# TESEO : AMBIENTE DE PROGRAMACION DE SISTEMAS TRANSACCIONALES DISTRIBUIDOS

**MARIA CONSUELO FRANKY**

Grupo de investigación SINBAD
Dpto. de Ingeniería de Sistemas
Universidad de los Andes
Apartado aéreo 4976
Bogotá - Colombia

**RESUMEN :**

Se presenta un sistema que permite al diseñador de una Base de Datos Distribuida especificar el esquema de fragmentación y distribución de los datos. Además le permite programar las transacciones que podrán ser solicitadas por los usuarios finales; esta programación se hace en términos de la base de datos global, tal como se haría para una base de datos centralizada. A partir de las especificaciones del diseñador, el sistema genera el protocolo de ejecución de las transacciones en un ambiente distribuido asegurando atomicidad y control de concurrencia. Este Ambiente de Programación de Sistemas Transaccionales Distribuidos "TESEO" es entonces un Manejador de Bases de Datos Distribuidas que permite que el diseñador se despreocupe totalmente de las técnicas de procesamiento distribuido y pueda concentrarse en los aspectos propios de su aplicación.

# 1. INTRODUCCION

**TESEO: Ambiente de Programación de Sistemas Transaccionales Distribuidos** es un sistema en desarrollo en la Universidad de los Andes, actualmente en fase de prototipo, cuyo propósito es automatizar la especificación e implementación de aplicaciones transaccionales que van a operar en ambientes distribuidos (i.e. redes locales o redes geográficamente dispersas).

TESEO permite al diseñador especificar el esquema de fragmentación y de distribución de los datos que maneja la aplicación. Le permite además programar las transacciones que desea ofrecer a los usuarios finales. Para programar transacciones el diseñador no requiere conocer técnicas de procesamiento distribuido, solo las técnicas usuales del manejo de bases de datos.

La implementación de una aplicación específica a través de TESEO implica 3 etapas : en la etapa de *especificación* el diseñador especifica los esquemas de la base de datos distribuida y programa las transacciones; en la etapa de *generación* el sistema genera un programa que permite ejecutar esas transacciones en un ambiente distribuido; finalmente en la etapa de *ejecución* los usuarios finales solicitan transacciones desde los distintos sitios que conforman el ambiente distribuido.

El presente artículo está dividido en 8 partes : a continuación de esta introducción, la parte 2 presenta las características actuales del sistema considerado como Sistema Manejador de Bases de datos Distribuidas. La parte 3 presenta los módulos en que está dividido el sistema. Las partes 4 y 5 explican cómo el diseñador interactúa con el sistema para especificar los esquemas global, de fragmentación y de distribución de los datos y los algoritmos de las transacciones. La parte 6 explica la generación del programa que implementa el protocolo de ejecución distribuida de lastransacciones a partir de las especificaciones del diseñador. En las partes 7 y 8 se presentan las extensiones previstas para este sistema así como algunas conclusiones.

# 2. CARACTERISTICAS ACTUALES DEL SISTEMA

TESEO es un Sistema Manejador de Bases de Datos Distribuidas (SMBDD) con las siguientes características (en referencia a la arquitectura de B.D.D. presentada en *[Cer 84]* ):

**ESQUEMA GLOBAL :**

La Base de datos Global sigue el modelo Relacional; el diseñador debe entonces especificar las relaciones globales que va a manejar la aplicación.

**ESQUEMA DE FRAGMENTACION :**

El sistema permite al diseñador fragmentar las relaciones globales de forma horizontal simple, es decir, por agrupación de tuplas que según el valor de sus atributos deben estar juntas para la aplicación.

**ESQUEMA DE DISTRIBUCION :**

El sistema permite al diseñador asignar cada fragmento a uno o varios sitios del ambiente distribuido. Para efectos de actualización una de las copias se considera "primaria" y las demás "secundarias".

**SISTEMA MANEJADOR DE BASE DE DATOS LOCAL :**

El sistema asume **homogeneidad** de los SMBD locales y por lo tanto no requiere Esquemas de Traducción. En la versión actual se implementó para cada sitio un SMBD sencillo capaz de realizar las operaciones básicas de consulta y actualización de fragmentos locales. Estas operaciones permiten insertar, modificar y eliminar una tupla de un fragmento local, así como seleccionar y proyectar las tuplas de un fragmento que cumplen con determinada condición.

Se ha previsto que en el futuro TESEO se acople con diversos SMBD comerciales disponibles para los sitios del ambiente distribuido.

**NIVEL DE TRANSPARENCIA OFRECIDA AL DISEÑADOR :**

Una vez que el diseñador especifica los Esquemas Global, de Fragmentación y de Distribución, puede programar transacciones en términos de las relaciones globales, es decir, de forma transparente a la fragmentación y distribución de datos.

En la versión descrita en este artículo el lenguaje que utiliza el diseñador para programar transacciones es **Turbo-Pascal** con servicios provistos por TESEO para manipular relaciones globales.

**FUNCIONES AUTOMATICAS INCLUIDAS EN LA APLICACION GENERADA :**

El sistema TESEO incorpora en la aplicación generada las siguientes funciones :

a- asignación de estampillas a las transacciones que se ejecutan asegurando un orden total
b- selección de fragmentos que participan en una consulta
c- distribución de instrucciones de cada transacción a los sitios participantes
d- atomicidad y recuperación de transacciones bajo un protocolo de validación
e- actualización consistente de todas las copias de un fragmento
f- control de concurrencia de transacciones utilizando candados y listas de espera para asegurar la serialización legal de transacciones
g- prevención de abrazos mortales

# 3. MODULOS DEL SISTEMA

La Figura 1 de la página siguiente muestra los módulos componentes del Ambiente de Programación de Sistemas Transaccionales Distribuidos TESEO.

