COMPILADOR DE EXPRESIONES DEL LENGUAJE RELACIONAL PARA EL SISTEMA TESEO

PERLA LEVY

Grupo de Investigación SINBAD
Dpto. de Ingeniería de Sistemas
Universidad de los Andes
Apartado Aéreo 4976
Bogotá-Colombia

RESUMEN

Tener las facilidades de un lenguaje relacional para manipular una Base de Datos Distribuida es muy importante. En este artículo se presenta un lenguaje relacional inspirado en SQL para el sistema TESEO, y las transformaciones que debe tener cada una de las instrucciones de este lenguaje para quedar en términos de los servicios ofrecidos por el sistema TESEO para manejar Bases de Datos Distribuidas.

Estas transformaciones incluyen optimizaciones al generar los planes de consulta, con lo cual se reduce el tiempo de respuesta de las operaciones aplicadas sobre una Base de Datos Distribuida.

1. INTRODUCCION

Este artículo describe la implementación del módulo COMPILADOR DE EXPRESIONES DEL LENGUAJE RELACIONAL para el sistema TESEO [Franky 87]. Los objetivos de este proyecto realizado como tesis de pregrado, son ofrecerle al diseñador de una Base de datos distribuida(BDD) un lenguaje relacional que le permita manipular la base de datos global(BD), y realizar algunas optimizaciones en la generación de planes de consulta a partir de instrucciones del lenguaje relacional.

El lenguaje relacional ofrecido al diseñador facilita el acceso a la BD: permite hacer consultas, modificaciones, etc., que involucren varias relaciones al tiempo y ofrece operadores como join, proyección, unión y producto cartesiano.

El módulo Compilador de TESEO se aplica en el contexto de BDD con las siguientes características:

- Bases de datos que sigan el modelo relacional.

- Definición estática de relaciones y de los esquemas de fragmentación y distribución.

- Fragmentación horizontal simple de las relaciones: sin replicación.

  El diseñador de la aplicación debe manejar las relaciones globales únicamente a traves del lenguaje relacional descrito en este artículo.

  El sistema está orientado hacia la compilación de transacciones especificadas por el diseñador, que pueden ser solicitadas con diversos parámetros por los usuarios finales.

2 - DESCRIPCION GENERAL

Debido a que el módulo **Compilador** de TESEO debe trabajar en forma coordinada con los módulos existentes del sistema TESEO, a continuación se dara una breve explicación de estos últimos. La descripción completa se encuentra en [Franky 87] y posteriormente se presentarán las ampliaciones hechas a los módulos existentes a fin de implementar el módulo **Compilador**.

2.1 MODULOS INICIALES DEL SISTEMA TESEO

La primera versión del sistema TESEO está conformada por dos módulos que deben ejecutarse en dos etapas sucesivas y que son:

A.- MODULO ESPECIFICADOR

El objetivo de este módulo es ofrecerle al diseñador las facilidades necesarias para especificar tanto los esquemas global, de fragmentación y de distribución de la base de datos, como las operaciones y los algoritmos de cada transacción (ver definición de esquemas de BDD en [Ceri 84]). Esto se hace con el fin de generar tablas y archivos que seran consultados tanto por el sistema (para efectuar verificaciones y validaciones), como por la aplicación (en ejecución).

El diseñador utiliza este módulo para:

- Especificar los sitios lógicos sobre los cuales se va a ejecutar la aplicación.
- Definir las relaciones globales de la base de datos.
- Especificar la fragmentación y distribución de relaciones.
- Definir y especificar las transacciones que podrán ser solicitadas por los usuarios finales.

B.- MODULO GENERADOR

En este módulo el sistema toma los archivos definidos por el diseñador en el módulo especificador y, a partir de ellos, va generando el programa que simula los procesos concurrentes que conforman la aplicación distribuida.

En esta fase ocurren una serie de verificaciones con respecto a las transacciones definidas, para detectar errores como:

- Transacciones mal estructuradas (Begin - End).
- Utilización de variables locales y de parámetros que no han sido definidos.

- Utilización de relaciones, llaves o atributos que no han sido definidos o que sus tipos no corresponden.

- Utilización inadecuada de los servicios ofrecidos por el sistema en uno o varios de sus parámetros.

- Existencia de algunos archivos utilizados.

4.2 - AMPLIACIONES AL MODULO ESPECIFICADOR

En el módulo Especificador de la primera versión de TESEO, el diseñador especifica las transacciones en lenguaje Turbo-Pascal utilizando los siguientes **servicios** del sistema, para manejar las relaciones globales:

- **Insertar** (relacion)

  Permite adicionar una tupla a relación. La llave principal e info constituyen los valores de todos los atributos y se separan por '/' así:

  /atr1/atr2/...../atrn/

- **Eliminar** (relacion, llave-ppal)

  Borra la tupla con **llave-ppal** de la relación especificada, es la función inversa de insertar.

