC para distribuirlo en varios procesadores y optimizar el tiempo de ejecución. Ya que se ejecutaría en paralelo.

Bibliografía

[1] P. Bottoni, Formalising Visual Languages, http://Kogs25.informatik.uni-hamburg.del haarslev/v195www/talks. 1995

[2] M. López, A. Martínez, R. Santaolaya, O. Fragoso. Especificación formal de un Lenguaje de Programación Visual para Desarrollo de Diseño Detallado de WARNIER, *CIMAF'*97, La Habana, Cuba, p.p. 106-114

[3] Jesús Vegas, "Introducción a las Bases de Datos", Departamento de Informática Universidad de Valladolid, Valladolid 1998, http://descartes.dcs.fi.uva.es/ jvegas.

[4] M. López Sánchez, Lenguaje Visual para el modelado de sistemas en tiempo real. Tesis de Doctorado, CIC-IPN México D.F. Mayo 2003.

[5] E. William, Abstraction and Organization in signs and Sign Systems, Cognitive Science Research Centre, School of Computer Science, The University of Birmingham, Edgu/baston, B15 2TT, West Midlands, Britain. 1995

[6] R. Pressman, *Ingeniería del Software*, *un Enfoque Práctico*. McGraw-Hill, 1998. Cuarta Edición.

[7] G. Costagliola, A. DeLucia, S. Orefice, G. Tortora. A parsing Methodology for the implementation of Visual System. *IEEE Transactions on Software Engineering*, vol. 23, No.12, p.p 777-799, 1997

[8] M. Stuart, Y. Masoud, A computer-based Iconoc Language, Departament of computer Science, University of Exeter-England. 1990

[9] M. Burnett, Baker y McIntyre, "Visual Programming, *Computer*, vol. 28, No. 3, p.p 14-16. 1995

[10] J. Kiper, E. Howard, Ch. Ames, Criteria for Evaluation of Visual Programming Languages, *Journal of Visual Languages and Computing*, vol.8, No. 2, p.p175-192. 1997

[11] M. Bunett, M. J. Baker, A classification System for Visual Programming Language, Departamento de Ciencias Computacionales de la Universidad del Estado de Oregon, Corvallis Oregon, U.S.A., *Journal of Visual Languages and Computing*, p.p. 287-300. 1994

[12] J. Gard Clark, C. Thomas Wu, DFQL : Dataflow Query Language for Relational Databases. U.S. Navy, Naval Postgraduate School, Departament of Computer Science, Code CS Monterey, CA p.p. 3-6.

[13] S. Dogru, V. Rajan, K. Rieck, J. R. Slagle, B. S. Tjan, Y. Wang, A Graphical Data Flow Language for Retrieval, Analysis, and Visualization of a Scientific Database, *Journal of Visual Languages and Computing* p.p 247-265, Computer Science Departament, University of Minnesota, Minneapolis Visualization of Scientific Databases. 1996

[14] S. V. Chapa Vergara, Programación automática a partir de descriptores de flujo de información. Tesis de doctorado presentada en la Sección de Computación del departamento de Ingeniería Eléctrica del Centro de Investigación y Estudios Avanzados del IPN. México D.F. Marzo 1991

[15] N. Soto Godínez, Interoperabilidad en Bases de Datos a través del Sistema Visual LIDAWEB. Tesis de licenciatura presentada en la Universidad Nacional Autónoma de México campus Acatlán, Marzo 2002.

[16] A. Silberschatz, H. F. Korth, S. Sudarshan. *Fundamentos de Bases de Datos*. McGraw-Hill, 1998. Tercera Edición

[17] H. V. McIntosh, G. Cisneros, The programming languages REC and Convert, *SIGPLAN Notices* 25, 7, p.p. 81-94. July 1990

[18] G. Cisneros, Configurable REC. SIGPLAN Notices 29, 5, p.p. 7-16. May 1994

[19] M. Margaret, B. A. Goldberg, T. G. Lewis, *Visual Object−Oriented Programming*. Prentice Hall. p.p. 46-65

[20] J. C. Worsley, J. D. Drake, *Practical PostgreSQL*. (O' Reilly Unix) Command Prompt Inc. January 2002.

[21] M. Fowler, K. Scott. *UML gota a gota*. Prentice Hall. 1997

[22] C. Larman. *UML y Patrones. Introducción al análisis y diseño orientado a objetos*. Prentice Hall. 2002.