UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DESARROLLO DE UN SISTEMA CLIENTE-SERVIDOR DE SINCRONIZACION DE BASES DE DATOS DISTRIBUIDAS BASADO EN COMUNICACION ENTRE PROCESOS

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N:

A N T O N I O P A Z I T U R B E
JULIO ARTURO GARRIDO MARTINEZ
ENRIQUE CAZARES MALDONADO
J O R G E P A Z I T U R B E
SAMUEL RICARDO TRUJILLO GONZALEZ

DIRECTOR DE TESIS:
ING. JUAN MANUEL GOMEZ GONZALEZ

MEXICO, D. F. 1998

Agradecimientos

Un especial agradecimiento a la institución que nos forjó como profesionales la Facultad de Ingeniería.

Agradecemos a la UNAM, en donde nuestras mentes se nutrieron de conocimientos nuevos y nuestro horizontes fueron ampliados.

Expresamos un sincero reconocimiento a nuestro asesor de tesis Juan Manuel Gómez, quien guió con mucho entusiasmo el presente trabajo.

Agradecemos también al Programa de Apoyo a la Titulación por las facilidades proporcionadas.

Es justo reconocer que este trabajo ha sido posible por las facilidades proporcionadas por ASERCA, institución donde se desarrollo el sistema

Agradezco el apoyo desinteresado que siempre me brindaron mis padres , la confianza que nunca deje de sentir por parte de mis hermanos , los conocimientos que adquirí de todos los profesores que conocí , la ayuda de mis compañeros y amigos de la facultad , el fuerte impulso que recibí de mi buen amigo Alfonso Mireles Belmonte , y sobretodo el amor y cariño de mi esposa que siempre me motivo a seguir a delante en ese entonces.

Antonio Paz Iturbe

Agradezco a mis padres, Elva Maldonado y Bernabe Cázares, más que a nadie el apoyo que me dieron en todo momento, su valiosa guía que me alentó a seguir adelante, y por la fe que depositaron en mi

A mis hermanas, Elvia, Jovita y Miriam por sus consejos y criticas.

A mi amigo Eduardo con quien participe en un primer proyecto en PEMEX para sincronizar información, y de este proyecto se gestaron las ideas básicas para el desarrollo de este sistema.

Enrique Cázares Maldonado

A Dios:
Por haberme dado la oportunidad de vivir.

A mis padres
Por que con su esfuerzo, paciencia y dedicación me dieron la oportunidad de estudiar está carrera

A mis hermanos
Que siempre me apoyaron, y creyeron en mi

A mis amigos
Con los cuales conviví durante mis estudios, y me apoyaron mucho.

A los profesores que tuve
Que me aportaron sus conocimientos

A todos aquellos que fueron mis jefes en los trabajos en los que estuve
Por que me dieron la oportunidad de trabajar y seguir estudiando

Jorge Luis Paz Iturbe.

Por todo el apoyo que me han brindado a lo largo de mi vida, por ser mi soporte en todo momento, por la educación y formación que han sabido darme, dedico el presente a mis padres

Armando y Laura.

También quiero agradecer a mis hermanos, amigos y a todas las personas que han estado conmigo a través de toda la carrera y que han compartido mis logros y fracasos sin interés alguno

Julio Arturo Garrido Martínez

Agradecimientos:

A Dios
El único que conoce el fin desde el principio, fuente inagotable de sabiduría, en el cual descansa todo lo que sé y lo que soy

A mi Esposa Mahumy:
Por el amor y apoyo *incondicional* que siempre me brindas, hecho palpable en cada éxito de nuestra vida, incluyendo este trabajo.

A mis Hermanas:
Raquel, Mary y Betty, gracias por los consejos, los regaños y la paciencia que me han tenido hasta hoy.

A mis Padres(✠):
Por los ejemplos e ideales que grabaron en mi mente y por su incomparable *compañía*, que hasta hoy la disfruto en cada recuerdo suyo.

Samuel Trujillo González

Tabla de Contenido

1. Introducción

La dinámica actual de la Administración Pública Federal, está abriendo paso a transformaciones de forma que impactan operativa y funcionalmente a las Dependencias. Elementos como la productividad, transparencia en el manejo de los recursos, modernización de sistemas, procedimientos y la demanda de la población por mejores servicios, requiere de la modernización de las dependencias públicas, presionando para aplicar nuevas tecnologías.

En este sentido, la presencia en el mercado de equipos y sistemas informáticos permite realizar cambios profundos en la organización, operación y prestación de servicios de las organizaciones dedicadas a la administración de recursos

Muchas de estas organizaciones han implantado sistemas de bases de datos distribuidas geográficamente en el país, comunicadas entre sí mediante redes de área extendida o también llamadas redes de gran alcance. Una de estas organizaciones es ASERCA (Apoyos y Servicios a la Comercialización Agropecuaria), la cual esta constituida por 16 centros regionales y las oficinas centrales. El objetivo de esta institución es estimular la producción agrícola mediante apoyos económicos a los productores en función de las hectáreas cultivadas en cada ciclo agrícola; llevándose a cabo esta tarea desde el año de 1993 a través del programa conocido como PROCAMPO.

El esquema conceptual de PROCAMPO en los centro regionales, se encuentra conformado principalmente por cuatro módulos: Registro Voluntario, Entrega de Apoyos, Mantenimiento y Control de Almacén. De estos módulos, la información referente a Entrega de Apoyos, es la que de manera primordial debe estar sincronizada con la base de datos de las oficinas centrales de ASERCA, toda vez que en dicho módulo se tiene identificado lo que se pagó a los productores, los pagos cancelados, los pagos complementarios a lo cultivado y las reexpediciones de los pagos que fueron necesarias de hacer.

Al iniciar este programa de apoyo a los productores, los sistemas de computo eran del tipo centralizado, teniendo una base de datos en las oficinas centrales; sin embargo, debido al rápido crecimiento del volumen de información y a la dinámica del programa, que requería su descentralización, fue necesario distribuir la información en 16 centros regionales

Durante el tiempo, que lleva funcionando el programa PROCAMPO se han utilizando diversos sistemas de transmisión de información entre los centros regionales y oficinas centrales; así en 1993 no se utilizo ningún sistema, en 1994 la transmisión de información se hacia, indistintamente en forma parcial y/o completa y para 1995 la transmisión se comenzó a realizar por medio de un Sistema de Replicas de bases de datos.

Realizado el análisis de la situación que guardaba del modelo administrativo de la base de datos así como los productos y servicios que se generaban, además de la problemática que el mismo modelo produce en la operación por las características de su estructura, se determinó que era necesario conformar un nuevo modelo que signifique un cambio más profundo, que aprovechará la tecnología informática y la aplicación de los principios de bases de datos.

El sistema de información que se propuso, se desarrolló y se implantó se basa en un esquema de replicas, existiendo en los centros regionales las bases de datos Maestras y en las oficinas centrales la

base de datos Replica; con la particularidad de que en la Replica es necesario hacer operaciones que modifican la información y simultáneamente, en forma independiente se lleva la producción normal de las bases de datos Maestras. El análisis y diseño de este modelo se realizó con base en metodologías orientadas a objetos, esto con la finalidad de poder identificar cualidades y propiedades de cada una de las entidades que intervienen en el esquema y darles así el tratamiento apropiado

La manera en que se comparten los recursos en el esquema conceptual de PROCAMPO es mediante el uso de una red de computadoras interconectadas entre sí mediante X25 RDI, utilizando TCP/IP como protocolo de comunicación. Por otro lado, la información que fluye entre los nodos de la red se comprime, con esto se asegura que los datos que se transportan viajen en forma rápida y segura; pero si por fallas en la línea de comunicación o en el hardware de algún nodo de la red no es posible utilizar este medio, se implemento la opción de transportar la información con el mismo formato pero utilizando algún medio magnético ya sea cintas de respaldo o disquetes, así se asegura que la producción de la base de datos distribuida no se interrumpa.

2. Fundamentos Teóricos

Este capítulo abarca los términos y metodología que soportan la aplicación desarrollada. Estos términos están agrupados en tres rubros principales: Bases de Datos, Redes y la Metodología de Desarrollo.

2.1 Bases de Datos[1]

Para poder reconocer la capacidad y alcance del sistema, es necesario dar algunas definiciones útiles con respecto a las Bases de Datos, y sobre todo de las distribuidas, que son el punto central de este capítulo.

2.1.1 Definiciones y Conceptos propios de la Teoría de las Bases de Datos

En primer lugar debemos de reconocer tres características principales de las bases de datos:

- Una base de datos es una colección de datos lógicos y coherentes con un significado esencial. Esto deja excluido cualquier archivo con datos aleatorios como lo son oficios, cartas, documentos explicativos, etc.
- Una base de datos es diseñada, construida y publicada con un propósito específico, con aplicaciones proyectadas, pensadas para un grupo de usuarios realmente interesados en dichas aplicaciones.
- Una base de datos interactúa con un conjunto de eventos del ambiente en que nos desenvolvemos, es decir el mundo real; algunas veces este conjunto es denominado "mundo pequeño". Los cambios en el mundo pequeño son reflejados en la base de datos.

En otras palabras se puede decir que en una base de datos, los datos derivan de los grados de interaccion con eventos del mundo real, y de una audiencia que esta activamente interesada en el contenido de la misma.

También una base de datos puede ser generada y administrada manualmente o por una computadora. Este proyecto en sus inicios era mantenido en forma híbrida, (es decir tanto manual como por computadoras) lo que traía consigo riesgos propios que involucra tal manejo como es pérdida, fuga y errores de la información. A continuación se darán otras definiciones propias de la administración en forma computarizada.

Un **Sistema Administrador de Bases de Datos** (*DBMS*[1] por sus siglas en inglés) es una colección de programas de computadora de propósito general que capacitan al usuario para crear y mantener bases de datos, de una manera fácil. La **definición** de una base de datos involucra la forma de modelar abstractamente los tipos de datos que se almacenarán en dicha base. La **construcción** de la base de datos es el proceso de almacenar los mismos datos en algún medio de almacenamiento que pueda ser controlado por el *DBMS*. La **manipulación** de la base de datos incluye funciones como consultas,

[1] Se utilizarán los términos en inglés que son los más frecuentemente usados en el ámbito profesional y comercial.

recuperación de ciertos registros, actualizaciones para reflejar cambios del mundo pequeño y generación de reportes de los propios datos.

Cabe decir que muchas veces no es necesario usar un *DBMS* para implementar una base de datos computarizada, sino que uno puede escribir código para crear un *DBMS* propio. Ya sea este o el caso anterior, siempre se tendrá la base de datos y una cantidad de software para manipular dicha base; a este conjunto de elementos se le **denomina Sistema de Base de Datos (*DBS*)**

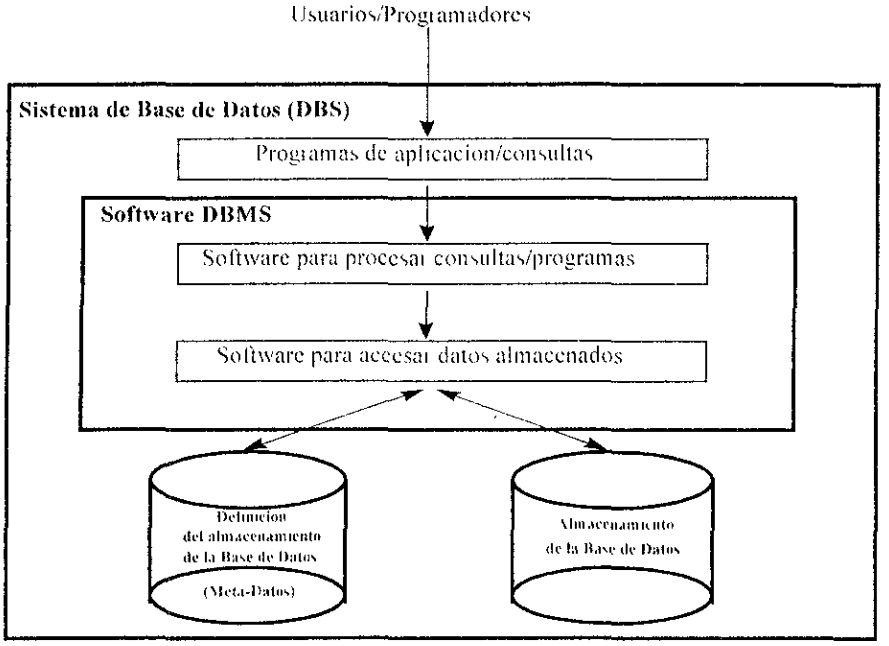


figura 2.1 Esquema simplificado de un sistema de bases de datos.

2.1.2 Modelo de Datos y Algebra Relacional

2.1.2.1 Antecedentes

El Modelo Relacional fue desarrollado en 1970 por el Dr. E. F. Codd de los Laboratorios de Investigación de IBM en San José Cal. Codd también incluyó el Algebra Relacional y fijó los fundamentos teóricos en algunos escritos publicados entre 1970 y 1974. Esta investigación le valió el premio Turing de la Association of Computing Machinery, que es el más alto honor que se ofrece en el campo de la computación. En una investigación de 1979, Codd vuelve a analizar su modelo creado y le incorpora los NULL al álgebra relacional. El modelo resultante es conocido como RM/T. Muchos investigadores han hecho nuevas propuestas y definiciones a este modelo por lo que hoy en día es el modelo más extendido y más aceptado entre los DBMS, a continuación se dará una explicación a dicho modelo y su teoría

2.1.2.2 Conceptos del Modelo Relacional

Como modelo de datos el relacional tiene los siguientes componentes:

- **Estructuras de datos:** son los conceptos de relación, entidades, atributos y dominios.
- **Definiciones de integridad:** está dada por el concepto de llave, posibilidades de valores nulos y dos reglas de integridad.
- **Operadores:** sus operadores incluyen los de actualización y la llamada álgebra relacional.

2.1.2.2.1 Estructuras de datos

El modelo relacional representa los datos en una base de datos como una colección de relaciones. Cada relación puede ser vista como una tabla o en algunos casos como un simple archivo. En el primer caso, cuando la relación es vista como una tabla de valores, cada renglón en la tabla representa una colección de datos conectados. Estos datos pueden ser interpretados como una descripción de una *entidad* o una *instancia de relación*. Aquí se puede ver la ventaja de este modelo, el cual no necesita como otros modelos conocidos, mezclar apuntadores y datos en la tabla para ubicarlos físicamente, sino que la posición es irrelevante. En vez de esto, los renglones son ligados por relaciones entre dos tablas que tienen atributos idénticos. En la terminología de las bases de datos relacionales, una tabla es llamada una *relación* (tabla), una columna dentro de la relación es llamada un *atributo* (campo) y un renglón en dicha relación es llamado una *tupla* (registro). El *tipo de datos o formato* describe el tipo de valores que pueden aparecer en cada columna, a este se le denomina *dominio*. Al dominio se le considera como un conjunto de valores *atómicos o indivisibles*, lo que quiere decir que el contenido no puede ser descompuesto en más partes

2.1.2.2.2 Definiciones de integridad

Los elementos que determinan la integridad para el modelo relacional son la *llave primaria y foránea*, los *valores nulos* y *2 reglas de integridad* que se enuncian a continuación.

Dentro de los atributos debe haber uno o varios que sirvan para distinguir cada entidad en la relación. Es a lo que se llama llave primaria. Esto significa que 2 *tuplas* no pueden contener la misma combinación de valores para todos sus atributos. Otra de las características de la llave principal es que debe ser invariante en el tiempo, de tal manera que si se adicionan nuevas *tuplas* no haya necesidad de cambiar la llave principal porque se tiene la seguridad que esta nunca presentará duplicidad en su contenido.

A fin de mantener la integridad a lo largo del tiempo en una base de datos relacional, se debe cumplir con 2 reglas en cuanto a los valores de las llaves primarias.

- ***Regla de Integridad de la Entidad-Relación***: Ningún componente de la llave primaria puede tener valores nulos. Esto se debe a que la llave primaria identifica *tuplas* individuales en la relación; si se tiene un valor nulo en la llave primaria, esto implica que no habrá modo de identificar una *tupla*. Esta regla se aplica solo a una relación individual.
- ***Regla de Integridad Referencial*** : A diferencia de la regla anterior esta se aplica entre dos relaciones y es usada para mantener la consistencia entre las *tuplas* de ambas. Esta regla en forma general indica que una *tupla* en una relación de la cual se hace mención en su llave primaria en otra relación

deberá existir en dicha relación. para definir este concepto más formalmente. se hará uso de *llave foránea*. Las condiciones para una llave foránea se dan específicamente en la regla de integridad referencial entre dos esquemas de relación R_1 y R_2.

Un conjunto de atributos *FK* en el esquema de relación R_1 es una llave foránea de R_1 si satisface las siguientes dos reglas:

1. Los atributos *FK* tienen el mismo dominio que los atributos *PK* de la llave primaria de otro esquema de relación R_2; los atributos *FK* se dice que hacen referencia o tiene referencia a la relación R_2.

2. Un valor de *FK* en una *tupla* t_1 de R_1 puede ocurrir como un valor de *PK* para una *tupla* t_2 en R_2 o bien puede ser nula, es decir $t_1[FK] = t_2[PK]$, y entonces se dice que la *tupla* t_1 es referenciada o referida de la *tupla* t_2 .

Estas reglas son las que soportan a todo el modelo relacional.

2.1.2.2.3 Operadores

2.1.2.2.3.1 Algebra Relacional

El álgebra relacional es un conjunto de operaciones que son usadas para manipular las relaciones. Estas operaciones se usan para seleccionar *tuplas* de relaciones individuales y combinarlas con *tuplas* extraídas a su vez de otras relaciones con el propósito de crear una consulta - una petición de recuperación - a la base de datos. El resultado de cada operación es una nueva relación la cual puede ser nuevamente manipulada por las operaciones del álgebra relacional.

Las operaciones del álgebra relacional se dividen en 2 grupos. El primer grupo incluye un conjunto de operaciones propias de la teoría matemática de conjuntos; dichas operaciones incluyen la UNION, INTERSECCION, DIFERENCIA y PRODUCTO CARTESIANO. El otro grupo consiste de operaciones desarrolladas específicamente para bases de datos relacionales e incluye los operadores PERMUTACION, SELECCION, PROYECCION y JUNTAR. A continuación se describen estas operaciones en términos puramente matemáticos utilizando la terminología del modelo relacional y reservando los ejemplos prácticos en la parte del lenguaje relacional estándar y su extensión.

Operación UNION

Dadas dos relaciones R y S, la operación UNION es la relación sobre los mismos dominios que contiene las *tuplas* que están en R o que están en S. Se denota como:

$$R \cup S$$

Operación INTERSECCION

Dadas dos relaciones R y S, la operación INTERSECCION es la relación con las *tuplas* que están en R y en S. Se denota como.

$$R \cap S$$

Operación DIFERENCIA

Dadas dos relaciones R y S, la operación DIFERENCIA es la relación con las *tuplas* que están en R pero no en S. Se denota como:

$$R - S$$

Operación PRODUCTO CARTESIANO

El producto cartesiano obtiene todas las *tuplas* que se construyen concatenando cada *tupla* de R con cada de S. En este caso los dominios de R y S no tienen que ser los mismos. Se denota como:

$$RxS$$

Operación PERMUTATION[2]

Esta operación se aplica a una sola relación y consiste en cambiar el orden de los atributos. Se denota por:

$$Pi(i\ j\ k)$$

en donde se indican el orden en que estarán los atributos de la relación original. En este caso el primer atributo es i, el segundo j y el tercero k.

Operación SELECT

Esta operación se utiliza para seleccionar un subconjunto de *tuplas* en una relación que cumplen con una condición (simple o compuesta) sobre los valores para uno o varios de los atributos. Se denota por:

$$\sigma\langle condicion\_de\_seleccion\rangle(\langle nombre\_de\_la\_relacion\rangle)$$

Operación PROJECT

Si se piensa en una relación como una *tabla*, entonces la operación SELECT actúa sobre los *renglones*, seleccionando algunos y descartando a otros. Por otro lado la operación PROJECT selecciona ciertas *columnas* de la tabla y descarta otras. Es decir si solo nos interesan conocer ciertos atributos de la relación, la operación PROJECT proyectará la relación solo sobre dichos atributos. Se denota por:

$$\pi\langle lista\_de\_atributos\rangle(\langle nombre\_de\_la\_relacion\rangle)$$

Operación JOIN

La operación JOIN, denotada por ⊠, se usa para combinar *tuplas* mezcladas de dos relaciones dentro de *tuplas* individuales. Esta operación es muy importante para cualquier base de datos relacional con más de una relación, ya que permite el proceso de conexión entre las relaciones. La forma general de la operación JOIN en dos relaciones $R(A_1,A_2,\ ..,A_n)$ y $S(B_1,B_2,..\ ,B_m)$ es:

$$R⊠\langle condición\ JOIN\rangle S$$

El resultado de la operación es una relación Q con $n+m$ atributos $Q(A_1,A_2,..\ ,A_n,B_1,B_2,\ \ ,B_m)$ en orden tal que; Q tiene una *tupla* para cada combinación de *tuplas* - una de R y una de S - siempre y cuando la

[2] Se utiliza el término inglés de la operación ya que es el más usado en los lenguajes SQL de los Manejadores de Bases de Datos.

combinación satisfaga la condición JOIN. Esta es la principal diferencia entre las operaciones PRODUCTO CARTESIANO y JOIN; en JOIN solo las combinaciones de *tuplas* que satisfagan la condición aparecerán, mientras en PRODUCTO CARTESIANO *todas* las posibles combinaciones de *tuplas* aparecerán en el resultado. La condición JOIN tiene por lo tanto esta forma:

<*condición*>AND<*condicion*>AND...AND<*condición*>

2.1 2.2 3 2 Operadores de Actualización

Para actualizar los valores de los atributos en las *tuplas* se pueden efectuar las operaciones de agregar, borrar o modificar. El manejo de llaves foráneas hace necesario establecer reglas que determinan como manejar las operaciones de actualización de relaciones para no introducir inconsistencias, a continuación se indican dichas reglas:

Reglas para agregar.- Al insertar una *tupla* en una relación, el valor de un atributo que es llave foránea puede ser nulo, o algún valor del dominio de la llave primaria.

Reglas para borrar.- Si se va a borrar una *tupla* en una relación R_1 con cierta llave primaria y otra relación R_2 tiene ese atributo como llave foránea, hay 3 casos:

a) Borrado restringido - No se puede borrar una *tupla* en R_1 que tenga *tuplas* en R_2 con el mismo valor como llave foránea.
b) Borrado en cascada.- Al borrar una *tupla* en R_1, se borrarán todas las *tuplas* en R_2 con ese valor.
c) Borrado con nulificación.- Al borrar la *tupla* en R_1, a todas las *tuplas* con igual valor en R_2 se les pone el valor nulo.

Reglas para Modificar

a) Modificación en cascada.- Al modificar una llave primaria en R_1 se le cambian los valores correspondientes en R_2.
b) Modificación con nulificación - Al cambiar los valores de la llave primaria en R_1 a los correspondientes en R_2 se les pone el valor nulo.

2 1.2.2.3 3 Operaciones Primitivas

En el álgebra relacional se han incluido las operaciones UNION, INTERSECCION, DIFERENCIA, PRODUCTO CARTESIANO, PROJECT, SELECT, PERMUTATION y JOIN. Sin embargo no es necesario tenerlas todas; las operaciones primitivas indispensables a partir de las cuales se pueden obtener las demás son: UNION, DIFERENCIA, PRODUCTO CARTESIANO, PROJECT y SELECT.

Por ejemplo la INTERSECCION se puede expresar en términos de la DIFERENCIA como sigue:

$$R \cap S = (R - (R - S))$$

JOIN se puede obtener a partir del SELECT de los elementos que cumplen con una condición en el producto cartesiano:

$$R \boxtimes <condición\ JOIN> S = \sigma(condicion\_de\_seleccion)(R\_S)$$

Esto es importante pues algunos sistemas manejadores de bases de datos comerciales proporcionan solo parte de los operadores y conociendo las operaciones primitivas a partir de las cuales se pueden obtener las otras no resultaría difícil crearlas como funciones

Otro punto importante del álgebra relacional es ver que propiedades tienen las operaciones para optimar su uso:

UNION es conmutativa y asociativa:

$$(R \cup S) \cup T = (S \cup T) \cup R$$

INTERSECCION también es conmutativa y asociativa:

$$(R \cap S) \cap T = (S \cap T) \cap R$$

Propiedad distributiva entre UNION e INTERSECCION:

$$R \cap (S \cup T) = (R \cap S) \cup (R \cap T))$$

También son conmutativos PRODUCTO CARTESIANO y SELECT.

A partir de los operadores del álgebra relacional también se pueden efectuar las operaciones de actualización. Para insertar una *tupla* se puede hacer como la unión de una relación y dicha *tupla*:

$$R \cup (t_1, \ldots t_n)$$

Para borrar se hace la diferencia entre la relación y la *tupla*:

$$R - (t_1, \ldots t_n)$$

En síntesis se puede decir que un sistema manejador de bases de datos se puede llamar totalmente relacional si soporta:

a) Una base de datos relacional incluidos los conceptos de dominio, llave, y las dos reglas de integridad y el almacenamiento de las relaciones.

b) Un lenguaje de manipulación de datos que sea al menos tan poderoso como el álgebra relacional

2.1.3 Extensiones del Modelo Relacional[ii]

Hoy en día existen diversidad de aplicaciones que por su complejidad no se adecuan al modelo relacional clásico que hemos estudiado, sin embargo cuando este fue dado a conocer en la década de los 70's, hubo una gran cantidad de proyectos que se iniciaron para mejorar el trabajo de Codd. La mayoría de estos proyectos tomaron como base el mismo modelo relacional por lo que fueron llamados *aproximaciones evolucionarias*, —a diferencia de las *aproximaciones revolucionarias* que fueron trabajos con nuevas propuestas y nuevos modelos de bases de datos como el modelo orientado a objetos—todos estos vinieron a enriquecer la semántica del modelo relacional existente.

A continuación se analizarán algunas de las extensiones del modelo que sirvieron en la aplicación del proyecto del presente trabajo.

2.1.3.1 Generalización, Agregación, Especialización y Ejemplos del uso de la extensión en Modelo Relacional

La agregación es una forma de abstracción donde una conjunción entre entidades o relaciones es considerada y modelada como una relación con sus propias características. En ese sentido, la agregación corresponde a la noción del producto cartesiano.

Las relaciones de diferentes tipos se combinan para formar otras denominadas *entidades agregadas*. Por ejemplo en la figura 2.2se visualiza que la relación 'ensamblaje' forma parte de cuatro diferentes relaciones y esta por si sola forma una relación agregada independiente con características propias

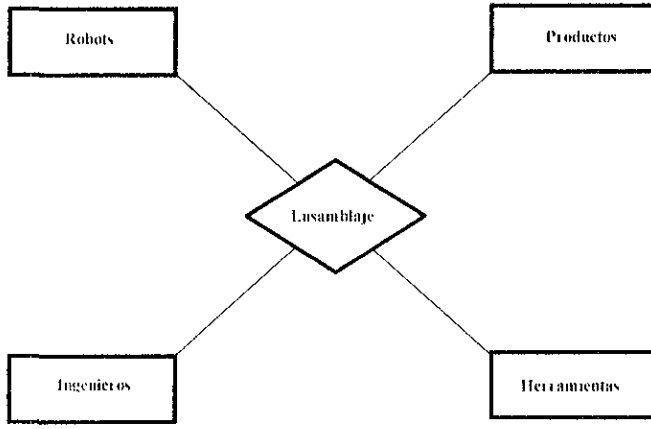


figura 2.2 Ejemplo de entidad agregada: "Ensamble".

Aunque la agregación es común en el modelo relacional clásico, la parte más importante la constituye la que deriva en la extensión del modelo, que es la abstracción de la conexión *parte de*; esta conexión existe como una parte principal y varias subpartes posiblemente de diferentes relaciones, esto frecuentemente forma jerarquías de agregación, por ejemplo una bicicleta puede ser modelada de manera agregada de la siguiente manera.

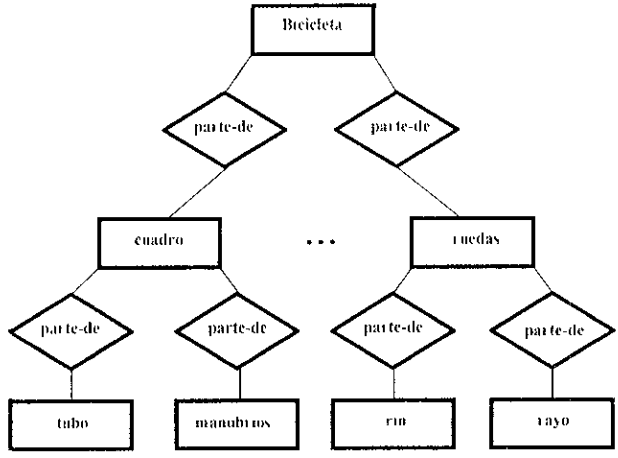


figura 2.3 Ejemplo de jerarquía de agregación de las partes de una bicicleta.

Por otra parte la generalización es una abstracción la cual da lugar a una clase de relaciones individuales que se conciben genéricamente como una relación especificada en su más mínima definición. La generalización se usa para agrupar relaciones similares que pueden ser descritas por una relación

denominada *entidad o relación genérica.* Así los detalles no relevantes de las relaciones individuales son omitidos en la relación genérica, o visto de otra manera, los detalles más importantes de las relaciones individuales adquieren mayor énfasis en la relación genérica que los atributos comunes. La generalización actúa sobre varios niveles por medio de la conexión *is-a* (es uno(a)). Como ejemplo se tiene la generalización de la bicicleta en relaciones genéricas más concisas.

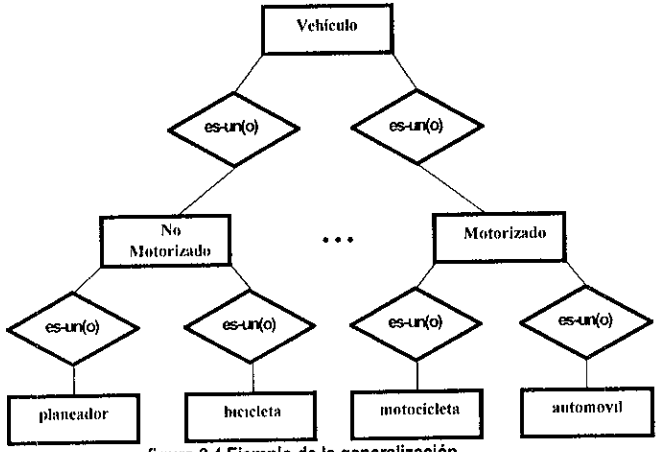


figura 2.4 Ejemplo de la generalización.

Cabe decir que se pueden crear relaciones que mezclen tanto la agregación como la generalización.

Teniendo estos antecedentes, podemos realizar la siguiente extensión: Mientras el modelo relacional provee al menos algún soporte de jerarquías para modelar objetos agregados, la representación de jerarquías para la generalización no es soportada por el modelo relaciona tradicional. Es por eso que se presenta la especialización que es semánticamente mejor que la agregación y generalización solas o combinadas. Para sustentar esta propuesta, se usa una abstracción denominada *jerarquía de generalización/especialización* que consiste solo de dos niveles como se presenta en la figura 2.5.

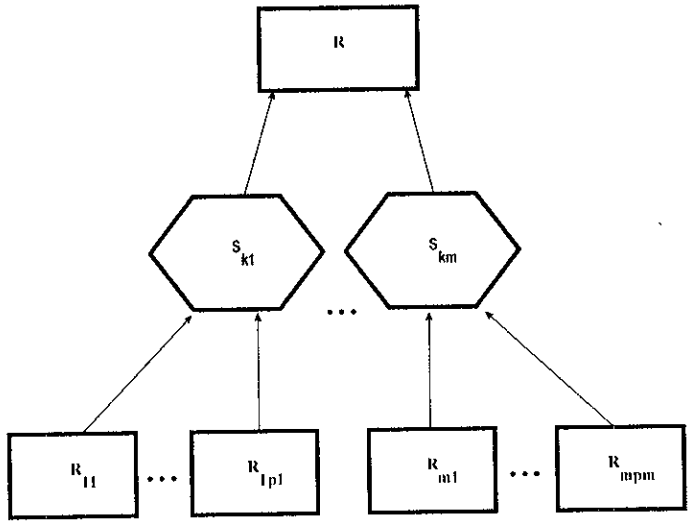


figura 2.5 Jerarquía de generalización/especialización.

La Relación genérica tipo R es especializada dentro de relaciones tipo R_{11}, ...R_{1p1},... ,R_{mpm}

Considerando ahora las llamadas dimensiones de especialización, en el esquema ejemplo, estas dimensiones son $S_{k1},\ldots,S_{km}$ Todas estas S_{ki} forman un subconjunto de todos los atributos $S_1,\ldots,S_n$ que describen la relaciones genéricas tipo R. Cada dimensión de especialización S_{ki} especifica un atributo particular de las entidades tipo R con la propiedad de que pueden ser agrupadas dentro de subconjuntos de especialización mutuamente excluyentes $R_{i1},\ldots,R_{ip}$. Además se asume que cada Relación de tipo R viene a ser exactamente uno de los conjuntos $R_{i1},\ldots,R_{ip}$ dependiendo de su propiedad S_{ki}. Así cada entidad de tipo R es especializada a lo largo de *m* diferentes tipos de relaciones especializadas.

En la notación gráfica las dimensiones de especialización se denotan por un hexágono. Los conjuntos especializados mutuamente excluyentes se conectan por medio de flechas que apuntan al hexágono y de este sale una flecha hacia el tipo genérico que ha sido especializado.

Para ilustrar esta explicación consideremos la figura 2.6 y la relación tipo *Personas* con tres dimensiones de especialización:

1. sexo.- que hace distinción del tipo *Personas* en mujeres y hombres
2. nacionalidad.- que agrupa todas las *Personas* en diversas nacionalidades
3. estado civil.- que agrupa a las *Personas* en subconjuntos especializados denominados solteras, casadas, divorciadas y viudas.

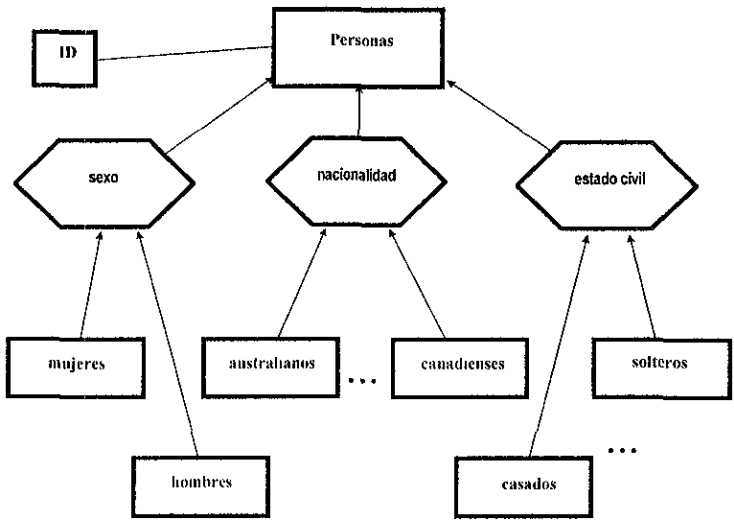


figura 2.6 Relación personas con tres dimensiones de especialización.

Hay que notar que estas tres dimensiones de especialización son completamente independientes entre si ya que por ejemplo el sexo no influye con el estado civil o la nacionalidad de una persona.

El identificador de la persona es un tipo especial de atributo el cual no se considera una especialización, ni requiere flechas que salgan o entren hacia él, sin embargo es necesaria su declaración para fines de modelado de la relación.