- **Modificar** (relacion, llave, atributos que cambian, info)

  Modifica los atributos con la nueva información de la tupla con llave de la relación especificada.

- **Seleccione-Proyeccion**(relacion,formula,proyeccion,archivo_resultado)

  Selecciona de **relación** aquellas tuplas que cumplen la **fórmula** de selección, devolviendo en **archivo_resultado** la proyección de esas tuplas los atributos especificados en **proyección**.

Respecto a la primera versión de TESEO el compilador permite al diseñador utilizar además de los servicios mencionados, un lenguaje relacional inspirado en SQL [Chamberlin 76] con las siguientes instrucciones para manipular la base de datos global.

a.- SELECT atributos INTO tabla
    FROM relaciones globales
    WHERE fórmula

  Significado:
  Seleccione las tuplas de las **relaciones globales** que cumplan con la **fórmula**, proyéctelas sobre **atributos** y guardelas en **tabla**.

b.- INSERT INTO relación
FROM tabla

Inserte las tuplas de **tabla** en la **relación**.

c.- UPDATE relación
SET atr1=<exp1>, atr2=<exp2>, ... , atrn=<expn>
WHERE fórmula

Actualice en **relación** las tuplas que cumplan con la **fórmula** de la siguiente manera: en el **atr-i** coloque la **<exp-i>**.

d.- DELETE relación
WHERE fórmula

Borre de **relación** las tuplas cumplan con la **fórmula**.

Para las instrucciones especificadas anteriormente se asume que:

tabla es una relación temporal local.

<EXPI> ::= <CONSTANTE> / <PARAMETRO>

<ATRIBUTOS> ::= <RELACION> <.> <ATRIBUTO> /
<RELACION> <.> <*>

<FORMULA> ::= <ATOMO> / <ATOMO> <Y> <FORMULA>

<ATOMO> ::= <ATRIBUTO> <OPREL> <ATRIBUTO> /
<ATRIBUTO> <OPREL> <CONSTANTE> /
<ATRIBUTO> <OPREL> <PARAMETRO>

<OPREL> ::= [<, >, =, <=, >=, <>]

Solo se acepta 'Y' como operador lógico de la fórmula porque una instrucción que tenga un 'O' en su fórmula de selección, se puede expresar como la unión de 2 tablas resultantes de un SELECT, UPDATE, etc.

En conclusión, la implementación del módulo compilador de TESEO permite que el diseñador programe las transacciones de su aplicación en lenguaje Turbo-Pascal pudiendo manipular las relaciones globales a través del lenguaje relacional presentado arriba.

2.3 - UBICACION DEL MODULO COMPILADOR EN EL SISTEMA TESEO

El siguiente diagrama ilustra la ubicación del módulo **Compilador** en el sistema TESEO.