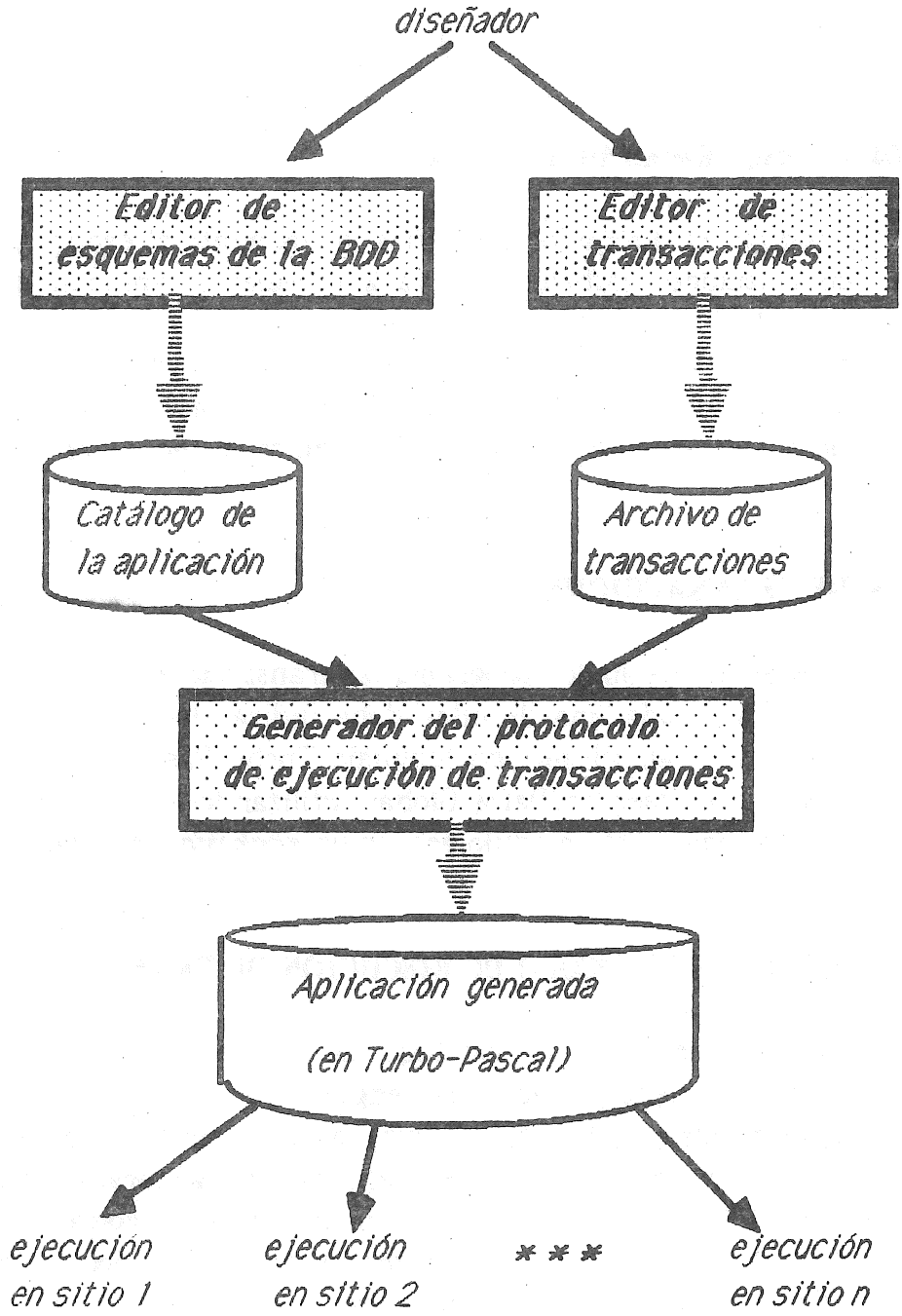


**Figura 1 : MODULOS DE TESEO**

Una explicación breve de los módulos es la siguiente :

Módulo **EDITOR DE ESQUEMAS DE LA B.D.D.** :

A través de este módulo el diseñador especifica inicialmente el esquema global de la B.D.D., es decir, las relaciones globales que va a manejar la aplicación. En seguida especifica el esquema de fragmentación de esas relaciones dando fórmulas de selección de tuplas para constituir fragmentos. Por último especifica el esquema de distribución asignando cada fragmento a uno o varios sitios lógicos. Este módulo genera el **catálogo** de la aplicación que posteriormente se va a duplicar en cada uno de los sitios.

Módulo **EDITOR DE TRANSACCIONES** :

A través de este módulo el diseñador programa las transacciones que quiere ofrecer a los usuarios finales. Para cada una de ellas debe indicar los parámetros que debe suministrar el usuario y el algoritmo en lenguaje Turbo-Pascal. Los algoritmos se expresan en términos de la Base de Datos Global utilizando los servicios del sistema para manipular las relaciones. Este módulo genera un **archivo de transacciones.**

Módulo **GENERADOR DEL PROTOCOLO DE EJECUCION DE TRANSACCIONES:**

Este módulo toma como entrada el catálogo de la aplicación y el archivo de transacciones y genera un programa fuente en lenguaje Turbo-Pascal, con la lógica para poder ejecutar transacciones en ambiente distribuido asegurando atomicidad y control de concurrencia. En la versión actual de TESEO el programa generado incluye un nivel de comunicación específico para una red local de microcomputadores conectados en anillo; en esta red se asume que a cada microcomputador (sitio físico) corresponde un único sitio lógico. En el futuro se planea generalizar el nivel de comunicación para que el diseñador pueda especificar la interfaz de TESEO con los servicios de comunicación que ofrezca la red específica en la cual va a operar la aplicación.

Para tener el sistema en funcionamiento, debe compilarse el programa resultante y ponerse en ejecución en cada uno de los sitios.

# 4. ESPECIFICACION DE ESQUEMAS DE LA BASE DE DATOS DISTRIBUIDA

Cuando se programa una aplicación con TESEO, el diseñador debe empezar por definir el **Esquema Global** especificando las relaciones globales que va a manejar la aplicación: para cada relación debe indicar su nombre, sus atributos y los tipos correspondientes; también debe indicar cúal es la llave (uno o más atributos). Por ejemplo, en una aplicación bancaria se especificaría la relación *CUENTAS* así :

atributos de *CUENTAS* :

| | | | |
|---|---|---|---|
| *código* | : tipo | *entero* | *( llave )* |
| *saldo* | : | *real* | |
| *propietario* | : | *cadena* | |
| *saldo inicial* | : | *real* | |

A continuación el diseñador debe especificar el **Esquema de Fragmentación** indicando para cada relación cómo deben agruparse las tuplas en fragmentos a través de fórmulas de selección. El objetivo aquí es agrupar tuplas que deben estar juntas desde el punto de vista de la aplicación; sin embargo, se sabe que hacerlo de manera óptima es un problema muy complejo de optimización que no tiene solución única *[Ceri-84]*.