2.1.3.1.1 Extensiones sintácticas del modelo relacional

La sintaxis para generar este modelo sería la siguiente.

var R [:**generic**

s_{k1}=(R_{11},…,R_{1p1});

…

s_{km}=(R_{m1},…,R_{mpm}),

of]

aggregate[keylist]

s_1:[**key**] R_1

s_n:[**key**] R_n;

end

Para el ejemplo de la gráfica 2.6 las sintaxis quedaría de la siguiente manera:

```
var Personas [:generic
        sexo=(hombres, mujeres);
        Nacionalidad=(australianos. . canadienses);
        Estado_civil=(solteros... viudos)
    of
    aggregate[ID]
        ID: identificador_de_persona
    end
```

2.1.3.1.2 Manipulación de datos

Hasta ahora solo se ha discutido lo referente al modelado de datos dentro de la extensión del modelo relacional, sin embargo hay que resaltar la importancia y el cuidado que se debe de tener al manejar la información. Esto es debido a que los esquemas modelados presentan redundancia, causada por la duplicación de atributos que no se pueden eliminar debido a los niveles de generalización y especialización que han tomado los datos

Para hacer una confiable actualización de datos hay que asegurar que la modificación en un atributo A dentro de alguna relación R se refleja en todas las relaciones generalizadas de R y obviamente contienen el atributo A. A esto se le denomina actualización en cascada y deberá ser ejecutada automáticamente por el sistema manejador de bases de datos para evitar violaciones de consistencia.

De manera general, las operaciones de actualización en cascada se pueden hacer hacia arriba o hacia abajo sobre diferentes niveles de jerarquías de generalización. Todo esto depende del número de niveles sobre los cuales los atributos fueron heredados *(cascada hacia arriba)* y sobre el número de niveles sobre los cuales los atributos han sido manejados como tipos especializados *(cascada hacia abajo)*

2.1.3.1.3 Discusión

Como todo modelado, la extensión del modelo relacional tradicional para soportar la generalización sufre de algunos inconvenientes que a continuación se enumeran.

1. Replicación de datos cuando se mapea hacia el modelo relacional tradicional
2. No existe una pauta para la manipulación de orientación a objetos, es decir, el modelo extendido, no permite incorporar operaciones específicas de la aplicación.
3. La semántica de la estructura de los objetos fundamentales no es fácil de apreciar cuando el desarrollador tiene una vista limitada de la aplicación.
4. Operaciones de actualización demasiado costosas para garantizar la consistencia de la información replicada (actualizaciones en cascada) Esta puede fácilmente degradar si existen jerarquías de generalización demasiado extensas.

2.1.4 Conceptos de los Sistemas Administradores de Bases de Datos Distribuidas (DDBMS)[iii]

2.1.4.1 Definición

Una base de datos distribuida puede ser definida como una colección consistente de datos ubicados físicamente en diferentes sitios o *sites* bajo el control de manejadores de bases de datos separados, corriendo en sistemas de cómputo independientes. Los sites que no son otra cosa que los servidores de archivos principales dentro del sistema de la red a la cual están interconectadas, mantienen una comunicación a su vez con otros sistemas y cada sistema tiene aplicaciones locales y servicios con capacidad de procesamiento autónomo. Cada sistema participa en la ejecución de una o varias aplicaciones globales. De esta manera las aplicaciones requerirán datos de más de un *site*. La naturaleza distribuida de las bases de datos es oculta para los usuarios que verán el sistemas como único; esta transparencia, más que un requisito, es una característica de las bases de datos distribuidas. Aunque hay un gran número de ventajas al usar bases de datos distribuidas, hay también un gran número de problemas y discusiones en la implementación. Finalmente los datos en una base de datos distribuida se pueden particionar (fragmentar) o replicar (duplicar) o ambos como más adelante se explicará.

2.1.4.2 Ventajas

Algunas de las ventajas potenciales de un sistema de bases de datos distribuidas se listan a continuación:

- **La naturaleza distribuida de algunas aplicaciones de bases de datos.-** Muchas aplicaciones son distribuidas naturalmente sobre diferentes *sites* o regiones. Por ejemplo un Banco con diferentes sucursales alrededor del mundo. La base de datos de ASERCA tiene de hecho esta naturaleza: cada región es reconocida como un *site* en donde se distribuyen los datos y cada región mantiene para sí una base de datos para almacenar información de tipo local. Muchos usuarios locales acceden solo los datos para dicho *site*, pero los usuarios globales como los gerentes generales, tendrán que acceder datos almacenados en diferentes *sites*. Cabe hacer notar que los datos en cada *site*, típicamente describen un "mundo pequeño". La fuente de datos, las aplicaciones y la mayoría de los usuarios, físicamente residirán en ese *site*. Por otro lado, los usuarios globales quizá solo accedan ocasionalmente los datos locales de determinada región.
- **Incremento en confiabilidad y disponibilidad e incremento gradual en el sistema.-** La *confiabilidad* es ampliamente definida como la probabilidad de que el sistema esté listo en un determinado momento en el tiempo. Por otra parte la *disponibilidad* es la probabilidad de que el sistema este continuamente en línea y listo durante un intervalo regular de tiempo. Cuando los datos y el software del Sistema de Bases de Datos Distribuido (*DDBS* por sus siglas en inglés), están distribuidos sobre varios *sites* alguno de ellos quizá pueda fallar, sin embargo los demás seguirán en operación. Solo los datos y el software que existan en el *site* que falló, no estarán disponibles, otros datos y su software si podrán ser usados. En un sistema centralizado, la falla de un simple *site* hace que el sistema no este disponible a ninguno de los usuarios. Una de las mejoras en confiabilidad y disponibilidad es la que radica en la réplica de los datos y software en más de un *site*. Por último el crecimiento de la base de datos de manera gradual bajo un esquema ya definido resulta una ventaja más para la implementación de un sistema distribuido, basta con crear nuevos *sites* y compartirlos con los ya existentes.

- **Permisos para compartir datos mientras se mantienen algunas medidas de control local.-** Algunos tipos de *DDBS* hacen posible el control de datos y software localmente en cada *site*. Sin embargo, ciertos datos pueden ser accedidos por otros *sites* remotos por medio del software *DDBMS*. Esto permite controlar los datos compartidos a través del sistema distribuido.
- **Mejoras en el desempeño.-** Al distribuir una base de datos de gran tamaño sobre diferentes *sites*, existirán pequeñas bases de datos en cada uno de ellos. Las consultas y transacciones locales operan con los datos en un solo *site*, lo que potencialmente demostrará mejor desempeño ya que se realizarán pequeños procesos en las base de datos local. Además cada *site* tendrá un menor número de transacciones ejecutándose, que es más eficiente que tenerlos en una sola base de datos centralizada. Para aquellas transacciones que involucran acceso a más de un *site*, el procesamiento de cada uno de estos, se puede llevar de manera paralela, la cual puede reducir la ejecución de la transacción y su tiempo de respuesta.

La distribución hace que un sistema se incremente tanto en complejidad como en diseño e implementación Para que las ventajas enumeradas no se empañen por características de complejidad, es necesario adicionar ciertas funciones especiales que los sistemas centralizados no contemplan:

a) Acceso remoto y transmisión de consultas y datos a lo largo de varios *sites* vía una red de comunicación.

b) Seguir una pista de la distribución de datos y su réplica en el catalogo de *DDBMS*

c) Proyectar estrategias de ejecución para consultas y transacciones que operen con datos en más de un *site*.

d) Decidir cual copia de los datos replicados será accedida

e) Recuperación de *sites* individuales que fallan y de nuevos tipos de fallas como son las debidas a las ligas de comunicación.

2.1.4.3 Desventajas

Las desventajas de la implementación de un modelo distribuido en un sistema manejador de bases de datos radican en su costo y su complejidad. Un sistema distribuido, el cual esconde su naturaleza distribuida al usuario final resulta más complejo que un sistema centralizado. El incremento de la complejidad significa que los costos de adquisición y mantenimiento de hardware, no muy pequeño por cierto, es mucho mayor que un sistema centralizado de bases de datos. La naturaleza paralela de los sistemas distribuidos indica que no es fácil evitar problemas y estos a su vez, dentro de las aplicaciones usadas, serán difíciles de identificar y posteriormente aislar y corregir. Además, por su propia naturaleza, el sistema distribuido, enlista una gran cantidad de coordinación de mensajes en la comunicación con los diferentes *sites* involucrados en el mismo. Si el sistema esta bien planeado y pensado las desventajas podrán ser controladas

2.1.5 Arquitectura de un Sistema de Bases de Datos

En el nivel físico de hardware, los principales factores que distinguen a un *DDBS* de un sistema centralizado son los siguientes:

a) Existen muchas computadoras llamadas *sites* o nodos

b) Estos nodos o *sites* deberán ser conectados a todo lo largo, por algún tipo de red de comunicaciones para la transmisión de datos y comandos.

Los *sites* podrán estar conectados vía una red de área local LAN, o bien podrán estar distribuidas geográficamente sobre grandes distancias conectadas vía una red de área amplia WAN. Las redes LAN generalmente usan cables, mientras las redes WAN utilizan líneas telefónicas digitales o satélites. También es posible una combinación de los dos tipos de red. La topología usada es un factor determinante en el desempeño del *DDBS*, sin embargo en arquitecturas de alto nivel, el tipo de red y la topología no son tan importantes como que el *site* sea capaz de comunicarse, directa o indirectamente con otro *site*.

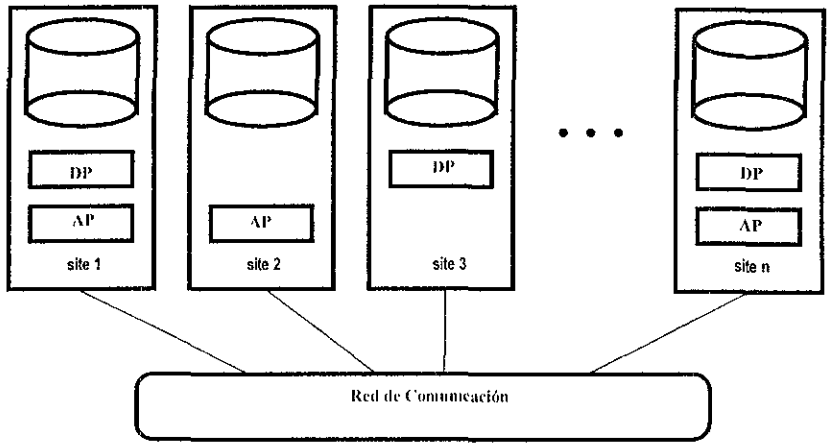


figura 2.7 Arquitectura física simplificada de un Sistema de Bases de Datos Distribuido.

Para manejar la complejidad de un *DDBMS*, este se subdivide en tres módulos principales:

a) ***El procesador de datos (DP).***- Este software es el responsable de la administración local de los datos en un *site*, es parecido al software *DBMS* en los sistemas centralizados.

b) ***El procesador de la aplicación (AP).***- Es el software responsable de la mayoría de las funciones de distribución, tiene el acceso a la información del catálogo, el cual se encarga de la asignación para encaminar y distribuir los datos, además es responsable de procesar todas las peticiones que requieren acceder a más de un *site* y mantiene información de la fragmentación y réplica en general.

c) ***Software de comunicaciones.***- Algunas veces en conjunción con un sistema operativo distribuido, es el que provee las primitivas de comunicación que son usadas por el PA para transmitir comandos y datos a lo largo de varios *sites* según se necesite. Este no es estrictamente parte del *DDBMS*.

Es posible que en un *DDBS*, algunos *sites* contengan tanto *AP's* como *DP's*, o bien solo *AP's* o sólo *DP's*. Un *site* que es usado sólo para funciones *DP* es llamado *máquina* ***back-end*** y uno que es usado solo para funciones *AP* es denominado *máquina* ***front-end***. Una máquina *back-end* puede incluir hardware especializado para búsquedas dentro de la base de datos de manera local, además es responsable del control de concurrencias y recuperación de errores. Una máquina *front-end* puede ser

una terminal "tonta" o bien puede ser una poderosa minicomputadora con varia terminales conectadas a ella y es responsable de mantener el catálogo y el control de la ejecución de distribución, el control de la concurrencia y la recuperación a un nivel global; la figura 2.8 muestra una vista lógica de un *DDBS*, donde las APs y las DPs se muestran sin especificar el *site* en el cual residen

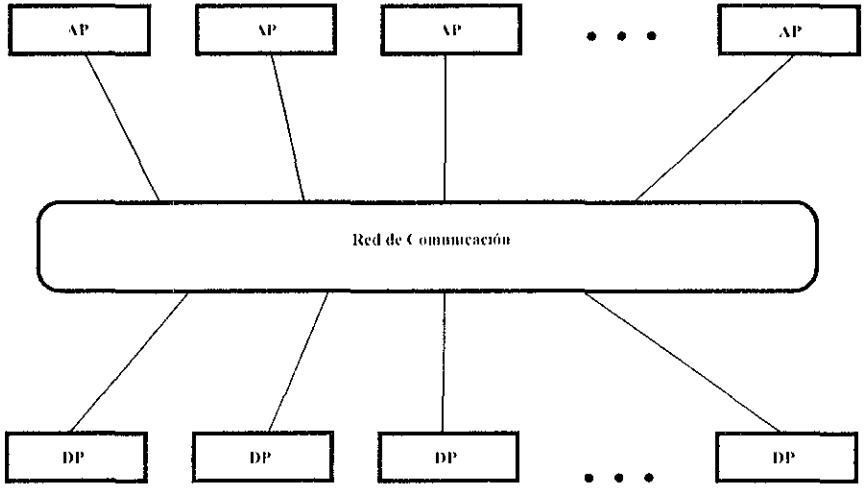


figura 2.8 Arquitectura lógica simplificada de un Sistema de Bases de Datos Distribuido.

Una importante función de los *AP's* es que esconda los detalles de la distribución de datos, de forma tal que el usuario no lo note, este último podrá hacer consultas globales, transacciones, modificaciones, etc. como si se tratara de un sistema centralizado, sin tener que especificar en que *sites* se encuentra la información solicitada, este tipo de propiedad se le denomina *transparencia de la distribución*. Cabe mencionar que muchos *DDBMS* no tienen esta propiedad por lo que es el usuario el responsable de asignar los *sites*, las tablas y los registros que el catálogo contiene. Este modelo es menos complejo que uno que cuente con la propiedad de transparencia, pero al interactuar el usuario con el se vuelve más complicado.

2.1.6 Fragmentación, Réplica, Procesamiento de Consultas y Optimización de Bases de Datos Distribuidas

2.1.6.1 Fragmentación y Réplica de Datos

La Fragmentación es la decisión que se toma de dividir la base de datos en pequeños fragmentos y distribuirlos dentro de diferentes *sites*. Este tipo de decisión se hace sobre un esquema relacional de bases de datos, ya que es el modelo que acepta la fragmentación. Antes de determinar en que *sites* residirá la información se debe determinar en cuales se ubicarán las *unidades lógicas* que no son otra cosa que las tablas que mantiene dicha base, aunque en algunos casos una relación puede subdividirse en otras unidades lógicas más pequeñas. La fragmentación de datos tiene dos técnicas:

a) ***Fragmentación horizontal.-*** Se lleva a cabo haciendo un subconjunto de coincidencias de determinado campo, es decir se genera otra condición una a muchos en la relación, permitiendo que determinados datos queden sujetos a una tabla que queda almacenada en un *site*, dicha tabla se verá como una división "horizontal" que presenta otros renglones relacionados de manera que hay consistencia en las relaciones.

b) ***Fragmentación vertical.-*** Se lleva a cabo cuando una tabla es dividida en dos o más categorías, haciendo que no haya consistencia en los datos pues se tendrían que agregar nuevos campos primarios para sustentar la relación, de esta forma la fragmentación se ve como una división "vertical" de la tabla en la cual los datos se subdividen por columnas. Este tipo de fragmentación no es del todo adecuada, a menos que esta división sea hecha en campos que no tienen atributos comunes.

Además se puede hacer una *fragmentación mixta* la cual sería tener presentes las dos técnicas mencionadas arriba. Para cualquiera de estas formas aplicadas, siempre será necesario tener un *Esquema de Fragmentación* que es la definición del conjunto de fragmentos en los cuales se divide la base de datos, dicha definición también debe de contener todos los atributos que satisfacen las condiciones de la base de datos entera, de manera que esta pueda ser reconstruida en todos sus fragmentos simplemente aplicando técnicas de unión.

Un *Esquema de Asignación* describe como será asignado el *site* para determinado fragmento del *DDBS*, es decir, es un mapeo hacia los lugares en los que se localizará y será almacenado determinado fragmento.

Si un fragmento es almacenado en más de un *site*, se dice que el dato es *replicado*. La réplica de datos es útil en la mejora y disponibilidad de los datos, el caso extremo es cuando una base de datos es replicada en su totalidad, lo cual se denomina una *réplica completa*. Esto mejora la disponibilidad del sistema, ya que este seguirá operando por mucho tiempo al menos en un *site*; también mejora el desempeño total, ya que las consultas globales se podrán realizar en ese mismo *site* La desventaja de este caso extremo es que el sistema puede hacerse sumamente lento, ya que una simple actualización deberá hacerse en todos los *sites* donde la base mantiene sus réplicas. El otro caso extremo es la falta de réplica, denominado *asignación no redundante*, donde no existe más que una sola copia o réplica de la base de datos, con sus resultado contundentes ya conocidos. Dentro de estos dos extremos, existe el de réplica parcial, que como su nombre lo indica sólo una parte de los datos es replicada en ciertos *sites*. Al tratar de utilizar la réplica es necesario llevar un control más estricto sobre las actualizaciones a los datos y sobre las concurrencias que puedan presentarse al acceder a los datos. Este control debe de ser transparente al usuario.

2.1.6.2 Procesamiento de Consultas y Optimización

Uno de los factores a considerar al implementar una base de datos distribuida es el costo de la transferencia de datos sobre la red. Estos datos incluyen archivos intermedios que son transferidos a otros *sites* para procesar hasta obtener el resultado requerido y después ser transferidos al que hizo la petición. Aunque el costo quizá no sea muy alto en aquellas redes que tienen un buen desempeño a nivel local, si debe de ser considerado para otro tipo de redes de tipo regional. De aquí que muchos algoritmos de optimización de un *DDBMS* consideren como meta el reducir la cantidad de datos transferidos como criterio principal para optimar. Este tipo de optimización se logra aplicando

algoritmos de compresión de datos o bien tratando de procesar consultas reduciendo el número de *tuplas* que serán transferidas en base a técnicas que utilizan los operadores relacionales antes mencionados. Cualquiera que sea el método a seguir, es recomendable hacer un análisis costo/beneficio de las técnicas a utilizar.

En lo que se refiere a las consultas, un *DDBMS* que soporte, fragmentación, réplicas y transparencia al usuario deberá ser capaz de permitir al usuario, especificar una consulta o actualización con base en un *módulo de descomposición de consultas*, que se encargará de dividir o descomponer la consulta total en varias subconsultas que podrán ser ejecutadas en diferentes *sites* Además deberá de existir otro módulo que combine los resultados de las diferentes subconsultas para formar el resultado de la consulta total.

Si existen réplicas de los datos consultados, el *DDBMS* será capaz de determinar y escoger o *materializar* una réplica en particular durante la ejecución de la consulta. Para determinar la elección, éste deberá consultar la información del catálogo de la base de datos. Para la fragmentación vertical, el catálogo guarda la lista de los atributos que cada fragmento tiene, para la fragmentación horizontal el catálogo tiene una condición, denominada *centinela*, que cada fragmento mantiene, esta es básicamente una condición de selección que especifica que *tuplas* existen en el fragmento; se le llama centinela ya que sólo las *tuplas* que satisfagan esta condición, se les permitirá ser almacenadas en dicho fragmento. Si la fragmentación es mixta, el catálogo guarda tanto la lista de atributos como la condición de centinelas para poder hacer los cálculos.

2.1.7 Métodos de Control de Concurrencia y Recuperación en Bases de Datos Distribuidas

Para el control de concurrencia y recuperación, existen algunos problemas en un ambiente distribuido que en un sistema centralizado nunca se darán. Algunos de estos problemas son los siguientes:

a) Manipulación de *múltiples copias* de los datos.- El método de control de concurrencia es el responsable de mantener la consistencia entre las diferentes réplicas. El método de recuperación es el responsable de hacer una copia consistente de los datos de un *site* en otro, si este llega a fallar y posteriormente recuperar los datos.
b) Falla en *sites* individuales.- El *DDBMS* deberá continuar operando cuando uno o más *sites* individuales fallen. Cuando este es recuperado, su base de datos local deberá ser actualizada y sincronizada a las demás antes de unirse al sistema.
c) Falla en líneas de comunicación.- El sistema deberá de operar con fallas de una o más líneas de comunicación entre los *sites* Un caso extremo que pudiera ocurrir, es el de *particionamiento de la red*, el cual divide los *sites* dentro de dos o más particiones, donde los *sites* pertenecientes a una misma partición se pueden comunicar entre sí, pero no con *sites* de otra partición.
d) Confirmaciones de transacciones distribuidas.- Cuando se confirma una transacción que actualizará bases de datos almacenadas en diferentes *sites*, el problema principal que se genera, es que hacer cuando algún o algunos de estos llegara a fallar. Existe varios métodos como el protocolo de confirmación de las dos fases que más adelante se explicará.
e) Bloqueo mutuo distribuido.- Cuando una o más peticiones requieren los mismos datos, hay que tomar en cuenta el bloqueo mutuo, este puede ocurrir en los diferentes *sites*, por lo que es necesario aplicar los mecanismos conocidos para evitar este problema.

A continuación se presentan algunas técnicas para el control de concurrencia y recuperación en base: de datos distribuidas.

2.1.7.1 Control de Concurrencia Distribuido basado en una copia Distinguida de los Datos

La idea de este método es designar una copia particular de los datos a la que se denomina *copia distinguida*, esta copia podrá residir o no en el mismo *site* que mantiene los datos originales. Cualquier petición que se haga de los datos se definirá sobre la copia distinguida, así mismo, las actualizaciones afectarán a la copia y no a los datos originales. El *DDBMS* será el encargado de actualizar a los datos originales. Existen 4 técnicas basadas en este método que a continuación se explicarán.

2.1.7.1.1 Técnica del Site Primario

Este método como su nombre lo indica, designa un único *site* que sirve de *coordinador* entre todos los elementos de la base de datos, éste será el responsable de procesar todas las peticiones de bloqueo, ya sea otorgando o denegando dichas peticiones. Este método es una extensión del método utilizado en bases de datos centralizadas. Una de sus ventajas es su simplicidad y su fácil implementación, sin embargo sus desventajas son vistas inmediatamente: un solo *site* encargado de recibir peticiones, puede ser saturado, causando cuellos del botella al sistema total, después de la saturación puede venir la falla de quedar indefinidamente fuera de línea.

Aunque en este método un solo *site* es el encargado de responder las peticiones, los datos se acceden desde cualquier otro dentro de la base de datos y como se mencionó anteriormente, si una transacción obtuvo un bloqueo para lectura o escritura, en realidad el acceso se hace a la copia y el *DDBMS* será responsable, si así se ha solicitado, de actualizar *todas las copia* replicadas de ese dato antes de liberar el bloqueo.

2.1.7.1.2 Técnica del *Site* Primario con copia de respaldo

Esta técnica es igual a la anterior sólo que además, mantiene otro *site* como copia de respaldo del *site* primario, cualquier operación en el primario se reflejará en el respaldo. La recuperación en esta técnica es más fácil, porque si ocurre una falla en el primario, el respaldo pasa a ser el coordinador manteniendo las mismas características, evitando recurrir a otros procesos más complejos de la recuperación. Sin embargo esta técnica resulta más lenta, pues las peticiones se tendrán que duplicar en dos *sites* diferentes antes de responderlas. La saturación y la lentitud de las peticiones son el principal problema esta técnica.

2.1.7.1.3 Técnica de la Copia Primaria

A diferencia de las dos técnicas anteriores, esta técnica utiliza como coordinador a varios *sites*, teniendo copias distinguidas de los datos en cada uno de ellos. En caso de que uno llegue a fallar, éste sólo afectará aquellas peticiones que procesaba y no a todo el conjunto mantenido por otros *sites*. Además esta técnica acepta también copias de respaldo y es mucho más rápida, confiable y disponible.

2.1.7.1.4 Técnica de Selección de un nuevo Coordinador en caso de falla

Cuando un *site* coordinador falla en cualquiera de las técnicas descritas, los que permanecen activos deberán de elegir un nuevo coordinador. En la primera técnica donde no hay respaldo, todas las transacciones son abortadas y el proceso de recuperación se torna largo y tedioso. Parte de esta recuperación consiste en escoger un *site* primario y crear un *administrador de procesos de bloqueo* con

la información que mantenía el *site* que falló. En la segunda técnica descrita, en caso de falla del coordinador, todas las transacciones se suspenden mientras el *site* respaldo es designado primario, a continuación se escoge otro *site* respaldo, se sincroniza y se envía copia de toda la información necesaria para su buen funcionamiento. El caso extremo de esta técnica es cuando tanto el primario como el respaldo fallan; para esto debe de existir un proceso llamado *elección* que es usado para escoger el nuevo coordinador. En este proceso, cualquier *site X* que intenta comunicarse con el coordinador repetidas veces sin obtener respuesta, asume que el coordinador está fuera de línea e inicia el *proceso elección*: envía un mensaje a todos los *sites* participantes para que propongan a *X* como el nuevo coordinador. Posteriormente *X* recibe la aceptación y pasa a ser el *site* primario. El algoritmo de elección es por si mismo demasiado complejo pero es la idea principal detrás del método de elección. El algoritmo debe de resolver cualquier caso de empate de dos o más *sites* que llegan a proponerse como coordinadores

2.1.7.2 Control de Concurrencia Distribuido basado en Votación

A diferencia de los anteriores, el método de votación no mantiene una copia distinguida, sino que las peticiones de bloqueo se envían a todos los *sites* que incluyen copia de los datos solicitados, cada copia mantiene sus bloqueos y puede otorgar o denegar la petición. Si una transacción solicita un bloqueo y se le otorga en la mayoría de los *sites* donde residen las copias, entonces dicha transacción mantiene el bloqueo y envía un mensaje a todos informando que el bloqueo le fue otorgado. Si a una transacción no se le otorga el bloqueo de los datos en la mayoría de los *sites* participantes durante un determinado período de tiempo, la petición se cancela y se informa a todos de dicha cancelación.

Este método es considerado un verdadero método de control de concurrencia distribuido, ya que la responsabilidad de las decisiones residen en todos los *sites* y no solo en uno. Los estudios de simulación muestran que este método tiene un más alto tráfico de mensajes entre los *sites* que los métodos de copia distinguida. Su implementación es compleja, más aún si se toman en cuenta las fallas, pero es un método distribuido que utiliza el trabajo en equipo, propio de la distribución.

2.1.7.3 Recuperación Distribuida

Los procesos de recuperación en una base de datos distribuida son complicados, pues la mayoría de las veces es difícil o hasta imposible determinar cuando un *site* presentará una falla. Una técnica de recuperación es el estar mandando constantemente mensajes entre estos. Pueden suceder tres casos cuando los mensajes no son recibidos:

1.- El mensaje no fue recibido porque falló el canal de comunicación
2.- El mensaje no fue recibido porque el *site* receptor esta fuera de línea
3.- Se recibe el mensaje y se responde pero la respuesta nunca llega

Con estos casos ya resulta posible hacer un chequeo exhaustivo para ver cual es el error y la solución a tomar.

Si no se contará con el envío de mensajes, sería difícil determinar que camino tomar cuando hubiera una falla.

Cuando se hacen transacciones, es necesario tener listas las técnicas de recuperación para poder aplicarlas cuando la información se pierda. Una de ellas es la de no hacer una confirmación hasta que se asegure que cada *site* participante mantiene una bitácora local que evite la pérdida de datos. *El protocolo de las dos fases* es frecuentemente usado para evitar pérdidas y asegurar la correcta

confirmación de la distribución. Este método consiste en dos acciones que se toman antes de realizar cambios en la base de datos:

La primera acción llamada fase 1, prepara a todas las bases en los diferentes *sites* participantes en la transacción para estar listas para escribir en los registros, estos deberán enviar una respuesta afirmativa, si alguno de ellos no está listo por alguna falla, envía una respuesta negativa, o bien si no puede enviar la respuesta debido a la misma falla el coordinador dará un determinado intervalo de tiempo en el cual si no hay respuesta se asume que dicho *site* no está listo y se cancela la transacción.

La segunda acción llamada fase 2, se realiza cuando todas las bases de datos participantes en los diferentes *sites*, han enviado una respuesta satisfactoria de que están listas para hacer la modificación, en ese momento, se debe generar un archivo de bitácora para hacer posible la recuperación y a continuación se confirma la transacción. El archivo de bitácora es para cada uno de los *sites* participantes, si alguno fallara en el momento de la transacción, la información en el archivo de bitácora serviría más adelante para restaurar y completar la acción a nivel local.

2.1.8 Conclusión

Las bases de datos distribuidas han resurgido entre las empresas como una opción más, a raíz de la expansión de los sistemas abiertos y la red Internet. Aunque su teoría resulta compleja, las técnicas existentes para su manejo son requeridas sobre todo en *DDBMS* que compiten en el mercado. El uso de la distribución en una base de datos es hasta hoy poco explotado, pero es la tendencia de los principales sistemas tanto gubernamentales como de negocios privados, por lo que es necesario aprovechar al máximo la capacidad que nos proporciona tal manejo para desarrollar sistemas más confiables y seguros.

2.2 Redes

2.2.1 Introducción.

Una red de computadoras esta formada mínimamente por dos computadoras que se comunican entre si, y son capaces de intercambiar información. Los objetivos de estas redes son básicamente:

a) Compartir recursos: hacer que programas, datos y recursos estén disponibles para cualquier usuario que se encuentre conectado en la red y así lo solicite.

b) Alta fiabilidad: en una red de computadoras se tienen fuentes alternas de suministros como son unidades de procesamiento y fuentes de almacenamiento, en los cuales como un ejemplo se pueden tener almacenados los mismos archivos en varias computadoras y en caso de una falla de hardware de alguna de ellas, poder utilizar la copia de los archivos de otra computadora.

c) Costo: las computadoras pequeñas tienen una mejor relación costo/rendimiento, comparada con la ofrecida por las maquinas grandes.

d) Medio de comunicación: las redes de computadoras ofrecen un medio de comunicación rápido y efectivo entre personas que se encuentran muy alejadas.

La conexión entre computadoras no necesita hacerse a través de un hilo de cobre; también puede hacerse mediante el uso de fibra óptica, microondas y satélites de comunicaciones. En general las redes se pueden clasificar en dos grupos:

- Redes de área local (LAN, *Local Area Network*), conecta computadoras cercanas una de la otra. Las tecnologías LAN proporcionan las velocidades de conexión mas altas entre computadoras, pero sacrifican la capacidad de recorrer largas distancias. Por ejemplo, una LAN común recorre una área pequeña, y opera dentro de un rango que va de los 10 Mbps a los 2 Gbps, como la tecnología LAN cubre distancias cortas, ofrece tiempos de retraso muchos menores que las WAN.
- Redes de área amplia (WAN, Wide Area *Network*) constan de computadoras que se encuentran en varias ciudades, estados e incluso países, se difiere a las redes LAN por el enorme trayecto que debe recorrer la información cuando se están comunicando las redes. Las WAN operan mas lentamente que las LAN y tienen tiempos de retraso mucho mayores entre las conexiones. La velocidad normal para una WAN llega a un rango que va de los 56 Kbps a 155 Mbps. Los retardos para una WAN pueden variar de unos cuantos milisegundos a varias decenas de segundos debido al envío de señales a los satélites en órbita alrededor de la tierra.

2.2.2 Topología

La topología de red describe la manera en que se configura una red. Existen tres topologías básicas: bus, anillo y estrella.

a) La topología de bus, tiene un canal común que utilizan todos los equipos conectados. Cada equipo se conecta a este cable y transfiere datos directamente a la computadora con la que desea comunicarse. La ventaja principal es que no requiere mucho cableado como otras topologías y cada equipo debe conectarse a un solo cable, la desventaja es que todos los equipos tienen que

compartir el cable y competir por el acceso, otra ventaja es que un falso contacto o una falla en un equipo no basta para desactivar el sistema de red. La figura 2.9 ilustra la topología de bus.

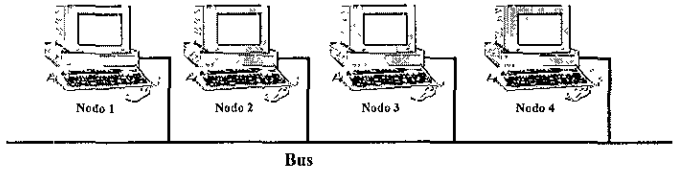


figura 2.9 Topología de bus.

b) *La topología de anillo*, conecta cada equipo a otros dos, los datos pasan de un equipo a otro alrededor del anillo hasta llegar a su destino. El adaptador de interfaz de red de cada equipo tiene que tomar los datos de una maquina y pasarlos a la siguiente, si no se cuenta con hardware especial, una falla en un solo equipo puede causar que falle toda la red. La figura 2.10 ilustra la topología de anillo.

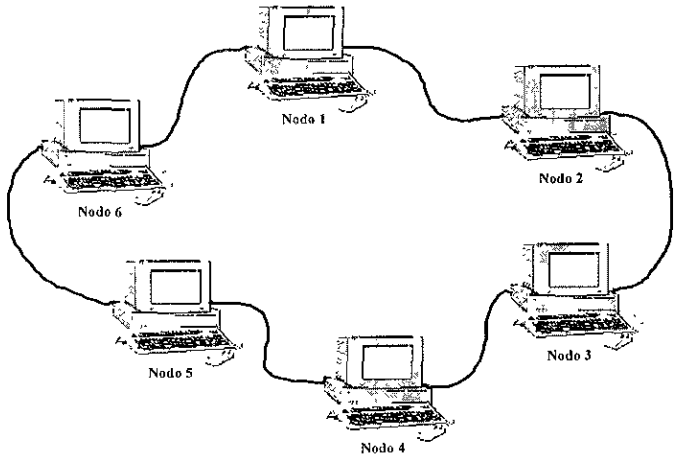


figura 2.10 Topología de anillo.

c) La topología de estrella emplea un equipo en el centro de la red y uno en cada extremo de cada pico de la estrella, todos los mensajes se canalizan a través del equipo central, esta topología requiere demasiado cableado, si este equipo central falla, falla toda la red. Una ventaja de esta topología es que cada conexión no tiene que soportar múltiples equipos en competencia por acceso, logrando altas frecuencias de transferencia de datos. La figura 2.11 ilustra la topología de estrella.

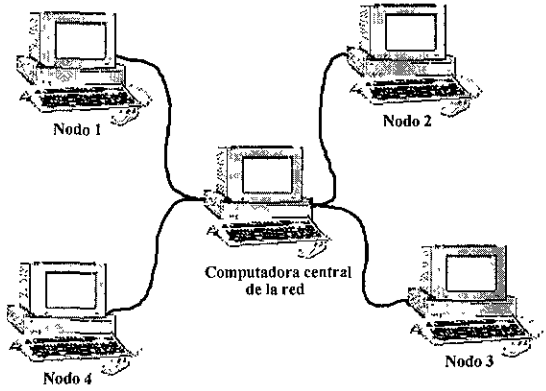


figura 2.11 Topología de estrella.

2.2.3 Arquitectura

La arquitectura de una red de computadoras es un arreglo ordenado y complejo de partes interconectadas. Una arquitectura de red muestra la manera en como están unidas las partes del sistema y como interactúan con el medio. El punto de contacto donde interactúan las partes del sistema se conocen como interfaz.

Las interfaces que interactúan de software a software son llamadas interfaces de aplicación del programador (API' s), las cuales hacen posible que se comuniquen los programas, sistemas operativos y todos los demás componentes de software.

Las redes se organizan en una serie de capas o niveles, con objeto de reducir la complejidad de su diseño. Cada capa se construye sobre su predecesora. El número de capas, nombre, contenido y función de cada una varían de una red a otra. Sin embargo, en cualquier red, el propósito de cada capa es ofrecer ciertos servicios a las capas superiores, liberándolas del conocimiento detallado sobre como se realizan dichos servicios.

El modelo OSI (Interconexión de Sistemas Abiertos, desarrollado por la Organización Internacional de Normas ISO) ; utiliza capas para organizar una red en módulos funcionales bien definidos. La figura 2.12 muestra las capas de la red en el modelo ISO/OSI.