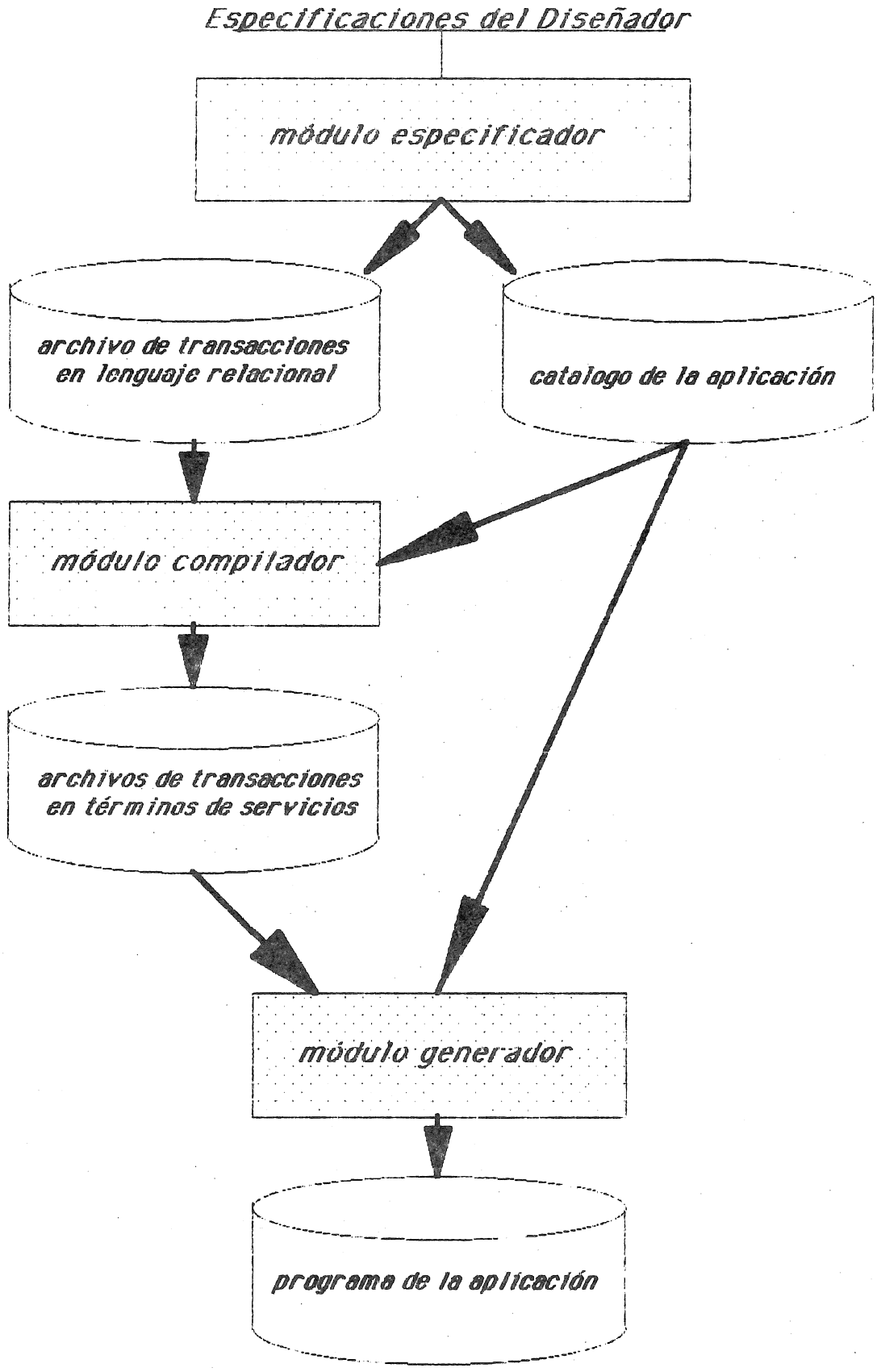

Figura 1: UBICACION DEL MODULO COMPILADOR EN EL PROGRAMA TESEO

Explicación:

Una vez que el diseñador especifica transacciones en lenguaje relacional, el módulo **Compilador** debe transformar el texto de estas transacciones en términos de servicios del sistema y de operaciones entre tablas locales.

El archivo resultante de esta transformación es el que debe tomar como entrada el módulo Generador.

2.4 - AMPLIACIONES AL MODULO GENERADOR

El procesamiento del lenguaje relacional de las transacciones, genera invocaciones a los servicios existentes (Insertar, Eliminar, Modificar, Seleccion-Proyeccion) y a nuevas operaciones locales que van a trabajar sobre tablas locales, resultado de servicios. Por lo tanto, es necesario adicionar al módulo Generador el código de las nuevas operaciones locales, que son las siguientes:

a.- UNION (TABLA1,TABLA2,TAB-RESP)

su función es unir las dos primeras tablas y dejar el resultado en la tabla dada como tercer parámetro (TAB-RESP).

b.- JOIN (TABLA1,TABLA2,FORMULA,PROYECCION,TABLA3)

realiza el join de TABLA1 con TABLA2, selecciona aquellas tuplas que cumplan con la FORMULA, las proyecta sobre los atributos de PROYECCION, y las devuelve en TABLA3.

c.- SEL-PROY (TABLA1,FORMULA,PROYECCION,TABLA2)

toma las tuplas de TABLA1 que cumplan con la FORMULA, las proyecta sobre los atributos de PROYECCION y las devuelve en TABLA2.

3. DISEÑO DEL MODULO COMPILADOR

El mayor trabajo del proyecto se concentra en la nueva etapa de compilación, la cual debe solucionar el problema de transformar las clausulas SELECT, INSERT, DELETE, UPDATE (presentadas en la sección 2), a instruciones en términos de los servicios del sistema: SELECCION-PROYECCION, INSERTAR, ELIMINAR, MODIFICAR y de las operaciones locales UNION, JOIN, SEL-PROY.

El módulo Compilador examina el archivo que contiene el texto de las transacciones y produce un archivo de salida con el texto de las transacciones ya transformado. Es este archivo de salida del Compilador el que debe tomar como entrada el módulo Generador.

3.1 - COMPILACION DEL SELECT

La compilación del SELECT implica una serie de pasos que serán explicados a continuación. En este tratamiento se aplicó una combinación de las transformaciones y optimizaciones expuestas en [Ceri caps 5 y 6, 84]

a. - PRIMERA INTERPRETACION GENERAL DE UN SELECT

La instrucción de consulta

```
SELECT a1,a2,.....,aN INTO tabla
FROM   R1,R2,.....,Rn
WHERE  atomo1 Y atomo Y ... Y atomoN
```

donde <a1> ::= <relacion> <.> <atributo>

<atomoI> ::= <atributo><operador relacional><atributo> /
<atributo><operador relacional><constante> /
<atributo><operador relacional><parametro>.

se interpreta inicialmente como:

π a1,..,aN (σ atomo1 Y..Y atomoN (R1 X R2 X ..... X Rn))

donde los atributos respuesta del SELECT (a1,..,aN) son los atributos de la proyección externa; luego sigue una selección que se aplica sobre los átomos del WHERE (atomo1 Y....Y atomoN), y por último se tienen joins entre las relaciones especificadas en el FROM (R1,R2,...,Rn).

