Diseño e implantación de un Optimizador de Consultas para un Sistema Manejador de Bases de Datos orientado por Objetos MIDBS

Domingo Hernández Hernández
Universidad de Los Andes, Facultad de Ingeniería, Escuela de Ingeniería de Sistemas, Departamento de Computación, Grupo de Investigación en Ingeniería de Datos y Conocimientos (GIDyC)
e-mail: dhh@ing.ula.ve
Mérida 5101 - Venezuela.

Resumen

En este artículo se presenta una propuesta para la arquitectura del optimizador de consultas que será empleado en el manejador de bases de datos orientada por objetos MIDBS [MONTILVA 93]. Este optimizador acepta una consulta expresada en un lenguaje de alto nivel no procedimental basado en el cálculo de predicados orientado por objetos (CPOO) presentado en [BERTINO 92]. Es importante destacar que para expresar una consulta en este lenguaje se requiere que el usuario conozca el esquema de la Base de Datos Orientada por Objetos (BDOO) sobre el cual se raealizará la consulta.

Una vez que el usuario ha realizado la consulta, el optimizador de consultas se encarga en la primera fase de verificar tanto de que existan las clases a las cuales pertenecen los objetos solicitados, como la no violación de las reglas de integridad semánticas asociadas a cada clase. En la segunda fase se elabora un conjunto de planes de ejecución capaces de dar respuesta a la interrogación propuesta por el usuario y en la tercera y última fase, se selecciona cuál es el plan de ejecución de menor costo; esto se realiza por medio del cálculo del orden de magnitud de cada uno de los planes propuestos en la segunda fase. Una característica importante de este optimizador de consultas es la capacidad que tiene de dar respuesta a algunas de las consultas realizadas por el usuario sin acceder a los objetos, sino a través de la evaluación del conjunto de reglas de integridad semánticas asociadas a cada clase.

Palabras claves: Bases de Datos Orientadas por Objetos. Optimizador de consultas.

1. Introducción

En este artículo se presenta la arquitectura del optimizador de consultas utilizado por el Sistema Manejador de Bases de Datos Orientado por Objetos (SMBDOO) llamado MIDBS, propuesto en [MONTILVA 93], el se basa en un modelo de datos y conocimiento denominado "D/K Model" [MONTILVA 93]. Sin embargo, es importante destacar que la arquitectura propuesta aquí también puede ser utilizada en cualquier otro SMBDOO siempre y cuando este utilice un lenguaje de consulta basado en el Cálculo de Predicado Orientado por Objeto (CPOO) propuesto en [BERTINO 92].

La entrada al optimizador de consulta es una interrogación realizada por el usuario en un lenguaje no procedimental de alto nivel basado en el CPOO antes mencionado. La consulta pasa por una serie de módulos en las que se verifica tanto la sintaxis como la semántica de la consulta. Una vez verificada la consulta, ésta se pasa a un proceso para determinar si satisface las reglas de integridad semánticas asociadas a las clases que se involucran en la consulta. El paso siguiente en la arquitectura del optimizador es generar el árbol de la consulta del usuario para identificar las distintas clases, atributos y operaciones necesarias para obtener la respuesta del usuario. Al finalizar la construcción del árbol de consulta se pasa al módulo que realiza la vinculación entre el árbol de consulta y el esquema físico de la BDOO con la finalidad de obtener el grafo de conexión, el cual permite estudiar la factibilidad de intercambiar las operaciones de juntura sobre las clases atómicas. Una vez generado el grafo de conexión se pasa al módulo que permite la generación de los distintos árboles de procesamientos con los que se les puede dar respuesta al usuario. Cuando el optimizador de consulta posee la información de que la lista de árboles de procesamiento ya están listo, éste le pasa el control al módulo de evaluación de la lista de árboles de procesamiento, el cual se encarga de aplicar las funciones de costo preestablecidas para cada operación. Estas funciones de costos estan basadas en la cardinalidad de las clases involucradas en la consulta. El costo total de procesmiento de cada árbol es la suma de los costos incurridos por evaluar cada nodo del árbol. Una vez que se han detrminado todos los costos, se

debe seleccionar aquel árbol que incurra en el menor costo de procesamiento, el árbol seleccionado se pasa al siguiente módulo del optimizador el cual se encarga de ejecutar el plan de procesamiento. El módulo de ejecución del plan transforma el árbol de procesamiento óptimo en un programa ejecutable que permite realizar las operaciones e iteractuar con el manejador de objetos del MIDBS.

2. Arquitectura del Optimizador de consultas para MIDBS

Los conceptos y paradigmas que envuelven tanto a la Programación Orientada por Objetos (POO) como el diseño de BDOO permiten al usuario modelar el mundo real, sin las restricciones a las que es sometido cuando desea realizar una aplicación utilizando el modelo relacional. Ahora bien, este poder de expresividad del modelo orientado por objetos, trae inmerso el problema de que al iniciar el proceso de optimizar una consulta, ésta puede involucrar: objetos complejos, atributos heredados de una clase que se encuentra en un nivel superior en la jerarquía de clases, atributos que se definen de manera recursiva como objetos de la misma clase a la cual pertenecen como atributos [HUGHES 91], y por último, para el caso de MIDBS el usuario puede realizar consultas en la que se involucran objetos espaciales. Por esta razón es necesario desarrollar un optimizador de consultas que sea capaz de manipular todas las características anteriormente mencionadas.

En la figura 1 se muestra la arquitectura del optimizador de consultas propuesto para MIDBS, la cual esta basada en un modelo de árbol para representar las consultas de los usuarios, un modelo de grafos de agregación para representar el esquema conceptual de la BDOO y un modelo de árbol para representar el espacio de búsqueda, en términos de un conjunto de planes de ejecución capaces de dar respuesta al usuario.

En las secciones siguientes se presenta un estudio detallado de cada uno de los módulos que conforman dicha arquitectura.