7	Capa de aplicación
6	Capa de presentación
5	Capa de sesión
4	Capa de transporte
3	Capa de red
2	Capa de enlace (de datos)
1	Capa física

figura 2.12 Capas de la red en el modelo ISO/OSI.

2.2.4 Capas de red en el modelo OSI.

En una red de capas, cada una de ellas emplea protocolos bien definidos para comunicarse con las que la rodean, en el modelo OSI se emplea el nombre de la capa para identificar el protocolo. Conceptualmente cuando dos computadoras hablan entre sí, las capas correspondientes en cada una sostienen una conversación. Toda comunicación entre dos computadoras sucede en la capa (física) inferior de la red, donde existe la conexión real en hardware, los datos enviados desde una computadora fluyen de modo vertical a través de las capas hasta llegar a la capa física y al llegar a esta capa los datos fluyen horizontalmente a través de las líneas de transmisión de la red hacia la computadora de destino.

Cada una de las capas del modelo OSI, proporciona servicios específicos de comunicación a la capa arriba de ella, esto es, cada una de las capas de red cumple una función especifica de la cual depende la capa de arriba de ella con el fin de incrementar su funcionalidad.

Cada capa de la red sigue reglas especificas (protocolos) que definen los servicios que ofrece. Un servicio define la habilidad que una capa ofrece a la capa superior, como la detección de errores. Un protocolo es el conjunto de reglas que la capa debe seguir para poner en marcha el servicio, estos protocolos definen los formatos de los paquetes y la estructura de los datos que debe usarse para el servicio solicitado.

Las capas pueden ofrecer dos tipos diferentes de servicios a las capas que se encuentran sobre ellas, uno orientado a conexión y otro sin conexión.

En el servicio orientado a conexión, el usuario establece una conexión, la utiliza y después termina la conexión, en cambio en un servicio sin conexión el usuario únicamente transmite su información a través de la red y espera a que llegue a su destino.

2.2.4.1 Capa Física

La capa física es la encargada de transmitir datos a través de los canales de la red. En esta capa se incluyen los elementos físicos (hardware) necesarios para cumplir esta función. Los métodos de transmisión, incluyendo las señales de transmisión, también forman parte de esta capa.

Esta capa se encarga de determinar las propiedades mecánicas y eléctricas de los canales de comunicación, así como los detalles de procedimientos relacionados con estas características.

La capa física determina las frecuencias de señal analógica que representan un 0 y un 1, y decide el punto en el que una frecuencia cambiante pasa de 1 a 0 y viceversa.

El propósito de la capa física consiste en transportar el flujo original de bits de una maquina a otra. Normalmente, se utilizan varios medios físicos para realizar una transmisión como por ejemplo par trenzado, cable coaxial de banda base, cable coaxial de banda ancha, fibras ópticas, transmisión por trayectoria óptica, etc.

2.2.4.2 Capa de Enlace

La capa de enlace da formato o transforma los datos binarios puros en algo con sentido para la capa de red, por lo regular, en tramas de datos. Además, acepta información de la capa de red y traduce los datos al formato binario correcto para la capa física. En todos los casos la capa de enlace asegura que

las transmisiones binarias entre anfitriones de la red carezcan de errores. La función principal de la capa de enlace es prevenir la corrupción de los datos.

La capa de enlace cumple con un número de funciones especificas, proporcionar una interfaz de servicio muy bien definida a la capa de red, determinar como los bits correspondientes a la capa física están agrupados en tramas, ocuparse de los errores de transmisión, y regular el flujo de las tramas, de tal manera que los receptores lentos no se vean desbordados por los transmisores rápidos y la de la administración del enlace en general.

El principal servicio es transferir datos a la capa de red de la maquina de origen, a la capa de red de la maquina de destino. En la capa de red de la maquina de origen hay una entidad, llamada proceso, que entrega algunos bits a la capa de enlace para su transmisión hacia la maquina de destino.

El trabajo que realiza la capa de enlace consiste en transmitir los bits a la maquina destino, de tal forma que puedan entregarse a la capa de red en el otro extremo de la red. Los servicios que la capa de enlace proporciona varían de sistema a sistema; los mas comunes son:

- Servicio sin conexión y sin asentimiento: este servicio consiste en hacer que, la maquina origen, transmita tramas independientes a la maquina destino, sin que esta proporcione un asentimiento. No se establece ninguna conexión previa, ni tampoco se libera posteriormente.
- Servicio sin conexión y con asentimiento: en este servicio cada una de las tramas transmitidas se asiente en forma individual. De esta manera, el transmisor sabe cuando la trama llega bien al otro extremo. Si la trama no llega dentro de un intervalo de tiempo especificado, entonces puede comenzar a transmitirla nuevamente.
- Servicio orientado a conexión. Con este servicio las maquinas origen y destino establecen una conexión antes de transmitir algún dato. Cada una de las tramas transmitidas a través de la conexión se numera, y la capa de enlace garantiza que cada trama transmitida sea recibida. También garantiza que cada una de las tramas se reciba exactamente una sola vez, y que todas las tramas se reciban en el orden correcto.

Otro aspecto importante en la capa de enlace es controlar el flujo de información que un emisor desea transmitir en tramas a una velocidad mayor a la que puede aceptar el receptor Para llevar a cabo este control se necesita algún mecanismo de retroalimentación, que le permita al emisor saber si escapar o no del receptor para seguirlo.

El protocolo contiene reglas bien definidas con respecto a cuando el emisor debe transmitir la siguiente trama. Estas reglas por lo general prohiben que las tramas se envíen hasta no tener conocimiento de que el receptor haya dado permiso para ello ya sea en forma implícita o explícita.

2.2.4.3 Capa de red.

La capa de red determina la ruta o trayectoria que siguen los datos para llegar a su destino. Por eso debe manejar el trafico, el congestionamiento y la velocidad a través de las líneas de transmisión, así como la corrupción de datos en la red. Esta capa define la interfaz entre las computadoras anfitrión y toda comunicación de paquetes ubicada entre las direcciones fuente y destino. Esta capa también es responsable de asegurar la secuencia adecuada. Las capas física, de enlace y de red comparten la responsabilidad de asegurar la integridad de los datos dentro de un canal de comunicación.

Como la función de la red consiste en el encaminamiento de paquetes, desde la maquina origen hasta la maquina destino. En la mayoría de las subredes, los paquetes necesitaran realizar múltiples saltos para terminar el viaje. El algoritmo de encaminamiento es aquella parte del software correspondiente a la capa de red, que es la responsable de decidir sobre que línea de salida se deberá transmitir un paquete que llega. Si la subred utiliza Datagrama internamente, esta decisión deberá tomarse de nuevo con cada paquete de datos que llega.

Cuando se tiene la presencia de muchos paquetes en la subred, el rendimiento se degrada. Esta situación se conoce con el nombre de congestión, en la figura 2.13 se describen sus síntomas.

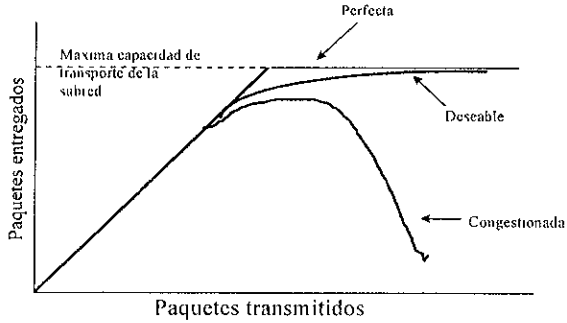


figura 2.13 Congestión en una red .

Cuando el número de paquetes que los nodos introducen en la subred se encuentran dentro de los limites de capacidad de transporte, dichos paquetes se entregan a sus respectivos destinos sin mayor problema y, la cantidad que se entrega es proporcional a la cantidad que se transmite. Sin embargo a medida que el trafico se incrementa en forma considerable, los paquetes de información comienzan a perderse y el rendimiento se degrada completamente y, casi ningún paquete se entrega. Este es problema es resuelto por la capa de red.

2.2.4.3.1 Protocolo IP

La capa de red es el corazón de cualquier red basada en el protocolo TCP/IP. La figura 2 14 muestra los módulos del software IP en la capa de red.

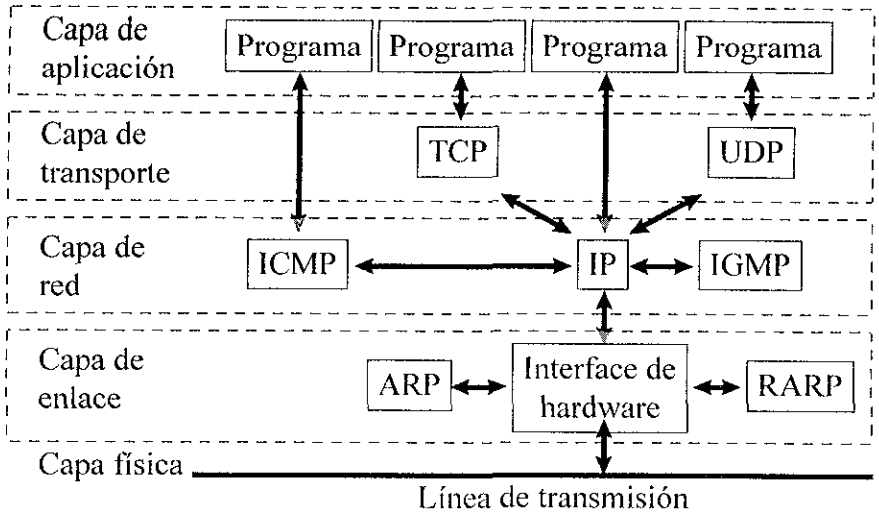


figura 2.14 Módulos del software IP en la capa de red.

Esta capa incluye al Protocolo de Internet (IP), el Protocolo de Control de Mensajes de Internet (ICMP, Internet Control Message Protocol) y el Protocolo de Manejo de Grupos de Internet (IGMP, Internet Group Management Protocol). El protocolo IP hace casi todo el trabajo dentro de la capa de red. ICMP e IGMP son protocolos de apoyo para IP, pues lo ayudan a manejar mensajes especiales de la red, como los mensajes de error y de transmisiones múltiples.

El Protocolo Internet es el sistema de entrega para el conjunto de protocolos TCP/IP. En una red TCP/IP, un datagrama del Protocolo Internet encapsula todos los protocolos excepto los de resolución de direcciones. El encapsulamiento consiste en guardar la información en el formato requerido por el protocolo siguiente del nivel inferior, conforme los datos viajan por la pila de protocolos, cada capa trabaja sobre lo que encapsuló la capa anterior.

El Protocolo Internet, proporciona tres definiciones importantes. Primero define la unidad básica para la transferencia de datos utilizada a través de una red, es decir especifica el formato exacto de todos los datos que pasaran a través de una red. Segundo, el software IP realiza la función de ruteo, seleccionando la ruta por la que los datos serán enviados. Tercero, además de aportar especificaciones formales para el formato de los datos y el ruteo, el IP incluye un conjunto de reglas que le dan forma a la idea de entrega de paquetes no confiable. Las reglas caracterizan la forma en que los anfitriones y ruteadores deben procesar los paquetes, como y cuando se deben generar los mensajes de error y las condiciones bajo las cuales los paquetes pueden ser descartados.

2.2.4.3.2 Direcciones IP

Una red TCP/IP asocia las direcciones IP con una tarjeta de interfaz, no con la computadora anfitrión. Cada tarjeta de interfaz de red conectada debe tener su propia dirección IP; sin embrago una sola computadora puede tener varias tarjetas de interfaz de red, lo que significa que una sola computadora puede tener varias direcciones IP validas.

Una dirección IP es de 32 bits, o 4 bytes de longitud. No obstante la mayoría de las veces se escriben las direcciones IP en notación decimal, la cual representa cada byte de dirección IP como una serie de números decimales separados por puntos, por ejemplo, las siguientes direcciones representan la misma dirección IP:

Dirección IP en números binarios: 10000110 00011000 00001000 01000010

Dirección IP en números enteros: 2,249,721,922 (o -2,045,245,374)

Dirección IP en números hexadecimales: 0x86180842

Dirección IP en notación decimal: 134.24.8.66

La dirección IP de 32 bits incluye un número de red y un número de dirección. De acuerdo a su diseño original, el byte de mayor orden en una dirección IP identifica el número de la red, y los tres bytes de menor orden identifican a la computadora anfitrión (interfaz).

2.2.4.3.3 Ruteo en una red con protocolo IP

El ruteo en una red con protocolo IP puede ser difícil en especial entre computadoras que tienen muchas conexiones físicas de red. De forma ideal, el software de ruteo examinaría aspectos como la carga de la red, la longitud del datagrama o el tipo de servicio que se especifica en el encabezado del datagrama, para seleccionar el mejor camino. Sin embargo, la mayor parte del software de ruteo en red es mucho menos sofisticado y selecciona rutas basándose en suposiciones sobre los caminos mas cortos.

El algoritmo de ruteo en una red de redes utiliza solo direcciones IP y se controla por medio de tablas. Aunque es posible que una tabla de ruteo contenga una dirección de destino de un anfitrión especifico, la mayor parte de ellas solamente contienen direcciones de red para mantenerse de un tamaño reducido, la utilización de una ruta asignada por omisión también puede ser útil para mantener reducida una tabla de ruteo, especialmente para los anfitriones que acceden a solo un ruteador.

2.2.4.4 Capa de transporte

Su tarea consiste en hacer que el transporte de datos se realice en forma segura y económica, desde la maquina fuente hasta la maquina destinataria, independientemente de la red o redes físicas que se encuentran en uso

El objetivo fundamental de la capa de transporte consiste en proporcionar un servicio eficiente, fiable y económico a sus usuarios, normalmente entidades (por ejemplo procesos) de la capa de sesión. Para alcanzar este objetivo, la capa de transporte utiliza los servicios que proporciona la capa de red.

La capa de red y de transporte son muy similares, pero la función de la capa de transporte hace posible que el servicio de transporte sea mas fiable que el proporcionado por la capa de red subyacente. Los paquetes extraviados, los datos dañados e incluso los N-RESET de la red pueden ser detectados y compensados por la capa de transporte. Además, las primitivas de servicio de la capa de transporte pueden diseñarse para ser independientes de las primitivas de servicio de la capa de red, que pueden variar considerablemente de red a red.

Gracias a la capa de transporte, es posible que los programas de aplicación puedan escribirse utilizando un conjunto normalizado de primitivas, y hacer que dichos programas funcionen en una gran variedad

de redes, sin tener que preocuparse de la manera de tratar con diferentes interfaces de cada subred y con transmisiones inseguras.

2.2.4.4.1 Puertos de la Capa de Transporte

En la terminología relacionada con TCP/IP, un puerto es como una dirección IP excepto que TCP/IP lo asocia con un protocolo, no con una computadora anfitrión. Del mismo modo que los Datagramas almacena las direcciones IP origen y destino, los protocolos de transporte almacenan los números de los puertos del origen y del destino.

Puesto que TCP es un protocolo confiable orientado a conexión la entrega de los datos se centra en la conexión, no en el puerto. Por ejemplo, las aplicaciones que utilizan TCP pueden abrir conexiones múltiples en el mismo puerto y así comunicarse sin problemas.

La capa de transporte enruta los paquetes de y hacia los programas de aplicación; por lo tanto, la capa de transporte necesita una forma de identificar cada aplicación. Es ahí donde entran los números de puerto. Cada aplicación, sin importar que sea cliente o servidor, tiene un número de puerto único. Cuando un programa crea una sesión se le asigna un número de puerto, un programa cliente no necesariamente necesita conocer el número de puerto que utiliza, pero un programa servidor sí.

2.2.4.4.2 Protocolo de Control de Transmisión (TCP)

Para asegurar la confiabilidad y la secuencia del flujo de bytes, TCP utiliza una técnica conocida como acuse de recibo positivo con retransmisión. Esta técnica requiere que un receptor se comunique con el origen y le envíe un mensaje de acuse de recibo (ACK) conforme recibe los datos El transmisor guarda un registro de cada paquete que envía y espera un acuse de recibo antes de enviar el siguiente paquete. El transmisor también arranca un temporizador cuando envía un paquete y lo retransmite si dicho temporizador expira antes de que llegue un acuse de recibo

La técnica de acuse de recibo positivo ocupa una cantidad sustancial de ancho de banda de red debido a que debe retrasar el envío de un nuevo paquete hasta que reciba un acuse de recibo del paquete anterior, para mejorar la capacidad de transferencia de los mensajes, TCP no envía un mensaje y espera hasta recibir el mensaje de confirmación antes de transmitir otro mensaje, sino que utiliza otro concepto llamado ventana deslizante (ver Apéndice A) , que permite transmitir varios mensajes mientras espera los mensajes de confirmación, la técnica coloca una ventana pequeña y de tamaño fijo en la secuencia, y transmite todos los paquetes que residan dentro de la ventana.

El objetivo del concepto de ventana deslizable es no tener ociosa la red esperando un mensaje de confirmación y de esta manera aprovechar mejor el ancho de banda de la red así como su desempeño.

El TCP proporciona mediante estas técnicas el servicio de flujo confiable deduciendo que el TCP es un protocolo de comunicación, no una pieza de software. TCP especifica el formato de datos y los acuses de recibo que intercambian dos computadoras para lograr un transferencia confiable, así como los procedimientos que la computadora utiliza para asegurarse de que los datos lleguen de manera correcta. También especifica como el software TCP distingue el destino correcto entre muchos destinos en una misma maquina, y como las maquinas en comunicación resuelven errores como la perdida o duplicación de paquetes.

2.2.4.5 Capa de sesión

La función principal de la capa de sesión consiste en proporcionar una manera por medio de la cual los usuarios de la capa de sesión (por ejemplo, las entidades de presentación) establezcan conexiones, llamadas sesiones, y transfieran datos sobre ellas en forma ordenada. Una sesión podría utilizarse para un acceso remoto desde una terminal a un ordenador remoto, o para una transferencia de archivos, o para cualquier otro propósito.

La característica mas importante de la capa de sesión es el intercambio de datos. Una sesión, al igual que una conexión de transporte, sigue un proceso de tres fases, la de establecimiento, la de utilización y la de liberación.

Existen importantes diferencias entre una sesión y una conexión de transporte. La principal entre estas es la forma como se liberan las sesiones y las conexiones de transporte. Otro servicio de la capa de sesión es la sincronización, la cual se utiliza para llevar las entidades de sesión de vuelta a un estado conocido, en caso de que haya un error o algún desacuerdo. La capa de transporte se ha diseñado cuidadosamente para que se pueda recuperar, en forma transparente, de todos los errores de comunicación, así como de fallos de las subredes. Sin embargo, un estudio mas detallado demuestra que la capa de transporte se ha diseñado para enmascarar solamente los errores de comunicación. Esta no se puede recuperar de los errores cometidos en la capa superior.

Los usuarios de sesión pueden dividir el texto en paginas, e insertar un punto de sincronización entre cada una de ellas. En caso de presentarse un problema, es posible restablecer el estado de la sesión a un punto previo de sincronización, para desde ahí continuar, para hacer posible este proceso, llamado resincronización, el usuario de sesión emisor, deberá continuar reteniendo los datos durante el tiempo que sea necesario

Otra de las características clave de la capa de sesión, es la administración de actividades. La idea tras la administración de actividades es la de permitir que el usuario divida el flujo de mensajes en unidades lógicas denominadas actividades. Cada actividad es completamente independiente de cualquiera de las demás que pudieron haber venido antes o que vendrán después de ella.

Por ejemplo, considérese una sesión que se haya establecido con el propósito de transferir varios archivos entre dos ordenadores Se necesita alguna forma para marcar el lugar en donde termina un archivo y comienza el siguiente. El empleo del carácter ASCII FS (Separador de archivos) no es la mejor idea, por que si los archivos contienen información binaria, estos caracteres podrían aparecer en los datos y señalar accidentalmente el final de un archivo, cuando no se trataba de esto. El empleo de algún tipo de inserción de carácter, como se hizo en la capa de enlace, es una posibilidad, pero en la capa de sesión la inserción tendrá que hacerse probablemente en software y no en hardware, por lo que el procedimiento será lento.

Las actividades están estrechamente relacionadas con los puntos de sincronización. Cuando se inicia una actividad, los números de serie de los puntos de sincronización vuelven a 1 y se inserta un punto de sincronización mayor. Dentro de una actividad es posible establecer puntos de sincronización adicionales, tanto mayores como menores.

Dado que el inicio de una actividad también corresponde a un punto de sincronización mayor, una vez que se inicia una actividad, ya no es posible resincronizar a un punto anterior a aquel correspondiente al inicio de dicha actividad. En particular no es posible resincronizar a un punto de sincronización de una actividad previa.

2.2.4.5.1 Modelo cliente-servidor

Se ha establecido que dos procesos que se comunican a través de una conexión de sesión o transporte son simétricos. En la práctica, esta suposición no se cumple frecuentemente. Un ejemplo común es una red de computadoras o estaciones de trabajo sin discos, llamados clientes, que se están comunicando a través de una red con un servidor de archivos que tiene un disco sobre el cual se almacenan todos los archivos. En este sistema, los clientes acceden a sus datos mediante la transmisión de solicitudes al servidor, el cual se encarga de llevar a cabo el trabajo y devuelve respuestas. La comunicación siempre toma la forma de parejas de solicitud-respuesta, siempre iniciadas por el cliente y nunca por el servidor A este modelo se le conoce con el nombre de modelo cliente-servidor.

Mientras que resulta obvia la posibilidad de establecer sesiones entre clientes y servidores, y después utilizar comunicación semiduplex en esas sesiones, la gran sobrecarga ocasionada por múltiples capas de conexiones viene a ser, por lo general, poco atractiva para aplicaciones, como servidores de archivos, en donde el rendimiento es sumamente crítico

Desde este punto de vista, un cliente que transmite un mensaje a un servidor y obtiene una respuesta, es equivalente a un programa que llama a un procedimiento y obtiene un resultado. En ambos casos el que hace la llamada inicia una acción y espera hasta que se complete, y los resultados estén disponibles. Aunque en el caso local, el procedimiento se ejecuta en la misma maquina del que hace la llamada, y con una llamada de procedimiento remoto que se ejecuta en una maquina diferente, el que hace la llamada no necesita preocuparse de esta diferencia.

Este esquema consiste ahora que la comunicación cliente-servidor se hace por medio de llamadas a procedimientos, y no como comandos de E/S o de interrupciones. Todos los detalles de la red se ocultan del programa de aplicación, poniéndolos en procedimientos locales. Estos procedimientos se llaman cabos (stub).

2.2.4.5.2 Llamadas a procedimientos remotos

El paso 1 consiste en el programa o procedimiento cliente llamando al procedimiento cabo montado dentro de su propio espacio de direcciones. Los parámetros pueden pasarse de la manera normal. El cliente no nota ninguna cosa rara en esta llamada, por que se trata de una llamada normal, de tipo local.

El procedimiento cabo del cliente, entonces, colecta los parámetros y los empaqueta en un mensaje. Esta operación se conoce como encapsulado de parámetros. Después de que el mensaje se haya construido, se pasa a la capa de transporte para su transmisión (paso 2). En un sistema de red tipo LAN sin conexión, la entidad de transporte probablemente solo unirá una cabecera al mensaje y lo pondrá en la red (paso 3). Cuando el mensaje llega al servidor, la entidad de transporte lo pasa al procedimiento cabo del servidor (paso 4), el cual se encarga de *desencapsular* los parámetros. El procedimiento cabo del servidor entonces llama al procedimiento servidor (paso 5), pasando los parámetros en la forma normal. El procedimiento servidor no tiene ninguna manera de notar que esta siendo activado de forma remota, por que el que esta haciendo la llamada, es un procedimiento local que obedece todas las reglas normales. Solo el procedimiento cabo sabe que esta sucediendo algo especial.

Después de que haya completado su trabajo, el procedimiento del servidor devuelve el control (paso 6), de la misma manera que cualquier otro procedimiento devuelve el control una vez que termina.

También, puede devolver un resultado al extremo que le hace la llamada. El procedimiento cabo del servidor, entonces, encapsula el resultado en un mensaje y lo entrega a la interfaz de transporte (paso 7), posiblemente haciendo una llamada al sistema, como sucedió en el paso 2. Después de que la respuesta llega a la maquina del cliente (paso 8), se entrega al procedimiento cabo del cliente (paso 9). Por ultimo el procedimiento cabo del cliente devuelve el control al extremo que hace la llamada, es decir, al procedimiento cliente. Cualquier valor que devuelva el servidor en el paso 6, es entregado al cliente en el paso 10.

2.2.4.6 Capa de presentación

La capa de presentación contiene funciones comunes que la red emplea repetidamente durante comunicaciones en la red. Estas funciones incluyen la interfaz de la red con impresoras, monitores y formatos de archivo. La capa de presentación determina como aparecen los datos ante el usuario. Los servicios de comunicación que proporciona la capa de presentación son los siguientes:

- Cifrado de datos(en ingles *encrypt*): El propósito del cifrado de datos es la seguridad. En una red, el cifrado de datos transforma la información inteligible en ininteligible antes de la transmisión. En el extremo receptor de la conexión, la capa de presentación debe descifrar la transmisión y transformar los datos en información utilizable.
- Compresión de datos e incremento del ancho de banda: La compresión de datos comprende la conversión de datos con el objetivo de reducir su tamaño. La capa de presentación reduce la cantidad de datos que debe de transportar la red. Por lo tanto la compresión efectiva de datos en la capa de presentación mejora el desempeño general.

2.2.4.7 Capa de aplicación

La capa de aplicación contiene todos los detalles relacionados con aplicaciones especificas o programas de computo diseñados para usuarios de red. Su propósito es proporcionar a las aplicaciones servicios de comunicación, establecer y controlar el ambiente en el cual las aplicaciones pueden realizar sus operaciones.

2.3 Metodología Orientada a Objetos

2.3.1 Introducción

A lo largo del tiempo, las organizaciones han reconocido la importancia de administrar los recursos claves, tales como el trabajo y la materia prima. Actualmente, la información ha sido colocada como un recurso clave. Los tomadores de decisiones han comenzado a comprender que la información no solo es un resultado de manejar empresas, sino la base para su organización y crecimiento.

Para maximizar la utilidad de la información en una empresa, esta debe ser administrada correctamente, de la misma forma que otros recursos, de ahí el uso de las computadoras, las que han contribuido para una buena administración y manejo de ella.

Debido a la complejidad y magnitud de los sistemas, el software se desarrolla en periodos mayores a los planeados, consumiendo mayores recursos económicos y humanos. A consecuencia el software creado tiene defectos, que en su mayoría se deben a la falta de visión a futuro en su diseño y conceptualización, ya que al ser liberados la organización ha sufrido cambios, los cuales no siempre se consideran. La mayoría del software corporativo es obsoleto por los cambios a los que están expuestas las organizaciones que los requieren.

De la necesidad de buscar y aplicar nuevos enfoques para la construcción de software para los sistemas de Información, la Ingeniería de Sistemas pretende mediante metodologías innovadoras como lo es la Orientada a Objetos, crear sistemas lo mayormente confiables, flexibles, mantenibles y capaces de evolucionar para satisfacer los requerimientos de cambio de cualquier organización, además de minimizar el tiempo de desarrollo invertido en su construcción, creación de bibliotecas y código reutilizable, entre otros.

2.3.2 Conceptos Básicos

2.3.2.1 Orientado a Objetos

El término "Orientado a Objetos" significa que el software se organiza como una colección de objetos discretos que contienen tanto estructuras de datos como un comportamiento, esto se opone a la programación convencional, en la cual las estructuras de datos y el comportamiento solamente están relacionados de forma débil. También se conoce como la técnica de construcción de sistemas basada en la combinación de componentes con límites bien establecidos, que enfatiza la creación de bibliotecas de componentes reutilizables.

2.3.2.2 Objeto.

Un objeto es una representación de una cosa ya sea física o abstracta, y que forma módulos de software ideales debido a que pueden definirse y mantenerse independientemente, formando cada uno un universo autocontenido. Todo lo que un objeto conoce esta expresado en sus variables. Todo lo que puede hacer esta expresado en sus métodos. Un objeto por tanto, tiene:

- Identidad
- Atributos

- Relaciones
- Comportamiento (conjunto de métodos)

Estos componentes se muestran en la figura 2.15.

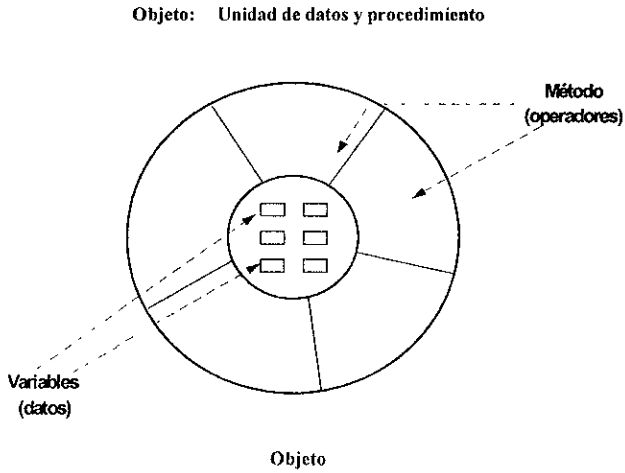


figura 2.15 Partes que conforman un objeto.

2.3.2.2.1 Identidad

Es una etiqueta única que distingue a un objeto de todos los demás.

2.3.2.2.2 Atributo

Es la abstracción de las características de un objeto, estos atributos deberán hacer al objeto: mas completo, exclusivo y mutuamente independiente, es decir, que tomen su valor independientemente uno de otro.

El conjunto de atributos se conoce también como variables de instancia porque definen el estado del objeto en cualquier momento. El valor de las variables puede cambiarse dinámicamente durante la vida de un objeto.

2.3.2.2.3 Relación

Es un evento que relaciona a dos o mas objetos por medio de atributos.

2.3.2.2.4 Métodos

Un método es como el conjunto de operaciones que son capaces de realizar los objetos de una clase. Es como un objeto, actúa o reacciona a los mensajes que otros objetos le envíen.

2.3.2.3 Mensaje

La forma en que los objetos interactúan entre si es enviándose mensajes pidiendo que se ejecute un método específico. Un mensaje consiste simplemente del nombre del objeto a quien va dirigido

seguido del nombre del método que el objeto receptor sabe como ejecutar. Si el método requiere información adicional, el mensaje incluye esta información como parámetros.

Los objetos del mundo real se pueden afectar en infinita variedad entre si: crear, agregar, mover, enviar, remover, etc.

El objeto que inicia el mensaje se conoce como transmisor y el que lo recibe como receptor. Un sistema Orientado a Objetos consiste de varios objetos interactuando unos con otros enviándose mensajes entre si. El objetivo de un mensaje consiste en la activación de alguno de los métodos que tiene el objeto receptor.

2.3.2.4 Encapsulación

Es el mecanismo que permite ocultar los detalles de la representación interna de un componente, presentando la interfaz disponible para el usuario.

2.3.2.5 Clase

Mientras un objeto es algo concreto que existe en tiempo y espacio, una clase representa solamente una abstracción, la esencia de un objeto. Podemos definir una clase como un grupo o un tipo marcado con un común atributo.

Una clase es aquel conjunto de objetos que tienen características y comportamiento semejantes, por lo tanto los objetos que no tengan atributos semejantes no pueden ser agrupados en una misma clase. La agrupación de objetos en clases es una forma de ordenar y organizar el dominio de objetos que conforman un sistema.

Las clases surgieron porque en raras ocasiones un sistema involucra un solo objeto de cada tipo. Es mucho mas común requerir mas de uno.

2.3.2.6 Herencia

La herencia es un mecanismo por el cual una clase de objetos puede definirse como un caso especial de una clase mas general, con lo cual automáticamente incluye toda la definición de métodos y variables de la clase general, es una relación entre clases, en donde una clase comparte la estructura o comportamiento definidas en una de ellas o mas con otras clase.

A la clase que hereda se le conoce como superclase y a la clase heredera se le denomina subclase. Una subclase puede añadir y/o modificar operaciones a su comportamiento, formando así una especialización de la superclase. Una clase puede ser subclase de una o mas superclases. En el primer caso se habla de herencia sencilla y en el segundo de herencia múltiple.

La herencia entre clases puede extenderse a cualquier grado. El resultado es una estructura arborescente conocida como **Jerarquía de Clases**.

Cuando hay herencia de clases, los métodos definidos en una superclase se heredan a los objetos de las subclases.

2.3.2.7 Polimorfismo

Polimorfismo, es la capacidad de un objeto para responder al mismo mensaje de una o mas clases, y cada uno de diferente forma. Esto significa que un objeto no necesita conocer quien envía el mensaje,

el solo necesita conocer cuantos tipos de objetos han sido definidos para responder a un mensaje en particular.

El polimorfismo nos permite reconocer y explotar semejanzas entre diferentes clases de objetos.

2.3.3 Características de Sistemas Orientados a Objetos

El desarrollo de sistemas con la metodología orientada a objetos tiene un doble propósito, por un lado la elaboración de herramientas que solucionen la problemática planteada y por otro, la generación de componentes que puedan ser integrados a la biblioteca de componentes, aumentando así el capital de software.

El impacto económico de la adopción de la metodología orientada a objetos no puede medirse con base a la construcción de un solo sistema, la ganancia del uso de esta metodología se muestra a largo plazo, donde la reutilización de los componentes capitalizados se vuelve importante.

La esencia de las metodologías orientadas a objetos son la abstracción de datos, tipos abstractos, reutilización de código, polimorfismo, ocultamiento o encapsulación de información. Entendiendo por:

- *Abstracción de datos.* La abstracción consiste en capturar una idea general, es decir, la esencia de una entidad mas compleja, para posteriormente representarla con algún elemento interno de un programa. El representar solo los atributos generales del objeto sin considerar características específicas ni el funcionamiento interno del objeto, permite concentrarse en la manipulación y operaciones a realizar sobre el.
- *Tipos Abstractos.* La creación de tipos abstractos permiten al programador adaptar el lenguaje de programación a sus necesidades específicas. Un tipo de datos abstracto, es un tipo de dato definido por el usuario y representa la abstracción de un objeto; por tanto se ajusta al problema que se esta resolviendo y puede ser manipulado con los tipos internos del lenguaje.
- *Reutilización de Código.* Consiste en emplear los recursos utilizados en proyectos anteriores. Las metodologías orientadas a objetos proponen dos estrategias para la reutilización de código: la composición y la herencia.
    1. Composición, permite definir una nueva clase de objetos mediante la unión de un conjunto de clases ya existentes.
    2. Herencia, esta puede ser Simple: crear una nueva clase basada en otra clase mas general; o Múltiple: crear una clase de un conjunto de clases, de tal forma que si utiliza algún método y no lo tiene, lo solicita a las clases de las cuales se derivo.
- *Polimorfismo.* Permite la manipulación de objetos de clases distintas como si fueran de la misma clase, con lo cual es posible definir interfaces uniformes para diferentes tipos de objetos.
- *Ocultamiento o Encapsulamiento de información.* Se da a raíz de la abstracción, ya que al extraer la esencia de una entidad, las características internas quedan ocultas. Un objeto debe ser transparente en cuanto a su uso, es decir únicamente debe mostrar sus atributos externos para su manipulación, ocultando su implementación, algoritmos para sus operaciones, estructuras de datos que manejan, etc.

Otra característica distintiva de los sistemas orientados a objetos es que los objetos "saben" qué operaciones se pueden aplicar sobre ellos. Este conocimiento se consigue combinando abstracciones de datos y de procedimientos en un solo componente de programa (denominado objeto).

Los objetos encapsulan datos y procesos. Las operaciones de procesamiento son parte del objeto y son iniciadas pasando un mensaje al objeto. Existe una representación especial para los sistemas orientados a objetos, y estos dependen de la metodología que se haya utilizado para desarrollarlo. En términos de mantenimiento de Software el manejo de objetos facilita la actualización.

Los tiempos promedios de corrección de fallas de un sistema, al implantarlo, se reduce notablemente, lo que da una flexibilidad para la oportuna corrección del sistema.

El uso de la tecnología de objetos ofrece la facilidad de crear prototipos, los cuales, ayudan de manera importante en el desarrollo de sistemas de información.