El árbol de consulta global es el resultado de ésta primera interpretación y tiene la siguiente forma:

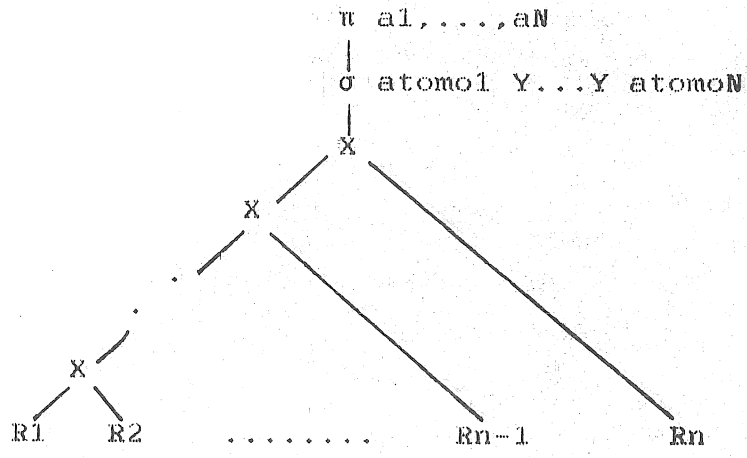

FIGURA 2: FORMA GENERAL DEL ARBOL INICIAL DE CONSULTA

B. - **BUEN ORDEN PARA HACER LOS JOINS**

Con el fin de evitar que los joins se conviertan en producto cartesiano hay que escoger un orden adecuado para hacerlos.

El metodo escogido para hacerlo es el siguiente:

- Se escoge como primera relación aquella que tenga en la fórmula de selección algún atributo relacionado con una constante. Esto se hace porque permite añadir en el árbol de consulta una selección directa sobre la relación, con lo cual se reduce el número de tuplas participantes en el join.

  En caso de que ninguna relación cumpla con esta condición, se escoge cualquiera como primera.

- Como segunda relación se escoge cualquiera que en la fórmula de selección tenga algún atributo relacionado con otro de la relación escogida como primera.

- Las demas relaciones se van escogiendo con el mismo criterio: que tengan en la fórmula algún atributo que las relacione con otro atributo de cualquiera de las relaciones anteriores.

Si quedan relaciones que no tengan nexo alguno con las anteriores, éstas se colocan al final de la secuencia e implicarán inevitablemente productos cartesianos.

Para ilustrar este método se da el siguiente ejemplo:

Se tienen las siguientes relaciones

```
EST     (CARNET,NOMBRE,PROM,....)
VIENDO  (CARNET-E,COD-C,........)
CURSO   (COD,CRED,..............)
```

y se desea hacer la siguiente consulta

```
SELECT  est.carnet, est.nombre
FROM    est,viendo,curso
WHERE   est.carnet=viendo.carnet-e  Y viendo.cod-c=curso.cod
          Y curso.cred > 3
```

Siguiendo los pasos especificados anteriormente, se escogería como primera relación CURSO por existir en la fórmula de selección (parte WHERE) el átomo curso.cred>3; luego se escogería VIENDO porque es la única relacionada con CURSO por medio del átomo viendo.cod-c=curso.cod, y por último se escoge EST por estar relacionada con VIENDO por el átomo est.carnet=viendo.carnet-e.

C. - GENERACION DEL ARBOL FINAL

Basándose en el árbol de consulta global y luego de haber escogido el orden de los joins, se procede a la generación del árbol de consulta final.

El árbol que se va a generar equivale a realizar una serie de transformaciones sobre el árbol inicial, a fin de incluir algunas de las simplificaciones y optimizaciones presentadas en [Ceri caps 5 y 6, 84].

El árbol final debe presentar las siguientes características, en relación al árbol inicial:

- debe tener las relaciones (nodos hojas) en un buen orden de acuerdo a los criterios ya expuestos.
- debe generar el máximo número de operadores de proyección y selección a fin de reducir el tamaño de los operandos antes de las operaciones binarias de join.
- debe bajar lo máximo posible esos operadores reductores (Π y σ) a fin de aplicarlos lo antes posible.
- debe colocar los átomos paramétricos en el nivel mas superior a fin de evaluar las variables de la consulta una sola vez, en compilación.