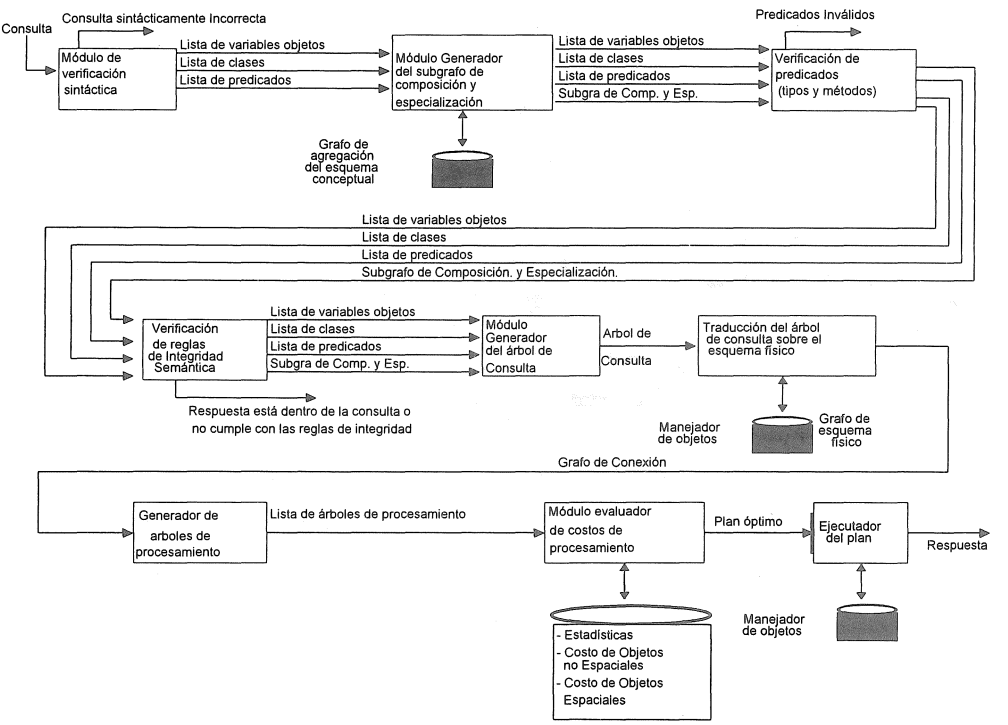

Fig.1 Arquitectura del Optimizador de consulta propuesto para MIDBS

Para ilustrar el proceso a realizar en cada uno de estos módulos, se usará el esquema conceptual de la BDOO presentado en la figura 2, y sobre él se realizarán algunas consultas con la finalidad de observar el comportamiento del optimizador de consultas de MIDBS

2.1 Módulo de verificación sintáctica

La consulta expresada por el usuario se almacena en una cadena a fin de ser procesada por el módulo de verificación sintáctica. Este módulo se encarga de dividir la cadena en tres grandes subcadenas llamadas cláusula destino, cláusula rango y cláusula cualificación. Esta división es fácil de obtener debido a que en las consultas expresadas en CPOO se emplean el delimitador punto y coma (;) para separar cada cláusula. Debe recordarse que una consulta expresada en CPOO tiene que poseer de manera obligatoria tanto la cláusula destino como la cláusula rango, mientras que la cláusula cualificación es opcional.

El módulo de verificación sintáctica recibe la consulta del usuario y la divide en subcadenas, cada una de las cuales se pasa respectivamente a los siguientes submódulos :

1. Verif_Sintac_clau_Dest: Este módulo recibe como parámetro de entrada la subcadena que contiene la cláusula destino de la consulta expresada en CPOO. Si la sintaxis está correcta este módulo devuelve una lista con los nombres de las variables objetos y los atributos requeridos por el usuario en la consulta.

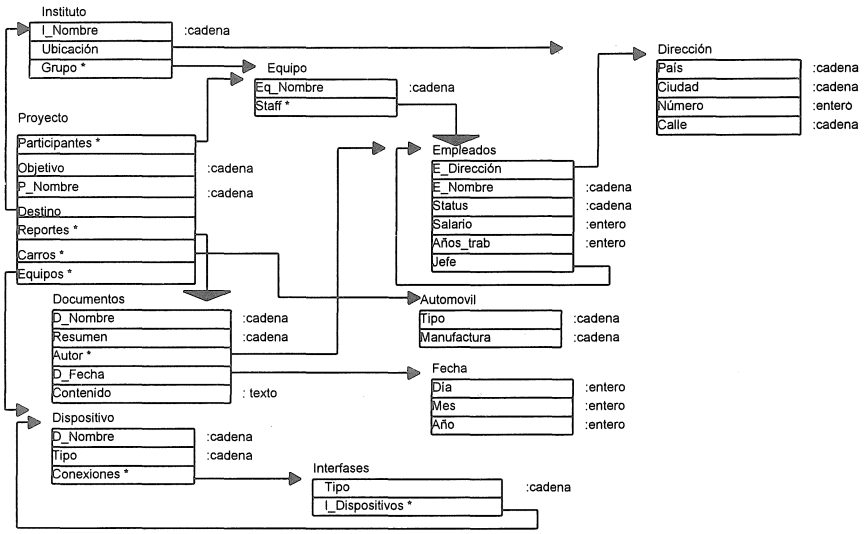

Fig. 2 Esquema de una base de datos orientada por objetos.

En el análisis sintáctico de la subcadena llamada cláusula destino se deben considerar los casos siguientes como correctos:

- Pueden existir variables objetos sin atributos asociados. Esto significa que todos los atributos de las clases asociadas a estas variables son requeridos por el usuario.
- El caracter punto (.) de la función punto establece la relación existente entre una variable objeto y el atributo que se desea recuperar.
- En la cláusula destino pueden existir una o más variables objetos asociadas a distintas clases que pueden referenciar a los respectivos atributos de las clases asociadas.

A medida que se realiza el análisis sintáctico de la subcadena llamada cláusula destino, se van insertando en los nodos de la lista llamada *Lista_Variables_objetos* las subcadenas formadas por las variables objetos y sus atributos referenciados. Los nodos de esta lista poseen el formato mostrado en la figura 3. Es importante destacar que sí al realizar el análisis sintáctico de esta subcadena existe algún error, la lista

Lista_Variables_Objetos debe salir vacía y esto le indicará al módulo de verificación de la sintaxis que la consulta realizada es incorrecta sintácticamente.

Variable_objeto

Nombre_variable	Nombre_Atributo	Punt_siguiente

Fig. 3 Formato del nodo de la lista Lista_Variable_Objetos

2. Verif_Sintac_Clau_Ran: Este módulo recibe como parámetro de entrada la subcadena que contiene la cláusula rango de la consulta expresada en CPOO. Si la sintaxis está correcta, este módulo devuelve una lista con los nombre de las variables objetos y sus respectivas clases asociadas.

   En el estudio de la sintaxis de la subcadena llamada cláusula rango se consideran los siguientes casos como sintácticamente correctos:

- Cada una de las variables objetos referenciadas en la cláusula destino deben estar asociadas a una clase.
- El caracter (/) establece la relación entre la variable objeto y la clase.
- Se utiliza el delimitador coma (,) para separar cada par formado por la variable objeto y su respectiva clase.