El manejo de esta tecnología soporta la adecuación de los sistemas de acuerdo a las necesidades cambiantes del medio organizacional. "El desarrollo del software mediante estas técnicas, además de ofrecer rapidez, ofrece una mayor confiabilidad."[IV]

2.3.4 Metodología Orientada a Objetos de Grady Booch

2.3.4.1 Definición de la Metodología Orientada al Objeto

Es un proceso creciente e interactivo, en el cual se identifican clases y objetos de acuerdo a las propiedades relevantes de un dominio en particular, con la finalidad de implantar un mecanismo que prevea el comportamiento que un modelo requiere.

La metodología Orientada a Objetos propone que durante el proceso de diseño se retome mediante el análisis de sí mismo, haciendo una mejora basado en una nueva apreciación, este proceso debe repetirse hasta estar seguro que el diseño es correcto y completo, a este proceso se le conoce como Round-trip Gestalt Design.

2.3.4.2 Principios de Modelado de Objetos

En la identificación de objetos, propone una serie de principios para facilitar su detección y modelado, los cuales son :

1. Abstracción
2. Encapsulación
3. Modularidad
4. Jerarquía
5. Tipificación
6. Concurrencia
7. Persistencia

De los cuales identifica a los primeros cuatro como fundamentales, y los últimos tres como no esenciales pero sí muy útiles.

2.3.4.2.1 Abstracción

Una abstracción denota las características esenciales de un objeto que lo distinguen de los otros objetos. Se pueden identificar cuatro tipos de abstracción :

- Abstracción de entidades, es un objeto que representa un modelo usado por un problema en específico.
- Abstracción de acciones, es la identificación de un objeto que provee una función generalizada, es decir, todas al funciones que pueda realizar este objeto serán del mismo tipo.
- Abstracción virtual de máquina, es aquel objeto que lleva a cabo un conjunto de operaciones, las cuales son realizadas mediante un control de alto nivel a éste.
- Abstracción de coincidencia, es aquel objeto que empaqueta un conjunto de operaciones definidas que no tienen relación entre ellas.

2.3.4.2.2 Encapsulamiento

Es el proceso que esconde todos los detalles de un objeto que no contribuyen con sus características esenciales, es decir, es la información escondida en un objeto.

2.3.4.2.3 Modularidad

Es el acto de particionar un programa en componentes individuales, que pueden reducir su complejidad. La modularidad es una propiedad de un sistema que descompone en un conjunto de módulos definidos con cohesión y libertad entre ellos. La identificación de la modularidad de objetos, clases o comportamiento de los mismos, nos facilitarán el establecimiento del mecanismo de implantación del sistema.

2.3.4.2.4 Jerarquía

Es una ordenación de abstracciones. La mayoría de las abstracciones son identificadas en un orden jerárquico, en el diseño de un sistema la jerarquización de los objetos simplifica el entendimiento de un objeto. Se pueden identificar dos tipos de jerarquización: la sencilla y la múltiple; básicamente, la jerarquía define la relación entre clases, cuando entre varias clases comparten la misma estructura y comportamiento por una clase es sencilla, en caso de ser su comportamiento definido por varias es múltiple.

2.3.4.2.5 Tipificación

Un tipo es una característica precisa de estructura o comportamiento, la cual es compartida por un conjunto de entidades. La tipificación es el coacción o ejecución de la clase de un objeto, lo cual hace que el objeto de diferentes tipos no cambie, es decir, sólo podrá hacerlo en casos especiales o restringidos.

2.3.4.2.6 Concurrencia

Es la propiedad que distingue un objeto activo de otro que no lo está , es decir, nos indica el estado de los objetos en su ciclo de vida.

2.3.4.2.7 Persistencia

Es la propiedad de un objeto de que a través de diferentes momentos y actividades existe o trasciende en el tiempo, es decir, un objeto puede continuar existiendo de un proceso a otro, ya sea por su propia existencia o a través de su cambio de dirección en el cual fue creado.

2.3.4.3 Proceso de la Metodología Orientada a Objetos

Round-trip Gestalt Design representa el proceso del diseño Orientada a Objetos y requiere de dos elementos:

1. De una rica notación, que nos deje un modelo del problema en diferentes formas y que además permita enfocarse en las partes relativamente independientes del problema, en diferentes tiempos. Ya que el hablar de un proceso interactivo significa poder trabajar con cualquier tipo de modelo.
2. La invención de algunos mecanismos que usen estas abstracciones, las cuales deberán ser documentadas en una serie de diagramas de objetos.

El diseño de los detalles de estos mecanismos nos lleva a la estructuración de clases, a través de las cuales se establece un protocolo, para cada una de ellas. Eventualmente, al final del proceso del diseño, se termina con la definición de diagramas que proveen diferentes vistas del sistema, todos ellos deben ser coherentes y consistentes, teniendo una evolución creciente de uno a otro.

Este desarrollo creciente e interactivo hace del tradicional ciclo de vida en cascada del desarrollo de software un proceso de retroalimentación de una etapa a otra. El ciclo de vida tradicional es fundamentalmente un proceso pobre, y generalmente viola muchos de los principios de la práctica de la ingeniería, lo cual no quiere decir que el diseño debe ser realizado sin una estructura.

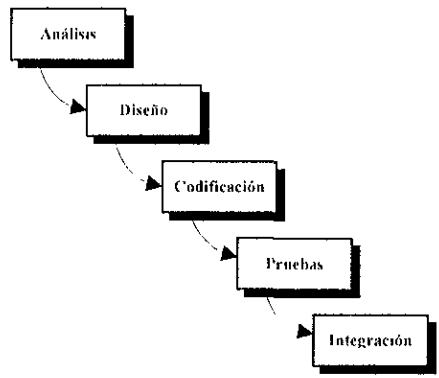


figura 2.16 Ciclo de vida del desarrollo de sistemas en la forma tradicional de cascada.

En su lugar, se sugiere que se permita un desarrollo evolucionado, es decir, un proceso espiral.

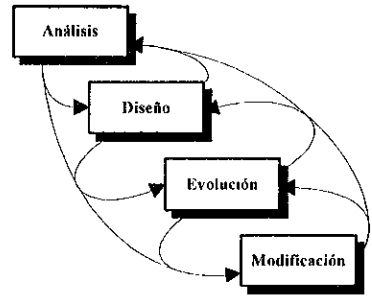


figura 2.17 Ciclo de vida del desarrollo de sistemas para la metodología orientada a objetos.

El Diseño Orientado a Objetos es un proceso que se obtiene mediante el seguimiento ordenado de los siguientes eventos:

1. ***Identificar las clases y objetos en un nivel de abstracción***
2. ***Identificar la semántica de estas clases y objetos***
3. ***Identificar las relaciones entre estas clases y objetos***
4. ***Implantar estas clases y objetos***

Este es un *proceso creciente*. la identificación de nuevas clases y objetos usualmente nos causa redefinir y mejorar la semántica de las relaciones entre clases y objetos. Es un *proceso interactivo*: la implantación de clases y objetos, lo cual regularmente nos lleva al descubrimiento o invención de nuevas clases y objetos, que son el resultado de la simplificación y generalización de nuestro diseño.

El inicio de nuestro proceso es descubriendo las clases y objetos que forman parte del vocabulario de nuestro dominio del problema, y nuestro objetivo final es identificar todas las entidades del dominio del problema, que tienen un papel en la aplicación, especificar la interacción entre entidades, la asociación de cada una de ellas, la asociación de cada entidad a la función que las realice, y el análisis de estas funciones a un nivel en que cada uno sea totalmente comprendido. El proceso termina cuando no encontramos mas indicadores o mecanismos del sistema, o cuando las clases y objetos ya han sido descubiertas y que son susceptibles de implantarse por componentes de ellas mismas de software ya existente.

2.3.4.3.1 Identificación de Clases y Objetos

Se consideran dos pasos para realizar esta tarea:

1. El descubrimiento de las abstracciones primarias en un problema definido, es decir, los principales elementos que intervienen en el proceso, son los candidatos a ser las clases y objetos.
2. La invención de los más importantes mecanismos que proveen el comportamiento requerido de los objetos que trabajan juntos para realizar una función. Para encontrar estos elementos y procesos a través del análisis usando técnicas para el modelado de objetos, el desarrollador debe aprender el vocabulario del dominio del problema, las reglas del juego, y los eventos que deben ocurrir para las clases y objetos de nuestro diseño, en el más alto nivel de abstracción.

Aquí se enfatiza la definición de los principales elementos que intervienen en el sistema, ya que son éstos los candidatos a clases y objetos, y que además son la base del diseño. Se inicia con la definición de los más importantes para nosotros, esperando que estos sean los mecanismos que se implanten en el diseño.

Al finalizar esta etapa se debe obtener una lista de nombres que puedan ser clases y objetos (usando nombres completos para mantener su significado), por el momento sólo le llamaremos lista de cosas, ya que posteriormente éstas se convertirán en clases y objetos y otros en simples atributos de objetos. Además de esta lista, se debe especificar el significado de estas abstracciones y mecanismos, para el llenado apropiado de las plantillas de las clases y objetos.

En la mayoría de los casos, este paso toma un corto tiempo, en consideración con el tiempo que pueden llevarse los otros tres pasos. Es recomendable se realice a esta lista una revisión, para definir un vocabulario común, que manejen los desarrolladores Es también apropiado para un diseñador dibujar algunos diagramas de objetos que muestren cómo varias clases y objetos trabajan juntos. Si la descomposición física del sistema es importante, los dibujos de diagramas de módulos también son de utilidad dibujar los diagramas de procesos necesarios.

2.3.4.3.2 Identificación de la semántica de Clases y Objetos

El segundo paso involucra al primero, ya que la estabilidad de éste depende del establecimiento de las clases y objetos identificados del paso anterior. Aquí el desarrollador es como un observador externo, es decir, su visión de cada clase debe ser desde la perspectiva de su interfaz, así como la identificación de las cosas que pueden ser una instancia de una clase y las cosas que cada objeto puede hacer con otro.

Este paso es mucho más difícil y requiere de más tiempo que el anterior. En esta fase se tienen fuertes debates durante las revisiones que se lleven por los diseñadores. Es en esta fase que se definen las clases y objetos, lo cual no representa una tarea fácil; decidir el protocolo para cada objeto resulta ser una tarea muy ardua. Por esta razón, el proceso del Diseño Orientado a Objetos se convierte en interactivo desde este punto. La preparación del protocolo para un objeto, requerirá decisiones que cambien el significado de otro objeto. En general, la existencia de las llaves primarias o principales elementos no cambian, sino solamente se transfiere su alcance.

Una técnica útil para guiarnos en esta actividad, es la escritura de cada objeto, donde se defina su ciclo de vida, desde su creación hasta su destrucción, incluyendo las características de su comportamiento.

De esta manera obtenemos un diseño orientado al objeto de crecimiento natural, así al documentar nuestras decisiones de diseño se respalda el significado de cada clase y objeto, nosotros generalmente refinamos la plantilla que dibujamos anteriormente, esto significa que debemos documentar toda la semántica estática y dinámica de cada llave primaria y mecanismo, de la mejor forma posible. También se debe dibujar nuevos diagramas que documenten el mecanismo que hemos inventado en este paso. Por último, se debe decidir cómo se integrarán las partes del diseño, es decir se debe crear el prototipo del sistema; para hacerlo se debe considerar nuestro actual diseño y evaluar las alternativas de desarrollo y los problemas que puedan representar un riesgo.

2.3.4.3.3 Identificando cuáles son clases y cuáles son objetos

Ubicar clases y objetos en el nivel adecuado de abstracción es difícil, algunas ocasiones nos encontramos en la dificultad de definir cuál es una clase y cuál no, lo que debemos hacer es identificar

la cohesión y libertad entre sus abstracciones, y con ello colocarla en la jerarquía que le corresponda, por los objetos que puedan crearse a partir de ella.

La identificación de las principales abstracciones involucra dos procesos: el descubrimiento y la invención. A través del descubrimiento, nosotros reconocemos las abstracciones usadas en el dominio del problema (regularmente el experto del problema lo menciona), y por medio de la invención se crean nuevas clases y objetos que son necesariamente parte del problema.

2.3.4.3.4 Identificando mecanismos

Usamos el término de mecanismo, para describir cualquier estructura de donde los objetos trabajan juntos para proveer algún comportamiento que satisfaga los requerimientos del problema.

Un mecanismo representa el diseño de decisión acerca de cómo una colección de objetos deben cooperar.

El mecanismo es como el alma del diseño, y representa la estrategia de decisiones que deben los diferentes objetos realizar cooperativamente para cumplir su función.

2.3.4.3.5 Identificación de relaciones entre Clases y Objetos

Este paso es de mayor número de actividades que el anterior. Aquí, estableceremos exactamente cómo las cosas deben interactuar en el sistema. Respecto a las llaves primarias o elementos principales, se deben afirmar el uso, herencia y otros tipos de relaciones entre clases. Para los objetos en nuestra implantación, deben establecerse la semántica y estática de cada mecanismo.

Hay dos actividades relacionadas aquí que nos causan refinar los productos de nuestro diseño. Primero, nosotros debemos descubrir patrones entre clases, lo que nos provoca la reorganización y simplificación de la estructura del sistema de clases y patrones entre la colección cooperativa de objetos, esto nos lleva a generalizar los mecanismos que comprenden nuestro diseño. En segundo lugar, debemos tomar decisiones de visión, es decir, cómo una clase ve a otra, cómo los objetos ven a otros, y de igual importancia, qué objetos deben ver o no a otros. Estas visiones nos ayudan a tomar decisiones de empaquetamiento inteligentes en el diseño de la arquitectura de módulos de nuestro sistema.

El descubrimiento de patrones y tomar decisiones de visión deben causar la refinación de protocolos de varias clases, definidas anteriormente, lo cual nos lleva a abstracciones comunes y mecanismos definidos en un solo lugar.

Una técnica que es útil para guiarnos en estas actividades es el uso de tarjetas CRC, definidas por Beck & Cuningham. Una tarjeta CRC (para clases, responsabilidades y colaboración) es usada para cada clase de objetos, en las cuales cada una contiene el nombre de la clase, sus responsabilidades (su comportamiento), y las clases que colaboran con ella. Un diseñador puede que organice el espacio de las tarjetas y defina su contenido, de acuerdo a la ejecución de escritos o del ciclo de vida de los objetos, a manera de introducir un paso previo.

En este punto del diseño aún no podemos tener una vista completa de las llaves primarias o elementos importantes y de los mecanismos existentes, ya que todavía es prematuro, porque hasta el momento sólo se ha considerado la representación de clases y objetos en nuestro sistema, y porque todavía no conocemos lo suficiente de cada abstracción y mecanismos que usaremos.

Este paso incluye el completar la mayoría de los modelos del diseño. Se refinan los diagramas de clases hechos anteriormente, tomando en cuenta los descubrimientos de patrones y la visibilidad que tiene un objeto del otro. También se completa los detalles de los diagramas de objetos que documentan la esencia del mecanismo de nuestra solución. En la práctica primero se dibujan los objetos que conocemos que cooperan en cierta forma, después, se seleccionan pares de objetos y se pregunta si hay relaciones entre ellos, si la respuesta es afirmativa, se realizan las siguientes dos preguntas: cómo estos objetos están relacionados y qué mensajes se envían el uno al otro. Lo cual no llevará a refinar el protocolo de las correspondientes clases, y este refinamiento debe causarnos el descubrimiento de nuevos patrones. Este es otro de los motivos por el que la metodología Orientada Objetos es un proceso interactivo.

Hasta este punto hemos creado la esencia del diagrama de módulos para nuestra solución, la cual contempla las decisiones de visibilidad tomadas. También hemos creado y refinado prototipos.

2.3.4.4 Relaciones entre Clases y Objetos

2.3.4.4.1 Tipos de Relaciones.

Los objetos contribuyen al comportamiento de un sistema con la colaboración de otros objetos. Las relaciones entre dos de éstos abarca la suposición de todo lo que uno puede hacer para el otro, incluyendo las operaciones que pueden ser realizadas entre sí y el comportamiento resultante. Existen dos tipos de jerarquías de relaciones entre objetos que son:

a) Uso de relaciones

b) Contenido de relaciones

a) **Uso de relaciones**

La figura 2.18 ilustra cómo se usan las relaciones entre varios objetos; la línea entre dos objetos representa la existencia del uso de una relación entre dos y significa que un mensaje pasa a través de este camino (relación). El mensaje que pasa entre dos objetos es usualmente unidireccional, sin embargo es posible la comunicación entre objetos bidireccionalmente.

Dada una colección de objetos que están relacionados, cada uno puede jugar Tres diferentes roles.

- *Objeto Actor* : que es un objeto que puede operar sobre otros objetos, pero que nunca es operado por otros objetos
- *Objeto Servidor* : que es aquel objeto que nunca opera sobre otros objetos, solamente es operado por otros objetos.
- *Objeto Agente* : que es aquel objeto que puede operar sobre otros objetos y que puede ser operado por otros objetos.

Cuando un objeto pasa un mensaje a otro deben estar sincronizados de alguna forma. Lo cual nos lleva a que un objeto puede ser utilizado como:

- *Objetos secuenciales* . que es un objeto pasivo cuya semántica está garantizada solamente en la presencia de una secuencia de control.

- *Objetos por bloques* : que es un objeto pasivo cuya semántica está garantizada en presencia de múltiples secuencias de control, y
- *Objetos concurrentes o activos* : que es un objeto activo cuya semántica está garantizada en la presencia de múltiples secuencias de control.

El que un objeto sea utilizado secuencialmente o por medio de una sola secuencia de control excluye las otras formas, y así con cada uno.

Entiéndase por objeto activo aquél que comprende su propia secuencia de control; y por objeto pasivo aquél que no comprende su propia secuencia de control.

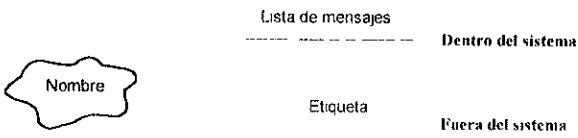


figura 2.18 Notación usada en la representación de objetos y sus relaciones.

b) **Contenido de relaciones**

Es la encapsulación de una relación en un objeto. El contener o encapsular una relación reduce el número de objetos que deben ser visibles a un nivel de objetos.

2.3.4.4.2 Relaciones entre Clases

Un objeto es una entidad concreta que existe en un tiempo y espacio, una clase representa solamente una abstracción, la esencia de un objeto.

Una clase es un grupo definido por atributos comunes, es una definición de objetos que comparten una estructura común y un común comportamiento.

La interfaz de una clase proporciona la vista exterior de si misma y por tanto enfatiza la abstracción mientras esconde en su estructura los secretos de su comportamiento. Esta interfaz primeramente consiste de las declaraciones de todas las operaciones aplicables a instancias de esta clase, esto podría incluir la declaración de otras clases, constantes y variables.

Una relación entre clases indican una forma de compartir y un tipo conexión semántica. Hay tres tipos de relaciones entre clases:

- *La generalización*, denotando "un tipo" de relación
- *La agregación*, la cual denota "una parte" de relación.
- *La asociación*, la cual denota alguna conexión semántica entre otro tipo de clase no relacionada.

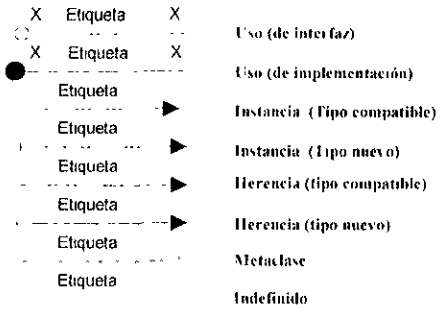


figura 2.19 Notación usada para representar relaciones entre clases.

Estas relaciones son soportadas por los lenguajes Orientados a Objetos o los basados en objetos; en la combinación de las relaciones entre las clases denominadas:

a) Relación de herencia, es cuando una clase comparte la estructura o comportamiento definidos en una herencia simple o múltiple (uno o varios objetos respectivamente). La herencia define un tipo de jerarquía, donde cada clase hereda de otra u otras clases (la de nivel mas alto se le denomina superclase y las clases de nivel inferior son llamadas subclases). Una subclase típicamente redefine la estructura definida y el comportamiento de su superclase. La herencia de relaciones es la base del polimorfismo, ya que por las diferentes relaciones con varias clases un objeto tiene la posibilidad de responder a una operación en diferentes formas. La herencia múltiple se usa cuando la relación simple no satisface las suficientes relaciones entre clases Su uso es parte de la existencia de relaciones de un objeto con más de una clase, debido a su naturaleza y las relaciones requeridas. Cuando una clase es construida primeramente por herencia múltiple y no se le aumenta su propia estructura o comportamiento se está hablando de una clase agregada.

b) Relaciones de uso. Hay dos tipos de uso de relaciones entre clases:

- interfaz de una clase, (uso de otra clase) en donde la clase usada debe ser visible a cualquiera de sus clientes.
- Implantación de una clase, (uso de otra clase) en donde la clase usada es escondida como parte del secreto de usar clases.

A través de las líneas que representa la relación se puede expresar también la cardinalidad de la relación.

c) Relaciones de instancia En la definición de una clase debe tener un estado preciso de los objetos que pueden pertenecer o ser definidos como instancias de una clase. Una definición es un ejemplo de un deposito de una clase cuyas instancias son colecciones de otros objetos. Un deposito de una clase denotan una colección homogénea, es decir, todos los objetos son de la misma clase, o que representan un conjunto heterogéneo, en donde cada objeto debe ser de diferentes clases, pero que comparten algunas superclases comunes.

d) Relaciones de metaclases, una metaclase es una clase, la cual sus instancias son la misma clase, es decir la clase de una clase es la metaclase.

2.3.4.5 Implantación de Clases y Objetos

Este paso no es necesariamente el último, se incluyen dos principales actividades: tomar decisiones de diseño concernientes a la representación de las clases que hemos inventado, ubicar las clases y los objetos en los módulos, y programas en procesos. Es en este punto cuando a partir de la vista de adentro de cada clase y módulo, se decide como su comportamiento se debe implantar.

Al menos las principales abstracciones y mecanismos son triviales, este es el punto donde regularmente se regresa al primero, para aplicar nuevamente este proceso, pero ahora con las clases y módulos existentes. En esta forma, repetimos el proceso de diseño, sin embargo en esta ocasión nos enfocamos en un nivel de abstracción menor. Lo cual no quiere decir que la metodología Orientada a Objetos es un proceso TOP-DOWN exactamente, porque diseñamos primero el nivel mas alto de las principales abstracciones y mecanismos, lo cual involucra estas clases y objetos que son directamente parte del vocabulario de dominio del problema, sin embargo, en cualquier nivel de diseño, nosotros nos saltamos el nivel mas bajo de abstracción de las principales abstracciones y de los mecanismos, porque ellos nos sirven más como clases y objetos primarios sobre los cuales se componen todos los altos niveles de clases y objetos Esto nuevamente hace de la metodología Orientada a Objetos un proceso Round-trip Gestalt Design.

En este paso se establece la interfaz concreta para cada clase y objeto importantes para nosotros en un nivel de abstracción. Nuevamente se realiza una refinación de la estructura de clases del sistema, y en particular el completar la implementación para cada parte de las plantillas de clases, de los diagramas de los módulos, y si el sistema lo demanda de los diagramas de procesos.

La implantación de una clase en su vista interior involucra el secreto del comportamiento de la clase. La implantación de una clase de divide en tres partes:

1. *Público* la declaración que forma parte de la interfaz de una clase y es visible a todos los clientes que son visibles a la clase
2. *Protegido* : la declaración que forma parte de la interfaz de una clase, pero no es visible a cualquier otro, excepto de sus subclases.
3. *Privada* la declaración que forma parte de la interfaz de una clase, pero no es visible por cualquier otra clase.

2.3.4.6 Notación

La notación sugerida por Grady Booch, contempla la estructura de las clases, los mecanismos de herencia, el comportamiento individual de los objetos y el comportamiento dinámico de un sistema.

La notación es la documentación del diseño de un sistema, esta notación debe usarse de acuerdo a las necesidades de cada proyecto. Tenemos cuatro tipos de diagramas:

1. Diagramas de clases
2. Diagramas de objetos
3. Diagramas de módulos

4. Diagramas de Procesos

Los primeros dos son parte de la vista lógica de un sistema, porque nos describen la existencia y el significado de las principales abstracciones y la forma del diseño. Los últimos dos diagramas son parte de la estructura física de un sistema, porque describen concretamente los componentes del software y hardware de una implantación.

La semántica Dinámica y Estática de un sistema se expresan a través de dos tipos de diagramas:

1. Diagramas de estados de transición
2. Diagramas de sincronización (Timing Diagrams)

Para cada clase deberá tener un diagrama de transición asociado a sus indicaciones de cómo el orden en eventos externos pueden afectar el estado de cada instancia de la clase. Se deberá usar diagramas de sincronización en conjunción con cada diagrama de objetos, para mostrar el orden de los mensajes que son enviados y evaluados.

2.3.4.6.1 Diagramas de clases

Muestran la existencia de clases y sus relaciones en el diseño lógico de un sistema. Un diagrama de clases representa todo o parte de la estructura de clases del sistema. Hay tres elementos de la estructura de clases: clases, las relaciones de las clases y las utilerías de clases.

Clases, su icono es un globo amorfo, llamado nube y esta pintado con línea punteada, la cual indica que los clientes que hacen uso de él generalmente operan solamente sobre las instancias de la clase y no de la clase en sí. La clase debe tener nombre, el cual es escrito dentro de la nube Si el nombre es particularmente largo, se puede usar mnemónicos, o en su defecto agrandar la nube. Cada clase debe ser única.

figura 2.20 Notación para el diagrama de clases.

Relaciones de las clases, los iconos para los diagramas de clase están especificados en la figura 2.20, cada relación debe tener etiqueta, la cual documenta el nombre de la relación entre las clases. El uso de una relación con doble línea con un circulo en uno de los extremos identifica que la clase usa recursos de otra, se dice que la relación es por interfaz cuando la clase utilizada es visible a todo cliente, y la relación es por implementación cuando la clase esta oculta como parte del secreto del uso de una clase

El símbolo • es usado para indicar la dirección en que la etiqueta debe ser leída. De igual forma los iconos para representar la cardinalidad en las relaciones entre clases se muestran en el mismo cuadro. También en casos complicados y donde se requiera, para completar la cardinalidad se pueden usar operadores relacionales : =, <>, >,<, <= y >=.

Utilerías de Clase, representa cualquier subprograma libre o una colección de programas libres, es decir que no pertenecen a una clase, y que a su vez pueden servir a aquellas que lo requieran. Su icono es también una nube, pero además tiene una sombra, y también necesitan tener un nombre.

Plantillas de clases, es la documentación de cada clase, subclase, sus campos, y sus operaciones.

Nombre:	Identificador	
Documentación:	Descripción de lo que se documenta	
Visibilidad:	Exportada, Privada, ImportadaCarnalidad	0 / 1 / n
Jerarquía:		
Superclases	Lista de clases	
Metaclases:	Nombre de la clase	
Parámetros	Lista de parámetros	
Interfaz / Implementación		
(Publico o Privada):		
Usos	Lista de clases	
Campos:	Lista de declaraciones de campos'	
Operaciones:	Lista de operaciones	
Estado·	Diagrama de estados de transición	
Concurrencia:	Secuencial, por bloques, activo	
Espacio de complejidad:	Texto	
Persistencia·	Estatico o dinamico	

figura 2.21 Plantilla de clases.

La visibilidad, indica si la clase es exportada, privada o importada en relación a su categoría de clases; la carnalidad, indica cuántas instancias permitirá la clase; la jerarquía dependerá del lenguaje que se utilice para su desarrollo y del diseño del mismo, lo mismo es con los parámetros ya que no todos los lenguajes lo permiten; la implementación o interfaz se refiere a la forma en que los diferentes atributos van a ser definidos, lo cual también dependerá del lenguaje que se utilice ya que no todos tienen la versatilidad del manejo de diferentes tipos; deben indicarse a aquellos campos que son propios de la clase con sus definiciones; las operaciones son una descripción general de las que son de la clase, usos se refiere a las clases de las cuáles hace uso; el espacio de complejidad, se refiere al tamaño por unidad de la clase; por ultimo la persistencia se refiere a que si la definición de la clase es estática o dinámica.

las clases asociadas, es decir, si se muestra que un objeto envía a otro un mensaje, entonces la operación a realizar al recibir el mensaje debe estar definido por su clase.

Objetos.- El icono para representar un objeto es semejante al de las clases, el nombre de los objetos debe ser único. Las propiedades de un objeto deben aparecer en el lado izquierdo inferior de su icono. Las mas importantes propiedades que incluyen son la concurrencia y la persistencia de cada objeto, en otras palabras si el objeto es inactivo o activo durante un proceso.

Relaciones entre objetos.- Representan el envío de mensajes de un objeto a otro. Cuando usamos una línea, se trata de un mensaje bidireccional; cuando usamos líneas sólidas es cuando el modelo podemos implantarlo en el software.

Objeto	
Nombre:	Identificador
Documentación:	Descripción
Clase:	Nombre de la clase a la que pertenece
Persistencia:	estatico / dinámico

figura 2.24 Plantilla de objeto.

Mensajes	
Operación:	Nombre de la operacion
Documentación:	Descripción
Frecuencia:	Periodo o frecuencia de ejecucion
Sincronización:	Simple síncrona, asíncrona

figura 2.25 Plantilla de mensajes.

Diagramas de Sincronización.

Se usan para representar la dinámica del envío de mensajes de un diagrama de objetos. Un diagrama sincronizado es una gráfica que establece el tiempo en el eje de horizontal, y los objetos a través del eje vertical. Como se avance sobre el eje horizontal, el tiempo, una operación debe ser invocada. Al marcar un proceso con un asterisco, indica que se crea el objeto, y cuando se marca el proceso con un signo de admiración (!) indica la destrucción del objeto.

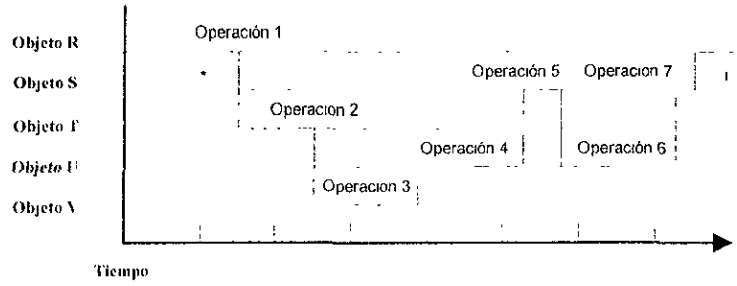


figura 2.26 Diagramas de sincronización.

2.3.4.6.4 Diagramas de módulos (Boochgrams/ Gradygrams)

Muestran la colocación de las clases y objetos en los módulos en el diseño físico de un sistema; un diagrama de módulos representa todo aparte de la arquitectura del sistema. Hay dos elementos en la arquitectura de módulos: los módulos y su visibilidad.

Existen varios iconos para representar los diferentes tipos de módulos que pueden existir.

Los tipos de módulos corresponden a tipos de módulos que soporten los diferentes lenguajes orientados o basados en objetos.

Módulos.- Es la clasificación de los programas que componen un sistema. El nombre del módulo es necesario, y se debe colocar en la parte superior del rectángulo, cada nombre debe ser único, cuando el nombre del módulo esta dentro de un rectángulo, indica que ese módulo es exportado por otro subsistema; en caso contrario, es decir, que solo sea un nombre representa un módulo privado, que solo esta incluido en ese sistema; y cuando el nombre esta subrayado, representa un módulo que es importado de otro sistema. La única relación entre módulos es la dependencia de compilación, la cual es representada por la línea representada en la figura anterior.

Subsistemas.-en los sistemas grandes es necesario contar con un nivel de abstracción mayor, este nivel es representado mediante los subsistemas, a través de estos diagramas un desarrollador puede comprender de forma general la arquitectura de un sistema. Cada subsistema denota otro diagrama de módulos, con su diagrama de categorías de clases, etc. El icono de un subsistema se muestra en la siguiente figura.

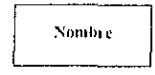


figura 2.27 Icono para subsistema.

Nombre:	Nombre del Subsistema
Documentación:	*Descripción*
Declaraciones:	Lista de declaraciones del Subsistema

figura 2.28 Plantilla de subsistemas.

2.3.4.6.5 Diagramas de Procesos

Un sistema contiene múltiples programas ejecutándose sobre una colección de computadoras distribuidas. Los procesos de diagramas visualizan y nos ayudan a ubicar procesos y procesadores.

Un diagrama de procesos representa todo o parte de la arquitectura de procesos de un sistema. Hay tres elementos importantes para los diagramas de procesos: procesadores, dispositivos y conexiones.

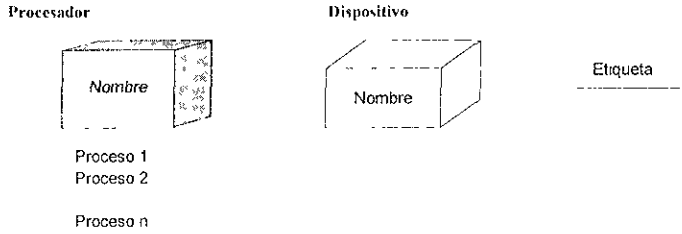


figura 2.29 Símbolos usados en los diagramas de procesos.

Procesadores y dispositivos.- Es una pieza de hardware capaz de ejecutar programas, regularmente en el icono de procesadores incluye la serie de procesos que va a ejecutar.

Conexiones.- Representa un acoplamiento de hardware, como lo es un cable Ethernet, o fibra óptica. Esta es representada por la línea mostrada en la figura anterior, de acuerdo a la dirección requerida se puede poner una flecha.

Procesador	
Nombre:	Nombre del Procesador
Documentación:	Descripcion
Características:	Características generales
Procesos:	Lista de procesos que realizara
Horario:	Ciclos de ejecución de los procesos

figura 2.30 Plantilla de procesadores.

Procesos	
Nombre:	Nombre del Proceso
Documentacion.	Descripción
Prioridad:	Grado de prioridad del proceso (numerico)

figura 2.31 Plantilla de procesos.

Dispositivos y Conexiones	
Nombre:	*Nombre del proceso*
Documentación:	Descripcion
Características:	Descripcion

figura 2.32 Plantilla para dispositivos y conexiones.

3. Desarrollo del sistema

Para el desarrollo del sistema es necesario la aplicación de métodos. Si bien los métodos basados en los mismos principios y conceptos conforman una *metodología*, puede ocurrir que se aplique una metodología diferente en alguna de las etapas del ciclo de vida. Tal es el caso del sistema a desarrollar, en el cual se utilizará una metodología orientada a objetos en el análisis y diseño, pero por la naturaleza de las herramientas de programación algunos módulos se codificará bajo los conceptos estructurados.

El desarrollo de este sistema tiene un *ciclo de vida* constituido por las cinco fases siguientes:

Análisis

Diseño

Programación

Implantación

Mantenimiento

Cada fase es un punto de transformación que se alimenta de la salida anterior, dando lugar a la siguiente secuencia de transformaciones:

1) Partiendo de un sistema real se elabora una representación y se identifica el problema (análisis).

2) La representación anterior se modifica para elaborar un modelo con soluciones (diseño).

3) El modelo de diseño se transforma en aplicaciones (programación).

4) Las aplicaciones se prueban, corrigen e instalan en el sistema real (implantación).

5) El sistema real no es estático consiguientemente nuevas necesidades surgen y las aplicaciones deben adecuarse a los cambios (mantenimiento) así se regresa al inciso 1.

Esta secuencia se muestra en la figura 3.1, donde las líneas continuas indican el flujo normal en un desarrollo puramente estructurado, pero la metodología orientada a objetos reconoce que en el ciclo de vida hay retroalimentación entre las etapas. Las retroalimentaciones entre las etapas se muestran en líneas punteadas en la figura 3.1.

3.1 Análisis

El análisis por definición "es el estudio que se realiza para separar las distintas partes de un todo"[1] Este concepto se encuentra fuertemente arraigado en la primera fase del ciclo de vida de un sistema independientemente de la metodología que se trate. Aunque cada metodología realiza la separación de los componentes de manera diferente.