Para ilustrar la forma final del árbol para el caso de 3 relaciones usemos el siguiente ejemplo:

```
SELECT  est.carnet, est.nombre
FROM    est, viendo, curso
WHERE   est.carnet=viendo.carnet-e Y viendo.cod_c=curso.cod
        Y curso.cred > 3  Y est.prom > xxx
```

El árbol final correspondiente es:

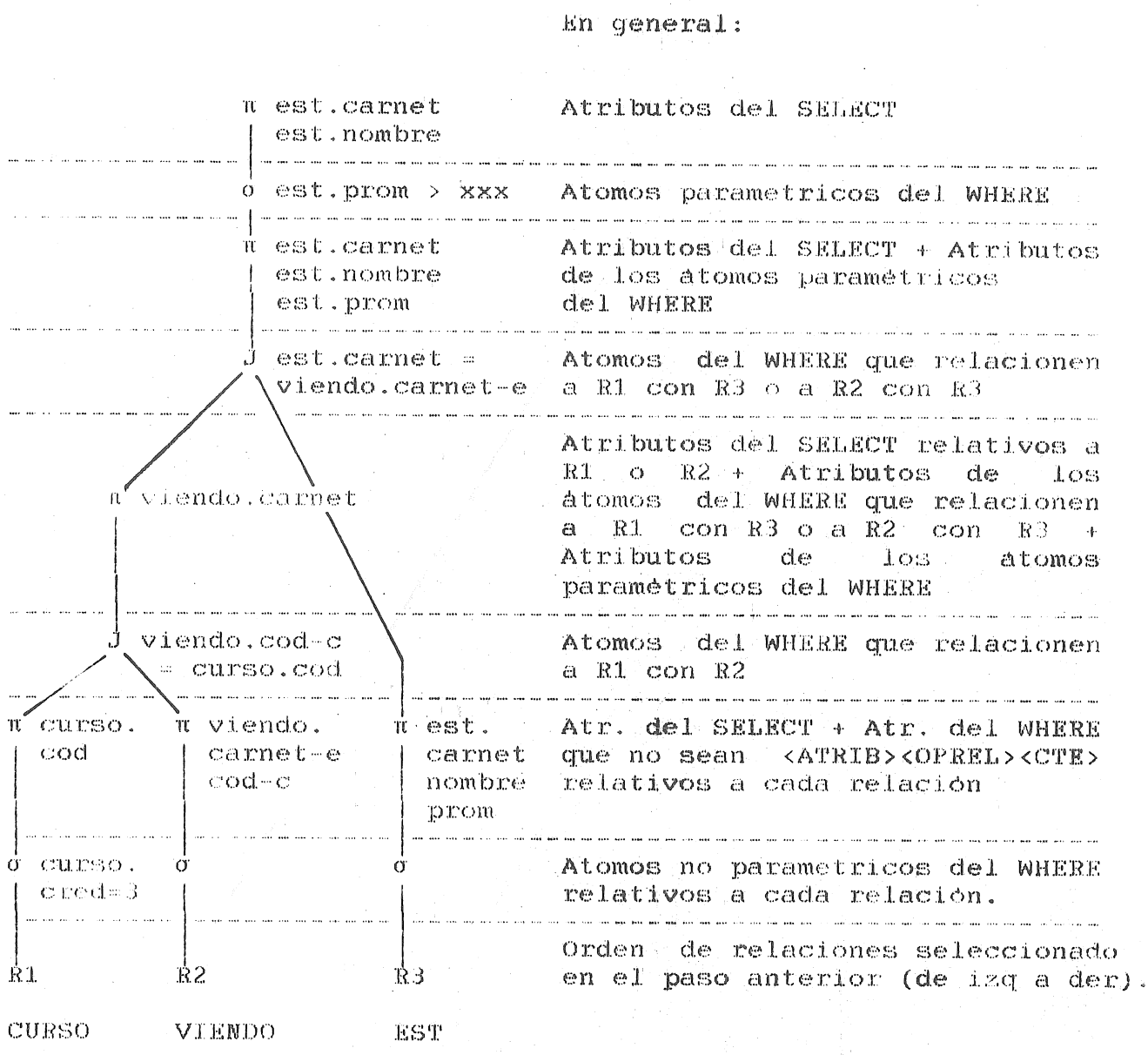

FIGURA 3: EJEMPLO DE ARBOL FINAL DE CONSULTA

Al lado derecho de la FIGURA 3 están especificados los atributos que debe tener cada una de las proyecciones, selecciones y joins en referencia a la forma general del SELECT.