La versión actual del sistema permite únicamente fragmentación horizontal simple de cada relación. Por ejemplo, para la relación *CUENTAS* se podrían indicar los siguientes fragmentos :

| | | |
|---|---|---|
| $CUENTAS_1$ | : | tuplas con *saldo inicial* $\geq$ 1.000.000 |
| $CUENTAS_2$ | : | tuplas con ( *saldo inicial* $<$ 1.000.000 ) y ( *código* $\leq$ 100 ) |
| $CUENTAS_3$ | : | tuplas con ( *saldo inicial* $<$ 1.000.000 ) y ( 100 $<$ *código* $\leq$ 200 ) |
| $CUENTAS_4$ | : | tuplas con ( *saldo inicial* $<$ 1.000.000 ) y ( *código* $>$ 200 ) |

En la versión actual del sistema es reponsabilidad del diseñador asegurar una

fragmentación correcta de cada relación, es decir, cada tupla debe quedar asignada exactamente a un solo fragmento. De esta forma los fragmentos son disyuntos y su unión es igual a la Base de Datos Global.

En el futuro se piensan incluir esquemas más generales de fragmentación, como es la fragmentación horizontal derivada, la vertical y combinaciones de las anteriores.

Finalmente el diseñador debe especificar el **Esquema de Distribución** de cada relación, asignando cada fragmento a uno o a varios sitios lógicos del ambiente distribuido en el cual va a ejecutarse la aplicación. Por ejemplo, los fragmentos de *CUENTAS* podrían ser asignados así :

$CUENTAS_1$ : $sitio_1$ , $sitio_2$ , $sitio_3$ , $sitio_4$
$CUENTAS_2$ : $sitio_2$
$CUENTAS_3$ : $sitio_3$
$CUENTAS_4$ : $sitio_4$

En el caso de un fragmento replicado su primera copia se denomina "**primaria**" y las demás "**secundarias**".

Una vez especificados por el diseñador los Esquemas Global, de Fragmentación y de Distribución, el sistema genera el **Catálogo de la aplicación** que posteriormente se replicará en cada uno de los sitios de ejecución. Este catálogo permitirá primordialmente la localización de datos que participan en una transacción solicitada por el usuario final.

# 5. ESPECIFICACION DE TRANSACCIONES

En la versión del sistema descrita en este artículo el lenguaje ofrecido al diseñador para programar transacciones es **Turbo-Pascal**.

El diseñador debe comenzar por enumerar las transacciones que podrán ser solicitadas por los usuarios finales : para cada una debe indicar nombre, parámetros y tipos. Por ejemplo, en la aplicación bancaria se podrían ofrecer, entre otras, las siguientes

transacciones:

transacción *ABRIR CUENTA* :
parámetros *nuevo-código* : tipo *entero*
*nombre-de-propietario* : tipo *cadena*
*saldo-al-comenzar* : tipo *real*

transacción *TRANSFERENCIA* :
parámetros *cuenta-fuente* : tipo *entero*
*cuenta-destino* : tipo *entero*
*cantidad* : tipo *real*

Conocidas las diferentes transacciones, el sistema TESEO incluye en la aplicación generada un "despachador de comandos" *[Toro 87]* que se encargará de ofrecer al usuario un menú con las diversas transacciones disponibles. En tiempo de ejecución, cada vez que el usuario final seleccione una transacción, el despachador de comandos pedirá los parámetros respectivos, activará la transacción respectiva y posteriormente presentará al usuario los resultados de ésta.

En un futuro se prevee brindar al diseñador la posibilidad de escoger entre divesos tipos de despachadores *[Toro 87]* para diversos tipos de terminales, e inclusive, permitir que el usuario programe su propio despachador.

Una vez enumeradas las transacciones, el diseñador debe especificar el algoritmo en Turbo-Pascal de cada una de ellas : el diseñador programa las transacciones en términos de las relaciones globales como si la Base de Datos fuera centralizada, despreocupándose aquí de la fragmentación y distribución de los datos. Para lograr esta transparencia, la programación de la manipulación de relaciones globales debe hacerse únicamente a través de procedimientos "**servicios**" provistos por el sistema TESEO . Estos servicios y sus parámetros son básicamente los siguientes :

- Servicio **Insertar** **(nombre de relación, valor de atributos)**
  Genera el código para que en tiempo de ejecución se inserte una nueva tupla en la relación nombrada. En ejecución la aplicación generada deducirá entonces el fragmento de la relación en el cual debe insertarse la nueva tupla y se encargará de hacer la inserción en cada una de las copias de ese fragmento.

  Es importante resaltar que el diseñador especifica el valor de cada atributo indicando la constante, variable o parámetro que contiene el valor. El nombre la relación debe ser una constante.

- Servicio **Eliminar** **(nombre de relación, valor de llave)**
  Genera el código para que en tiempo de ejecución se elimine la tupla correspondiente a la llave especificada de la relación nombrada. En ejecución la aplicación generada eliminará la tupla de todas las copias del fragmento al cual pertenece. En forma similar al servicio anterior, el diseñador especifica el valor de la llave indicando la constante, variable o parámetro que contiene tal valor; el nombre de la relación debe ser una constante.

- Servicio **Modificar** **(nombre de relación, valor de llave, nombres de atributos que cambian, nuevos valores de atributos)**
  Genera el código para que en tiempo de ejecución se modifique la tupla de la relación nombrada correspondiente a la llave especificada. Así, en ejecución la aplicación generada modificará la tupla especificada en todas las copias del fragmento al cual pertenece. El nombre de la relación y los nombres de los atributos que se quieren modificar deben ser constantes mientras que el valor de la llave y los nuevos valores de los atributos pueden ser constantes, variables o parámetros.