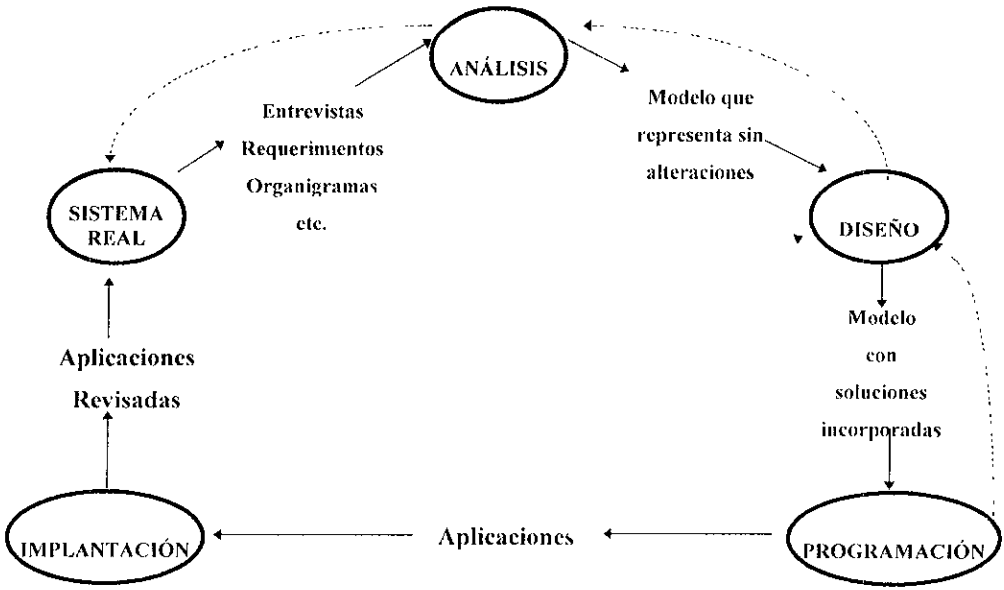


figura 3.1 Ciclo de vida del desarrollo de sistemas.

En las metodologías estructuradas la separación de los componentes es de tipo funcional mientras que en las metodologías orientadas a objetos se centran en una separación por objetos y clases.

La separación de los elementos de una entidad material o abstracta tiene como propósito hacer una representación del sistema en estudio con el fin de entenderlo de manera que la fase posterior de diseño se construyan soluciones con base en una representación realista.

Los modelos que se elaboren en el análisis no deben estar contaminados con elementos de diseño además debe tenerse en cuenta que el análisis se va refinando poco a poco y el modelo obtenido primeramente puede no ser el más apegado a la realidad

Las principales actividades ha realizar en esta fase son:

1) Recopilar información general del sistema.

2) Determinar problemas y sus causas.

3) Delimitar el problema.

4) Determinar requerimientos

5) Identificar los principales componentes del sistema.

6) Obtener la factibilidad de solución con base en sistemas de cómputo.

3.1.1 Información general del sistema

Para el establecimiento de un modelo que describa el funcionamiento del sistema bajo estudio se deben establecer la estructura estática, posteriormente se podrá elaborar un modelo dinámico. Por lo cual se comenzará con la descripción de las entidades que participan en el sistema.

El sistema real que será objeto de análisis es el mecanismo de emisión y conciliación de apoyos directos de Apoyos y Servicios a la Comercialización Agropecuaria (ASERCA). Institución que se creó el 16 de abril de 1991, como órgano administrativo desconcentrado de la Secretaría de Agricultura y Recursos Hidráulicos, el que tendrá por objeto fomentar la producción agrícola mediante apoyos a la comercialización de productos agropecuarios. ASERCA no interviene directamente en el proceso de compra-venta, sino que subsidia a quienes comercializan de origen los productos agrícolas para hacer posible una venta justa para los productores nacionales y al comprador se le compensa la diferencia entre el precio nacional e internacional. Esta mecánica ha sido utilizada por ASERCA en los casos de trigo, sorgo, arroz, cártamo, algodón y soya.

La dependencia maneja diferentes programas de apoyo a los productores de los que destaca el Programa de Apoyos Directos al Campo, cuya vigencia será de 15 años a partir del ciclo agrícola OI/94 con pagos constantes durante los primeros 10 años y decrecientes en los últimos 5 años. PROCAMPO, que tiene por objeto transferir recursos en apoyo a la economía de los productores rurales, que reúnan los requisitos y cumplan con la normatividad operativa del mismo. El monto del apoyo está en proporción a la superficie cultivada y se entrega una vez que el cultivo ha brotado. Este apoyo se entrega sin otro compromiso por parte del productor más que el de cosechar, consiguientemente el productor puede utilizar el producto cultivado de acuerdo a sus necesidades.

ASERCA es la institución encargada de establecer la normatividad y vigilar su cumplimiento De igual forma es responsable de la custodia de la base datos de los productores beneficiados, de la determinación de la elegibilidad de todas y cada una de las solicitudes y de la autorización de la entrega del apoyo. Que a grandes rasgos sigue el proceso siguiente·

1. Cada ciclo agrícola ASERCA Central autoriza a cada Regional la emisión de formatos de reinscripción y listas de productores y predios correspondientes.
2. ASERCA Regional emite y distribuye formatos y listados a las Delegaciones Estatales de la SAGAR que le correspondan.
3. Las Delegaciones Estatales distribuyen los formatos y listados a los Distritos de Desarrollo Rural (DDR) correspondientes.
4. Cada DDR distribuye los formatos y listados a los Centros de Apoyo al Desarrollo Rural (CADER) correspondientes.
5. En el CADER se publican los listados y se avisa a los productores para reinscribir la solicitud de apoyo.
6. El productor con ayuda de un técnico llena la solicitud de apoyo informando cuánta superficie va a cultivar y de que cultivo.
7. El CADER concentra las solicitudes que reinscriben a los productores en el programa de apoyo y las entregan al DDR correspondiente.

8. El DDR concentra las solicitudes de los CADER's y se acumulan totales de las superficies cultivadas a apoyar revisando que el valor sea consistente con los datos cartográficos disponibles. Las solicitudes revisadas se entregan al Subcomité de Control y Vigilancia(SCCV).
9. El SCCV es un grupo de productores quienes confirman que la información contenida en las solicitudes es correcta, posteriormente las regresa al DDR.
10. El DDR las entrega a la Delegación Estatal correspondiente.
11. La Delegación Estatal concentra las solicitudes y se verifica que el total de superficie cultivada sea congruente con los datos cartográficos a nivel estado. Las solicitudes se envían al Centro Regional correspondiente.
12. La información de cada solicitud es validada y procesada en el Centro Regional donde se emite un documento de pago que es proporcional a la superficie cultivada.
13. Los documentos de pago son distribuidos por ASERCA Regional a través de las Delegaciones Estatales.
14. Las Delegaciones Estatales distribuyen los documentos a su vez a los DDR's.
15. Los DDR's entregan los documentos correspondientes a cada CADER.
16. En el CADER el productor recoge su documento de pago.
17. El documento de pago circula en el sistema comercial hasta que entra al sistema bancario.
18. El sistema bancario compensa el cobro de los pagos en la Tesorería de la Federación (TESOFE).
19. Tesorería de la Federación proporciona la información de los cheques cobrados a ASERCA Central.
20. ASERCA Central confirma que el monto cobrado en el sistema bancario corresponda con el emitido originalmente

El proceso de entrega de apoyos en PROCAMPO, se distribuye básicamente en ocho instancias de responsabilidad[VI]:

- ASERCA Central
- ASERCA Regional
- Delegación Estatal
- DDR
- SCCV
- CADER
- Productor
- TESOFE

En términos generales las actividades que desarrolla cada una de estas instancias se sintetiza a continuación:

ASERCA Central

- Actualización de la normatividad y procedimientos operativos.
- Autoriza a ASERCA Regional de la impresión de los formatos del Programa de Apoyos.
- Autoriza a ASERCA Regional de la impresión de documentos de pago o reexpedición.
- Conciliación de los pagos y cancelaciones efectuadas con base en la información proporcionada por TESOFE.
- Dictaminación jurídica de las modificaciones a los conceptos del directorio de PROCAMPO de ASERCA Regional.
- Evaluación de los aspectos normativos y operativos del Programa de Apoyos Directos.

ASERCA Regional

- Emisión de los formatos de reinscripción, dictaminación y listados de control y seguimiento.
- Coordinación del proceso de verificación de los predios con cada una de las Delegaciones Estatales.
- Captura, validación y proceso de la información de los formatos de reinscripción y dictaminación.
- Impresión de los documentos de pago.
- Revisión de los documentos de pago, y en su caso, actualización y reimpresión.

Delegación Estatal

- Recepción y distribución a los Distritos de Desarrollo Rural de los formatos de reinscripción y dictaminación.
- Proporcionar listados de control y seguimiento del programa de verificación de predios a los Distritos de Desarrollo Rural.
- Supervisión de la operación de PROCAMPO.
- Concentración de la información y remisión a ASERCA Regional.
- Tramitación de cancelaciones, bajas y modificaciones en el Directorio de PROCAMPO.
- Revisión y cancelación de cheques incorrectos y en su caso tramitación de reexpedición.

Distrito de Desarrollo Rural (DDR)

- Recepción y distribución a los Centros de Apoyo al Desarrollo Rural de los formatos de reinscripción y dictaminación.
- Difundir las normas de ASERCA a los integrantes del Subcomité de Control y Vigilancia.
- Supervisar la operación de PROCAMPO en las ventanillas de atención.
- Formalizar la dictaminación de los predios reinscritos en forma conjunta con el Subcomité de Control y Vigilancia.
- Concentración de la información y remisión a la Delegación Estatal.
- Concentración de documentos de pago incorrectos y de modificaciones a los conceptos del Directorio del PROCAMPO y remisión para el trámite a la Delegación Estatal.

Subcomité de Control y Vigilancia (SCCV)

- Recepción y revisión de los formatos de reinscripción.
- Verificación física y documental de las solicitudes de reinscripción.
- Proporciona recomendación de las solicitudes de reinscripción al Distrito de Desarrollo Rural.
- Remisión de la documentación al Distrito de Desarrollo Rural.

Centro de Apoyo al Desarrollo Rural (CADER)

- Recepción de los formatos de reinscripción y dictaminación, listados de control y seguimiento, tamaño de la muestra y casos a verificar, además de los documentos de pago.
- Atención a los productores en las ventanillas.
- Revisión de las normas de operación sobre las solicitudes y la documentación.
- Verificación física y documental de las solicitudes con información dudosa y de las solicitudes seleccionadas por ASERCA Regional como muestra a verificar
- Concentración de la información y remisión al Distrito de Desarrollo Rural.
- Revisión de documentos de pago y entrega al productor.
- Remisión al Distrito de Desarrollo Rural de los documentos de pago incorrectos
- Recepción y tramitación de modificaciones a conceptos del Directorio de PROCAMPO.

Productor

- Realización del trámite de reinscripción.
- Tramitar en su caso la actualización de los conceptos del Directorio PROCAMPO.
- Revisión del documento de pago respecto a la dictaminación.
- Recepción del apoyo correspondiente.

Tesorería de la Federación (TESOFE)

- Proporciona a ASERCA Central formas valoradas en las que se emitirá el pago en ASERCA Regional.
- Compensa el cobro de los documentos de pago realizados en la Sistema Bancario.
- Proporciona a ASERCA Central la información de los documentos cobrados
- Concentra los documentos de pago cancelados y los reporta a ASERCA Central.

Por el ámbito en que operan las instancias de responsabilidad se consideran de ámbito federal (SAGAR, ASERCA Central, ASERCA Regional, TESOFE) y de ámbito estatal (Delegaciones estatales, DDR y CADER).

ASERCA cuenta con 16 Centros Regionales distribuidos en la República Mexicana y con una instancia Central. Además ASERCA Regional se auxilia de la estructura de la SAGAR que cuenta con 33 Delegaciones Estatales, 193 Distritos de Desarrollo Rural (DDR) y 712 Centros de Apoyo (CADER).

Si bien el proceso descrito previamente se aplica actualmente, en los primeros años de operación el programa de PROCAMPO los centros regionales no contaban con infraestructura de cómputo por lo que el procesamiento de la información (captura validación y ratificación de solicitudes, así como la emisión de los apoyos) se realizaba de manera central.

Como parte de la segunda fase de operación del programa, se instalan los centros regionales, los cuales están distribuidos en toda la república para agilizar el registro de inscripción así como la entrega de apoyos a los productores.

En 1994 en ASERCA central se crea una copia de la información de todos los centros regionales. Esta copia era necesaria ya que las operaciones de captura y validación se realizaban en los regionales pero las operaciones de cálculo, la emisión y la cancelación de los apoyos sigue siendo a nivel central.

Después a fines de 1994 la emisión de apoyos es calculada y emitida desde los centros regionales, sin embargo la cancelación y conciliación de cheques se realiza en ASERCA Central. Ya que la información bancaria proporcionada por TESOFE se da a nivel central y ASERCA central la recibe y concilia cancelaciones y pagos efectuados, además distribuye esa información al centro regional que emitió el pago correspondiente.

3.1.2 Problemas y sus causas

Los principales problemas encontrados en la operación del Programa de Apoyos Directos al Campo se listan en la tabla 1. No todos estos problemas pueden tener soluciones basadas en sistemas de cómputo algunos de ellos podrían solucionarse cambiando procedimientos de operación.

3.1.3 Delimitación del problema

De la lista presentada en el punto 0 el problema 1 la validación de la información proporcionada en las solicitudes requiere de la realización de inspecciones en campo, pero debido a los costos que implica la inspección de todos los predios, actualmente se realizan muestreos. En cuanto a los problemas 2 y 3 es necesario el establecimiento de mecanismos para determinar la identificación única del predio y del productor. Así estos problemas implican modificaciones en procedimientos que están fuera del alcance de sólo el desarrollo de sistemas de cómputo. Por otro lado, el problema 4 es referente a la sincronización de las operaciones realizadas en bases de datos distribuidas geográficamente, proceso que puede ser sistematizado.

En particular el problema sincronización de las bases de datos ha absorbido cientos de horas-hombre de trabajo debido a que es continuo el desfasamiento de la información entre la base de datos copia y la base de datos original, aunado a esto las aplicaciones realizadas hasta el momento se enfocan a información muy específica por lo cual no pueden ser reutilizadas para otros casos. Además la sincronización de la información de los cheques emitidos y cancelados es importante para ASERCA, ya que deben tomarse decisiones de erogación de dinero con base en la información copia de los Centros Regionales. Por estas razones se elige como problema a resolver la sincronización de la información.

Problema	Causa
1. Emisiones de apoyos que no corresponden con lo realmente cultivado, ocasionando cancelaciones de los apoyos, reexpediciones y devoluciones	• Errores en la captura de las solicitudes. • La falta de una validación exhaustiva en campo de los predios y productores.
2. Duplicidad de predios repercutiendo en desperdicio de espacio en disco. Con base en los datos de predios duplicados detectados la proporción es de un 0.005% Por otro lado, debido a un registro de predio por cada ciclo (primavera-verano u otoño-invierno evita que el mismo predio físico se apoye doble en un ciclo.	• Un predio se registró en dos ciclos presentado un documento legal diferente de posesión en cada ciclo. • En zonas donde la propiedad de la tierra no esta regularizada no existe un documento oficial que acredite la propiedad.
3. Duplicidad en el padrón de productores resultando en desperdicio de espacio en disco. Se ha encontrado que este problema lo tiene un 0.02 %	• No se cuenta con un mecanismo que identifique plenamente a cada persona a nivel nacional aún. • El productor puede registrar un alta con diferentes documentos de identificación y aún tan solo con el acta de nacimiento y dos testigos.
4. Versiones diferentes de la misma información en Regionales y ASERCA Central.	• En el Regional los datos se corrigen para mantener el padrón al día, pero estas correcciones suelen omitirse en la copia en ASERCA Central. Sumado a ello en la base de datos de ASERCA Central una actividad rutinaria fue la corrección de nombres y/o superficies cultivadas para reexpedir un cheque con el nombre correcto. Sin embargo estos cambios no se efectuaron en la información de ASERCA Regional.
5. *Lenta transmisión de la información entre* los Centros Regionales y ASERCA Central.	• *Sólo dos regionales están enlazados con fibra* óptica a ASERCA Central el resto utilizan líneas telefónicas. • De ASERCA Central a un Centro Regional se abren en promedio de 3 a 5 enlaces que consultan información En muchas ocasiones información que había sido consultada por un usuario se vuelve a consultar por otro duplicando con ello la carga en las líneas de comunicación.

tabla 1 Lista de los principales problemas y sus causas (continua en la página siguiente).

3.1.4 Determinación de requerimientos

El sistema a realizar tendrá las características siguientes:

- Mantendrá copias de cualquier información de la base de datos de ASERCA Regional en ASERCA Central.
- Permitirá la operación con los datos tanto en ASERCA Central como en ASERCA Regional.
- Deberá contar con mecanismos alternos a las líneas de la red con el objeto de mantener los datos sincronizados a pesar de que las líneas no funcionen.
- Tolerante a caídas temporales de las líneas de comunicación, deberá continuar en el punto donde ocurrió la interrupción cuando las líneas se restablecen.
- El uso de la línea de comunicación deberá ser óptimo con el fin de no degradar por tiempos prolongados las comunicaciones.

3.1.5 Identificación de los componentes del sistema

Para construir un modelo general del sistema es necesario determinar las clases relevantes y sus relaciones.

Para determinar las clases se identifican los sustantivos de la descripción que potencialmente podrían considerarse como clases, estos sustantivos son:

ASERCA	SAGAR	compra-venta
Institución	órgano administrativo	comercialización
ciclo	PROCAMPO	apoyo
documentos de pago	productor	superficie cultivada
base de datos	solicitud	autorización
economía	productos agrícolas	elegibilidad
normatividad	Instancia de responsabilidad	ASERCA Central
ASERCA Regional	Delegación Estatal	DDR
SCCV	CADER	TESOFE
formatos	dictaminación	Operación
Supervisión	Cancelaciones	Bajas
Normas	Ventanillas	Formas Valoradas
Ámbito federal	Ámbito estatal	Sistema Bancario
Sistema Comercial		

La selección de las clases debe estar sujeta a las consideraciones siguientes:

- Las clases deben derivarse de los conceptos que conforman el dominio del problema y no del dominio del diseño o del dominio de la aplicación, de ahí que el sustantivo base de datos se descarte (aún cuando aparece en el texto fuente) para la elaboración del modelo en la etapa de análisis.

- Dado que el modelo a construir inicialmente se enfoca a la estructura organizacional se excluyen las clases situadas en niveles de abstracción diferente, tales como ciclo, apoyo, formatos, solicitud, documento de pago, ventanillas, elegibilidad y autorización.
- Se excluyen las clases redundantes tal como institución lo es con respecto a órgano.
- Los sustantivos de operaciones que son de carácter dinámico como compra-venta, cancelación, supervisión y dictaminación también son descartados.

Con base en los criterios anteriores la lista de clases resultante es la presentada a continuación:

- Ámbito Federal
- Ámbito Estatal
- ASERCA Central
- ASERCA Regional
- Delegación Estatal
- Distrito de Desarrollo Rural
- Centro de Apoyo al Desarrollo Rural
- Sistema Comercial
- Sistema Bancario
- Tesorería de la Federación
- Productor

Una vez obtenida la lista de clases se identifican las relaciones existentes entre éstas para ello se determinan en la descripción del sistema las referencias de cada clase a otras. Las relaciones comúnmente corresponden a verbos que indican un estado o a enunciados verbales. De los cuales se incluyen verbos de propiedad (compuesta por, cuenta, posee, ...), de ubicación (contenido a, adyacente a, ...), de autoridad (norma, regula, gobierna, ...), de comunicación (solicita a, habla con...), de cooperación (colabora, trabaja con ...). Además las relaciones deben desarrollarse si la frase contiene relaciones implícitas entre más de dos clases. De acuerdo con este criterio se determinaron las relaciones siguientes:

- ASERCA Central autoriza a ASERCA Regional impresión de formatos y documentos de pago.
- ASERCA Regional coordina proceso de verificación con las Delegaciones Estatales.
- Delegación Estatal proporciona listados y formatos a los Distritos de Desarrollo Rural correspondientes.
- Distrito de Desarrollo Rural proporciona formatos y listados a sus Centros de Apoyo de Desarrollo Rural.
- Distrito de Desarrollo Rural difunde normas a los Subcomités de Control y Vigilancia.
- Subcomité de Control y Vigilancia remite documentación a Distrito de Desarrollo Rural.
- Centro de Apoyo al Desarrollo Rural brinda atención a los productores.

- Centro de Apoyo al Desarrollo Rural remite información al Distrito de Desarrollo Rural.
- Productor tramita apoyo en el Centro de Apoyo al Desarrollo Rural.
- Productor cobra el cheque en el sistema comercial.
- El Sistema Comercial hace efectivo el cheque en el Sistema Bancario.
- El Sistema Bancario compensa el cobro del cheque en la Tesorería de la Federación.
- TESOFE proporciona información de los cheques cobrados a ASERCA Central.
- SAGAR, ASERCA Central, ASERCA Regional y TESOFE operan en un ámbito federal.
- Delegación Estatal SAGAR, Distrito de Desarrollo Rural y CADER operan en un ámbito estatal.
- ASERCA Central cuenta con 16 Centros Regionales.
- ASERCA colabora con la SAGAR.
- La SAGAR cuenta con 33 Delegaciones Estatales, 193 Distritos de Desarrollo Rural (DDR) y 712 Centros de Apoyo al Desarrollo Rural (CADER).
- Cada Delegación Estatal tiene a su cargo un número determinado de *Distritos de Desarrollo Rural*.
- Cada Distrito de Desarrollo Rural consta de más de un Centro de Apoyo al Desarrollo Rural.
- ASERCA Regional se auxilia de las Delegaciones Estatales de la SAGAR.

En esta lista de relaciones candidatas se han omitido deliberadamente las relaciones existentes con las clases no seleccionadas tal como la relación: ASERCA Regional emite formatos de reinscripción. Sin embargo se debe depurar aún más la lista de relaciones precedente, por ello se realiza la selección de las relaciones que participarán en el modelo con base en los criterios siguientes:

- Deben omitirse relaciones que se encuentren fuera del dominio del problema.
- Se deben incluir las relaciones de carácter permanente y no aquellas referentes a sucesos transitorios, tales como las relaciones de estructura. Con base en este criterio se omiten las relaciones siguientes:

  ASERCA Central autoriza a ASERCA Regional impresión de formatos y documentos de pago.

  Delegación Estatal proporciona listados y formatos a los Distritos de Desarrollo Rural correspondientes.

  Distrito de Desarrollo Rural proporciona formatos y listados a sus Centros de Apoyo de Desarrollo Rural.

  Productor cobra cheque en el Sistema Bancario.

  Distrito de Desarrollo Rural difunde normas a los Subcomités de Control y Vigilancia.

  Subcomité de Control y Vigilancia remite documentación a Distrito de Desarrollo Rural.

- Las relaciones derivadas en términos de otras se omiten por ser redundantes. En esta situación se encuentra la relación:

SAGAR cuenta con 193 Distritos de Desarrollo Rural.

que puede ser inferida de las relaciones:

SAGAR cuenta con Delegaciones Estatales.

Una Delegación Estatal consta de Distritos de Desarrollo Rural.

Además de la relación anterior, las relaciones que se eliminan por este criterio son:

ASERCA Regional opera en ámbito federal.

SAGAR cuenta con 712 Centros de Apoyo al Desarrollo Rural.

- *Se determinan relaciones que no se mencionan textualmente pero que están implícitas en otras relaciones.* Así la relación:

  El productor cobra el cheque en el sistema comercial.

  Se traduce en la relación permanente:

  El productor interactúa con el sistema comercial.

  La relación temporal:

  El Sistema Comercial hace efectivo el cheque en el Sistema Bancario.

  que se traduce en la relación:

  El Sistema Comercial interactúa con el Sistema Bancario.

  La relación:

  El Sistema Bancario compensa el cobro del cheque en la Tesorería de la Federación.

  Se convierte en:

  TESOFE pacta compensación con el Sistema Bancario.

  La relación:

  TESOFE proporciona información de los cheques cobrados a ASERCA Central

  se obtiene la relación:

  ASERCA mantiene convenio con TESOFE.

Finalmente las relaciones que participan en el modelo son:

- SAGAR cuenta con 33 Delegaciones Estatales.
- La Delegación Estatal tiene a su cargo Distritos de Desarrollo Rural.
- El Distrito de Desarrollo Rural consta de Centros de Apoyo al Desarrollo Rural.
- ASERCA Central cuenta con 16 Centros Regionales.
- ASERCA Central opera en ámbito federal.
- SAGAR está en un ámbito federal.

- Una o varias Delegaciones Estatales de la SAGAR colaboran con una Regional de ASERCA.
- Varias Delegaciones SAGAR se desempeñan en un ámbito estatal.

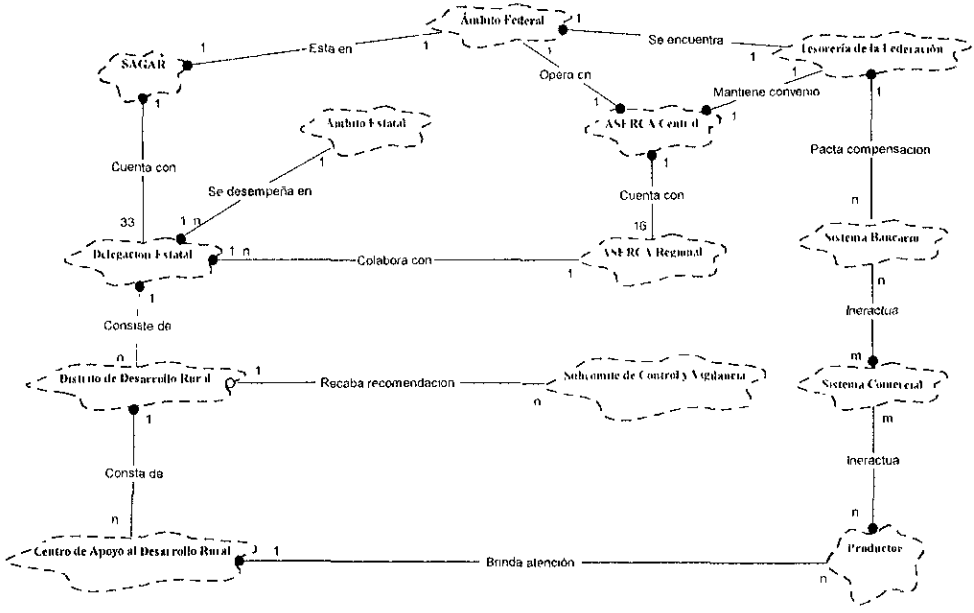


figura 3.2 Diagrama de clases que participan en el Programa de Apoyos Directos al Campo.

- Centro de Apoyo al Desarrollo Rural brinda atención a los productores.
- Los productores interactúan con el Sistema Comercial.
- El Sistema Comercial interactúa con el Sistema Bancario
- Varios Subcomités de Control y Vigilancia colaboran con un Distrito de Desarrollo Rural.
- La Delegación Estatal auxilia a ASERCA Regional.
- TESOFE se encuentra en un ámbito federal.
- ASERCA mantiene convenio con TESOFE.

Con las relaciones y clases seleccionadas se elabora el diagrama de clases mostrado en la figura 3.2. No es coincidencia que este diagrama asemeje a un diagrama entidad-relación en el cual se presenta la cardinalidad en las relaciones y además con el símbolo "•" que indica la dirección de la relación, la clase que tiene próximo el símbolo es la que realiza el verbo y la clase en la cual se omite es la que recibe la acción.

Una vez establecida la parte estática del sistema procedemos a establecer la parte dinámica (comportamiento). De la descripción general del sistema obtenemos los sucesos. En general los sucesos están asociados donde ocurre un intercambio de información, además deben determinarse las clases participantes en los sucesos. Los sucesos seleccionados con base en las clases que participan en el diagrama de la

figura 3.2 son:

- ASERCA Central autoriza emisión de solicitudes a ASERCA Regional.
- ASERCA Regional entrega solicitudes a la Delegación Estatal.
- La Delegación Estatal dispersa las solicitudes a los DDR.
- Los DDR distribuyen las solicitudes a los CADER.
- Los CADER's entregan las solicitudes al productor.
- El productor llena la solicitud y la entrega al CADER.
- El CADER verifica documentación y entrega solicitudes al DDR.
- El DDR revisa totales y entrega las solicitudes a la Delegación Estatal.
- La Delegación Estatal concentra las solicitudes y las entrega a ASERCA Regional.
- ASERCA Regional procesa la información y solicita a ASERCA Central la emisión de documentos de pago.
- ASERCA informa a TESOFE sobre el monto de los apoyos.
- TESOFE confirma recursos a ASERCA Central.
- ASERCA Central autoriza la emisión de los pagos.
- ASERCA Regional emite los pagos y envía a las Delegaciones Estatales.
- Las Delegaciones Estatales distribuyen los pagos a los DDR.
- Los DDR a su vez a los CADER.
- El CADER entrega el pago al productor.
- El productor cobra el cheque en el sistema comercial.
- El sistema comercial hace efectivo el cheque en el sistema bancario.
- TESOFE compensa el cobro del cheque al sistema bancario.
- TESOFE proporciona información de los cheques cobrados y cancelados a ASERCA Central.
- ASERCA Central concilia los cheques.

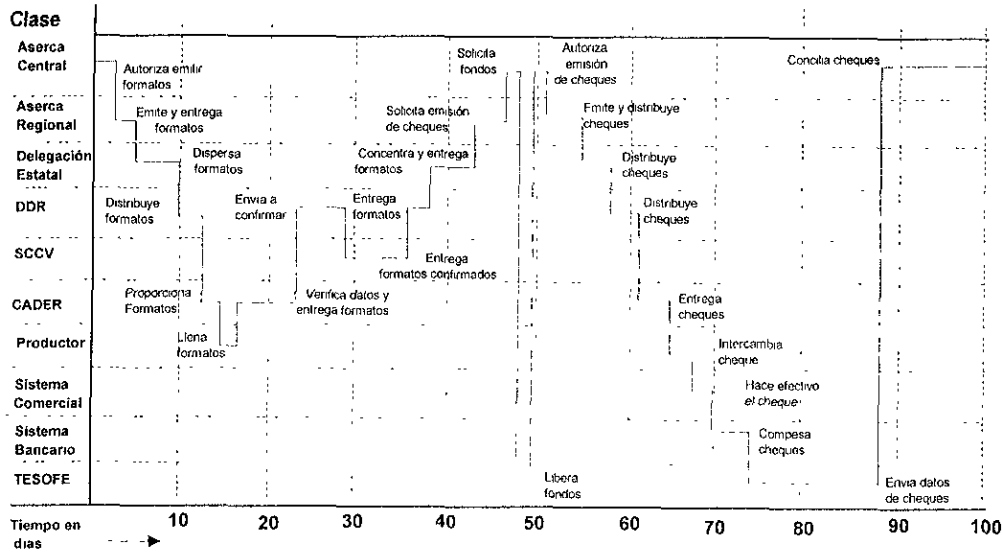


figura 3.3 Diagrama de sucesos del sistema.

Estos sucesos se modelan en el diagrama mostrado en la figura 3.3 donde se gráfica las clases en el eje vertical, el tiempo en el eje horizontal y los sucesos se ubican en las intersecciones clase-tiempo.

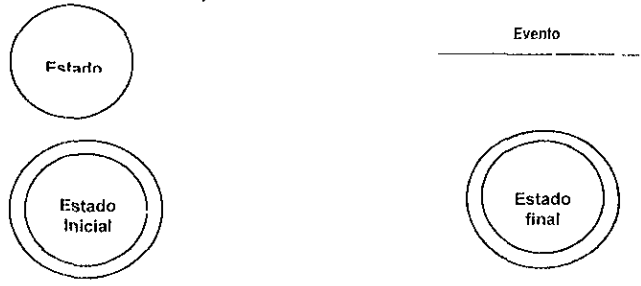


figura 3.4 Notación para Diagrama de Transición de Estados

El diagrama de sucesos debe ser consistente y completo En todo suceso debe existir un emisor y un receptor. A menos de los sucesos identificados como iniciales y finales, todo suceso debe tener un predecesor y sucesor. El diagrama de sucesos mostrado en la figura 3.3 cumple con estas características. Cada vez que la línea del diagrama cambia de una clase a otra el emisor es la clase a la izquierda y la clase receptor es la de la derecha De esta manera cuando en el diagrama se cambia de la clase ASERCA Central a la clase ASERCA Regional con el suceso Autoriza emitir formatos indica que ASERCA Central emite la autorización de formatos a ASERCA Regional.

Para obtener mayor información del comportamiento dinámico de las clases es necesario la construcción de diagramas de transición de estados para cada clase. Este tipo de diagramas muestran los sucesos o eventos que activan el cambio de estado en una clase además de las acciones que resultan

del cambio de estado. Los estados son representados mediante círculos y los eventos mediante flechas (ver figura 3.4). Dado que en el diagrama se asocia una acción para cada línea de transición de estado, el modelo usado es el Mealy.

Con base en la lista de sucesos se elaboran los diagramas de estado para cada una de las clases cuyo comportamiento sea no trivial y que se encuentren en marco en que se elabore la solución. Con base en estas premisas se seleccionaron ASERCA Central y ASERCA Regional elaborándose los diagramas de estados mostrados en la figura 3.5 y figura 3.6.

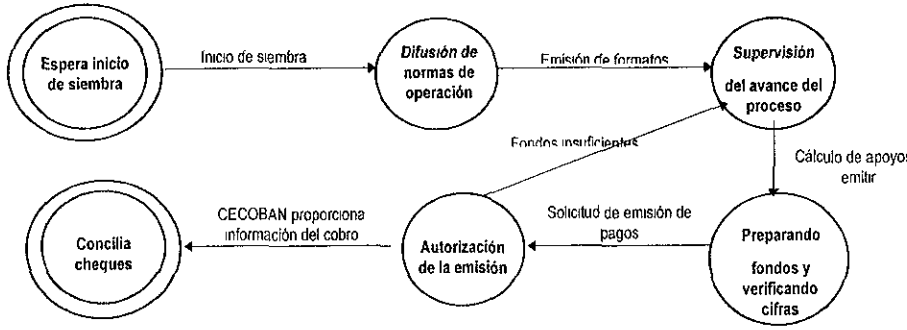


figura 3.5 Diagrama de estados de ASERCA Central.

El diagrama de estados de ASERCA Central parte del inicio de siembra el cual varía dependiendo de la región del país en que se ubique y generalmente es al comienzo de las estaciones primavera y otoño. Una vez iniciada la siembra ASERCA Central difunde a sus regionales las normas de operación del ciclo, mismas que establecen los procedimientos, tiempos y aspectos legales del programa de apoyos. Posteriormente al emitirse los formatos, se da seguimiento del avance en el proceso de los formatos, para lo cual se elaboran diariamente reportes de los principales indicadores, como totales de superficie a apoyar, número de productores que se han registrado, cultivos registrados, etc.

Cuando se determina el monto del apoyo, ASERCA Central obtiene los totales del monto para los apoyos y con base en ellos se gestionan los fondos con TESOFE. Una vez que los fondos están disponibles se autoriza la emisión de los apoyos. Finalmente la información de los cobros realizados por concepto de apoyo directo se envía de CECOBAN a ASERCA Central, donde se realiza la conciliación de los apoyos.

La autorización de emisión de formatos es el evento que activa la operación de ASERCA Regional, donde con base en el padrón de predios y productores se generan los formatos de inscripción, los cuales se distribuyen cuando los cultivos se han arraigado. Una vez llenos los formatos se procesan, corrigen y por último se emite el apoyo correspondiente.