Generalizando a n relaciones, el árbol final es el siguiente (Figura 4):

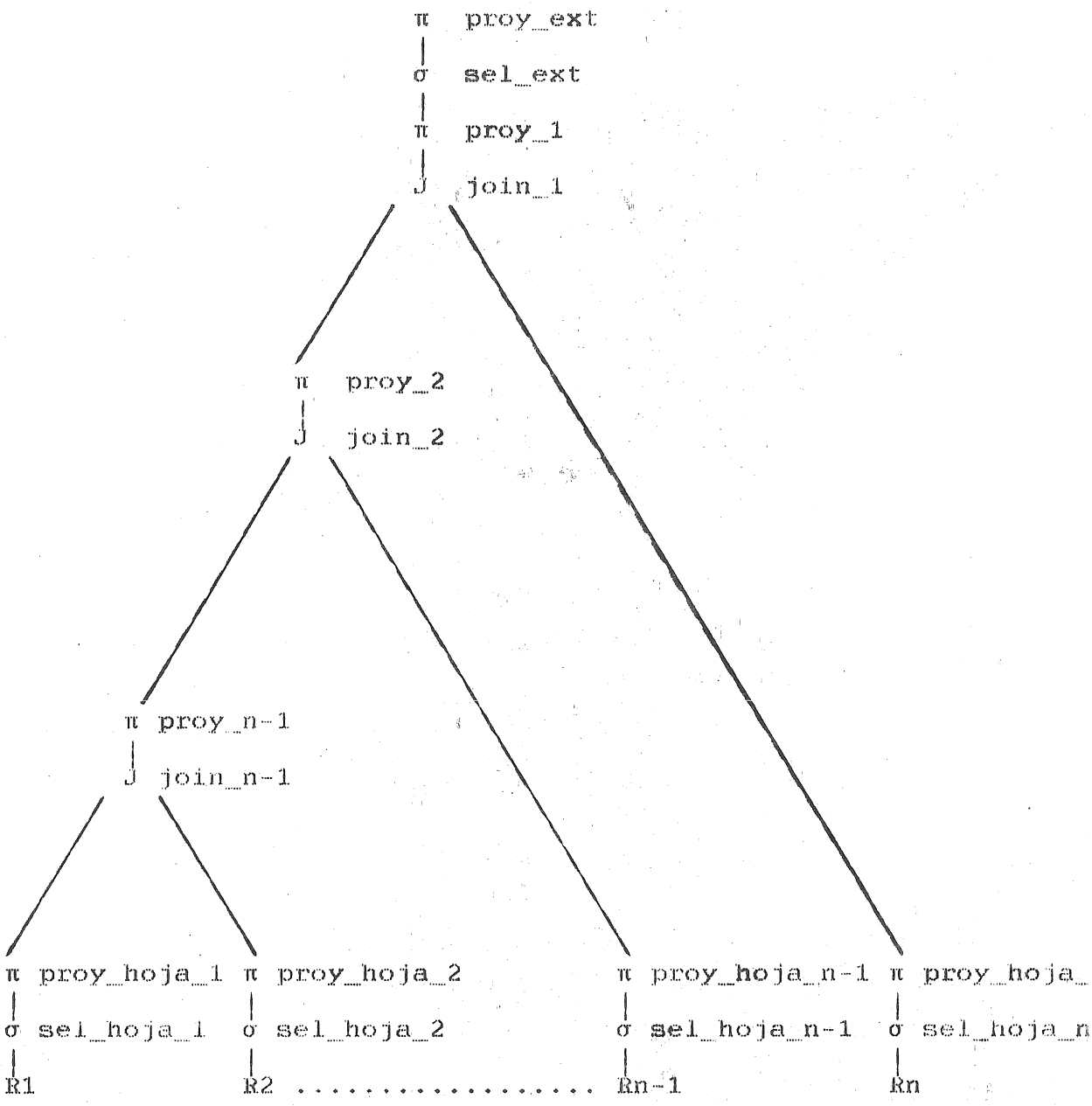

FIGURA 4: FORMA GENERAL DEL ARBOL FINAL DE CONSULTA

La explicación de los atributos correspondientes a cada nodo y la forma como se va generando el árbol de consulta, se explica en el siguiente macroalgoritmo correspondiente al programa que genera el árbol final directamente a partir del árbol inicial.

Sea n el número de relaciones que intervienen en la consulta.

```
MACROALGORITMO ( N )

  SI  hay átomos paramétricos en el where ENTONCES
  |
  |  PROY_EXT   <---   Atributos del select
  |  SEL_EXT    <---   Atomos paramétricos del where
  |
  FIN_SI
  K  <---  N
  PARA  I:=  1, N-1  HAGA
  |
  |  PROY_I     <---   Atributos del select + Atributos de los átomos
  |                    paramétricos del where relativos a las primeras
  |                    K relaciones + Atributos de los átomos
  |                    del where que sean <ATRIB><OPREL><ATRIB> relati-
  |                    vos a las primeras K relaciones, que
  |                    relacionen a cualquiera de las primeras K
  |                    relaciones con cualquiera de las restantes.
  |
  |  JOIN_I     <---   Atomos no paramétricos del where que relacionen
  |                    a REL[K] con cualquiera de las anteriores.
  |
  |  PROY_HOJA_K  <---   Atributos del select + Atributos del where
  |                      que no sean <ATRIB><OPREL><CTE> relativos a
  |                      REL[K].
  |
  |  SEL_HOJA_K   <---   Atomos del where que sean <ATRIB><OPREL><CTE
  |                      relativos a REL[K].
  |
  |  K  <---  K - 1
  FIN_PARA
  PROY_HOJA_K  <---   Atributos del select + Atributos del where
                      que no sean <ATRIB><OPREL><CTE> relativos a
                      REL[K].

  SEL_HOJA_K   <---   Atomos del where que sean <ATRIB><OPREL><CTE>
                      relativos a REL[K].

FIN_MACROALGORITMO
```