- Servicio **Selección-Proyección** **(nombre de relación, fórmula de selección, atributos de proyección, nombre de archivo respuesta)**
  Genera el código para que en tiempo de ejecución en el archivo respuesta se copien las tuplas de la relación nombrada que cumplen la fórmula de selección (dada en forma normal conjuntiva). En ejecución la aplicación generada deducirá los fragmentos a los cuales pertenecen las tuplas y realizará la selección en una copia (i.e. sitio) de cada uno de esos fragmentos. Hecho ésto, la aplicación generada efectuará la proyección sobre los atributos nombrados y los almacenará en el archivo de respuesta.

  Si la fórmula de selección es nula el servicio equivale a una proyección simple; de manera análoga, si los atributos de proyección son nulos el servicio equivale a una selección simple.

A continuación se da como ejemplo el algoritmo dado por el diseñador para la transacción *TRANSFERENCIA* mencionada al comienzo de la sección 4. Se observará el uso del caracter " / " para separar los distintos valores que pueden ir en un parámetro, por ejemplo, los nombres de los atributos de proyección en un servicio Selección_Proyección, o los atributos que conforman la llave en un servicio Modificar.

```
transacción TRANSFERENCIA  ( cuenta_fuente  : integer;
                             cuenta_destino : integer ;
                             cantidad       : real);

    Var      respuesta                 : file of real;
             saldo_actual, nuevo_saldo : real;

    Procedure Avisar ( texto : string );
    begin
          writeln; writeln; writeln;
          writeln ( '          ', texto )
    end Avisar;

begin
    Selección_proyección ( CUENTAS, código = cuenta_fuente, /saldo/, respuesta ) ;
    read ( respuesta, saldo_actual );
    If   saldo_actual ≥ cantidad  then
         begin
            nuevo_saldo := saldo_actual - cantidad ;
            Modificar ( CUENTAS, /cuenta_fuente/, /saldo/, /nuevo_saldo/);
            Selección_proyección ( CUENTAS, código = cuenta_destino,
                                   /saldo/, respuesta );
            read ( respuesta, saldo_actual) ;
            nuevo_saldo := saldo_actual + cantidad ;
            Modificar ( CUENTAS, /cuenta_destino/, /saldo/,/nuevo_saldo/ )
         end
    else Avisar ( ' transacción rechazada : saldo insuficiente ')
end
fin_transacción
```

Como se puede observar en el ejemplo anterior, las transacciones pueden incluir procedimientos locales (como *Avisar)* ; sin embargo, el sistema impone como restricción que estos procedimientos no incluyan servicios de manipulación de las relaciones globales.

# 6. GENERACION DEL PROTOCOLO DE EJECUCION DE TRANSACCIONES PARA AMBIENTE DISTRIBUIDO

Una vez que el diseñador ha interactuado con TESEO para especificar los esquemas de la B.D.D. y las transacciones, el sistema invoca el **Generador del Protocolo de Ejecución de Transacciones.** Este módulo toma como entrada el catálogo de la aplicación y el archivo de transacciones, producidos por los módulos anteriores, y genera la aplicación que habrá de ser ejecutada en cada uno de los sitios del ambiente distribuido.

## 6.1 CARACTERISTICAS DEL PROTOCOLO DE EJECUCION DE TRANSACCIONES

Cuando se genera la aplicación distribuida cada una de las transacciones que había especificado el diseñador es descompuesta automáticamente en diversos trozos. Estos trozos están básicamente delimitados por las invocaciones a los servicios de manipulación de la base de datos global. Los diversos trozos obtenidos y las invocaciones a los servicios de la base de datos son entonces "sumergidos" dentro de un esquema de ejecución que asegura los siguientes aspectos:

### *ATOMICIDAD Y RECUPERACION DE TRANSACCIONES :*

Una vez que una transacción comienza a ejecutarse por demanda de un usuario final se asegura su **atomicidad**, es decir, sus efectos se realizarán o bien totalmente (si no hay fallas del sistema), o bien ninguno de ellos (en caso falla). Para lograr esta atomicidad cada transacción se ejecuta siguiendo un **protocolo de validación en 2 etapas** : de manera general, en una primera etapa la transacción reserva los datos que necesita y construye "listas de intención" en cada uno de los sitios participantes, incluyendo en esas listas las modificaciones que se desean hacer a los datos locales; en la segunda etapa la transacción modifica efectivamente los datos y los libera. Este protocolo permite entonces anular los efectos de la transacción por simple destrucción de las listas de intenciones si ocurre una falla que interrumpa la transacción durante su primera etapa *[Le Lann 81]*.

En el caso de que algunos de los datos modificados por una transacción sean fragmentos con copias se aplica una adaptación del protocolo de *[Stonebraker 79]* : durante la primera etapa se reservan los fragmentos primarios y se construyen listas de

intenciones en los sitios correspondientes; durante la segunda etapa se difunden las intenciones a los fragmentos secundarios y se modifican efectivamente. Cuando un sitio estuvo aislado al recuperarse debe actualizar sus fragmentos secundarios pidiendo las modificaciones faltantes a los sitios donde se encuentran los fragmentos primarios.

Para consultar un fragmento que participa en una Selección_Proyección se reserva el fragmento primario; adicionalmente, si existe un fragmento secundario a nivel local, éste también se reserva y la consulta se realiza sobre el fragmento local.

### *CONTROL DE CONCURRENCIA DE TRANSACCIONES :*

En la aplicación generada se asegura el control de concurrencia de transacciones distribuidas que tratan de accesar los mismos datos en uno o varios sitios. Para ello se sigue un protocolo de control parcialmente distribuido que incluye :

* uso de **estampillas** para identificar transacciones (y sus mensajes correspondientes) asegurando un orden total, según el método de *[Lamport 78]*.
* uso de **candados** de reservación y **listas de espera** en cada sitio para serializar de forma consistente las transacciones que quieren accesar los mismos datos; se logran así las propiedades definidas por *[Eswaran 76]* : serialización "legal" de transacciones "bien-formadas" y a "dos-fases".
* **prevención de abrazos mortales** entre transacciones aplicando el método "herir o esperar" de *[Rosenkratz 78]* el cual implica anular una transacción que durante su primera etapa entra en conflicto con otra transacción de estampilla menor.