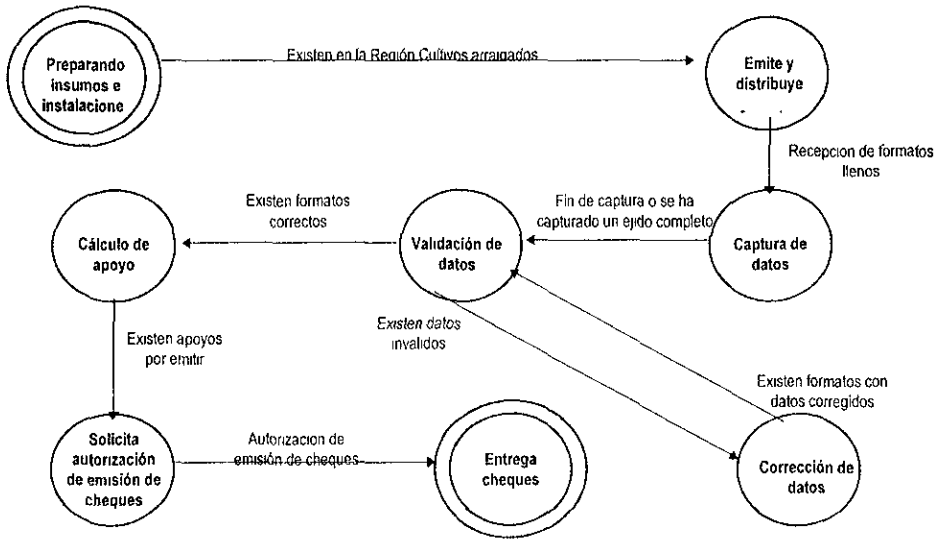


figura 3.6 Diagrama de estados de ASERCA Regional

Para completar el modelo dinámico del sistema es necesario realizar el diagrama de objetos donde se da importancia al flujo de mensajes e información. Es importante notar que no describe el tiempo y secuencia en que se da dicho flujo. Los objetos se representan mediante figuras amorfas que asemejan nubes de línea continua y se identifican por un nombre. Las líneas entre los objetos indican una relación de comunicación que se complementa con los mensajes que se representan con flechas que indican la dirección en que fluye el mensaje. Cuando un mensaje se deriva como respuesta de otro se indica con una flecha precedida por un círculo

El diagrama de objetos correspondiente al proceso de entrega de apoyos se muestra en la figura 3.7. En el diagrama se separan los objetos externos a la realización de la solución mediante la figura amorfa identificada como ASERCA que abarca ASERCA Central y ASERCA Regional.

Con respecto a la clase ASERCA existen dieciocho casos específicos o instancias, éstas son:

- ASERCA Central México D.F.
- Regional Hermosillo
- Regional Chihuahua
- Regional Lerdo
- Regional Reynosa
- Regional Culiacán
- Regional Zacatecas
- Regional Guadalajara
- Regional Celaya

- Regional Morelia
- Regional Toluca
- Regional Xalapa
- Regional Puebla
- Regional Chilpancingo
- Regional Oaxaca
- Regional Tuxtla
- Regional Mérida
- Regional Hidalgo

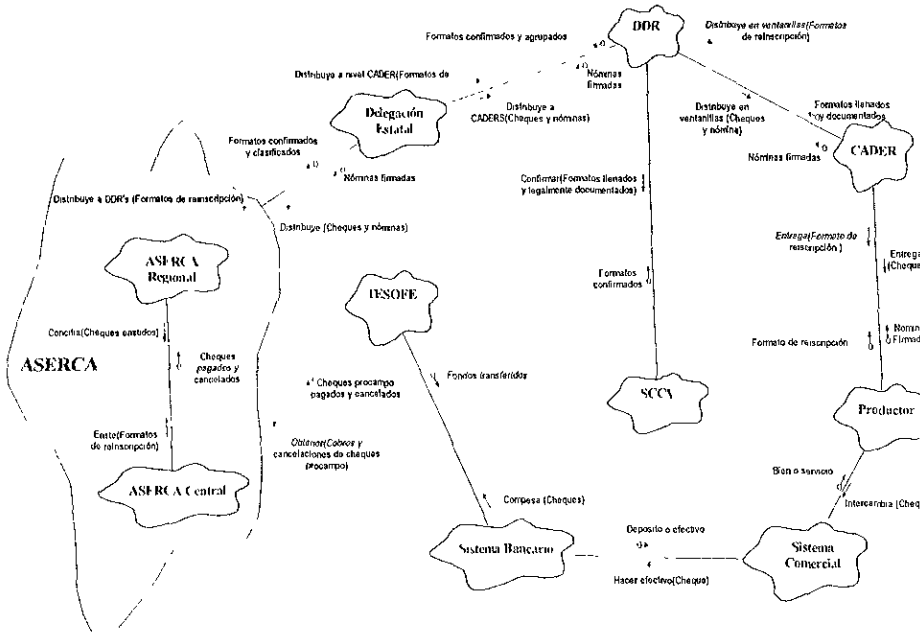


figura 3.7 Diagrama de Objetos de la emisión de apoyos directos.

3.1.6 Evaluación de alternativas de solución

El problema a resolver es el de compartir la información de los centros regionales con ASERCA Central y viceversa, soportando modificaciones de la misma en los nodos y a la vez evitando la

generación de versiones de la información inconsistentes entre sí. En la elaboración de las alternativas de solución se considera el estado actual en cuanto equipo y software de ASERCA. Las posibles soluciones se enumeran a continuación:

1. Copia de la información origen.
2. Acceso directo entre ASERCA Regional y ASERCA Central.
3. Implantación de una herramienta de réplica comercial.
4. Implantación de un esquema de réplica propio.

3.1.6.1 Copia de la información origen

En ASERCA Central se crea una copia de la información a compartir del regional, de igual manera se crea una copia en el regional de la información de ASERCA Central. Este esquema independiza la disponibilidad de los datos respecto de la línea de comunicaciones. Sin embargo la información contenida en la copia es siempre una versión anterior con respecto a la información fuente. Bajo este esquema no existe el control de las actualizaciones en la información original, por ello la copia se actualiza eliminando totalmente su información y transfiriéndole la información origen completa. Por otro lado las copias se restringen a consultas solamente ya que si se aplican operaciones de modificación a la información en ambos puntos resultaría versiones inconsistentes entre sí, provocando que la tarea de discernir los datos más actuales en cada versión sea un trabajo complejo y algunas ocasiones irrealizable.

3.1.6.2 Acceso directo entre ASERCA Regional y ASERCA Central

En esta solución la información se accede directamente desde ASERCA Central a ASERCA Regional y en sentido inverso el regional accede la información que le corresponde directamente en ASERCA Central. Lo cual permite disponer de los cambios más recientes de la información. Por otro lado muchos manejadores de bases de datos soportan las transacciones distribuidas en línea bajo el concepto denominado **commit** de dos fases. Sin embargo este esquema esta sujeto al desempeño y disponibilidad de la red de comunicaciones.

3.1.6.3 Solución basada en una herramienta de réplica comercial

Los esquemas de réplicas comerciales se basan en las operaciones aplicadas a la información y no sólo en los datos por lo cual en la actualización de las copias ya no es necesario reinicializarlas y transmitir totalmente la información origen, sino solamente los cambios efectuados desde la fecha en que se realizó la copia anterior. Por tanto se reduce el volumen de información a transmitir. Por otro lado la mayoría de los esquemas de réplica disponibles son de tipo asimétrico, es decir sólo permiten realizar consultas en las copias y en éstas se impide cualquier otra operación. Las modificaciones sólo se permiten donde se encuentra la información origen.

3.1.6.4 Implantación de un esquema de réplica propio

El esquema de réplica se basa en el control de las operaciones aplicadas diariamente a la información para la creación de las réplicas, de esta manera el envío de la información se efectúa en incrementos

con lo cual se reduce el volumen de información en la red. Además este esquema contará con las características siguientes:

- La información compartida por ASERCA y sus regionales debe modificarse tanto en el origen como en las copias para lo cual se implante un esquema de réplica simétrico.
- Mejor desempeño de la transferencia de las operaciones a las copias, lo cual se logra al comprimir los datos antes de ser enviados.
- Permitirá enviar las operaciones en otro medio diferente de la red, como por ejemplo disco o cinta.

3.1.6.5 Selección de la solución

De las alternativas propuestas en los puntos 3.1.6.1, 3.1.6.2, 3.1.6.3 y 3.1.6.4 se escogerá aquella cuyos costos sean menores y permita que la información se comparta sin que existan inconsistencias en los datos.

Las condiciones del entorno a las que estará sujeta la solución son:

- La red de comunicaciones de ASERCA esta soportada principalmente por vía telefónica entre puntos muy separados resultando lenta y poco confiable. Existen sólo dos enlaces rápidos con líneas de fibra óptica RDI y quince son enlaces telefónicos. A partir de la bitácora durante un año del nodo rs980 de ASERCA Central se ha encontrado que en promedio seis días en un mes no existe conexión a alguno de los nodos regionales.
- Durante tres años agrícolas (1994,1995 y 1996) el volumen de operación promedio de solicitudes operadas en cada año para el Programa de Apoyos Directos ha sido de 4.2 millones a nivel nacional[VII] y un volumen promedio de pagos de 3,878,596 tanto cobrados como cancelados.
- Se elabora quincenalmente un reporte de emisión y conciliación de pagos a nivel nacional en oficinas centrales además de otras consultas no planeadas que afectan el desempeño de los sistemas.

Los costos a evaluar son de tres tipos:

- Tiempo de respuesta.
- Económico.
- Esfuerzo.

Tiempo de respuesta

El tiempo de respuesta al aplicar las transacciones afecta directamente las actividades de ASERCA debido a que el gran volumen de transacciones implica un tiempo significativo tanto en procesos en lotes (batch) y procesos en línea. Existen dos tipos de costo en tiempo y estos son:

- El costo en tiempo al aplicar una transacción en horario de operación. Dado que en el esquema de acceso directo las transacciones se efectúan en los nodos remotos, la información debe ser transferida entre los nodos a la vez que se aplica la transacción y con ello se incrementa el tiempo por transacción durante la operación. Por otro lado en los esquemas de copia y réplica, el tiempo de transferencia no afecta el tiempo de aplicación de la transacción si la transferencia ocurre en un horario distinto al de operación.
- El costo en tiempo de transferencia de la información. Este costo está en función de la velocidad de transmisión, del uso de algoritmos de compresión y el horario en que se efectúa la transferencia.

El tiempo de respuesta de una transacción esta determinado por el número, velocidad y arquitectura de los procesadores, del número de accesos a los dispositivos de entrada/salida, del número de procesos corriendo en el sistema, además de la ubicación de la información en la red. Por ello para determinar el tiempo promedio de ejecución de una transacción se tomaron lecturas de ejecuciones en una sola plataforma (RS6000 modelo rs570) y para transacciones equivalentes. Las ejecuciones de las aplicaciones de ASERCA se registran en las bitácoras correspondientes, éstas contienen la hora de inicio, el número de transacciones realizadas y la hora final. Dado que la solución a implantar esta dirigida al proceso de conciliación de pagos se eligen las bitácoras de los procesos relacionados a dichos procesos para obtener el promedio del tiempo para procesar una transacción.

a) Copia de la información.

En el esquema de copia, la información se ubica localmente por lo cual las transacciones no hacen accesos remotos. Un inconveniente de este esquema es que no existe un control de las operaciones aplicadas en el día. Este esquema operó durante los primeros dos años en ASERCA para la conciliación de pagos. De las bitácoras de los procesos de conciliación aplicados localmente se obtiene la tabla 2. En la cual se muestra para cada corrida el número de transacciones aplicadas, la hora de inicio, la hora de terminación, la duración (estas tres expresadas en horas: minutos: segundos) y finalmente el tiempo promedio por transacción que se obtiene de dividir la duración expresada en segundos entre el total de transacciones. Para obtener el promedio definitivo se suma el total de transacciones que es de 642,622 transacciones y la suma de los intervalos de tiempo que es de 9 horas 3 minutos y 44 segundos (en total 32,624 segundos) dando un promedio de 0.05 segundos por transacción aplicada localmente en horario de operación.

Por otro lado, falta determinar el tiempo de transferencia, el cual se expresa en segundos por transacción. Cabe mencionar que este esquema no transfiere la información en incrementos ya que no hay distinción entre la información enviada y la información agregada, eliminada o modificada desde la última copia Por ello la información con cambios o sin ellos se envía, lo cual hace de este esquema de bajo desempeño. En particular de datos obtenidos de las bitácoras de ASERCA se transfirieron vía *ftp* 118 Megabytes (123,731,968 bytes) en 78 horas 40 minutos (283200.00 segundos) para 60,416 transacciones de pagos que idealmente deberían corresponder a 14.75 Megabytes (15,466,496 bytes) ya que en promedio cada transacción ocupa 256 bytes, lo que implica que un promedio de 8 veces se transfirió cada registro; consecuentemente el tiempo por transacción es amplificado por este efecto al valor de 4.69 segundos por transacción.

Cuando la línea de comunicaciones esta interrumpida la transferencia en este esquema es a través de cintas de 2 Gigabytes las cuales se envían por paquetería, tardando 2 días en promedio desde que la información es extraída en el Regional hasta su integración en ASERCA Central. Con base en la bitácora de los dos años en que se aplicó el esquema de copia se transfirieron 1.24 Gigabytes para 582,206 transacciones de pagos, que idealmente correspondería a un volumen de 142.14 Megabytes, esta diferencia se debe a que se transfirieron tanto registros con cambios como los que ya existían en la copia, lo cual resultó en promedio 9 transferencias para un mismo registro. Esta transferencia ocurrió durante 24 días (no se incluye el tiempo de aplicación de las operaciones), arrojando un tiempo de transferencia de 3.56 segundos por transacción.

Cabe notar que las copias se realizan fuera del horario de operación consecuentemente el tiempo de transferencia a través de la red o en cintas para realizar la copia no afecta la operación directamente,

tampoco las interrupciones en las líneas de comunicación afectan la respuesta de los procesos ya que éstos operan localmente con la copia, aún cuando esta no contenga la última versión de la información.

b) Acceso directo.

El tiempo por transacción aplicada directamente a objetos remotos se obtiene de las bitácoras de las aplicaciones de conciliación de ASERCA, las cuales se basan en el producto sql*net de ORACLE.

Con base en una muestra de 27 corridas en las cuales se procesaron 76,831 registros en 98 horas 40 minutos y 21 segundos (un total de 355,221 segundos) se obtiene un promedio de 4.62 segundos por transacción aplicada en horario de operación. En este esquema, además se debe considerar las consultas que se realizan periódicamente y las no planeadas, las cuales deben acceder a los datos remotamente, y son: la consulta quincenal de los pagos emitidos y conciliados, las consultas de auditoría y consultas de pagos con error al conciliar. Mismas que se efectúan en horarios de operación. Bajo este esquema se consultaron en promedio 69,436 registros por regional mediante sql*net, promediando 21 horas 3 minutos y 47 segundos lo que arroja 1.09 segundos por registro consultado. Así el tiempo total por operaciones en cada pago es de 5.71 segundos siempre y cuando exista el enlace de red.

La falta de enlace se presenta en promedio una vez cada semana durante todo el día a un regional, por tanto el retraso para acceder a la información es de un día de operación. De cuatro ocasiones que se interrumpió la comunicación 15,763 registros se procesaron en 37 horas y 05 minutos (133499.79 segundos) lo que resulta en 7.96 segundos por transacción más los tiempos por consultas afectadas en un día por falta de acceso a la red en las cuales se consultaron 10,456 pagos en 5 ocurrencias que tardaron 40 horas 10 minutos 18 segundos (155418.39 segundos) con un tiempo de 14.86 segundos por transacción consultada.

Por otra parte, dado que seis días de cada mes de operación se interrumpe la comunicación la probabilidad de contar con enlace es de 0.8.

Número de corrida	Transacciones	Hora de inicio	Hora de terminacion	Duración	Tiempo promedio por transacción (Duración en Segundos/Transacciones)
1	113	12 34 32	12 35 20	0 00 48	0.424778761
2	19 866	12 01 41	12 31 15	0 29 34	0.089298299
3	322	11 59 59	12 00 40	0 00 41	0.127329193
4	36,349	11 13 32	11 57 40	0 44 08	0.072849322
5	122	11 13 12	11 13 21	0 00 09	0.073770492
6	311	11 12 30	11 12 50	0 00 20	0.064308682
7	99,818	10 01 28	11 11 32	1 10 04	0.042103908
8	368	10 00 58	10 01 11	0 00 13	0.048507463
9	184	10 00 44	10 00 52	0 00 08	0.043478261
10	20 791	11 33 00	11 42 27	0 09 27	0.027271116
11	29,361	12 00 23	12 20 34	0 20 11	0.041240975
12	57 539	11 21 06	11 58 15	0 37 09	0.038738942
13	12 855	13 44 27	13 55 39	0 11 12	0.052275379
14	579	13 43 25	13 43 58	0 00 33	0.056994819
15	13 737	13 30 55	13 42 43	0 11 48	0.051539637
16	810	12 28 37	13 29 07	1 00 30	1.481481481
17	15 723	13 05 32	13 21 02	0 15 30	0.059149017
18	11,148	12 42 48	12 49 34	0 06 46	0.036419089
19	12 951	17 14 15	17 20 37	0 06 22	0.029495792
20	11 978	20 19 45	20 31 21	0 11 36	0.046468153
21	9 916	20 11 19	20 19 16	0 07 57	0.048104074
22	10 988	20 05 01	20 10 16	0 05 15	0.028667637
23	9 373	14 46 00	14 49 39	0 03 39	0.023364985
24	25 300	14 31 19	14 41 38	0 10 19	0.024466403
25	701	14 17 23	14 17 44	0 00 21	0.029957204
26	828	20 10 52	20 12 00	0 01 08	0.082125604
27	661	20 09 32	20 10 21	0 00 49	0.074130106
28	25 728	19 45 37	20 06 03	0 20 26	0.017652363
29	548	19 43 42	19 44 01	0 00 19	0.034671533
30	19 277	19 32 37	19 43 09	0 10 32	0.032785184
31	1 381	19 31 09	19 32 05	0 00 56	0.040550326
32	14 931	17 47 18	18 00 19	0 13 01	0.052296772
33	7 012	17 39 38	17 45 00	0 05 22	0.045921778
34	2 150	17 37 31	17 38 50	0 01 19	0.036744186
35	16,901	12 54 22	13 08 12	0 13 50	0.04910952
36	19,146	12 39 19	12 54 03	0 14 44	0.046171524
37	23,593	20 49 09	21 07 31	0 18 22	0.046710719
38	27 969	20 25 15	20 48 52	0 23 37	0.050679542
39	39,223	16 38 12	17 10 35	0 32 23	0.049537261
40	17 645	15 44 24	15 54 58	0 10 34	0.035930859
41	21,500	15 06 43	15 18 25	0 11 42	0.032651163
Total	642,622			9 03:44	0.050767014

tabla 2 Muestra de corridas del proceso de conciliación en horario de operación.

c) Réplica comercial.

El tiempo por transacción para un esquema de réplica se obtiene de los resultados de pruebas con el producto *Snapshot* de ORACLE. Las pruebas consistieron en implantar el producto para la conciliación de pagos como un mecanismo alterno al de acceso directo. Este producto permite crear un esquema de réplica asimétrica, es decir con transacciones solo en uno de los nodos que comparte la información, cualidad que impidió atender la necesidad de realizar operaciones tanto en regionales como en oficinas centrales. La creación del esquema de *Snapshots* añade controles para almacenar las operaciones diarias que afectan el tiempo de respuesta de las transacciones de conciliación. Con base en los datos de las bitácoras de 5 corridas, el procesamiento de 279,686 transacciones requirió de 8 horas con 30 minutos y cuarenta segundos (30,640 segundos) lo que da un promedio de 0.11 segundos por registros. De igual forma que para el esquema de copias, los tiempos de transferencia de transacciones entre nodos e interrupciones de la red no alteran el tiempo promedio en esta primera parte, ya que ocurre en horarios fuera de la operación diaria.

En segundo termino se determina el tiempo promedio de transferencia a partir de los datos de las bitácoras de operación los cuales son: para 279,686 transacciones se requirió de 68 horas 16 minutos y 58 segundos promediando un tiempo de 0.89 segundos por transacción a través de la red. La falta de conexión repercute en retrasos para la transferencia ya que no existe un medio alterno como la cinta magnética. En operación se presentó una interrupción de ocho horas que se sumo a las seis horas cuarenta y un minutos que tomó la transferencia de 26,753 transacciones resultando en un tiempo de 1.97 segundos por transacción.

d) Esquema de réplica propio.

En este caso se asume que el tiempo de respuesta por transacción será similar al del esquema de *Snapshots* debido a que el control de las operaciones aplicadas diariamente para generar las réplicas se basará en las mismas herramientas de *triggers* proporcionadas por ORACLE.

El tiempo de transferencia a través de la red se espera disminuir en una quinta parte del tiempo de transferencia de la réplica comercial ya que la información se comprimirá antes de ser enviada. Por lo cual el tiempo de transferencia esperado es de 0.18 segundos por transacción.

Además el esquema contará con un control de transferencia de información mediante cintas cuando la red este interrumpida. Dado que este valor no se cuenta se presupone un comportamiento similar al esquema de copias pero se evitaría traer más de una vez una misma información a menos de que sea modificada, de esta manera si para el esquema de copia el tiempo de transferencia es de 3.56 segundos por transacción y con un promedio de 9 transferencias para el mismo registro, se espera que el tiempo en la réplica se reduzca en esa proporción, lo que arroja un tiempo de 0.39 segundos por transacción.

Comparativo de tiempos

La selección de la solución debe considerar las condiciones del entorno, con este propósito se elabora un árbol de decisión. En primer termino se resumen los resultados en la tabla 3 donde se muestran los tiempos de ejecución de cada transacción en horario de operación para cada una de las soluciones propuestas.

Tiempo en operación	Condición del enlace de red Unidades en segundos por transacción	
Solución	Disponible c1	Suspendido (1 día) c2
a) Copia de información	0.05	0 05
b) Acceso directo	5.71	14.86
c) Réplica comercial	0.11	0.11
d) Réplica propia	0.11	0.11
Probabilidad de presentar la condición	0.8	0.2

tabla 3 Tiempo de respuesta por transacción en horario de operación

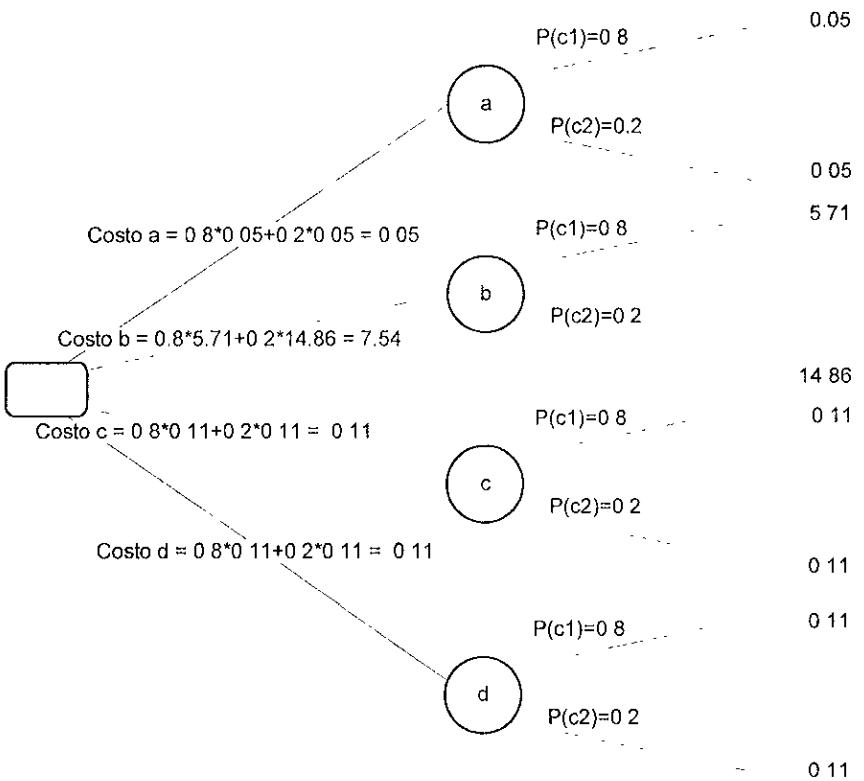


figura 3.8 Árbol de decisión para la evaluación del tiempo por transacción en horario de operación.

El árbol para determinar el costo en tiempo por transacción en horario de operación se elabora a partir de estos datos y se compone de un nodo de inicio representado por un cuadrilátero de bordes suaves del cual se ramifican las soluciones en los nodos que van de la etiqueta "a" a la etiqueta "d" correspondientes a los incisos de la tabla, de cada nodo solución se pueden presentar una de las dos condiciones con su costo correspondiente. El criterio de selección es la rama con el mínimo costo, la cual corresponde a la solución basada en copias de la información.

Ahora se procede de manera similar para el comparativo en el costo en tiempo, debido a la transferencia de información. En la tabla 4 se resumen los resultados para la transferencia de información.

Tiempo en transferir	Condición del enlace de red Unidades en segundos por transacción	
Solución	Disponible c1	Suspendido (1 día) c2
a) Copia de información	4.69	3 56
b) Acceso directo	5.71	14 86
c) Réplica comercial	0 89	1.97
d) Réplica propia	0 18	0.39
Probabilidad de presentar la condición	0.8	0.2

tabla 4 Tiempo de transferencia por transacción.

Los tiempos de transferencia para los esquemas de acceso directo y copias se ven amplificados debido a que una misma información, sin importar que haya cambiado o no, es transferida varias veces mientras en los esquemas de réplica una vez transferida una información posteriormente ya no se transfiere a menos de que sufra cambios.

Como se observa en la tabla el esquema de acceso directo es el más afectado cuando la línea esta suspendida en tanto los demás esquemas disminuyen el efecto negativo de la falta de línea con el uso de cintas enviadas por paquetería. A partir de los datos de la tabla 4 se elabora el árbol de decisiones presentado en la figura 3.9 que con base en las posibles condiciones que puede presentar la red de ASERCA, la solución a elegir sería la "d" con un costo de 0.22.

Costo económico y esfuerzo

El costo de desarrollo de la aplicación se determina a partir de las actividades a realizar, las cuales son:

Análisis y diseño del esquema de réplica.

Programación de los módulos.

Pruebas de integración de los módulos.

Implantación del sistema.

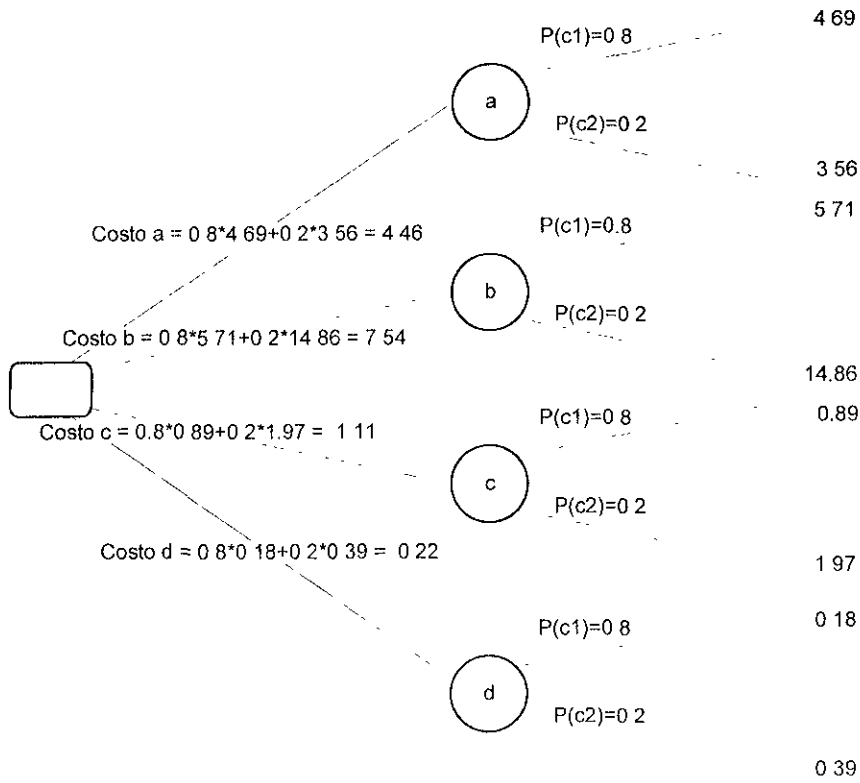


figura 3.9 Árbol de decisión para determinar el costo en tiempo de la transferencia de datos.

a) Copia de información.

Para el costo de este esquema se toma como referencia el esquema de copia implantado para la reexpedición de pagos usado en los ciclos OI93 y PV94 de ASERCA. Este sistema requirió de un esfuerzo estimado en 3,270 horas hombre con un costo de $230,600.

b) Acceso directo

Dado que actualmente ASERCA cuenta con la opción de commit en dos fases para la base de datos no se requiere de software adicional. Para cambiar las aplicaciones de conciliación de pagos bajo este esquema se requiere de un tiempo de un mes para un equipo de 3 personas: un líder de proyecto y dos programadores.

Persona	Precio unitario (costo/ hora)	Hora trabajadas	Número de personas	Costo (pesos M.N.)
Líder de proyecto	$125.00	160	1	$20,000
Programador	$43 00	160	2	$13,760
Total			3	$33,760

tabla 5 Costo económico de esquema de acceso directo.

El costo bajo el esquema de acceso directo es de $33,760 con un esfuerzo de 480 horas/hombre.

a) Réplica comercial

El costo de compra de una base de datos con opción distribuida para un plataforma IBM RS/6000 modelo 570 o rs980, en el primer semestre de 1997 se sitúa en 16,000 dólares. Considerando 18 nodos modelo rs570 (17 regionales y un nodo de desarrollo) y 2 nodos rs980 el costo total es de 320,000 dólares, considerando una paridad de 7.95[VIII] el costo es de 2,544,000 pesos. Adicionalmente ASERCA debe cambiar el esquema actual para la conciliación de pagos de manera que aproveche opción distribuida del producto. Para lo cual se estima un tiempo de dos meses para un equipo de 3 personas, un líder de proyecto y 2 programadores.

Persona	Precio unitario (costo/ hora)	Hora trabajadas	Número de personas	Costo (pesos M.N.)
Líder de proyecto	$125 00	320	1	$40,000
Programador	$43.00	320	2	$27,520
Total			3	$67,520

tabla 6 Costo del desarrollo del esquema de réplica comercial.

El costo total es de $2,611,520 con un esfuerzo de 960 horas/hombre.

a) Réplica propia.

Se propone un equipo de 4 programadores, un analista y un líder de proyecto quienes tendrán a punto el sistema transcurridos 6 meses. Durante el primer mes trabajan el analista y el líder para delimitar el problema y elaborar los diseños iniciales, después de transcurrido ese lapso se integran los cuatro programadores. El analista trabajará tres meses: el primero y segundo plantea los requerimientos, posteriormente hasta el sexto mes verifica con el usuario que los programas atiendan los requerimientos planteados al principio del proyecto. El líder esta al tanto del proyecto durante los seis meses de duración del proyecto. Con base en estos datos y los precios unitarios para cada uno de los elementos del equipo se determina el costo de desarrollo de la solución el cual se muestra en la tabla 7.

Persona	Precio unitario (costo/ hora)	Hora trabajadas	Número de personas	Costo (pesos M.N.)
Líder de proyecto	$125.00	960	1	$120,000
Analista	$175.00	480	1	$84,000
Programador	$43.00	640	4	$110,080
Total			6	$314,080.00

tabla 7 Costo del desarrollo del esquema de réplica propio.

El costo del desarrollo es de $314,080.00 con un esfuerzo de 4,000 horas/hombre.

Cuadro comparativo.

Para cada uno de los rubros evaluados se compara entre cada una de las soluciones los cuales se muestran en la tabla 8. Fijando el costo económico y de esfuerzo como criterio de selección, la opción elegida es el esquema con acceso directo. Si la red fuese confiable y permitiera transferir a mayor velocidad sería realmente la solución más práctica. Sin embargo, las condiciones de la red impiden la implantación del esquema, ya que con un tiempo de respuesta en operación de 7.54 segundos por transacción para 3,878,596 transacciones por cada año agrícola en 17 regionales implica un retraso de 20 días por regional. La siguiente opción a elegir por su costo económico es el esquema de copia, éste tiene la desventaja de programarse específicamente para cada grupo de tablas y cada vez que cambia la estructura es necesario volver a programar los módulos. Además este esquema restringe a las copias a modo lectura ya que de otra forma se crearían versiones inconsistentes de la información. Por esta inflexibilidad a la larga el costo de la copia se incrementa. Para este esquema el tiempo consumido durante un año de operación es para cada regional de 3 horas 10 minutos, sin embargo en la transferencia el tiempo anual es de 19 horas 17 minutos por regional.

Por otra parte el esquema de réplica propio que es económicamente casi un 30 % mayor al esquema de copia, en se espera que el tiempo de operación sea de 6 horas 58 minutos por cada regional. En cuanto al tiempo de transferencia el costo será de 13 horas 56 minutos por cada regional, que aventaja al tiempo de transferencia de una réplica comercial en 5 veces, ya que este es de 2 días 21 horas. La implantación de esquema de réplica tiene la ventaja de facilitar la implantación a nuevas tablas o el mantenimiento debido a cambios en la estructura de la base de datos. De ahí que finalmente se elija el esquema de réplica propio.

Solución	Costo en tiempo en operación (seg. / transacción)	Costo en tiempo de transferencia (seg. / transacción)	Costo económico (M.N.)	Costo en esfuerzo (horas/ hombre)
Copia de Información	0.05	4 46	$230,000.00	3,270
Acceso directo	7 54	7.54	$33,760.00	480
Réplica comercial	0 11	1.11	$2,611,520 00	960
Réplica propia	0.11	0.22	$314,080 00	4,000

tabla 8 Comparativo de costos para las soluciones propuestas

3.2 Diseño

Una vez planteado el problema y construido los modelos que representan el proceso de apoyos directos se puede iniciar la construcción de la solución seleccionada. En esta etapa se incorporan los componentes en equipo y programas que forman parte del sistema y se conforma por los siguientes puntos:

Arquitectura de los subsistemas.

Esquema de la base de datos.

Mecanismo de réplica.

3.2.1 Arquitectura de los subsistemas

La solución a implantar estará bajo un entorno de bases de datos distribuidas horizontalmente, donde 19 nodos contienen parte de la información de los pagos efectuados que habrán de integrarse en el nodo central donde se realiza el proceso de conciliación. Estos nodos se distribuyen geográficamente a través de la República Mexicana formando una red de área extendida (WAN) la cual cuenta actualmente con dos redes de área locales (LAN), ver figura 3.10; físicamente los nodos se comunican a través de dos medios:

a) Par trenzado.

b) Fibra óptica.

Por el alcance de los enlaces se considera que existen dos tipos:

a) Remotos.

b) Locales.

Estas dos características se combinan tal como se muestra en la tabla 9 donde se presenta el protocolo y/o servicio utilizados. A través de fibra óptica se llevan conexiones locales y remotas, las primeras utilizan el protocolo de paso de testigo en anillo (Token Ring), mientras las remotas utilizan el servicio de RDI. Además de la fibra óptica, existen conexiones remotas y locales en par trenzado, en este caso las conexiones remotas están bajo el protocolo X.25 y para enlaces locales entre terminales y el servidor se utiliza el protocolo RS-232-C.

La infraestructura de comunicaciones actual consiste de encriptores, modems, switches y multiplexores. La información es procesada por los encriptores cada vez que entra o sale del nodo, posteriormente si existen los dos tipos de conexión a RDI y X.25 los datos pasan a través de un switch el cual determina si los datos se envían mediante modems (en X.25) o mediante el multiplexor (en RDI). La velocidad de los modems es de 9600 bps.

Medio Físico/ Alcance	Par trenzado	Fibra óptica
Local	RS-232-C	Paso de testigo
Remoto	X 25	RDI

tabla 9 Identificación de los tipos de enlace en ASERCA.

En la figura 3.10 se muestra la distribución de los nodos donde los enlaces remotos se representan mediante una línea quebrada en forma de rayo, los enlaces locales mediante líneas continuas, la comunicación basada en X.25 así como la basada en RDI se representan mediante una nube y la etiqueta correspondiente, para representar la conexión vía paso de testigo se representa con un círculo. Los nombres de los nodos están definidos bajo el dominio de direcciones IP *central.ASERCA*.

Cabe notar que no todos los nodos se conectan directamente a ASERCA Central. Existe un grupo de nodos con conexión X.25 directa, estos nodos están ubicados al centro y por consecuencia están más cercanos respecto a la Ciudad de México, este grupo se forma por: Puebla, Xalapa, Celaya, Toluca, Morelia, Chilpancingo.