El funcionamiento del macroalgoritmo es el siguiente:

Primero se deducen los nodos superiores del árbol PROY_EXT y SEL_EXT (en caso que existan) y luego se deducen los demás nodos en ciclos repetitivos así:

Habiendo deducido los nodos PROY_I, JOIN_I, se deducen los nodos derechos PROY_HOJA_K, SEL_HOJA_K; luego se decrementa K para que en el siguiente ciclo se tomen los nodos hojas correspondientes a la rama derecha de PROY_I, JOIN_I, y por último se incrementa I para desplazarse hacia los nodos izquierdos PROY_I, JOIN_I que serán deducidos en el siguiente ciclo.

d. - TRANSFORMACION DEL ARBOL DE CONSULTA DEL SELECT A PEDIDOS DE SERVICIOS Y A OPERACIONES SOBRE TABLAS LOCALES

Una vez que se obtiene el árbol final de consulta, el compilador debe traducirlo a instrucciones en términos de invocaciones de servicios, de manipulación de relaciones y de operaciones sobre tablas locales. Para ello, el Compilador recorre el árbol en postorden para que cada vez que se vaya a analizar una operación ya estén evaluados sus operandos, y a medida que se vaya avanzando se vayan haciendo las traducciones correspondientes, dependiendo del nodo a evaluar.

Hay 3 clases de nodos: el nodo raiz, los nodos intermedios y los nodos hojas y la traducción para cada uno de ellos es la siguiente:

- Para el nodo raiz (proyección - selección superior del árbol) se debe generar la operación local

  SELECCION-PROYECCION (TABLA, FORMULA, PROYECCION, TABLA-RTA)

  donde TABLA debe contener las tuplas resultantes de todas las operaciones inferiores del árbol y TABLA-RTA va a contener el resultado total de la consulta.

- Para cada uno de los nodos intermedios (proyección - join intermedias) se debe generar la operación local

  JOIN (TABLA1,TABLA2,FORMULA,PROYECCION,TABLA3)

  donde TABLA1 debe contener las tuplas resultantes del subárbol inferior izquierdo del join, y TABLA2 las del subárbol inferior derecho.

- Para cada uno de los nodos hojas (proyección - selección del nivel más bajo del árbol, las que se aplican sobre las relaciones) se genera un llamado al servicio

  SELECCION-PROYECCION (RELACION,FORMULA,PROYECCION,
  TABLA_RESULTADO)

3.2 - COMPILACION DE INSTRUCCIONES DIFERENTES AL SELECT

La compilación de esas instrucciones se realiza así:

a. - INSERT

Forma General: INSERT INTO relación
FROM tabla

El INSERT permite insertar a **relación** varias tuplas que se encuentran en **tabla**.

La tabla puede ser el resultado de otra consulta, pero debe tener como campos todos los atributos de **relación**, en el mismo orden en que fueron especificados al definir la relación.

Al compilar el INSERT se genera un ciclo repetitivo que llama al servicio

INSERTAR (RELACION,LLAVE INFO)

por cada registro de la tabla.

Para poder hacer ésto, por cada INSERT existente en la transaccion, se genera un registro **ZREGIn** de tipo **ZREGINSn** donde n >= 1 es un contador que se incrementa con cada INSERT que aparezca en el texto de la transacción, el cual va a contener todos los atributos de la relación involucrada, y se le asocia a la tabla un descriptor **ZARCHIn** que será de tipo **FILE OF ZREGINSn**.

Con estas estructuras de datos el ciclo repetitivo generado para un INSERT es:

```
REPITA
 Lea un registro de ZARCHIn;
 Coloque a INFO la información correspondiente;
 Coloque a LLAVE la información correspondiente;
 INSERTAR (RELACION,LLAVE,INFO);
HASTA QUE ( fin(ZARCHIn) = verdadero )
```

b. - **DELETE**

Forma General: DELETE Relación
WHERE fórmula

La compilaciòon del DELETE implica en primer lugar, la generación de un SELECT de la siguiente forma:

```
SELECT  llaves  INTO  ZZi                    i >= 1
FROM    Relación
WHERE   Fórmula
```

donde: ZZi es un nombre que se le asigna a la tabla resultado. y <llaves> incluye todos los atributos que sean llave en relación.