### *SIMULACION DE PROCESOS PARALELOS:*

La aplicación generada implementa el protocolo de ejecución de transacciones a través de **varios procesos por sitio**, encargados de asegurar la atomicidad y control de concurrencia entre transacciones. En la versión actual TESEO genera aplicaciones para Sistemas Operacionales monoproceso (CP/M, MS-DOS); en consecuencia, el programa generado para cada sitio contiene un núcleo de control que da turnos sucesivos a los diversos procesos de acuerdo a los mensajes que van llegando al sitio. Más específicamente, la programación de estos procesos se hizo siguiendo una metodología derivada de las redes gráficas de Nutt, la cual se presenta ampliamente en *[Franky 86]*.

***INCLUSION DE UN NIVEL DE COMUNICACION SIMPLE :***

El programa generado incluye un nivel de comunicación simple que permite intercambiar mensajes entre sitios lógicos asociados a microcomputadores conectados en topología de anillo. En un futuro se planea que TESEO disponga de diversas librerías de comunicación que se adapten a diferentes estándares y topologías de redes.

## 6.2 PROCESOS GENERADOS

El programa generado para cada sitio corresponde a la ejecución simultánea de 3 procesos que aseguran los objetivos del protocolo de ejecución de transacciones; estos procesos se sincronizan por mensajes utilizando el nivel de comunicación para enviar y recibir mensajes, ya sea a nivel local (procesos dentro de un mismo sitio) o a nivel remoto (procesos dentro de sitios diferentes).

La figura 2 de la siguiente página ilustra la relación entre procesos de un sitio y las principales estructuras de datos que manejan. Las funciones principales de los procesos son las siguientes :

***PROCESO DESPACHADOR :***

Está encargado de realizar la **interfaz con el usuario y de coordinar cada transacción** demandada por él.

El proceso Despachador muestra por pantalla el Menú de Transacciones de la aplicación, detecta la clase de transacción que solicita el usuario, captura los parámetros correspondientes y va invocando sucesivamente a los diferentes trozos en que fue descompuesta la transacción. Cuando se invoca alguno de los servicios de manipulación de las relaciones globales (i.e. al final de cada trozo) el Despachador envía el mensaje local correspondiente al proceso Monitor y se queda esperando el mensaje de respuesta para poder proseguir con la ejecución de otro trozo.

Adicionalmente, cuando el proceso Despachador termina los trozos de una transacción difunde el mensaje de validación a los sitios participantes a fin de iniciar la segunda etapa de la transacción en la cual deben modificarse efectivamente los datos. En caso de falla, también es el proceso Despachador el encargado de difundir el mensaje de anulación.

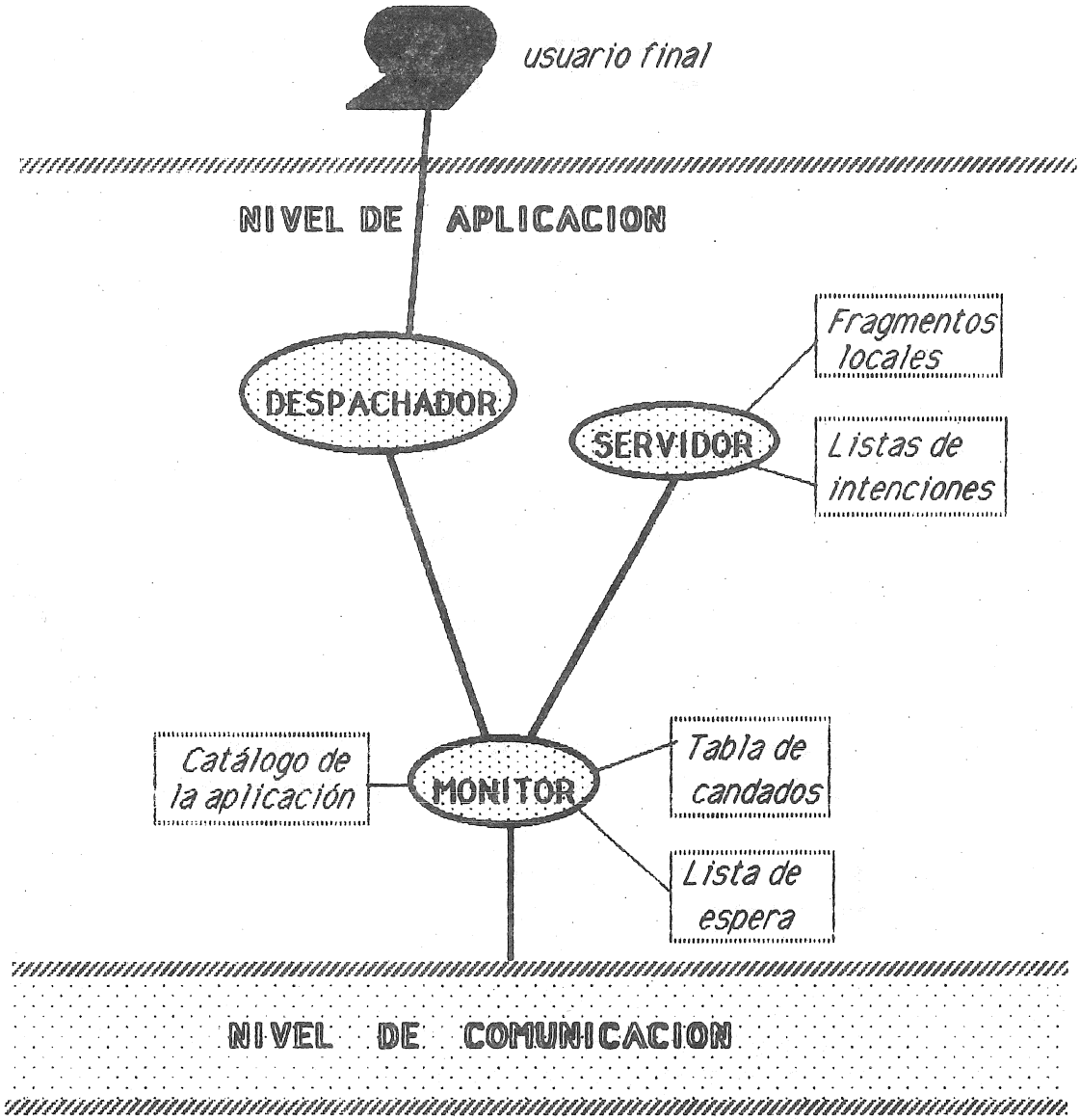


Figura 2 : **PROCESOS GENERADOS Y ESTRUCTURAS DE DATOS QUE MANEJAN**

*PROCESO MONITOR :*

Este proceso está encargado de **localizar los datos** que intervienen en cada transacción y de **controlar la concurrencia** entre transacciones que quieren accesar datos locales.