Un segundo grupo de nodos esta conectado directamente vía RDI, estos nodos son Mérida y Guadalajara.

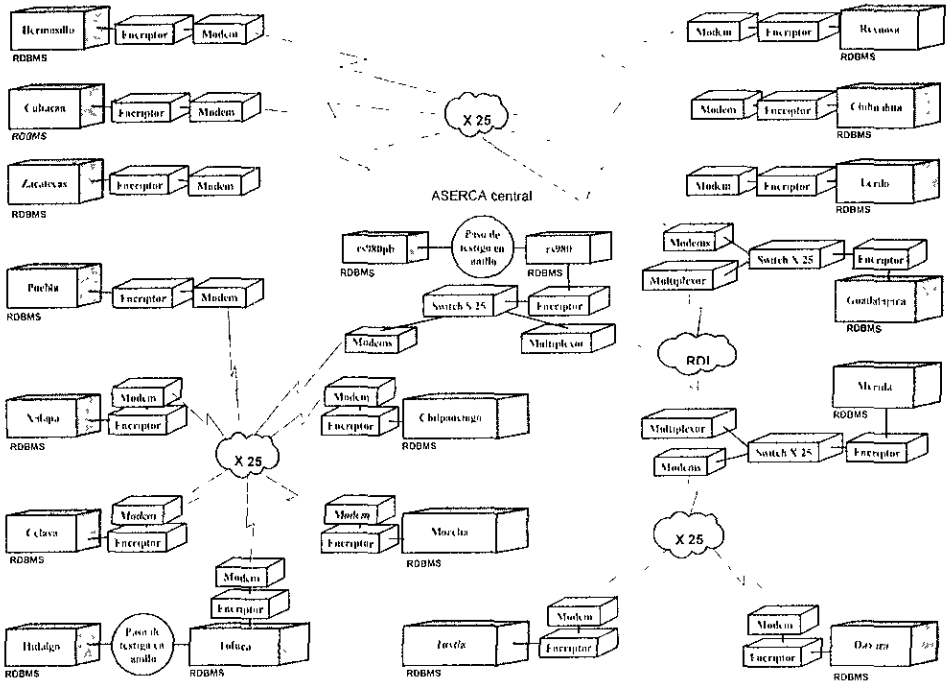


figura 3.10 Distribución del equipo de cómputo en ASERCA.

El tercer grupo de nodos realizan su conexión a ASERCA Central través del nodo de Guadalajara, al cual se conectan vía X.25, estos nodos se ubican en la parte Norte del país y son. Hermosillo, Culiacán, Zacatecas, Reynosa, Chihuahua y Lerdo.

El cuarto grupo consiste de los nodos cuyo enlace a ASERCA Central se realiza mediante el nodo de Mérida, al cual se conectan a través de X.25, estos nodos se ubican al Sur de la República y comparten la línea de RDI hacia ASERCA Central, los nodos que conforman este grupo son: Tuxtla y Oaxaca.

El nodo de Hidalgo se enlaza a ASERCA Central a través del nodo de Toluca, los cuales están conectados en una red local de paso de testigo en anillo y se ubican físicamente en las mismas oficinas en Toluca

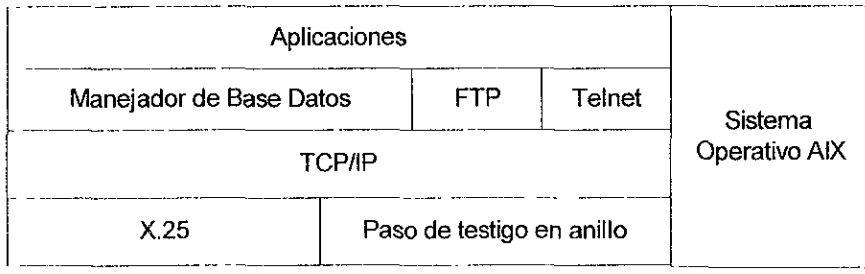


figura 3.11 Componentes de Software.

Cada nodo RISC RS6000 cuenta con el sistema operativo AIX y se conecta al red bajo el protocolo de TCP/IP. Además en el nodo existe una instancia de base de datos en la cual se registran las transacciones diarias. Las instancias existentes se dividen por su estructura en dos tipos: instancia regional e instancia central. En cada regional existen aproximadamente 25 terminales en promedio, en las cuales se realizan operaciones de captura, corrección, cálculo de apoyos etc. Los componentes de software se presentan en la figura 3.11, la cual incluye los principales componentes y algunos servicios de uso común como ftp y telnet.

3.2.2 Esquema de base de datos

En ASERCA se cuenta con el sistema automatizado de información PROCAMPO para el control del programa de apoyos directos. Este sistema controla las transacciones de la operación diaria en los centros regionales y oficinas centrales, análogamente a estas categorías se distinguen dos esquemas conceptuales de bases de datos:

- Un esquema conceptual de PROCAMPO para los centros regionales el cual se identifica con el nombre de instancia REG.
- Un esquema conceptual de PROCAMPO en ASERCA central, este se identifica con el nombre de instancia ASERCA.

La interfaz de comunicación entre los esquemas conceptuales está basada en la herramienta sql*net de ORACLE. Cada uno de los esquemas a su vez consisten de esquemas externos que son el medio por el cual el usuario accede a los datos. A continuación se listan los componentes de cada esquema conceptual:

Esquemas externos del esquema conceptual de un centro regional:

- Entrega de apoyos.
- Registro voluntario.
- Mantenimiento.
- Control de almacén.

Esquemas externos del esquema conceptual de oficinas centrales:

- Conciliación de apoyos.
- Autorización de emisión de apoyos.
- Indicadores de gestión.

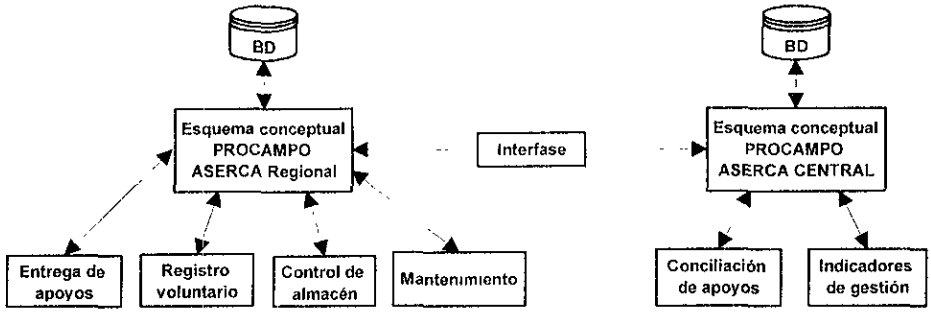


figura 3.12 Arquitectura global de los subsistemas de ASERCA.

La arquitectura de los esquemas de la base de datos de ASERCA se muestran en la figura 3.12. Los esquemas involucrados en la realización de la solución son el esquema externo de entrega de apoyos y el esquema externo de conciliación de pagos Por ello es necesario detallar mediante diagramas de clases estos esquemas.

El esquema conceptual externo de apoyos en ASERCA Regional consiste de nueve entidades las cuales se listan a continuación:

- Productor. Persona registrada como agricultor en cualquiera de los ciclos agrícolas y que puede ser beneficiada por sembrar alguno de los cultivos básicos.
- Solicitud de apoyo. Petición de apoyo económico, hecha por un productor a ASERCA durante el año agrícola para la siembra de alguno de los cultivos básicos.
- Apoyo. Monto derivado de una superficie sembrada con cultivos básicos correspondiente a una solicitud.

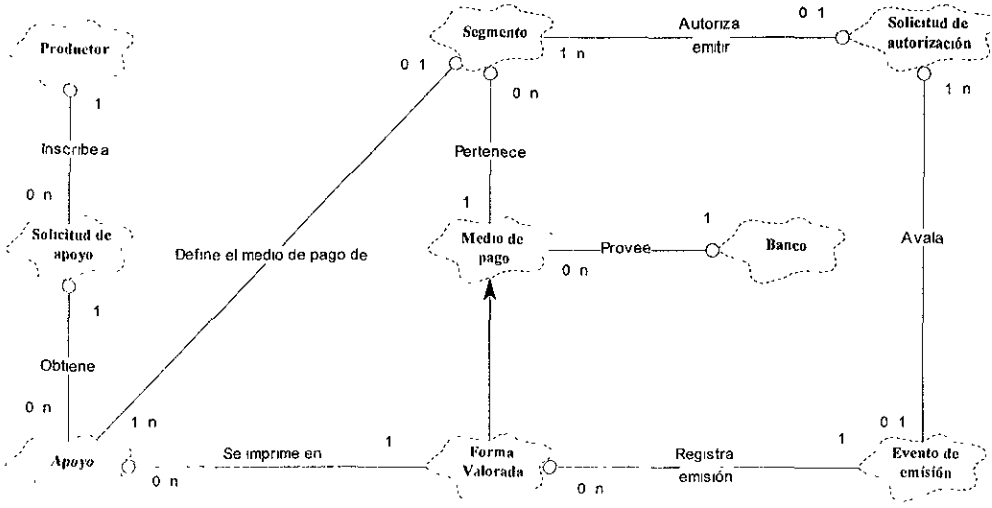


Figura 3.13 Diagrama de Clases del esquema de Entrega de Apoyos en el Regional.

- *Forma Valorada.* Documento de pago expedido por alguna institución bancaria que ingresa en la bóveda del regional. Es una generalización de los distintos documentos para pago como son cheques y giro telegráfico.
- Solicitud de autorización. Petición del Regional a ASERCA Central para emitir apoyos.
- *Evento de emisión. Registro de control de los cheques impresos,* por medio del cual se determina cuando, quien y cuanto se imprimió
- Segmento. Apoyos agrupados de acuerdo a la superficie a apoyar y la cartografía, asignado a un medio de pago.
- Medio de pago. *Es una generalización de las formas en que se puede pagar el apoyo de* PROCAMPO. Estas formas incluyen la expedición de formas valoradas (caso particular son cheques) y depósitos. Actualmente se opera con expedición de cheques.
- Banco. Institución financiera con la que se ha establecido un contrato de apertura de cuenta para la emisión y cobro de documentos de pago (por ejemplo cheques).

Las relaciones existentes entre las tablas del esquema de entrega de apoyos se muestra en la Figura 3.13. El símbolo "○" al final de las líneas que conectan las clases en el diagrama indica que la relación debe leerse partiendo de la clase a la que esta próximo. Así en la figura se lee un productor puede inscribir ninguna o varias solicitudes y una solicitud corresponde a un productor. Una solicitud puede obtener ninguno o varios apoyos. Un apoyo sólo corresponde a una solicitud además un apoyo no puede existir sin solicitud de apoyo. Los apoyos deben asignarse a un medio de pago que puede ser cheque de TESOFE, cheque de BANCOMER o cheque de BANAMEX. Para ello se agrupan en segmentos que indican el medio de pago. Posteriormente se pide autorización a ASERCA Central para emitir los apoyos mediante una solicitud de autorización. ASERCA Central solicita fondos para el monto de las solicitudes de autorización y libera la impresión de los apoyos. Los apoyos se imprimen

en formas valoradas y se registra un evento de emisión. Este evento es la suma de los montos de una o varias solicitudes de autorización liberadas.

El esquema conceptual de la Base de Datos en ASERCA Central se compone de 6 entidades que se lista a continuación:

- Banco. Institución bancaria con la que se ha establecido un contrato de apertura de cuenta para la emisión y cobro de documentos de pago (i. e. cheques).
- Cinta de conciliación. Cinta que ha sido proporcionada por CECOBAN y contiene la información de los cheques cobrados o cancelados de un banco en particular.

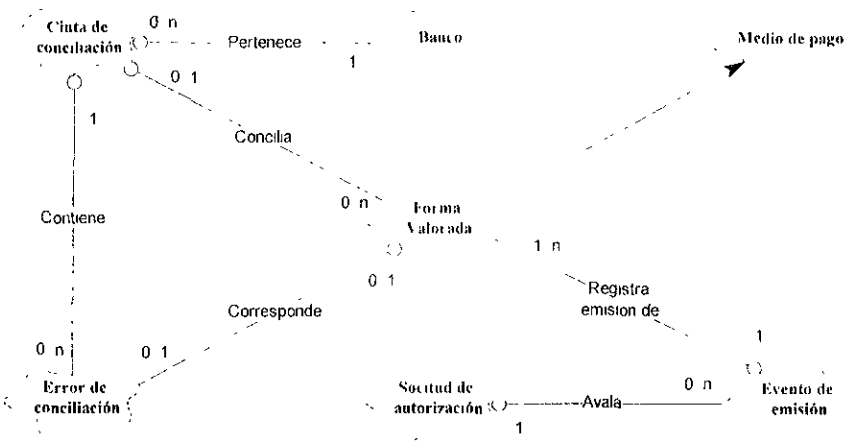


Figura 3.14 Diagrama de clases para el esquema conceptual de ASERCA Central

- Forma Valorada. Documento de pago foliado utilizado para impresión de los apoyos en alguno de los ciclos
- Error de conciliación. Información de alguno de los cheques cancelados o cobrados que no coincide con la información de la Base de Datos de PROCAMPO.
- Solicitud de Autorización. Petición del Regional a ASERCA Central para emitir apoyos durante el año agrícola.
- Evento de emisión. Información del monto y documentos de pago emitidos durante el año agrícola.

Las relaciones entre las entidades del esquema de ASERCA Central se muestran en la Figura 3.14. *Aquí se observa que las cintas entregadas por* CECOBAN a ASERCA contienen información de un banco a la vez. Además cada cinta puede conciliar varias formas valoradas o puede contener errores de conciliación. Las formas valoradas son una especialización de los medios de pago y son impresas en un evento que esta avalado por una o varias solicitudes de autorización lo cual además implica que el total en pagos e importe del evento de impresión debe coincidir con la suma de pagos e importe de las solicitudes de autorización.

Los esquemas antes presentados son esquemas de como está modelado actualmente la operación de la emisión de apoyos directos, sin embargo el objetivo del presente diseño es crear la interfaz para

compartir la información de los regionales con ASERCA Central y aplicable en general en sistemas realizados sobre AIX con RDBMS ORACLE con información distribuida en distintos nodos. Los esquemas Regional y Central muestran las diferencias siguientes:

- Las entidades solicitud de apoyo, productor, segmento, apoyo no son necesarias para las funciones realizadas en ASERCA Central.
- Las entidades cinta de conciliación y error de conciliación son innecesarias en ASERCA Regional.

Por otro lado ASERCA Regional y ASERCA Central deben compartir la información de las entidades siguientes:

- Forma valorada.
- Solicitud de autorización.
- Evento de emisión.
- Banco.

Estas tres últimas son las entidades de interés para el desarrollo del esquema de réplica, bajo el cual se compartirá la información.

3.2.3 Mecanismo de réplica

Las entidades Forma Valorada, Solicitud de autorización y Evento de emisión están fragmentadas horizontalmente en cada una de las regionales. Es decir, en cada uno de los regionales las entidades mantienen los mismos atributos pero contienen un subconjunto de las tuplas que conforman la entidad a nivel nacional. Por otro lado, la entidad Banco aun cuando está compartida no está fragmentada ya que se trata de un catálogo que se actualiza a nivel central con copias en los regionales.

Cada entidad consiste de 17 fragmentos, uno por cada nodo regional de ASERCA, que posteriormente el mecanismo de réplica deberá integrar en el nodo central.

Los fragmentos fungen en una primera instancia como información maestra u original, por otro lado las copias en el nodo central conformarán la información réplica. La cual se mantendrá actualizada aplicando las mismas transacciones aplicadas en los fragmentos maestros. Una transacción consiste de una o varias operaciones básicas para la manipulación de datos, las cuales son:

- Inserción de tuplas.
- Actualización de tuplas.
- Borrado de tuplas.

Para precisar los conceptos así como simplificar la exposición del mecanismo de réplica se desarrollarán continuación las definiciones de los componentes y sus reglas bajo el contexto establecido.

Un fragmento de información esta determinado por la entidad de la cual forma parte y el nodo en el que se ubica. Expresado de manera compacta un fragmento se denota como $F_{e,n}$ donde e es la entidad compartida y n el nodo de ubicación del fragmento.

En particular para la solución a desarrollarse, los dominios de los valores de los parámetros *e* y *n* se definen como siguen:

$e \in \{Forma\ valorada,\ Solicitud\ de\ autorización,\ Evento\ de\ emisión\}$

$n \in \{Hermosillo,\ Chihuahua,\ Lerdo,\ Reynosa,\ Culiacán,\ Zacatecas,\ Guadalajara,\ Celaya,\ Morelia,\ Toluca,\ Xalapa,\ Puebla,\ Chilpancingo,\ Oaxaca,\ Tuxtla,\ Mérida,\ rs980\}$

En particular el fragmento de la entidad Forma Valorada en el nodo Hermosillo se expresa como: $F_{Formas\ valoradas,\ Hermosillo}$

Un fragmento cambia a través del tiempo debido a las operaciones diarias, por fines prácticos se definen fechas u horas de corte en las que se obtiene un resumen que describen al fragmento en ese instante. Para diferentes cortes un mismo fragmento presenta diferentes datos y número de tuplas, el conjunto concreto de tuplas en un instante de corte dado define una versión del fragmento, consecuentemente un fragmento tiene tantas versiones como cortes se realicen.

Definición. Sea T el periodo de tiempo que transcurre entre cada corte y $F_{e,n}$ un fragmento de una entidad, entonces una versión del fragmento se expresa como $V_{F_{e,n},k}$ donde k es un entero que indica el número de corte o momento en que se obtuvo la "foto" el fragmento.

El conjunto de operaciones aplicadas durante un periodo de tiempo T comprendido entre un corte k y $k+1$ para la versión del fragmento $V_{F_{e,n},k+1}$ se designa como $O_{F_{e,n},k+1}$ que cumple la igualdad siguiente:

$$V_{F_{e,n},k+1} = O_{F_{e,n},k+1}(V_{F_{e,n},k}) \quad \text{.....} \quad (3.1)$$

En la ecuación (3.1) se define una versión como el resultado de aplicar las operaciones del periodo correspondiente a la versión inmediata anterior.

Hasta este punto se han definido las dos partes principales de los fragmentos:

- La versión que nos determina los datos particulares del fragmento en un momento dado
- Las operaciones que ocurren en un periodo para el fragmento.

Una vez definidos los componentes de un fragmento se pueden establecer las relaciones que existen entre estos.

Definición: Sea $R_{e,a}$ un fragmento de la entidad e ubicado en el nodo a con p versiones $V_{R_{e,a},i}$ donde $i \leq p$. Y sea $M_{e,b}$ otro fragmento de la misma entidad e ubicado en el nodo b con q versiones $V_{M_{e,b},j}$ donde $j \leq q$. Donde $0 \leq p \leq q$ y $a \neq b$ entonces el fragmento $R_{e,a}$ es réplica de $M_{e,b}$ si y sólo si se cumple la igualdad:

$$V_{R_{e,a},k} = V_{M_{e,b},k} \text{ para toda } k \text{ donde } 0 \leq k \leq p \quad \text{.....} \quad (3.2)$$

Es decir un fragmento es réplica de otro denominado maestro ubicado en un nodo distinto si cada una de las versiones de la réplica son idénticas a las versiones del maestro hasta la versión actual de la réplica.

Además sea $O_{M_{e,b},k}$ el conjunto de operaciones aplicadas al fragmento $M_{e,b}$ para un momento k sustituyendo $V_{f_{e,b},k+1}$ por $V_{M_{e,b},k}$ en la ecuación (3.1)

$$V_{M_{e,b},k} = O_{M_{e,b},k}(V_{M_{e,b},k-1}) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (3.3)$$

Ahora sustituyendo ecuación (3.1) en (3.3) se obtiene la igualdad siguiente:

$$V_{R_{e,b},k} = O_{M_{e,b},k}(V_{R_{e,b},k-1}) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (3.4)$$

La ecuación (3.4) nos indica que para obtener una versión *k* del fragmento réplica es necesario aplicar las operaciones ocurridas entre el momento *k*-1 y el momento *k* a la versión inmediata anterior. Esta característica sustenta en gran parte el mecanismo de réplica en el cual el fragmento réplica se actualiza al aplicar las operaciones efectuadas en el fragmento maestro desde el momento en que se realizó la actualización anterior.

Otras características del mecanismo de réplica son:

- Cada fragmento (réplica o maestro) se ubica en un nodo solamente y pertenece a una entidad.
- Un fragmento maestro puede tener uno o más fragmentos réplica para casos donde la información debe copiarse en más de un nodo. En este caso se ubican los fragmentos de la entidad *Banco*, cuyo fragmento maestro es $M_{Banco,1980}$ con fragmentos réplicas $R_{Banco,Hermosillo}$, $R_{Banco,Chihuahua}$, ..., $R_{Banco,Hidalgo}$.
- A su vez un fragmento réplica puede contener la copia de uno o más fragmentos maestros como ocurre en la información integrada de una entidad fragmentada horizontalmente. En este caso se encuentra los fragmentos de la entidad *Forma valorada*, para la cual los fragmentos maestros son $M_{Forma\,valorada,Hermosillo}$, $M_{Forma\,valorada,Chihuahua}$, ..., $M_{Forma\,valorada,Hidalgo}$ los cuales conforman el fragmento réplica es $R_{Forma\,valorada,1980}$.

El diagrama de clases, figura 3.15, muestra los componentes y relaciones a nivel conceptual del mecanismo de réplica.

Para describir el comportamiento de los componentes del mecanismo de réplica considérese un fragmento de réplica $R_{e,a}$ de otro fragmento maestro $M_{e,b}$ ubicados en los nodos *a* y *b* respectivamente. En cada nodo existe un entorno operativo que se sirve de los fragmentos para llevar su información, además se establece en cada nodo un manejador de operaciones para llevar el control de las transacciones y su aplicación. El mecanismo de réplica se describe en dos etapas, a saber:

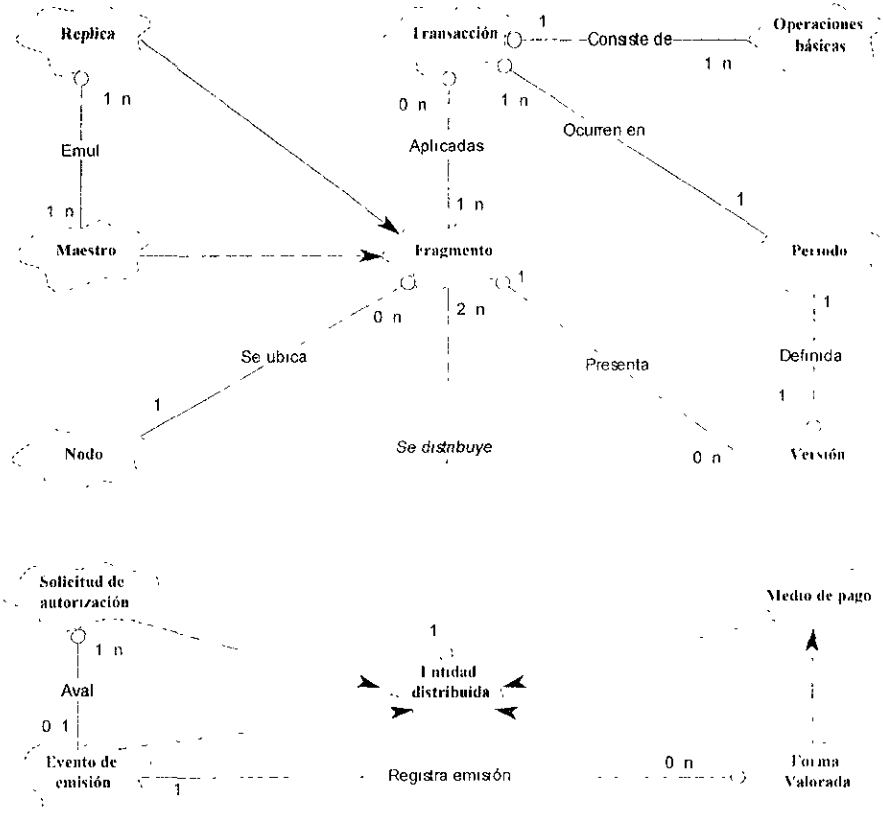


figura 3.15 Relaciones entre los componentes del mecanismo de réplica.

Etapa I) Inicialmente se opera sobre el fragmento maestro donde se inserta tuplas que posteriormente se actualizan o borran. Para crear la réplica de la información en esta etapa se efectúa el proceso siguiente:

1. Del entorno operativo del nodo *b* se generan operaciones dirigidas al fragmento maestro.
2. El fragmento maestro $M_{e,b}$ obtiene la operación y envía una petición al manejador de operaciones *b* para almacenarla antes de aplicar dicha operación en los datos del fragmento maestro.
3. El manejador de operaciones *b* envía una confirmación del almacenamiento de la operación al fragmento $M_{e,b}$.
4. El fragmento maestro evalúa la confirmación y si fue exitosa aplica la operación en las tuplas, de otra forma la cancela dicha operación.
5. El manejador de operaciones *b* mantiene las operaciones hasta alcanzar la hora de transferencia o una petición de transferencia del entorno operativo se presente.
6. Cuando el manejador de operaciones del nodo *b* recibe una petición de transferencia evalúa la línea de comunicaciones. De existir contacto con el manejador de operaciones del nodo *a* comprime y transmite las operaciones acumuladas desde la última transferencia. Si la línea de comunicaciones no permite el enlace con el nodo *a* se genera un archivo en cinta para ser

enviado por paquetería. Cabe notar que debido que la transferencia de las operaciones es asíncrona se representa en el diagrama mediante una flecha con media punta.

7. El manejador de operaciones del nodo *a* atiende las peticiones de aplicación de operaciones en el fragmento réplica, las cuales pueden provenir de la línea de comunicación o desde un archivo. Cuando las operaciones a aplicar se obtienen desde la línea de comunicaciones, el manejador aplica un algoritmo para expandir la información que comprimió el manejador de operaciones *b*.
8. El fragmento réplica $R_{e,a}$ recibe valida y aplica las operaciones que proceden desde el manejador de operaciones *a*. El resultado de la aplicación de cada operación se envía al manejador de operaciones *a*.
9. El manejador de operaciones *a* recibe y almacena las confirmaciones.
10. Cuando la hora de transferencia es alcanzada o se hace una petición explícita de la transferencia de confirmaciones, el manejador de operaciones *a* evalúa la línea de comunicación hacia el nodo *b*. Si existe enlace contacta el manejador de operaciones *b*, comprime las confirmaciones y las envía al manejador de operaciones *b*. Si no existe enlace se crea un archivo con confirmaciones que puede ser enviado a través de cinta o disco flexible.
11. El manejador de operaciones *b* recibe las confirmaciones de las operaciones aplicadas en el fragmento réplica a través de la línea de comunicaciones o desde un archivo. Las operaciones almacenadas originalmente son borradas por el manejador *b* siempre y cuando se reciba la confirmación de que se aplicó correctamente la operación correspondiente en el fragmento réplica de otro modo las operaciones se marcan con el error que indique la confirmación y tal operación se marca como pendiente de aplicarse.

El flujo de mensajes entre los objetos del proceso antes descrito se muestra en la figura 3.16.

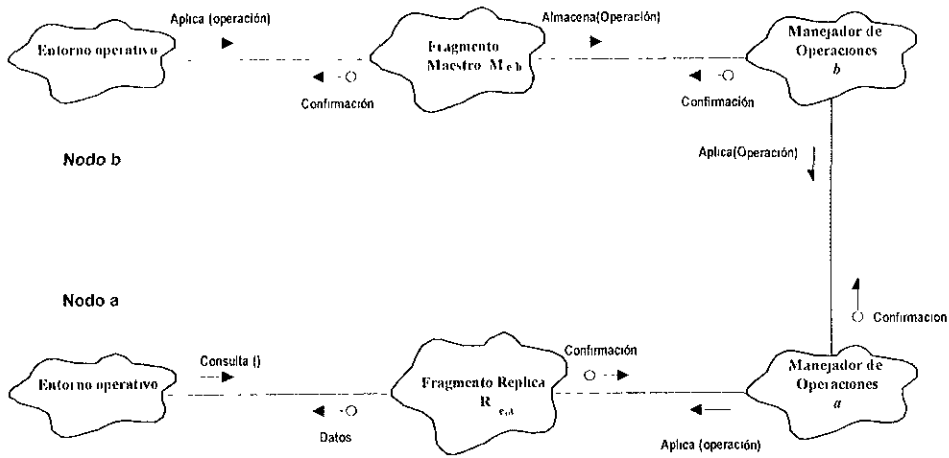


figura 3.16 Aplicación de operaciones desde el nodo con el fragmento maestro.

Etapa II) Se realizan operaciones en el nodo *a* sobre el fragmento réplica para lo cual se sigue los pasos siguientes:

1. Cuando las operaciones se efectúan desde el entorno operativo del nodo *a* sobre el fragmento réplica $R_{c,a}$, éste no afecta sus datos y envía una petición de almacenamiento al manejador de operaciones *a*.
2. El manejador de operaciones *a* se activa cuando se alcanza la hora de envío de operaciones. Revisa si existen operaciones a enviar. Si existen, primero verifica si puede enlazar al manejador de operaciones *b* a través de la red, si es posible esto envía las operaciones comprimidas. Si la red no está disponible genera un archivo que puede enviarse mediante cinta o disco flexible.
3. El manejador de operaciones *b* recibe las operaciones a aplicarse en el fragmento maestro $M_{c,b}$ desde la red o en un archivo. Se aplican las operaciones y se almacena la confirmación correspondiente.
4. El fragmento maestro recibe la operación del manejador *b*, valida, modifica sus datos y envía la confirmación al manejador *b*. Si la operación es exitosa entonces se envía además una petición de almacenamiento de la operación para aplicarse en el fragmento réplica $R_{c,a}$.
5. El manejador de operaciones *b* espera hasta la hora de transferencia o a una petición para enviar operaciones y confirmaciones al manejador *a*. El envío se realiza en línea o mediante archivo.
6. El manejador de operaciones *a* recibe las confirmaciones con base en las cuales elimina las operaciones aplicadas sin error y marca como pendientes las operaciones que no se efectuaron. Además recibe las operaciones y las aplica en $R_{c,a}$
7. El fragmento $R_{c,a}$ modifica los datos e indica el resultado de tal modificación como confirmación de aplicar la operación al manejador de operaciones *a*.
8. El manejador de operaciones *a* envía las confirmaciones al manejador de operaciones *b*.
9. El manejador de operaciones elimina las operaciones correspondientes a confirmaciones exitosas y marca como pendientes aquéllas que corresponde a confirmaciones con error.

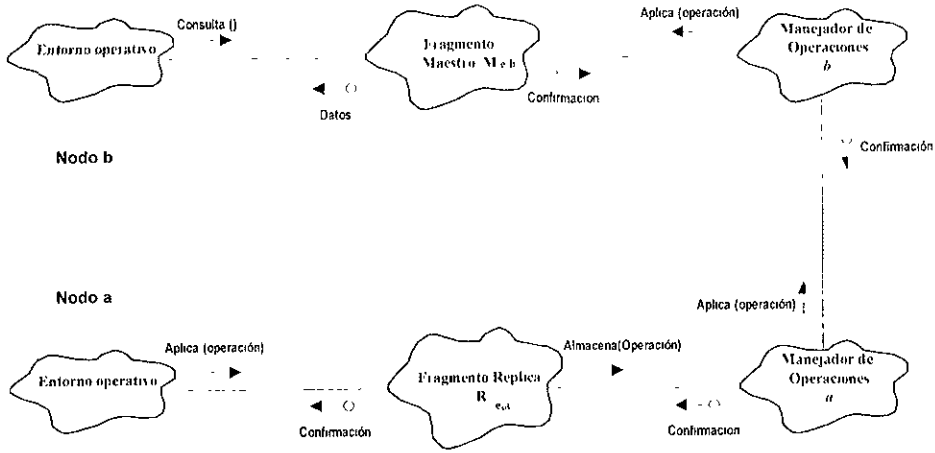
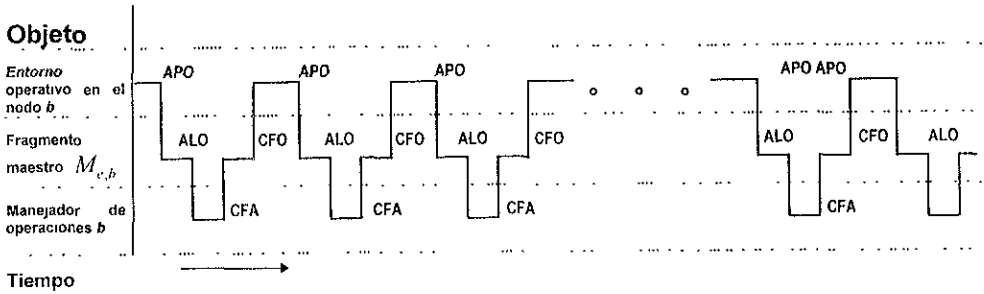


Figura 3.17 Aplicación de operaciones desde el nodo del fragmento réplica.

Los diagramas de objetos anteriores solo muestran como cooperan los componentes del esquema de réplica y no expresan la secuencia en que se presentan los mensajes, por ello estos diagramas se complementan con los diagramas de sucesos correspondientes.

El envío de operaciones aplicadas desde el nodo b sobre el fragmento maestro $M_{c,b}$ al fragmento réplica presenta dos diagramas de sucesos (ver Figura 3.18 y Figura 3.19) correspondientes a la aplicación de operaciones y a la transferencia de información, éstos se muestran en diagramas separados ya que son eventos asíncronos.



Mensajes

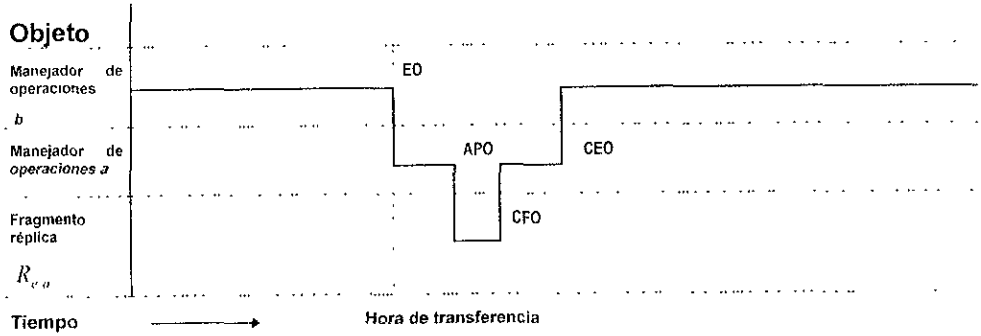
APO: Aplica operación.

ALO: Almacena operación

CFA: Confirma almacenado de operación

CFO: Confirma la realización de la operación

Figura 3.18 Diagrama de sucesos para la aplicación de una operación en el fragmento maestro.



Mensajes

EO . Envía operación

CFO: Confirma la realización de la operación

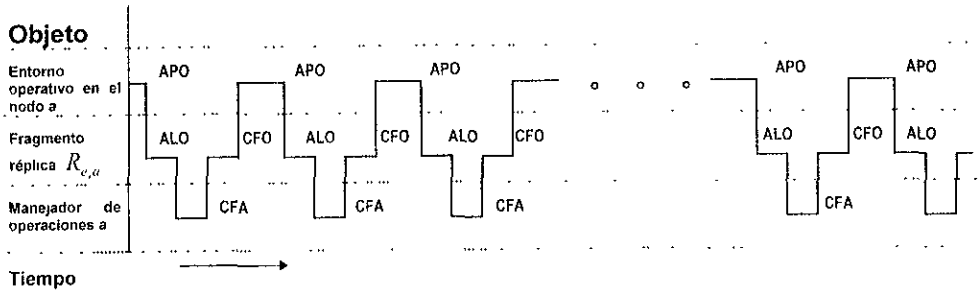
CEO Confirma envío de la operación

Figura 3.19 Diagrama de sucesos para la transferencia de operaciones del fragmento maestro al fragmento réplica.