Del mismo modo que en el módulo anterior, a medida que se realiza el análisis sintáctico de la subcadena llamada cláusula rango se van insertando dichas subcadenas formadas las variables y las clases asociadas en los nodos de la lista llamada *Lista_de_Clases*. Los nodos de esta lista poseen el formato que se muestra en la figura 4. Sí al realizar el análisis sintáctico de esta subcadena existiese algún error la lista Lista_Clases sale vacía.

Nom_clase

Nombre_variable	Nombre_clase	Punt_siguiente

Fig. 4 Formato del nodo de la lista Lista_Clase

3. Verif_Sintac_Clau_Cual: Este módulo recibe como parámetro de entrada la subcadena que contiene la cláusula cualificación de la consulta expresada en CPOO. Sí la sintaxis está correcta, este módulo devuelve una lista llamada *Lista_Predicados* con los distintos predicados asociados a la consulta del usuario. Los nodos de esta lista poseen el formato que se ilustra en la figura 5.

Predicado

Predicado	operador	Punt_predicado_asociado

Fig. 5 Formato del nodo de la lista Lista_Predicados

La subcadena llamada cláusula cualificación contiene los predicados que deben satisfacer los objetos que se deben recuperar de la base de datos.

Una vez concluida la verificación sintáctica pueden ocurrir uno de los siguientes resultados:

1. La consulta es sintácticamente incorrecta, en cuyo caso se termina el proceso de optimización enviándole un mensaje al usuario para informarle del problema. Esto se detecta si al menos una de las listas que debe salir de este módulo está vacía.
2. La consulta es sintácticamente correcta lo que permite seguir el proceso de optimización.

2.2 Módulo generador del subgrafo de composición y jerarquías de clases

El esquema de una BDOO puede representarse a través de dos grafos. El grafo de composición de objetos en el cual se representan los objetos compuestos, y el grafo de la jerarquía de clases. Estos dos grafos deben estar a disposición del optimizador de consulta con la finalidad de que este conozca las posibilidades de navegación a través de los objetos del sistema, ya sean clases compuestas o jerarquías de clases. Ahora bién, cada vez que el usuario realiza una consulta a la BDOO, el optimizador requiere que estos dos grafos se encuentren en la memoria principal. Sin embargo, en lugar de llevar a la memoria principal todas las clases asociadas a estos dos grafos, se debe generar un subgrafo que solo incluya las clases involucradas en la consulta del usuario, esto debe hacerce con el fin de ahorrar memoria principal y hacer mucho más rápido el recorrido por el grafo. Este subgrafo debe ser capaz de englobar tanto las clases compuestas, las clases

generales y sus respectivas especializaciones. Con este enfoque se evita tener grupos de clases ocupando memoria principal sin que nunca el usuario las consulte. Otra ventaja es que se tiene en un mismo grafo tanto el grafo de composición como el grafo de las jerarquías de clases. La combinación de estos dos tipos de grafos se logra mediante la manipulación de dos tipos de enlaces entre las clases del sistema, un enlace para establecer la composición de clases y otro enlace para establecer la jerarquía de clases. Para visualizar como se genera el subgrafo de la consulta considere el grafo del esquema de la base de datos propuesto en la figura 2 y la consulta siguiente: "Recupere los nombres de aquellos empleados que viven en la calle cuyo nombre es Miranda". Esta consulta expresada en CPOO posee la forma siguiente:

{ x.E_Nombre; x / Empleado; x.E_Dirección.Calle = "Miranda"}

Una vez realizada la verificación de la sintaxis de la consulta, el módulo de generación del subgrafo de la consulta recorre la lista de clases recibida del módulo anterior y busca cada una de estas clases, tanto en el grafo de composición de clases como en el grafo de jerarquía de clases. Sí la clase se encuentra en el grafo de jerarquías de clases, se extrae de este, un subgrafo con todas las clases relacionadas con la clase involucrada en la consulta. Sobre este subgrafo se verifica si éste posee clases compuestas y si ello es cierto, se va al grafo de composición y se extrae el subgrafo con las clases involucradas. Una vez terminado este proceso, se lleva cada una de estas clases al subgrafo de la consulta, previa verificación de que este subgrafo no esté ya contenido en el subgrafo de la consulta. Sí la clase que se está procesando no se encuentra en el grafo de jerarquía de clases, entonces la búsqueda se realiza sobre el grafo de composición y se realiza el mismo proceso. Aplicando lo anteriormente expuesto, el subgrafo de consulta para la interrogación antes realizada se presenta en la figura 6.

2.3.1 Implantación del grafo de la consulta

Como se ha observado en la sección anterior, el grafo de la consulta se implanta como un grafo dirigido donde los nodos representan las clases involucradas en la consulta, los enlaces entre nodos representan la relación de composición o de jerarquía. Para implantar este grafo de conexión se ha decidido emplear un grafo diseñado como una lista de listas tal y como puede observarse en la figura 7.

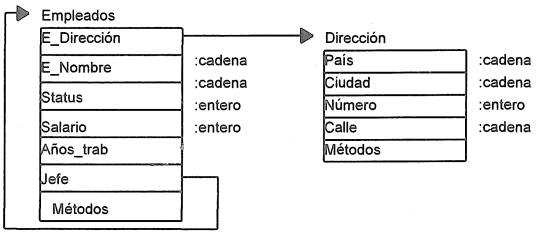

Fig. 6 Subgrafo de la consulta

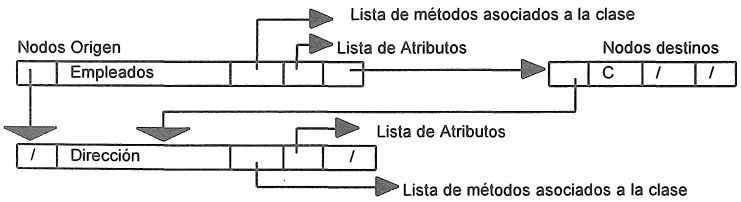

Fig. 7 Implantación del grafo de la consulta

El formato del nodo origen del grafo de la consulta se muestra en la figura 8

Nodo_Origen

Nodo_Sig.	Nombre_clase	Punt_lis_met	Punt_lis_atr	Enlace

Fig. 8 Formato del Nodo Origen.