Para realizar la primera de estas funciones el proceso Monitor maneja el Catálogo de la Aplicación, el cual representa los esquemas global, de fragmentación y de distribución de los datos de la aplicación. Esta información permite deducir los sitios que deben participar en la resolución de un servicio invocado por una transacción.

Así por ejemplo, cuando el proceso Despachador invoca el servicio *Selección_Proyección*, el proceso Monitor deberá recibir el mensaje local correspondiente. Con base en la información que lleva este mensaje (i.e. la fórmula de selección) y en el Catálogo de la Aplicación el proceso Monitor deberá deducir los sitios participantes. A estos sitios participantes el proceso Monitor envía entonces un mensaje externo *Selección_Proyección* destinado a los procesos Monitores correspondientes.

Posteriormente cuando lleguen mensajes de respuesta de los sitios participantes, en el sitio origen de la transacción se irán reuniendo las tuplas seleccionadas en un archivo local. Finalmente el proceso Monitor pasará un mensaje local de respuesta al proceso Despachador para avisarle que ya se recibieron los datos que había solicitado, y permitirle así que pueda continuar su ejecución.

Para realizar la segunda función relativa al control de concurrencia, el proceso Monitor maneja una Tabla de Candados para registrar el estado de reservación de los datos locales y así poder detectar conflictos de concurrencia. Siguiendo con el ejemplo, cuando un proceso Monitor recibe un mensaje externo *Selección_Proyección* examina la Tabla de Candados para ver si el fragmento local sobre el cual se quiere hacer la selección ya está reservado por otra transacción en modo conflictivo : si no hay conflicto el proceso Monitor registra en la Tabla la nueva reservación del fragmento y envía un mensaje local al proceso Servidor para que éste efectúe el servicio; la respuesta que el proceso Servidor entregue será enviada por el proceso Monitor al sitio origen de la transacción.

En caso de conflicto de concurrencia, el proceso Monitor coloca el mensaje externo de petición de servicio en una Lista de Espera, postergando la entrega del mensaje al proceso Servidor para cuando terminen las transacciones que originan el conflicto. Sin embargo, si una de esas transacciones posee estampilla mayor que la transacción que

está solicitando el servicio, el Monitor provocará la anulación de la primera con el fin de prevenir un abrazo mortal (método "herir o esperar" de *[Rosenkratz 78]*).

***PROCESO SERVIDOR :***

Este proceso está encargado de **accesar los fragmentos de datos locales**, **manejar las listas de intenciones** y asegurar la **actualización consistente de copias de fragmentos.**

Cuando este proceso Servidor recibe un mensaje de servicio *Selección_Proyección* selecciona tuplas del fragmento local correspondiente y conforma con ellas un archivo de respuesta que el proceso Monitor local deberá transmitir al sitio origen de la transacción. Cuando recibe un mensaje de servicio *Insertar, Eliminar* o *Modificar* registra la intención correspondiente en la Lista de Intenciones, y únicamente acualiza efectivamente los datos cuando recibe el mensaje de Validación de una transacción : solo en ese momento realiza las intenciones correspondientes a esa transacción.

Para el caso de fragmentos replicados, es el proceso Servidor el encargado de difundir las intenciones relativas a un fragmento primario local hacia los sitios donde estén localizadas sus copias secundarias. Esta difusión ocurre después de haber actualizado el fragmento primario. Para asegurar la consistencia mutua entre copias de un fragmento, el proceso Servidor de un sitio que se recupera después de una falla debe solicitar las intenciones acumuladas por el fragmento primario de cada una de los fragmentos secundarios que posea.

## 6.3 DESCOMPOSICION DE LOS ALGORITMOS DE TRANSACCIONES

En términos generales la programación de los procesos **Monitor** y **Servidor** se hizo enunciando las acciones que se debían realizar cada vez que se recibiera un mensaje de petición de servicio ( *Insertar, Eliminar, Modificar* o *Selección_Proyección)* , o un mensaje de respuesta a un servicio.

A diferencia de lo anterior, la programación del proceso **Despachador** debe seguir en cada caso la lógica del algoritmo de la transacción escogida por el usuario, y al mismo tiempo debe generar un mensaje de petición de servicio (para enviar al Monitor) cada vez que en la transacción se accesa la base de datos global. Esta programación plantea entonces el problema de descomponer el algoritmo de cada transacción en una secuencia de "trozos" que serán ejecutados uno a uno, en forma intercalada con el

envío de mensajes de petición de servicios y la recepción de mensajes de respuestas. Más exactamente en cada turno de ejecución el proceso Despachador recibe y procesa el mensaje de respuesta al servicio que solicitó en el turno anterior, ejecuta un trozo según la secuencia establecida, y finaliza el turno enviando al Monitor un mensaje de solicitud de servicio.

La descomposición del algoritmo de cada transacción plantea adicionalmente el problema de la preservación del estado del proceso Despachador cada vez que pierda el turno de ejecución al enviar un mensaje de petición de servicio al Monitor, es decir, al final de cada trozo. En particular, si se está ejecutando un trozo correspondiente a una parte de una estructura de control condicional o iterativa (i.e. IF o WHILE) y se encuentra una petición de servicio, se debe poder registrar en qué estado queda la ejecución para que el siguiente trozo pueda retomarlo después de la recepción y tratamiento del mensaje de respuesta al servicio solicitado.

En el estado actual de TESEO se permite que una transacción invoque servicios de manipulación de la B.D.D. desde una estructura de control condicional o iterativa situada a nivel de anidamiento cero pero no lo permite desde estructuras de control anidadas a niveles mayores.