Este SELECT se genera internamente y se interpreta como si fuera una instrucción más de la transacción; por lo tanto en el archivo de salida se verá como un llamado al servicio

SELECCION-PROYECCION (RELACION,FORMULA,PROYECCION,TABLA_RESULTADO)

La razón por la cual se hace el SELECT, es poder tener en TABLA_RESULTADO las llaves de todas las tuplas que deben ser eliminadas, y así poder llamar dentro de un ciclo repetitivo al servicio

ELIMINAR (RELACION,LLAVE)

por cada tupla (registro) leída en el archivo ZZi (TABLA).

Para poder leer del archivo, se genera un registro ZREGDn de tipo ZREGDELn donde n >= 1, el cual tiene como campos todos los atributos que sean llave en la relación implicada, y se le asocia a la tabla un descriptor ZARCHDn de tipo **FILE OF ZREGDELn**.

Con estas estructuras de datos para un DELETE se genera lo siguiente:

```
SELECCION-PROYECCION(RELACION,FORMULA,PROYECCION,TABLA);
ZARCHDn <--- TABLA
REPITA
 Lea un registro de ZARCHDn;
 Coloque a LLAVE la información correspondiente;
 ELIMINAR(RELACION,LLAVE);
HASTA QUE ( fin(ZARCHDn) = verdadero )
```

c. - UPDATE

Forma General:

```
UPDATE   Relación
SET      atr1=exp1,.....,atrn=expn
WHERE    Fórmula
```

Al igual que en el DELETE, al compilar el UPDATE se genera inicialmente un SELECT para poder tener en una tabla las tuplas que deben ser modificadas.

El SELECT sería de la forma:

```
SELECT llaves INTO ZZi   i >= 1
       FROM   Relación
       WHERE  Fórmula
```

donde: ZZi es el nombre que se le asigna a la tabla resultado, y <llaves> incluye todos los atributos que sean llave en relación.

El SELECT se genera y se interpreta internamente tal como si fuera una instrucción más de la transacción y en el archivo de salida del módulo **Compilador** se encuentra como un llamado al servicio

SELECCION-PROYECCION (RELACION,FORMULA,PROYECCION,TABLA)

Luego de haber sido generado el SELECCIONE, se leen las tuplas (registros) del archivo ZZi y por cada una de ellas, se hace un llamado al servicio

MODIFICAR (RELACION,LLAVE,INFO)

Para poder leer del archivo, se genera un registro **ZREDUn** de tipo **ZREGUPDn** donde n >= 1, el cual contiene como campos todos los atributos que sean llave en la relación implicada, y se le asocia a la tabla un descriptor **ZARCHUn** de tipo **FILE OF ZREGUPDn.**

Con estas estructuras de datos para un UPDATE se genera lo siguiente:

```
SELECCION-PROYECCION (RELACION,FORMULA,PROYECCION,TABLA);
ZARCHUn <--- TABLA_RESULTADO
REPITA
 Lea un registro de ZARCHUn;
 Coloque a INFO la información correspondiente;
 Coloque a LLAVE la información correspondiente;
 MODIFIQUE(RELACION,LLAVE,INFO);
HASTA QUE ( fin(ZARCHUn) = verdadero )
```

4. CONCLUSIONES

La realización del módulo COMPILADOR del sistema TESEO, permite ofrecer al diseñador de aplicaciones distribuidas un lenguaje relacional con total transparencia a la fragmentación y distribución de la Base de Datos Distribuida, y que aprovecha todas las facilidades ofrecidas por TESEO (facilidades expuestas en [Franky 87]).

La implementación de este módulo se encuentra actualmente a nivel de prototipo (en microcomputador), pero se espera una implementación más ambiciosa en la siguiente fase del sistema TESEO que será realizada sobre una red local Ethernet de equipos Vax.

BIBLIOGRAFIA

[Franky 87]: Maria Consuelo Franky,
"TESEO: Ambiente de programación de Sistemas Transaccionales Distribuidos",
XIII Conferencia Latinoamericana de Informática (CLEI),
Bogotá- Colombia, Nov. 1987

[Ceri 84]: S.Ceri, G. Pelagatti,
"Distributed Databases: principles and systems",
McGraw-Hill, 1984

[Alvarado y Muñoz 86]: Armando Alvarado, Adriana Muñoz,
"Sistema Generalizado de Transacciones Distribuidas sobre Bases de Datos Homogéneas", tesis de pregrado en Ingeniería de Sistemas ISC-86-I-9,
Universidad de los Andes, Sept. 1986

[Chamberlin 76]: D.D. Chamberlin et al.,
"SEQUEL 2: A unified aproach to Data Definition, Manipulation and Control", IBM Journal of Research and Development,
Vol. 20 No. 6, Nov. 1976