Donde:
Nodo_Sig. : puntero a la siguiente clase involucrada en la consulta.
Nombre_clase : Nombre de la clase involucrada.
Punt_lis_met: Puntero a la lista de métodos asociados a la clase.
Punt_list_atr: Puntero a la lista de atributos, con sus respectivos tipos.
Enlace: Puntero al nodo donde se almacena el tipo de enlace que existe entre esta clase y la clase asociada.
El formato del nodo enlace es presentado en la figura 9

Nodo_Enlace

Nodo_Relacionado	Tipo_Relación	Pun_Lis_Com	Nodo_Siguiente

Fig. 9 Formato del Nodo Enlace.

Donde:
Nodo_Relacionado: es el nodo con el cual se enlaza el nodo origen.
Tipo_Relación: es un campo que puede tomar los dos valores siguientes:
H si la clase relacionada es hija de la clases almacenada en el nodo origen.C si la clase relacionada es una clase que compone la clase origen.
Punt_list_Com: indica la lista de clases que componen a la clase origen, si este campo es nulo, solo la clase apuntada por el nodo relacionado compone la clase origen.
Nodo_siguiente: puntero a la siguiente clase con la que la clase origen mantiene relación.

3. Módulo de verificación de predicados (tipos y métodos)

Este módulo se encarga de verificar los tipos y métodos asociados a los predicados que forman la cláusula de cualificación del CPOO. Para lograr ello, el módulo hace uso del subgrafo de la consulta y de las listas (*listas de variables objetos, listas de clases, listas de predicados de cualificación*) generadas en el módulo de verificación sintáctica. Estas listas poseen el formato expuesto en las sección 2.1.

Cuando un usuario desea realizar consultas en CPOO, debe asociar una variable objeto a una clase. Ahora bién, este tipo de subcadena está almacenada en la lista llamada lista de clases, lo que permite iniciar la búsqueda de las clases en el subgrafo de la consulta. Es importante destacar, que en el subgrafo de la consulta se encuentra almacenada la información referente a los tipos de los atributos que forman la clase, así como los métodos asociados a cada clase. Con esta información el módulo de verificación de predicados toma cada uno de los predicados almacenados en la lista de predicados, para verificar si el tipo de los operandos se corresponde con el tipo declarado en el esquema del subgrafo de la consulta. Además, se verifica que el predicado formado por operando, operador, operando sea semánticamente correcto, es decir, que los tipos de los dos operandos puedan ser manipulados al mismo tiempo por el operador que los relaciona. También es importante recordar que un operando puede expresarse por medio de la función punto como la vinculación de una variable objeto con un método; en este caso el módulo de verificación de predicados busca la clase asociada a la variable objeto para verificar si el método asociado a la variable objeto pertenece a esta clase. Si en la verificación tanto de los tipos como de los métodos existiese algún error, el módulo de verificación de predicados suspende la ejecución del proceso de optimización enviándole un mensaje de error al usuario, en caso contrario se sigue con el proceso.

4. Módulo de verificación de reglas de integridad semántica

En el modelo D/K cuando el usuario especifica una clase, debe definir si dicha clase posee restricciones de integridad semántica sobre sus instancias. Como ya es conocido las reglas de integridad semántica se emplean para darle un mayor sentido lógico a los datos almacenados en la base de datos y de esta manera mantener la consistencia de los datos. Sin embargo la mayoría de los MBDR y MBDOO no manejan reglas de integridad semánticas directamente en el esquema conceptual de la base de datos. Ellos implantan el uso de algunas reglas de integridad semántica por medio de programas o métodos comúnmente llamados

disparadores ("trigers") que se ejecutan cuando la consulta involucra un atributo asociado a este disparador. Esto no permite un verdadero modelaje de las reglas de integridad en forma completa.

El modelo de datos D/K del MIDBS si permite la definción de reglas semánticas, lo que implica un ahorro de tiempo de procesamiento de una consulta, ya que si ésta no cumple con las reglas de integridad, el proceso de optimización se cancela. También puede suceder que la respuesta a la consulta se encuentre dentro del conjunto de reglas de integridad semánticas asociadas a la clases involucradas en la consulta del usuario. En cualquiera de estos dos casos, el optimizador de consulta puede dar respuesta al usuario sin acceder a las instancias de las clases almacenadas en la base de datos. Para ilustrar esto considérese el esquema presentado en la figura 2 y asúmase que la clase Empleado posee las siguientes reglas de integridad semánticas:

1. $\forall$ x $\in$ Empleado $\wedge$ x.Status ="Pasante de Investigación" $\Rightarrow$ x.Salario = 10.000
2. $\forall$ x $\in$ Empleado $\wedge$ x.Años_trab > 6 $\wedge$ x.salario > 100.0000 $\Rightarrow$ x.Status = "Investigador"

Supóngase que se realiza la consulta siguiente: "Recupere los nombres de aquellos empleados cuyo status sea pasante de investigación y cuyo salario sea mayor a 10.000 Bs.". Esta consulta expresada en CPOO posee la forma siguiente:

{x.E_Nombre; x / Empleado; x.Status = "Pasante de Investigación" $\wedge$ x.Salario >10.000}

Cuando el optimizador de consulta inicia el proceso en el que se ejecuta el módulo de verificación de reglas de integridad semántica, se da cuenta que la clase Empleado posee dos reglas de integridad semántica; por lo que debe verificar si los atributos involucrados en el predicado de la consulta satisfacen dichas reglas de integridad. Así el módulo de verificación se da cuenta que el predicado involucrado en dicha consulta no cumple con la primera regla de integridad, por lo tanto se cancela el proceso y se le envía el mensaje al usuario. Ahora considérese la consulta siguiente: "Recupere el status de aquellos empleados cuyo salario sea mayor de 100.000 Bs. y los años trabajados sean mayor o igual a 7 ".Esta consulta en CPOO posee la forma siguiente:

{x.Status; x / Empleado; x.Salario > 100.000 $\wedge$ x.Años_trab >7}

Cuando esta consulta llega al módulo de verificación de las reglas de integridad semántica, este se entera que la respuesta a la consulta se encuentra dentro de la regla de integridad semántica por lo tanto el optimizador de consulta puede dar respuesta automáticamente.

De los casos anteriormente expuestos se observa que si el esquema conceptual de la BDOO puede contener reglas de integridad asociadas a las clases y está disponible al optimizador de consulta, entonces éste puede dar respuesta a cierto tipo de consultasd del usuario sin tener que acceder a los datos almacenados en la base de datos.