El siguiente ejemplo ilustra el algoritmo de una transacción y su descomposición en trozos realizada por TESEO. El algoritmo contiene además de instrucciones simples de Turbo-Pascal, una estructura condicional y una estructura iterativa que contienen invocaciones de servicios :

```
transacción T (parámetros .....)
   declaraciones .... ;
begin
   instrucción a ;
   Selección_Proyección (parámetros...) ;
   instrucción b ;
   If condición1  then
        begin
           instrucción c ;
           Modificar (parámetros....) ;
           instrucción d
        end
   else
        begin
            instrucción e
        end  ;
```

```
    instrucción f ;
    While condición2 do
    begin
        instrucción g ;
        Insertar (parámetros....) ;
        instrucción h ;
    end ;
end
fin_transacción
```

**Descomposición de la transacción *T* en trozos :**

**trozo 1 de *T* :**

```
begin
    instrucción a ;
    enviar (Monitor, "Selección_Proyección", parámetros ....)
end
```

**trozo 2 de *T*:**

```
begin
    instrucción b ;
    var1 := condición1 ;
    If var1 then
        begin
            instrucción c ;
            enviar (Monitor, "Modificar", parámetros....)
        end
    else
        begin
            instrucción e ;
            enviar (Despachador, "nulo" )
        end
end
```

**trozo 3 de T :**

```
begin
   If  var1  then
        begin
            instrucción d
        end ;
   enviar (Despachador, "nulo")
end
```

**trozo 4 de *T* :**

```
begin
    instrucción f
    enviar (Despachador, "nulo" )
end
```

**trozo 5 de *T* :**

```
begin
   var2 := condición2 ;
   If  condición2  then
        begin
            instrucción g ;
            enviar  (Monitor, "Insertar", parámetros.....)
        end
   else
        begin
            núm_trozo  :=  núm_trozo + 1 ;
            enviar (Despachador, "nulo" )
        end
end
```

**trozo 6 de *T* :**

```
begin
    instrucción  h ;
    num_trozo  :=  núm_trozo - 2
    enviar (Despachador, "nulo" )
end ;
```

**Aclaraciones :**

Para facilitar la descomposición del algoritmo de la transacción, el sistema impone un fin de trozo antes de empezar una estructura de control condicional o iterativa, y al final de las mismas. Cuando ese fin de trozo no coincide con una petición de servicio se genera un envío de mensaje "nulo" dirigido al mismo proceso Despachador para permitir que este proceso pueda retomar el control cuando se le de turno de ejecución (de acuerdo a la metodología de programación que se siguió *[Franky 86]*, se otorga turno de ejecución a un proceso sólo si existe un mensaje destinado a él).

Por otra parte, el proceso Despachador maneja una variable estática *num_trozo* que indica cúal es el trozo que se debe ejecutar al recibir control, una vez que haya tratado el último mensaje de respuesta recibido (en caso del mensaje "nulo" ese tratamiento es también nulo). Teniendo en cuenta que el proceso Despachador incrementa en 1 la variable *num_trozo* antes de ejecutar el trozo indicado, es necesario alterar el valor de esa variable en los trozos correspondientes a una estructura iterativa *While* a fin de seguir la lógica de la estructura (i.e. poder salir del *While* cuando la condición ya no se cumple, o poder retornar al trozo que evalúa la condición para realizar un nuevo ciclo del *While*).

# 7. EXTENSIONES PREVISTAS PARA EL SISTEMA

La primera versión del sistema TESEO fue implementada en el trabajo de tesis de pregrado de *[Alvarado y Muñoz 86]* . Posteriormente el trabajo de *[Quintero 87]* amplió el protocolo de ejecución de transacciones para incluir el manejo de copias de fragmentos, aspecto no considerado en la versión inicial.

Además se han hecho otros 3 trabajos de tesis que desarrollan aspectos adicionales que no fueron descritos en este artículo :

- El trabajo de *[Morales 87]* implementó el manejo dinámico del catálogo de la aplicación : se trata aquí de ofrecer al diseñador un conjunto de transacciones para modificar dinámicamente el esquema global, de fragmentación o de distribución de una aplicación existente. Este tipo de transacciones permite no solamente la actualización del catálogo (que está replicado en todos los sitios del sistema), sino

también, la migración de datos de un sitio a otro, o la replicación de los datos de un fragmento en otro sitio. Para ejecutar estas transacciones del sistema se aplicó un protocolo adaptado del propuesto por *[Ellis 77]*.

El trabajo de *[Pedraza 87]* añade a la versión inicial del sistema un manejo flexible del control de concurrencia : teniendo en cuenta los diferentes servicios de manipulación de relaciones globales, el sistema maneja candados de reservación no solo a nivel de fragmento sino también a nivel de tupla, elevando así la concurrencia efectiva. Para ello se utilizan los candados de "intención" propuestos por *[Gray 75]*.

Además se ofrece al diseñador 3 opciones para escoger la técnica de tratamiento de abrazos mortales :

* prevención según la técnica "herir o esperar" de *[Rosenkratz 78]* expuesta en este artículo
* prevención por "timeouts" *[Ceri 84]*
* detección por construcción de un grafo global en un sitio "central", con posibilidad de que otro sitio asuma ese rol cuando el primero se aisla (basado en *[Stonebraker 79]* ).

El sistema tiene entonces la posibilidad de generar 3 protocolos diferentes de ejecución de transacciones. Para que el diseñador pueda evaluar qué tan acertado es escoger determinado protocolo para su aplicación, el sistema lleva algunas estadísticas sencillas cuando se ejecutan las transacciones (como número de mensajes intercambiados entre sitios).