5. Módulo generador del árbol de la consulta

Una vez verificada la sintaxis y la semántica de la consulta del usuario, se pasa al módulo generador del árbol de la consulta. Este árbol posee tres tipos de nodos, el nodo raíz, el nodo predicado, donde se guarda el conjunto de predicados propuestos en la cláusula de calificación del CPOO, y los nodos hijos que representan las clases involucradas en la consulta. El nodo raíz contiene las variables y los atributos asociados requeridos por el usuario. El nodo predicado almacena un conjunto de predicados constituidos por la cláusula de cualificación del CPOO. Los nodos hijos son subárboles de clases relacionadas con la consulta. Para ilustrar considérese el siguiente conjunto de consultas, expresadas sobre el esquema conceptual de la figura 2.

Consulta 1: "Recupere los nombres de los empleados que son miembros del staff del equipo de BD y cuyo status es el de investigador". En esta consulta se presènta como se genera el árbol de la consulta cuando involucra un atributo compuesto, y en CPOO se expresa de la forma siguiente:

{x.E_Nombre; x / Empleado; $\exists$ t / Equipo (x.Status = "investigador" $\wedge$ x $\in$ t.Staff $\wedge$ t.Eq_Nombre = "BD")}

El árbol de la consulta generada para esta interrogación se muestra en la figura 10.

Consulta 2: "Recupere los nombres de aquellos equipos con al menos un empleado que tenga un jefe que gane más de 100.000 Bs." En este caso se presenta un ejemplo de una consulta recursiva, y en CPOO se expresa de la forma siguiente:

$$\{x.Eq\_Nombre;\ x\ /\ Equipo;\ x.\exists\ Staff.\ \forall\ \rho(Jefe)\ .Salario > 100.000\}$$

El árbol de procesamiento para esta consulta se presenta en la figura 11.

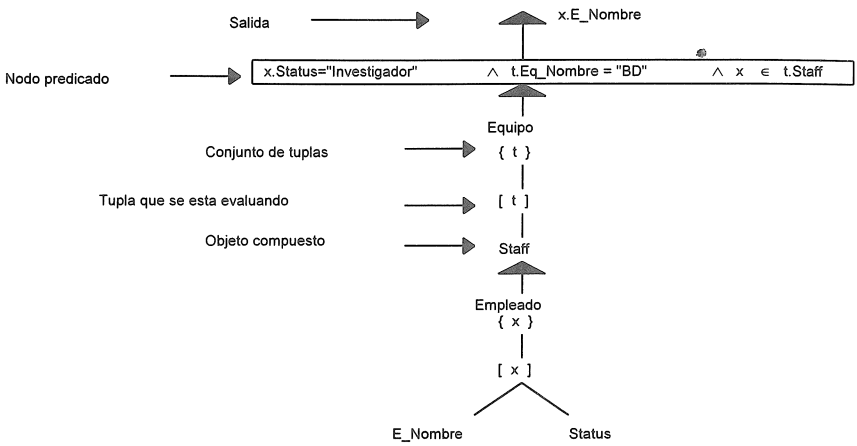

Fig. 10 Arbol de consulta para la interrogación de la consulta 1

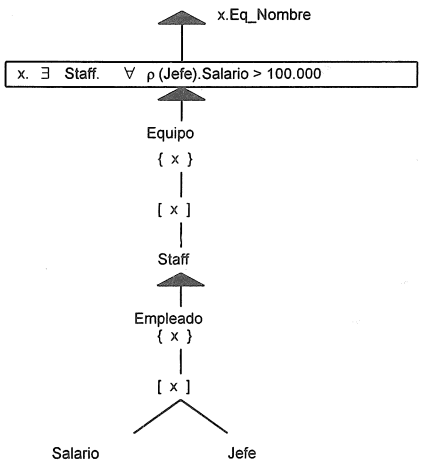

Fig. 11 Arbol de consulta para la interrogación de la consulta 2.

Consulta 3: "Recupere el número de las cédulas de aquellas personas que posean un salario mayor que 20.000 Bs. Si estas poseen un empleo o si son estudiantes recupere aquellos que comenzaron a estudiar en un año distinto a 1988". En este caso se considera una consulta en la que se encuentra involucrado un atributo heredado de un clase superior y se expresa en CPOO como:

{x.P_C.I. ; x / Persona; CLASS_OF(x)=[Empleado:x.Salario > 20.000; Estudiante:x.Año_Inicio ≠ 1988]}
El árbol de esta consulta se presenta en la figura 12

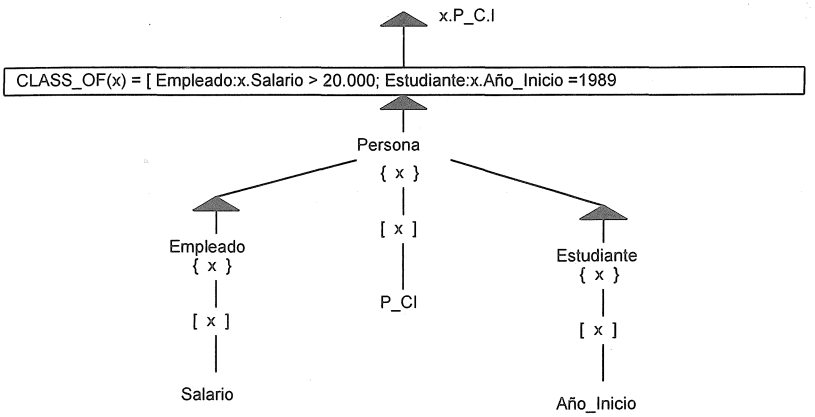

Fig. 12 Arbol de consulta para la interrogación de la consulta 3.

En el caso de que un usuario desee emplear un método dentro de una consulta, éste debe utilizarse como un operador más que actúa sobre las variables objetos que el usuario indique. Para visualizar este caso considere la consulta: "Recupere el nombre del proyecto cuyo objetivo es desarrollar un MBDOO que emplea un automóvil marca Mercedes para ir de viaje de negocios al instituto París IX" Esta consulta expresada en CPOO posee la forma siguiente:

$$\{x.P\_Nombre;\ x/Proyecto;\ x.Objetivo="MBDOO" \wedge \exists\ i / Instituto\ (i.I\_Nombre = "ParísIX" \wedge x.Disponibilidad\_de\_carro(i.E\_Dirección.Ciudad).Manufactura="Mercedes")\}$$

y posee el árbol de consulta presentado en la figura 13.

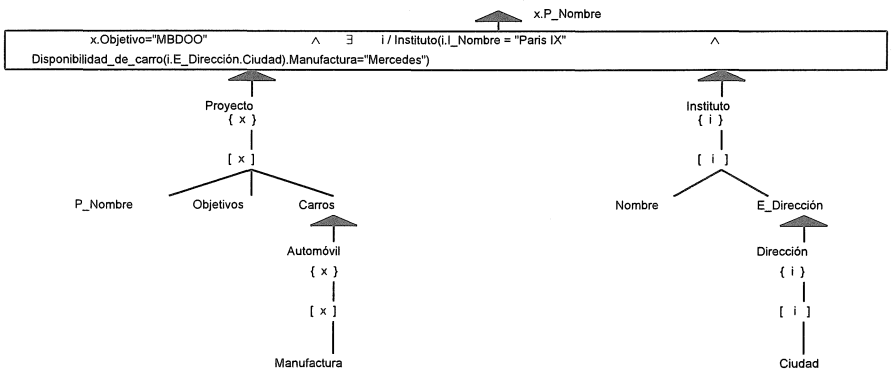

Fig. 13 Arbol de consulta para la interrogación de la consulta 4.

Como se ha visto anteriormente un árbol de consulta está formado por tres componentes: Nodo de salida,. nodo de predicado y nodos hijos o árboles de clases

Implantación del nodo salida

El nodo salida está formado por un puntero a una lista donde se almacenan todos los atributos requeridos por el usuario. El nodo salida tiene el formato siguiente:

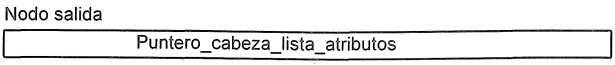

Fig. 14 Formato del nodo salida del árbol de consulta

Como se expresó anteriormente este nodo contiene el puntero hacia una lista de variables que están asociadas con los atributos requeridos por los usuarios. Esta lista está formada por nodos con el formato siguiente:

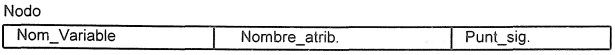

Fig. 15 Formato del nodo de la lista Lista_Nodo_Salida

Como un ejemplo de como se visualiza el nodo salida considere la figura 16:

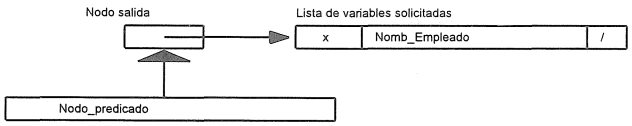

Fig. 16 Visualización del empleo del nodo salida

Implantación del nodo predicado

El nodo predicado describe las condiciones que deben satisfacer los objetos que se debe recuperar. Sin embargo este predicado puede no existir, en cuyo caso se recuperan todas las tuplas de las clases indicadas en los nodos hijos. El nodo predicado se implanta como un nodo que contiene una cadena de longitud indefinida, por lo que es necesario utilizar un delimitador que indique el final de ésta.

Implantación de los nodos hijos o árboles de clases

Cada nodo hijo del árbol de la consulta es a su vez un árbol implantado de manera individual. Cada uno de estos nodos son empleados para representar las clases involucradas en una consulta expresada en CPOO.

Cada nodo hijo del árbol de consulta posee el formato siguiente:

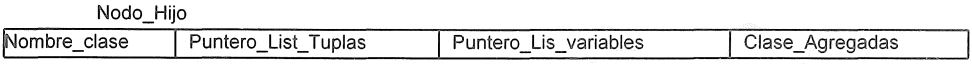

Fig. 17 Formato del nodo hijo del árbol de consulta.

A continuación se explica cada uno de los componentes del nodo llamado Nodo_Hijo

Nombre_clase: es el nombre de la clase que es raíz del nodo hijo.

Ult_Tupla_procesada: Este atributo almacena el último objeto suministrado por el manejador de objeto. Sí esta tupla satisface el predicado almacenado en el nodo predicado, entonces ésta se almacena en la lista direccionada por el atributo Puntero_List_Tuplas.

Puntero_List_Tuplas: es un puntero a la cabeza de la lista que almacena los identificadores de objetos que satisfacen los predicados que involucran dicha clase. Esta lista de identificadores se puede visualizar gráficamente en la figura 18:

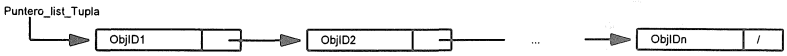

Fig. 18 Visualización del empleo de la lista de identificadores

Puntero_Lis_variables: Es un puntero a la lista de variables asociadas a los atributos requeridos por el usuario.

Clase agregada: este es un puntero hacia un nuevo nodo hijo que representa a una clase agregada del nodo hijo actual.

6. Generador del grafo de conexión

Una vez generado el árbol de consulta, éste se le suministra al módulo que realiza la traducción del árbol de consulta al esquema físico para así obtener el grafo de conexión. La consulta realizada por el usuario se expresa sobre el esquema conceptual; sin embargo a medida que el proceso de optimización avanza, los módulos requieren conocer el esquema físico. Por lo tanto, se hace necesario traducir la consulta al esquema físico, a fin de poder generar los planes de ejecución con los que se dará respuesta al usuario. Antes de proseguir con la explicación de cómo funciona este módulo, se debe describir como el manejador de objetos del MIDBS almacena las instancias de las clases. El manejador de objetos del MIDBS propuesto por Besembel en [BESEMBEL 94], sigue el enfoque de almacenar en un mismo grupo de disco tanto los objetos padres como sus objetos agregados. En el caso de que los objetos no puedan almacenarse juntos, el manejador de objetos agrupará en el mismo vecindario o paquete de discos los objetos restantes. Este enfoque permite la localización rápida de tanto los objetos padres como sus objetos agregados. Es importante resaltar que el manejador de objetos del MIDBS hace uso de un grafo de esquema físico, el cual permite mantener la correspondencia entre el esquema conceptual de la BDOO y los objetos almacenados en cada grupo. El uso del esquema físico le permite al manejador de objetos localizar de una manera rápida los objetos requeridos por una consulta. El grafo de esquema físico propuesto para MIDBS está compuesto por dos tipos de nodos: uno que se utiliza para representar los grupos que contienen objetos atómicos y otro, para representar los grupos que contienen objetos compuestos. Adicionalmente, se tienen dos tipos de enlaces: uno para indicar caminos de agrupamiento y otro para expresar caminos de índices. La figura 19 muestra el grafo del esquema físico de la BDOO propuesta en la figura 2. Puede observarse que existen los grupos F1, F2, F3, F4, F5 donde se almacenan todos los objetos compuestos. El manejador de objetos es el encargado de construir el grafo de esquema físico a medida que el usuario introduce el esquema conceptual de la BDOO.