El trabajo de *[Levy 87a, 87b]* añade a la versión inicial del sistema un lenguaje relacional tipo SQL *[Chamberlin 76]* para que el diseñador pueda expresar más cómodamente los algoritmos de las transacciones. Los algoritmos se pueden entonces expresar en *Turbo-Pascal* de forma transparente a los esquemas de fragmentación y distribución permitiendo además la manipulación de las relaciones globales a través de las siguientes instrucciones relacionales :

*INSERT INTO* relación
*FROM* tabla

*SELECT* atributos *INTO* tabla
*FROM* relaciones
*WHERE* fórmula_de_selección

*UPDATE* relación
*SET* atributo$_1$ = expresión$_1$, ..., atributo$_n$ = expresión$_n$
*WHERE* fórmula_de_selección

*DELETE* relación
*WHERE* fórmula_de_selección

En este proyecto se preprocesa el texto de los algoritmos de transaciones para transformar cada instrucción relacional en un conjunto de invocaciones a los servicios de más bajo nivel que ofrece la versión inicial del sistema. Se incluye también una optimización de la consulta derivada de cada *SELECT*, *UPDATE* o *DELETE* mediante una combinación de las transformaciones presentadas en *[Ceri-84]*.

Los trabajos realizados alrededor del sistema TESEO son todavía prototipos de experimentación, pues están implementados en microcomputadores con serias limitaciones de memoria, conectados por una red local muy simple y poco eficiente. A corto plazo se planea implementar TESEO en una red local *Ethernet* de equipos *Vax*. Se espera así ofrecer en un mediano plazo un sistema eficiente que pueda ser usado en el diseño de aplicaciones destinadas a organizaciones de tamaño considerable.

A más largo plazo se espera incluir en TESEO las siguientes ampliaciones :

* Encadenar TESEO con diversos SMBD locales, disponibles a nivel comercial, que realicen de forma eficiente el rol del proceso Servidor descrito en este artículo.
* Disponer de diversas librerías de comunicación que permitan adaptar TESEO a diferentes topologías y protocolos de redes.
* Disponer de un repertorio amplio de Despachadores (i.e. manejadores de la interfaz con el usuario) correspondientes a diferentes esquemas de interacción, diversos tipos de pantallas, o bien, permitir que el Diseñador componga su propio Despachador.

# 8. CONCLUSIONES

El **Ambiente de Programación de Sistemas Transaccionales Distribuidos TESEO** expuesto en este artículo es un ambiente de desarrollo de gran utilidad para el diseñador de aplicaciones sobre Bases de Datos Distribuidas. El diseñador no está obligado a conocer las técnicas de procesamiento distribuido, y puede entonces diseñar las aplicaciones de la misma forma como lo haría para una Base de Datos Centralizada.

Teniendo en cuenta que en Colombia entra a funcionar próximamente la Red Nacional de Datos *COLDAPAQ* se hace necesario en nuestro medio no solo dominar una nueva tecnología de comunicación, sino también reconceptualizar la forma de diseñar y planear sistemas de información para las organizaciones.

El proyecto TESEO se enmarca dentro del objetivo de apoyar y facilitar ese cambio tecnológico fundamental. La meta final es aún más ambiciosa : se desea ofrecer, a través de *COLDAPAQ*, un ambiente completo de diseño, evaluación, simulación, prototipificación y programación de Sistemas de Información Distribuidos.

En la actualidad la implementación de la siguiente etapa de TESEO en la red *Ethernet* de equipos *Vax* depende de la obtención de financiación para este proyecto.

# BIBLIOGRAFIA

*[Alvarado y Muñoz 86]:* Armando Alvarado y 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.

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

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

*[Ellis 77]:* C.A. Ellis,
"Consistency and Correctness of Duplicate Database Systems", Proc. of 6th Symposium on Operating Systems Principles (ACM), Nov. 1977.

*[Eswaran 76]:* K.P. Eswaran et al.,
"The notions of consistency and predicate locks in a database system", CACM Vol.19 nº 11, Nov. 1976.

*[Franky 86]:* M. Consuelo Franky,
"Una metodología y ambiente de Programación de Sistemas Distribuidos a partir de redes de Nutt", XII Conferencia Latinoamericana de Informática (CLEI), Montevideo - Uruguay, Nov. 1986.

*[Gray 75]:* J.N. Gray et al.,
"Granularity of locks and degrees of consistency in a shared data base", R.J. 1654 IBM Research Laboratory, San José - California, Sept. 1975.

*[Lamport 78]:* L. Lamport,
"Time, clocks and the ordering of events in a distributed system", CACM Vol. 21 nº 7, July 1978.

*[Le Lann 81]:* G. Le Lann,
"A distributed system for real-time transaction processing", IEEE Computer Magazine, Vol. 14 nº 2, Feb. 1981.

*[Levy 87a]:* Perla Levy,
"Sistema Generalizado de Transaciones Distribuidas : Compilador de expresiones de lenguaje relacional", tesis de pregrado en Ingeniería de Sistemas ISC-87-I-23, Universidad de los Andes, Sept. 1987.

*[Levy 87b]:* Perla Levy,
"Compilador de expresiones de lenguaje relacional para el sistema TESEO", XIII-Conferencia Latinoamericana de Informática (CLEI), Bogotá - Colombia, Nov. 1987.

*[Morales 87]:* Luis Germán Morales,
"Manejo Dinámico de catálogos para el Sistema Generalizado de Transacciones sobre BDD", tesis de pregrado en Ingeniería de Sistemas ISC-86-II-20, Universidad de los Andes, Enero 1987.

*[Pedraza 87]:* Gilberto Pedraza G.,
"Control de concurrencia eficiente para el Sistema Generalizado de Transacciones sobre BDD", tesis de pregrado en Ingeniería de Sistemas ISC-86-II-24, Universidad de los Andes, Feb. 1987.

*[Quintero 87]:* José Ignacio Quintero A.,
"Sistema Generalizado de Transacciones Distribuidas : protocolo para datos parcialmente replicados", tesis de pregrado en Ingeniería de Sistemas ISC-86-II-29, Universidad de los Andes, Feb. 1987.

*[Rosenkratz 78]:* D.J. Rosenkratz et al.,
"System level concurrency control for distributed database systems", ACM Transactions on Database Systems, Vol. 3 nº 2, June 1978.

*[Stonebraker 79]:* M. Stonebraker,
"Concurrency control and consistency of multiple copies of data in distributed INGRES", IEEE Transactions on Software Eng., Vol. SE-5 nº 3, May 1979.

*[Toro 87]:* Víctor M. Toro,
"Especificación, Diseño y Prototipificación de Sistemas Interactivos", XIII-Conferencia Latinoamericana de Informática (CLEI), Bogotá - Colombia, Nov. 1987.