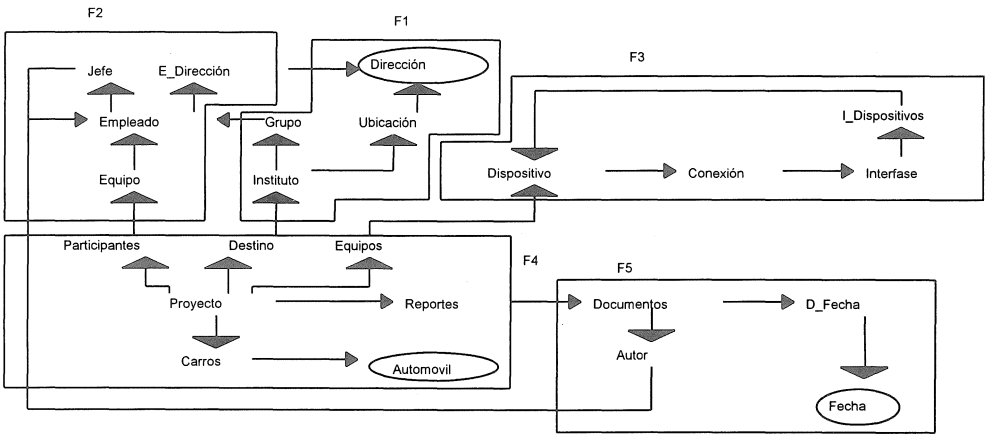

Fig. 19 Grafo del esquema físico del esquema conceptual de la BDOO de la figura 2

Teniendo en cuenta lo mencionado anteriormente , la función de este módulo es traducir la consulta del usuario que viene expresada en función del esquema conceptual al esquema físico para así obtener el grafo de conexión de la consulta. Para lograr dicha traducción este módulo recorre al mismo tiempo el árbol de la consulta y el grafo de esquema físico, con la finalidad de relacionar cada clase del árbol de consultas con su respectiva extensión de clase en el grafo del esquema físico. Como resultado de esta traducción se obtiene el grafo de conexión, en el que los nodos representan las extensiones de las clases que se encuentran involucradas en la consulta.

En el grafo de conexión existen tres tipos de enlaces: dos para expresar las junturas que pueden ser explícitas o implícitas y el tercer enlace para expresar las operaciones de selección de objetos. Es importante destacar que este grafo de conexión posee sus raíces en el grafo de conexión propuesto por [WONG 76] para utilizarse en las bases de datos relacionales. La figura 20 muestra el grafo de conexión obtenido a partir del grafo del esquema físico presentado en la figura 19 y del árbol de consulta de la figura 12.

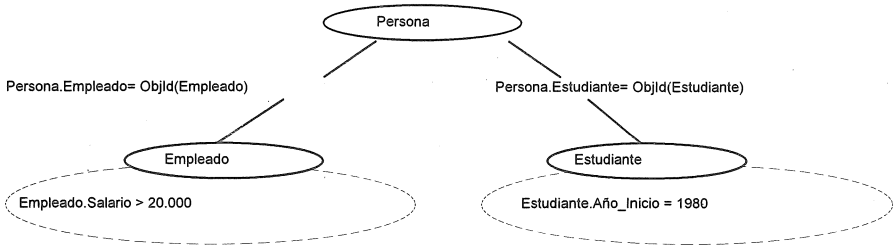

Fig. 20 Grafo de conexión para la consulta 3

El grafo de conexión se implanta como un grafo no dirigido donde los nodos representan las clases involucradas en la consulta, los enlaces entre nodos representan las junturas tanto explícitas como implícitas y los bucles representan la operación de selección. Para implantar este grafo de conexión se ha decidido emplear un grafo diseñado como una lista de listas, tal y como se puede observar en la figura 21

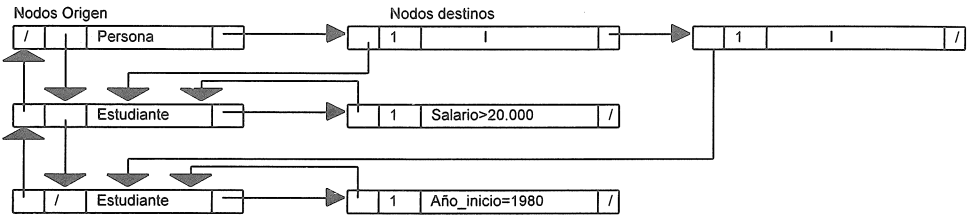

Fig. 21 Implantación del grafo de conexión

El formato del nodo origen del grafo de conexión es el presentado en la figura 22 y el formato del nodo enlace se presenta en la figura 23

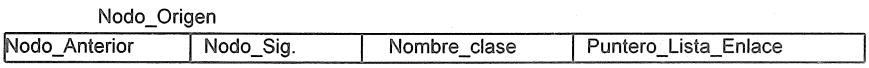

Fig. 22 Formato del nodo Origen.

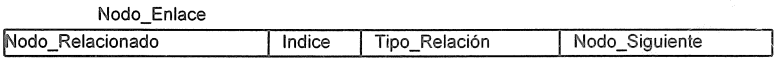

Fig. VII.22 Formato del nodo Enlace.

El atributo llamado Tipo_Relación del nodo enlace, almacena una cadena de caracteres que representa la relación existente entre los nodos que son vinculados por dicho enlace. Si la cadena es nula significa que existe una relación de juntura explícita que debe entenderse de la forma siguiente:

Nodo_origen.Nodo_Relacionado = ObjId (Nodo_relacionado)

En caso de que la cadena no sea nula se trata de una juntura implícita o una operación de selección. El campo índice del nodo enlace puede almacenar solo dos valores, 0 si no existe un índice sobre los atributos involucrados en la juntura y 1 en el caso de que exista un índice.

7. Módulo generador de árboles de procesamiento

Este módulo se basa en algunas ideas tomadas de las proposiciones de [LANZELOTTE 91] y [BERI 90]. Estos autores definen un árbol de procesamiento como un árbol no completo y no balanceado, donde los nodos hojas son entidades del esquema físico (Clases) y cada nodo no hoja representa una operación que puede ser de selección, juntura o proyección. Los resultados de estas operaciones sobre los nodos hojas pueden ser guardados como clases temporales en memoria principal, las cuales pueden considerarse como

nuevos nodos hojas. El orden de los nodos en los árboles de procesamiento especifican el camino a recorrer para obtener la respuesta del usuario, así como el orden en el cual las operaciones que se deben ejecutar. Este enfoque permite la construcción de árboles de procesamiento en el que el camino directo a un objeto complejo pueda ser intercalado con otras operaciones del mismo árbol, esto quiere decir que se pueden generar árboles de consultas donde primero se ejecuten las operaciones de selección de objetos, después las operaciones de juntura explícita y por último las operaciones de juntura implícita.

Para generar los árboles de procesamiento se requiere que se suministre como parámetro de entrada el grafo de conexión generado en el módulo anterior.

Puesto que pueden existir varios árboles de procesamiento distintos con los cuales se puede dar respuesta a una consulta del usuario. Por lo tanto, cada árbol de procesamiento generado en este módulo será una instancia de la clase árbol.

8 Módulo evaluador de costo de procesamiento

A este módulo se le suministra como parámetro de entrada una lista de árboles de procesamiento, los cuales deben evaluarse para determinar el costo en el que se incurre al ejecutar ese plan. Este costo de procesamiento está en función del tiempo de procesamiento del árbol. Los costos de procesamiento de los planes son almacenados en el campo llamado costo para determinar cuál de los planes es el óptimo. Esto se logra recorriendo la lista de árboles de procesamiento para así buscar aquel nodo de la lista que posea el menor valor en el campo costo.

Como se ha meniconado anteriormente, el modelo D / K puede soportar tanto datos espaciales como datos no espaciales. Ello trae como consecuencia que se debe estudiar qué funciones de costos utilizar cuando se trabaja con un tipo de dato espacial o cuando se trabaja con un tipo de dato no espacial.

Funciones de costo para evaluar operaciones sobre datos no espaciales

Las funciones de costo para evaluar los árboles de procesamientos dependen del tiempo de ejecución de la operación que se desea ejecutar. A continuación se presenta una tabla con el tiempo de ejecución de cada una de las operaciones empleadas en el procesamiento de consulta que involucran objetos no espaciales.

Tipo de operación	Tiempo de ejecución	Condición
Selección de objetos	O (Card(Clase))	
Juntura de clases	O (Card(Clase)*Log(Card(Clase)))	Utilizando Ordenamiento
Juntura de clases	O(Card(Clase))	Utilizando Hashing

De la tabla anterior puede observarse que existen dos tiempos de ejecución para la operación de juntura; ellos dependen del uso de un método de ordenamiento o de un método hashing para el acceso de objetos involucrados.

Función de costo para evaluar operaciones sobre datos espaciales

Las funciones de costos empleadas aquí fueron propuestas en [KIM 93]. Este autor propone un modelo de costo elemental para el cálculo de los predicados de selectividad para datos espaciales, partiendo que el sistema mantiene la información siguiente:

- Indices sobre los mínimos rectángulos que cubren los objetos espaciales.
- Cardinalidad. Total de objetos espaciales.
- Escala por cada dimensión, es decir, el valor máximo y mínimo por cada dimensión.

9. Módulo de ejecución del plan

Este módulo recibe como parámetros de entrada un puntero a la raíz del árbol de procesamiento que posee el menor costo. Este árbol de procesamiento indica cuáles son las clases que se encuentran involucradas en la consulta e indica el orden en el que se deben ejecutar las operaciones, para así obtener la respuesta que el usuario está requiriendo. Este módulo inicia la ejecución del plan a partir de los nodos hojas hasta llegar al nodo raíz. A medida que este módulo va ejecutando los nodos no hojas del árbol se va invocando el código ejecutable de las operaciones requeridas en estos nodos. Cuando estas operaciones se ejecutan, estas invocan al manejador de objetos para que le suministre las instancias requeridas. Los resultados intermedios de las

operaciones que se va realizando se almacenan en una clase temporal en memoria principal con la finalidad de servir como nodo hoja en el nivel siguiente. Cuando el módulo procesador del plan llega al nodo raíz se almacenan los atributos requeridos por el usuario en las respectivas variables de salida.

10 Conclusiones

La arquitectura del optimizador de consulta propuesta en este trabajo puede utilizarse en cualquier SMBDOO que posea las condiciones mínimas siguientes:

- Un lenguaje de consulta basado en el CPOO como el propuesto en [BERTINO 92].
- Un modelo de datos capaz de representar todos los conceptos y paradigmas de la orientación por objeto, y que permita al usuario asociar un conjunto de reglas de integridad semánticas a cada clase dentro del esquema conceptual.
- Posea un manejador de objetos que permita construir el grafo de esquema físico a medida que el usuario defina e implante su esquema conceptual.

El hecho que el modelo D/K posea la capacidad de asociar reglas de integridad semánticas a una clase, permite que el optimizador de consulta pueda dar respuestas a ciertas consultas formuladas por el usuario sin necesidad de acceder a las instancias de las clases almacenadas en la BD. Es importante destacar que en la bibliografía consultada ninguno de los SMBDOO poseen esta característica; esto se debe a que los modelos de datos que estos sistemas soportan no contemplan el manejo de reglas de integridad semántica como parte de su esquema

El concepto de encapsulamiento u ocultamiento de la información provisto por la orientación por objetos es violado por el optimizador de consultas debido a que este necesita saber como es la estructura interna de las clases para procesar la consulta. Por esta razón, se ha considerado en éste trabajo al optimizador de consultas como un dispositivo de confianza para el sistema.

Con respecto a las estructuras de datos que se ha empleado en el diseño de todos los módulos del sistema puede observarse que son estructuras de datos muy conocidas y de fácil implantación en cualquier lenguaje, y que además se adaptan muy bién tanto a los requerimientos funcionales como a los de almacenamiento de cada módulo.

BIBLIOGRAFIA

[BERI] Beri C. Kornastky Y., "Algebraic Optimization of Object-Oriented Query Languaje" In proceedings International Conference on Database Theory, Paris, France, Diciembre de 1990.

[BERTINO 92] Bertino E., Negri M., Pelagatti G. y Sabatella L., "Object-oriented Query Language: The Notion and the Issues". Knowledge and Data Engineering, Volumen 4, Número 3, Junio 1992.

[BESEMBEL 93] Besembel Isabel " D/K object manager ", Reporte Técnico del grupo GIDYC Diciembre de 1993.

[HUGHES J. G] Hughes J. G Object Oriented Database Prentice Hall 1991.

[KIM 93] Kim W., Garza J. F., "Operator for Spatial Data" Lecture Notes in Computer Science Springer-Verlag 1993.

[LANZELOTTE 91] Lanzelotte Rosana, Valduriez P., " Optimization of Nonrecursive Queries in OODBs" Lecture in Computer Science Springer - Verlag, Septiembre 1991.

rabajo de ascenso Julio 1989.

[MONTILVA 93] Montilva Jonás, "An integration method applied to the desing of data / knowledge model for multimedia and spatial applications " Enero de 1993.