La aplicación de operaciones originadas en el nodo a sobre el fragmento réplica $R_{c,a}$ presenta tres diagramas de sucesos:

- El primero corresponde a la aplicación de operaciones en $R_{e,a}$ que es similar a su homólogo del fragmento $M_{e,b}$, pero internamente las operaciones no modifican las tuplas del fragmento $R_{e,a}$, sólo envían la petición de almacenamiento de las operaciones .
- El segundo muestra la transferencia de las operaciones, las que al aplicarse en el fragmento $M_{e,b}$, también generan la operación de regreso al fragmento $R_{e,a}$.
- Finalmente el tercer diagrama muestra la transferencia de las operaciones de regreso con lo que finalmente se modifican las tuplas del fragmento $R_{e,a}$.

Una vez en operación el mecanismo, las operaciones de regreso y las operaciones originadas en el *nodo del fragmento maestro* se envían conjuntamente y ordenadas.



Mensajes

APO: Aplica operación

ALO: Almacena operación

CFA: Confirma almacenado de operación

CFO: Confirma la realizacion de la operación

Figura 3.20 Diagrama de sucesos para la aplicación de operaciones sobre el fragmento réplica.

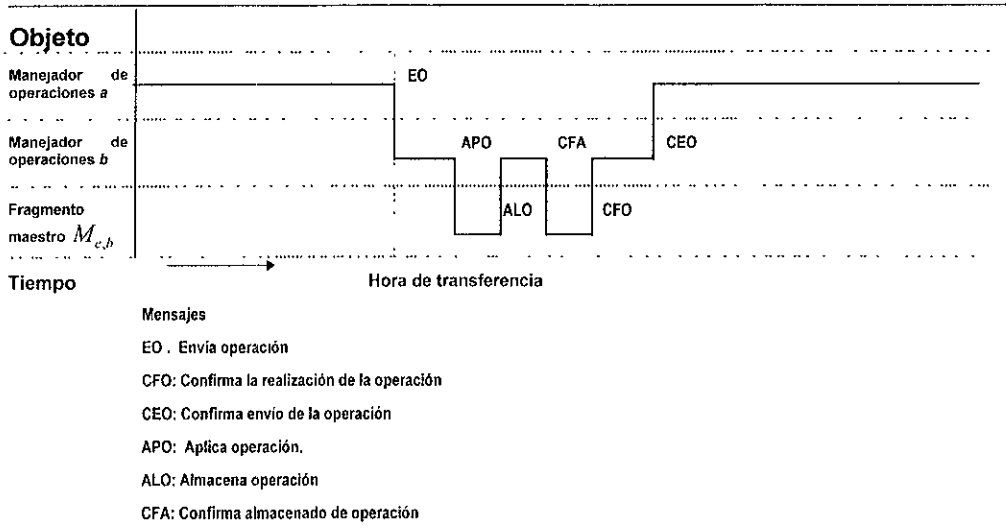


Figura 3.21 Diagrama de sucesos para la transferencia de operaciones originadas en nodo *b* sobre el fragmento réplica.

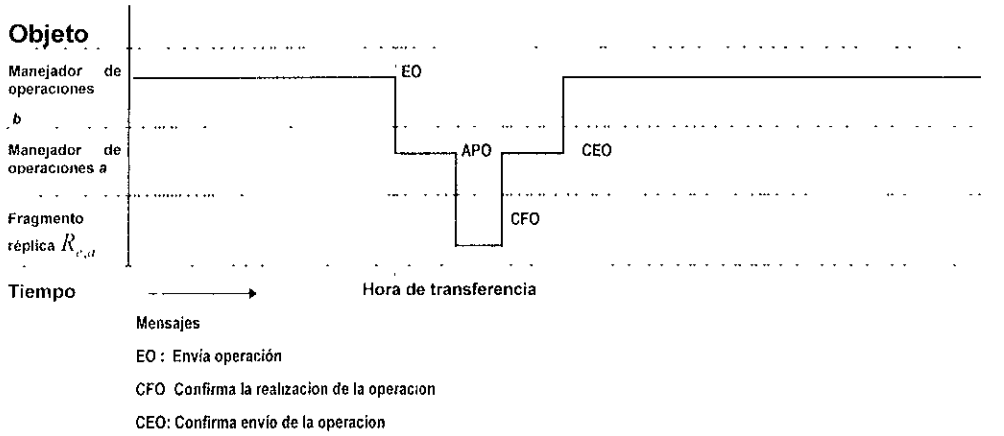


Figura 3.22 Diagrama de sucesos para la aplicación definitiva de las operaciones en el fragmento réplica.

3.3 Implantación

Una vez definido el diseño lógico y conceptual de la solución se procede con la transformación de estos conceptos en un diseño físico, así cada componente lógico se hace corresponder a uno o varios componentes físicos, tales como tablas, archivos y programas.

3.3.1 Componentes de la base de datos

De las clases determinadas en el diseño solo una parte se traducen en objetos del manejador de la base datos.

En el manejador de base de datos se pueden llevar a cabo tanto los componentes estáticos como dinámicos. Los componentes estáticos se realizan a través de tablas y sus restricciones de integridad, mientras la parte dinámica mediante paquetes de PL-SQL almacenados y *triggers*

3.3.1.1 Determinación de tablas

La clase *entidad distribuida* que representa las entidades con datos dispersos en diferentes nodos, físicamente es una lista de las tablas tal como se muestra a continuación:

Entidad distribuida: {Tabla}

Tabla
FORMAS_VALORADAS
SOLICITUDES_AUTORIZACION
TIPOS_EMISIÓN

Cabe notar que los campos que fungen como llaves primarias están subrayados.

La clase *Nodo* representa cada equipo servidor con los sistemas de información operando entorno a una instancia de base de datos. A cada uno de los cuales se denomina centro operativo, por esto se considera como más acertado denominar a la tabla física como *Centros* antes que *Nodos*. Así la definición resulta ser:

$$Centros \quad \{\underline{Centro}, Descripción, DB\_Link\}$$

Donde *DB_Link* contiene la cadena de conexión a la instancia del centro.

Las clases *Fragmento réplica* y *Fragmento maestro* son especializaciones de la clase fragmento. O en sentido inverso *fragmento* es una generalización de *réplica* y *maestro* y por ello debe cumplirse que:

$$Atributos(Fragmento) = Atributos(Fragmento\ replica) \cap Atributos(Fragmento\ maestro)$$

De esta manera las definiciones de las especializaciones son:
Fragmento réplica {*Nodo* , *Entidad*}
Fragmento maestro {*Nodo* , *Entidad*}

Entonces los atributos de la clase *Fragmento* son *Entidad*, *Nodo* Sin embargo Entidad corresponde físicamente a nombres de un centro, y *Nodo* al nombre de centros operativos de ASERCA, por ello con el fin de utilizar nombres de atributos de un nivel menos abstracto se sustituyen por términos más

familiares como Tabla y Centro. Además dado que el objetivo es obtener tablas bajo Oracle es importante considerar el espacio que ocupan. Cada tabla creada en Oracle ocupa dos bloques de espacio como mínimo, es decir 8 Kilobytes sin importar se ocupe o no. Por ello es conveniente reunir las tablas *réplica* y *maestro* en la generalización *Fragmento* distinguiendo las tuplas del *Fragmento réplica* de las tuplas del *Fragmento maestro* a través de una columna denominada Tipo. Así se obtiene la tabla siguiente:

Fragmento: {Centro, Tabla, Tipo}

Centro	Tabla	Tipo
HERMOSILLO	FORMAS_VALORADAS	MAESTRO
CHIHUAHUA	FORMAS_VALORADAS	MAESTRO
LERDO	FORMAS_VALORADAS	MAESTRO
REYNOSA	FORMAS_VALORADAS	MAESTRO
CULIACAN	FORMAS_VALORADAS	MAESTRO
RS980	FORMAS_VALORADAS	RÉPLICA

Para la clase *Transacción* se identifican sus campos como sigue:

Transacción: {Movimiento, Creación}

Número de transacción	Fecha de creación
1	07-ene-1997
2	07-ene-1997
3	07-ene-1997

En *Transacción* el campo número de transacción indica el orden en que se realizaron los cambios, la fecha de creación indica el día cuando se originaron las sentencias.

Cada tupla de la tabla *Transacción* consiste de una o varias tuplas de la clase *Operaciones básicas* cuya tabla es:

Operación básica: {Serie, Sentencia, Error}

Serie	Sentencia	Error
1	INSERT INTO FORMAS_VALORADAS (FOLIO, TIPO_FORMA, FECHA_STATUS, EVENTO) VALUES (10082400, 3, 'CEA', '07-01-1997', 1)	
2	INSERT INTO FORMAS_VALORADAS (FOLIO, TIPO_FORMA, FECHA_STATUS, EVENTO) VALUES (10082401, 3, 'CEA', '07-01-1997', 1)	

El campo Serie indica el orden en que se aplicó la sentencia de la transacción, además identifica una operación de otra. El campo *Sentencia* corresponde al texto de la operación de manipulación de datos y el campo Error indica el error que el RDBMS envía cuando no se puede aplicar la operación

La primera forma normal del modelado relacional dicta que cada campo debe ser atómico y no una composición de varios. En el caso del campo sentencia se puede observar que la sentencias se pueden dividir en cuatro partes:

- El tipo de operación de manipulación de tuplas (INSERT, UPDATE o DELETE).
- Los datos, que son los valores que entran o se actualizan en la condición.
- La condición de búsqueda para la modificación de los datos.
- La tabla a la que afecta.

Lo cual resulta en la definición siguiente para *Operación básica*:

Operación básica: {Serie, Tipo_Operación, Datos, Condición, Error, Tabla}

En los centros operativos de ASERCA se manejan el concepto de eventos para identificar un grupo de cambios en la información durante un intervalo de tiempo, así se retoma este concepto para el establecimiento físico de la clase *Periodo* en la tabla *Eventos*, ésta se describe a continuación:

Evento {*Evento, Fecha_Inicio, Fecha_Fin, Registros*}

Donde *Evento* es un número que identifica el período, *Fecha_Inicio* y *Fecha_Fin* delimitan temporalmente cuando ocurre el evento y *Registros* indica el número de registros sobre los cuales se aplicaron las operaciones.

Las definiciones de los campos hasta este punto no involucran las relaciones existentes entre las clases. Del diagrama de clases de la figura 3.15 se obtienen las relaciones siguientes:

Relación	Cardinalidad	Campos
Consiste de: Transacción→Operación básica	1:1..n	R:{Movimiento, serie}
Se distribuye: Entidad distribuida→Nodo	0..n:2..n	R:{Tabla, Centro}
Aplicadas sobre: Transacción→Fragmento	0..n:0..n	R:{Movimiento, Centro, Tabla}
Se ubica en: Fragmento→Nodo	0..n:1	R:{Centro, Tabla }
Ocurre en: Transacción→Período	1..n:1	R:{Movimiento, Evento}
Definida por: Versión→Período	1:1	R:{Evento, Versión}
Emula a: Réplica→Maestro	1..n:1	R:{{Centro, Tabla}$_{Replica}$, {Centro, Tabla}$_{Maestro}$}

El problema a resolver en este punto es ubicar los campos de las relaciones en las tablas correspondientes. Por ello a continuación se procede para cada relación:

- Para la relación entre *Transacción* y *Operación básica* se determina por la cardinalidad que los campos de la relación deben almacenarse en *Operación básica*, ya que para cada tupla existente en *Operación básica* si y sólo si existe una tupla en *Transacción*. Por esto último se redefine la llave de la tabla *Operación básica*, y la tabla queda del forma siguiente:

  Operación básica {*Movimiento, Serie, Clave_Mov, Datos, Condición, Error*}

- La relación *Se distribuye en: Entidad distribuida→Nodo* es de cardinalidad muchos a muchos, lo que impide el traslado de alguna de las llaves primarias a una de las tablas. Para realizar físicamente esta relación se identifican dos alternativas:

  1) La creación de una tabla intermedia que establezca la relación.

  2) A través de un trigger de base de datos asociado a las tablas de los fragmentos réplicas y maestras de la entidad distribuida donde se tenga la lista de nodos con los que se relaciona.

  Se elige la alternativa 2 debido a que en ésta no es necesaria la creación de la tabla de relación, tabla que sería pequeña y por ello no justifica su creación como tal. Por otro lado para el caso de tablas que contienen réplicas de varios fragmentos, se da una relación derivada de muchas tuplas de la

tabla a una tupla de la tabla *Centros,* la cual se lleva a cabo trasladando la llave primaria del nodo a la tabla.

De esta manera para la entidad *Forma valorada* en regionales tiene la definición de tabla siguiente:

Formas_Valoradas: {*Folio*, *Apoyo*, *Status*, *Evidencia*, *Fecha_Status*, *Ciclo*, *Tipo_Forma*, *Pago*, *Pasadas*, *Archivo*, *Beneficiado*, *Transmisión*, *Cinta*, *Regional*, *Fecha_Pago*, *Tipo_Apoyo*, *Evento*, *Estado*}

Y para la tabla que contiene las réplicas en el nodo central de la entidad *Forma valorada* se define como sigue:

Formas_Valoradas: {*Folio*, *Apoyo*, *Status*, *Evidencia*, *Fecha_Status*, *Ciclo*, *Tipo_Forma*, *Pago*, *Pasadas*, *Archivo*, *Beneficiado*, *Transmisión*, *Cinta*, *Regional*, *Fecha_Pago*, *Tipo_Apoyo*, *Evento*, *Estado*, *Regional*}

- En la relación *Aplicadas sobre: Transacción→Fragmento* la cardinalidad de la relación es de muchos a muchos, cuya realización física requiere de la creación de una tabla adicional, ésta crecería en un volumen directamente proporcional al número de transacciones a aplicar por el número de fragmentos involucrados. En el caso concreto de catálogos, de un volumen anual promedio de 246 tuplas por todos los catálogos de ASERCA, cada transacción se multiplica por 17, resultando en 4,182 transacciones anuales y para el caso de las tablas de operación se sabe que el volumen promedio anual de pagos de 3,878,596 los cuales crean transacciones al insertar, actualizar la entrega y al actualizar la confirmación de pago o cancelación del banco, resultando en 11,635,788 transacciones y por uno ya que casi siempre las transacciones se dirigen hacia ASERCA Central. Con base en los promedios anteriores en la tabla de operaciones existirían en promedio anualmente 11,636,034 tuplas en la tabla transacciones y en la tabla de relación 11,639,970 tuplas dando un factor 1.00033826, él que indica que en la práctica el número de tuplas en la relación es el mismo que la de transacción.

  Por lo cual se traslada la relación a la tabla *Transacción* quedando como:

  Transacción:{Movimiento, Fecha_Creación, Centro_Destino}

- Por la definición de la tabla *Fragmento* especificada en un principio ya incluyen los de la relación *Se ubica en: Fragmento→Nodo* con cardinalidad muchos a uno. Cabe aclarar que fragmento es un bloque concreto de tuplas que forma parte de la realización física de una entidad pero no es la entidad, y a diferencia del fragmento, la entidad puede ubicarse en diferentes nodos, un fragmento solo se ubica en un sólo nodo.

- La relación *Ocurre en: Transacción→Período* es de cardinalidad de muchos a uno lo que permite aplicar la regla de traslado de la llave primaria de la tabla de cardinalidad a uno a la tabla de cardinalidad a muchos. Así la tabla transacción se modifica como sigue:

  Transacción:{Movimiento, Fecha_Creación, Centro_Destino, Evento}

- Cuando existe una relación uno a uno entre dos clases puede tratarse en realidad de una sola clase percibida como dos, debido a diferentes puntos de vista. En este caso se encuentra la relación *Definida por: Versión→Período*, donde las clases participantes son una sola y físicamente corresponden a la tabla eventos definida como:

Evento. {Evento, Fecha_Inicio, Fecha_Fin, Registros}

- Las relaciones con cardinalidad muchos a muchos requieren de la creación de una nueva tabla donde se almacenen los campos de las mismas. Sin embargo para la relación *Emula a. Réplica→Maestro* el número de tuplas es aproximadamente 68 tuplas (una tupla por cada fragmento en cada nodo de las entidades *Formas valoradas, Solicitudes autorización, Eventos emisión* y *Medio de pago*) ocupando 2,156 si cada campo de la relación fuera una cadena de 32 caracteres que es un volumen pequeño para justificar la creación de una tabla adicional, por ello se deja el *trigger* que controla la aplicación de operaciones de esta tabla. Con esto último se separa el control de la dirección de las operaciones de cada fragmento en una cápsula.

El objetivo ahora es determinar los campos que finalmente constituirán cada tabla. Con tal propósito se hacen consideraciones de tipo práctico tal como el hecho de que las operaciones pueden ser enviadas a través de archivos con campos separados por comas y en particular en esta situación para confirmar la aplicación de la operación es necesario incluir el centro de origen en cada transacción. En operación es necesario determinar cuando una operación fue aplicada y en cuanto tiempo con el fin de disponer de estadísticas de tiempo que permitan prever el tiempo en contar con la información actualizada. Así que lo que implicaba un borrado en el modelo dinámico del mecanismo será sustituido por la presencia de un fecha de aplicación en el registro de operación y las operaciones se borrarán a través de un proceso en batch que corra diferido al momento en que se realiza la aplicación de las operaciones, con ello además se agiliza el proceso de transferencia de operaciones al ahorrar tiempo en el borrado de operaciones. Dado que la mayoría de las transacciones incluyen una operación se puede simplificar (aunque en contra de la normalización) y llevar a cabo ambas como una sola tabla operaciones y transacciones. Para esto último es necesario agregar un campo que distinga el conjunto de operaciones que forman la transacción. Así se unen las tablas *Operaciones* y *Transacciones* en la tabla *Movimientos* donde se elimina el campo *Serie* y la llave primaria se forma con el número de movimiento (campo *Movimiento*) y el centro de origen (campo *Centro_Origen*).

Por otro lado, es necesario catalogar los tipos de operaciones a aplicar además de fijar desde un inicio los tipos de transacciones posibles. Con esto último se permite distinguir conceptos concretos de la operación de ASERCA como de cancelación de cheques, autorización de emisión de pagos, o cualquier otro que se especifique Cada tipo de transacción se relaciona a un solo *Fragmento* y un fragmento tiene varios tipos de transacción así que la llave primaria de *Fragmento* se transporta a tipo de transacción. Como *Fragmento* por sí solo no justifica la creación de una tabla ya que gran parte del espacio físico de la tabla estaría desaprovechado (6 Kilobytes de los 8 que se requieren como mínimo para una tabla de Oracle). Por consecuencia conviene que *Fragmento* este implícito en la tabla de tipos de transacciones. Así se crea la tabla de tipos de movimientos por transacción tal como se muestra en seguida:

Tipos_movimietos_trans:{Clave_mov, Clave_asociada, Tipo_Mov, Tabla, Sentencia, Descripcion, Tipo_Tabla, Número_Campos}

Con base en la definición de *Tipos_Movimientos_Trans* y la relación existente con *Movimientos* se traslada la llave primaria de *Tipos_Movimientos_Trans* resultando la definición siguiente:

Movimientos: {Movimiento, Centro_Origen, Clave_Mov, Datos, Condición, Error, Fecha_Creación, Fecha_Aplicación, Centro_Destino, Evento, Mov_Padre, Total_Mov, Status}

Finalmente se reduce a tres tablas a crearse para el mecanismo de réplica:

- *Centros*
- *Tipos_Movimientos_Trans*
- *Movimientos*

Por otro lado, las tablas particulares al problema de conciliación de pago ya han sido creadas de acuerdo a un análisis y diseño del subsistema de entrega de apoyos de PROCAMPO y por ello no se abordan con el mismo detalle. Estas tablas son *Formas_Valoradas*, *Eventos_Expedición*, *Solicitudes_Autorización*, *Tipos_Formas* que corresponden a las clases *Forma valorada*, *Eventos de emisión*, *Solicitudes de Autorización*, y *Medio de Pago* respectivamente. El esquema planteado es suficientemente general para aplicarse a cualquier otra tabla o conjunto de tablas que participen en una transacción definida.

3.3.1.2 Determinación del espacio de almacenamiento

Para cada campo de las tablas resultantes se determina el tipo de datos y tamaño de los campos. En el caso del lenguaje de definición de datos de Oracle, los tipos son fijos y pueden ser: NUMBER, CHAR, VARCHAR, VARCHAR2 y LONG. Dependiendo de los valores posibles que cada campo puede tener se selecciona el tipo y tamaño. Así para el campo *Datos* se establece como VARCHAR2 de 2000 ya que este campo debe ser lo suficientemente amplio para almacenar cualesquiera de los campos que se actualizan en los fragmentos. De igual forma se define el campo *Sentencia* como VARCHAR2 de 2000 caracteres. Los campos que son referencias a la llave primaria de *Centros* como *Centro_Origen*, *Centro_Local* y *Regional* se definen como numérico de dos posiciones permitiendo hasta el registro de hasta cien centros, si posteriormente fuese necesario registrar más de cien centros, el manejador permite ampliar el campo después de creada la tabla.

En el código para crear la tabla *Centros* se especifica el tamaño y tipo de datos tal como se muestra a continuación:

```
CREATE TABLE CENTROS
(
     CENTRO                                NUMBER (2,0) NOT NULL,
     DESCRIPCION                           VARCHAR2 (50) NOT NULL,
     DB_LINK                               VARCHAR2 (32),
     CONSTRAINT CENTROS_PK PRIMARY KEY (CENTRO)
            USING INDEX TABLESPACE INDICES
            STORAGE (INITIAL 4K NEXT 1K PCTINCREASE 0)
)
PCTFREE 5
STORAGE (INITIAL 8K NEXT 4K PCTINCREASE 0) TABLESPACE TABLAS
/
```

Los tamaños y tipos de los campos se especifican en el código de definición de cada una de las tablas, de esta forma para la tabla *Tipos_Movimientos_Trans* se tiene lo siguiente:

```
CREATE TABLE TIPOS_MOVIMIENTOS_TRANS(
  CLAVE_MOV                    NUMBER (4,0) NOT NULL,
  CLAVE_ASOCIADA               NUMBER (4,0) NOT NULL,
  TIPO_MOV                     CHAR (1) NOT NULL,
  TABLA                        VARCHAR2 (40) NOT NULL,
  SENTENCIA                    VARCHAR2 (2000) NOT NULL,
  DESCRIPCION                  VARCHAR2 (100),
  TIPO_TABLA                   CHAR (1) NOT NULL,
  NUMERO_CAMPOS                NUMBER (2,0),
  CONSTRAINT TIPOSMOV_TRANS_PK PRIMARY KEY (CLAVE_MOV)
            USING INDEX TABLESPACE INDICES
            STORAGE (INITIAL 4K NEXT 1K PCTINCREASE 0)
)PCTFREE 5
STORAGE (INITIAL 8K NEXT 4K PCTINCREASE 0) TABLESPACE TABLAS
/
```

Debido a que el volumen de información es bajo para las tablas *Tipos_Movimientos_Trans* y *Centros* se define el espacio inicial mínimo de 2 bloques, el tamaño del bloque se define para cada instancia y en particular en las instancias en los centros operativos de ASERCA el tamaño de bloque es 4 Kilobytes por bloque. Se crea además la llave única que determina la identidad de cada tupla la que ocupa el espacio mínimo para índices que es de un bloque. En el caso de la tabla *Tipos_Movimientos_Trans* la llave única se determina por el campo clave_mov que es un número que identifica un tipo de transacción en el entorno operativo.

El código de definición de la tabla *Movimientos* cambia en sus parámetros de almacenamiento de acuerdo al volumen de operaciones manejado en cada centro regional, pero básicamente está dada por el código siguiente:

```
CREATE TABLE MOVIMIENTOS(
  CLAVE_MOV                              NUMBER (4,0) NOT NULL,
  MOVIMIENTO                             NUMBER (9,0) NOT NULL,
  MOV_PADRE                              NUMBER (9,0),
  TOTAL_MOV                              NUMBER (3,0) NOT NULL,
  DATOS                                  VARCHAR2 (2000),
  STATUS                                 VARCHAR2 (4) NOT NULL,
  ERROR                                  VARCHAR2 (30),
  FECHA_CREACION                         DATE NOT NULL,
  FECHA_APLICACION                       DATE,
  CENTRO_ORIGEN                          NUMBER (2,0) NOT NULL,
  CENTRO_DESTINO                         NUMBER (2,0) NOT NULL,
  EVENTO                                 NUMBER NOT NULL,
  CONDICION                              VARCHAR2 (600),
  CONSTRAINT MOVIMIENTOS_FK FOREIGN KEY (CLAVE_MOV)
            REFERENCES TIPOS_MOVIMIENTOS_TRANS (CLAVE_MOV),
  CONSTRAINT MOVIMIENTOS_PK PRIMARY KEY (MOVIMIENTO,CENTRO_ORIGEN)
            USING INDEX TABLESPACE INDICES
            STORAGE (INITIAL &INICIAL_PK NEXT &SIGUIENTE_PK PCTINCREASE 0)
)PCTFREE 5
STORAGE (INITIAL &INICIAL_TABLA NEXT &SIGUIENTE_TABLA PCTINCREASE 0 PCTFREE 5)
TABLESPACE TABLAS
/
```

En la definición de la tabla *Movimientos* se especifica la llave primaria formada por los campos *Movimiento* y *Centro Origen*, además se crea la restricción referencia hacia la tabla *Tipos_Movimientos_Trans* a través del campo *Clave_Mov*

La determinación de los parámetros de almacenamiento (Cláusula *storage* en la definición) de la tabla y el índice debe hacerse con base en el volumen de información esperado a almacenar en cada uno de los centros regionales.

Para los parámetros de almacenamiento de la tabla se aplica la fórmula[ix] siguiente:

$$Número\ de\ Bloques = X * \frac{5 + y * (1 + len)}{(TB - 90) * (1 - PCTFREE / 100)}$$

Donde X: Es el número de tuplas en la tabla. Este valor cambia de regional en regional.

y: Es el total de columnas de la tabla. En este caso es de 14.

len: Es la longitud de columna promedio. Para *Movimientos* es de 191 bytes por columna.

TB: El tamaño del bloque. En las instancias de ASERCA es de 4096 bytes.

PCTFREE: Es el porcentaje de espacio extra reservado en caso de actualizaciones a los campos de una tupla de la tabla. El valor de 5, en este caso se elige debido a que la actualización posterior a la inserción será solo en campos *Fecha_Aplicación, Error, Status* que serían el uno por ciento del tamaño del registro siempre y cuando se ocuparan todos los campos pero en la mayoría de las operaciones no se ocupa la totalidad de los campos y el porcentaje se incrementa así que se deja un margen.

Regional	Volumen anual promedio	Operaciones	Volumen mensual promedio	Número de bloques	Total Megabytes	INITIAL	NEXT
Hermosillo	53870	161610	13467.5	8,622	34	24	10
Chihuahua	85565	256695	21391.25	13,695	53	37	16
Lerdo	139815	419445	34953.75	22,378	87	61	26
Reynosa	134847	404541	33711.75	21,583	84	59	25
Culiacán	124195	372585	31048.75	19,878	78	54	23
Zacatecas	313264	939792	78316	50,140	196	137	59
Guadalajara	220831	662493	55207.75	35,346	138	97	41
Celaya	274450	823350	68612.5	43,928	172	120	51
Morelia	247575	742725	61893.75	39,626	155	108	46
Toluca	368401	1105203	92100.25	58,965	230	161	69
Xalapa	265156	795468	66289	42,440	166	116	50
Puebla	446820	1340460	111705	71,517	279	196	4
Chilpancingo	206938	620814	51734.5	33,122	129	91	9
Oaxaca	374409	1123227	93602.25	59,927	234	164	70
Tuxtla	271506	814518	67876.5	43,457	170	119	51
Mérida	129530	388590	32382.5	20,732	81	57	24
Hidalgo	221424	664272	55356	35,441	138	97	42
ASERCA CENTRAL	3878596	11635788	969649	620,799	2,425	1,697	727

tabla 10 Parámetros de almacenamiento para la tabla *Movimientos* en cada centro operativo.

Oracle organiza los datos almacenados en *extents*, cada uno de los cuales agrupa varios bloques en forma, cada grupo se identifica a través de un encabezado de control y el tamaño de cada grupo afecta el número de accesos de entrada/salida que debe hacer el manejador. Un número pequeño de *extents* con una definición de lectura de multibloque disminuye el número de accesos a disco. Para la creación de la tabla se debe definir el *extent* inicial en la cláusula INITIAL a un setenta por ciento del tamaño máximo esperado y el resto se deja al tamaño del siguiente *extent* de la cláusula NEXT. Con base en datos de las bitácoras y considerando que la tabla se depura cada mes, se estima un volumen de operaciones mensuales esperadas con base en las cuales se calculan los parámetros de almacenamiento, mismos que se muestran en la tabla 10.

Para la estimación de los parámetros de almacenamiento del índice se procede con los pasos siguientes:

1) Determinar el tamaño del encabezado de bloque.

   Para el caso de índices el tamaño del encabezado de bloque es de 159 bytes.

2) Determinar el espacio disponible en cada bloque para los datos.

   Espacio para datos = (*Tamaño del Bloque - Tamaño del encabezado*)

   (*Tamaño del Bloque - Tamaño del encabezado*)*(PCTFREE/100)

   = (4096-159) - (4096 - 159)*(5/100)

   =3937 bytes.

3) Calcular el tamaño promedio de las columnas combinadas que forman el índice.

   En este caso es una columna numérica sin decimales de cuatro posiciones.

4) Calcular el tamaño promedio de cada tupla en el índice.

   La tupla del índice esta formada además de las columnas incluidas en el índice, por la dirección física que asigna ORACLE, por byte de encabezado, almacena un byte de control por cada columna menor a 128 bytes y 3 bytes de control por cada columna mayor a 128 bytes.

   Tamaño promedio = encabezado de la tupla + longitud de la dirección física + longitud de las columnas combinadas + número de columnas menor a 128 bytes + 3(número de columnas mayores a 128 bytes)*

   Tamaño promedio = 1 + 6 + 4 + 1= 12 bytes

5) Calcular el número de bloque y bytes ocupados para el índice con base en la fórmula siguiente:

$$Bloques = 1.1 \frac{Tuplas\ con\ columnas\ no\ nulas}{\mathrm{Re}\ dondeo(espacio\ disponible\ para\ datos\ /\ Tamaño\ promedio\ de\ un\ tupla\ de\ indice)}$$

   El factor de 1.1 se debe a que se agrega un 10 por ciento de espacio adicional requerido para la creación de bloques extra generados por el índice.

   Con base en los resultados de los pasos anteriores y con el total de tuplas esperado para cada regional se obtiene la tabla 11.

Regional	Volumen anual promedio	Operaciones	Volumen mensual promedio	Número de bloques	Total Megabytes	INITIAL	NEXT
Hermosillo	53870	161610	13467.5	542	2.12	1.48	0.64
Chihuahua	85565	256695	21391.25	861	3.36	2.35	1.01
Lerdo	139815	419445	34953.75	1407	5.50	3.85	1.65
Reynosa	134847	404541	33711.75	1357	5.30	3.71	1.59
Culiacán	124195	372585	31048.75	1250	4.88	3.42	1.46
Zacatecas	313264	939792	78316	3152	12.31	8.62	3.69
Guadalajara	220831	662493	55207.75	2222	8.68	6.08	2.60
Celaya	274450	823350	68612.5	2761	10.79	7.55	3.24
Morelia	247575	742725	61893.75	2491	9.73	6.81	2.92
Toluca	368401	1105203	92100.25	3706	14.48	10.13	4.34
Xalapa	265156	795468	66289	2668	10.42	7.30	3.13
Puebla	446820	1340460	111705	4495	17.56	12.29	5.27
Chilpancingo	206938	620814	51734.5	2082	8.13	5.69	2.44
Oaxaca	374409	1123227	93602.25	3767	14.71	10.30	4.41
Tuxtla	271506	814518	67876.5	2732	10.67	7.47	3.20
Mérida	129530	388590	32382.5	1303	5.09	3.56	1.53
Hidalgo	221424	664272	55356	2228	8.70	6.09	2.61
ASERCA CENTRAL	3878596	11635788	969649	39022	152.43	106.70	45.73

tabla 11 Estimación de los parámetros de almacenamiento del índice asociado a la llave primaria de la tabla *Movimientos*.

Hasta este punto se ha determinado la estructura de base de datos básica para el esquema de réplica, adicionalmente se debe definir para cada aplicación específica que transacciones se someten al esquema de réplica y las reglas que han de seguirse para la actualización de los datos, dado que el esquema es simétrico se permite la actualización tanto en el nodo maestro como el nodo referencia y por ello podrían generarse conflictos en la modificación de las tuplas, para resolver dichos conflictos se especifican prioridades en tiempo, usuario o por el valor de los campos en las tuplas. Todas estas reglas se especifican en *triggers* de base de datos. Un *trigger* es un procedimiento almacenado en la base de datos, el cual es ejecutado automáticamente como resultado de una sentencia DML: INSERT, UPDATE o DELETE, no importando si el usuario utiliza sql*plus una aplicación en PRO*C, sql*forms, sql*report, etcétera. El uso de trigger permite simular métodos asociados a un objeto, en este caso por cada tabla se programan las reglas de operación respectivas y la modificación de los datos es sólo a través de las reglas ocultando con ello validaciones que por su complejidad no podrían ser implementadas con restricciones de base de datos relacional (referenciales y unicidad).

3.3.1.3 Realización de reglas de operación en el RDBMS

Para cada tabla se determinan sus reglas de operación y se programa su trigger correspondiente, a continuación se desarrolla el trigger para la tabla *Formas_Valoradas*.

Los pagos expedidos por ASERCA se realizan a nivel regional y la información de estos se registra en la tabla *Formas_Valoradas*. En ésta se registran cada una las formas que pueden tomar valor

económico. Estas formas pueden ser cheques de TESOFE, ordenes de pago, Fichas para Concentración Empresarial, cheques de Bancomer. Cada una de estas formas se clasifica mediante el campo *Tipo_Forma* de la tabla. Los valores posibles para este campo se muestran en la tabla 12.

Tipo de forma	Descripción
1	Depósito a cuenta
2	Orden de Pago.
3	Cheque de TESOFE
4	CEI BANCOMER
5	Cheques de BANCOMER

tabla 12 Valores posibles de medios de pago para el campo *Tipo_Forma*.

Las modificaciones de los valores posibles del campo *Tipo_Forma* ya se validan a través de una restricción referencial hacia la tabla *Tipos_Formas*, la cual representa a la clase *Medios de Pago*.

Los pagos o depósitos expedidos deben conciliarse contra la información proporcionada por el banco. Para efectuar esto la información debe conciliarse a nivel central por dos razones :

- Los bancos no tiene un manejo de regionales con correspondencia uno a uno con los regionales de ASERCA
- Los bancos siempre centralizan sus operaciones a través de CECOBAN.

Por lo cual la información de conciliación proporcionada por los bancos debe concentrarse en ASERCA central y después distribuirse en los regionales Para la distribución de la información de conciliación se presentan dos opciones:

- Con base en los datos proporcionados (folio e importe) buscar en cada nodo si existe el folio y entonces comparar importes. La búsqueda en cada nodo sería necesaria debido a la ausencia de referencia cartográfica de origen del cheque.
- Los registros de cheques expedidos (deben ser formas valoradas asociadas a un pago y no formas en blanco) en cada centro regional se envían primero a ASERCA discerniendo regional de origen. Al llegar la información de conciliación la búsqueda se realiza en las formas valoradas concentradas en ASERCA Central y con base en su regional de origen se construye el movimiento con la transacción de conciliación.

La elección de una de las opciones debe considerar aquélla con un mínimo de conexiones. En la primera opción existiría la posibilidad que a la primera conexión encontrará la forma valorada y la conciliará, sin embargo también existe el extremo de que al décimo séptimo intento encontrara la forma realizando con ello 16 conexiones adicionales, promediando 8 intentos de conexión. Mientras en la segunda solución se harían dos conexiones: la primera para traer la forma valorada y la segunda envía los datos conciliados. Con base en el criterio de selección la opción dos es la que se lleva a cabo.

Para identificar los registros a enviar y regresar, se comienza con la clasificación de las formas valoradas en dos grandes grupos: formas en blanco y formas impresas. Una forma entra por primera vez como forma en blanco y se caracteriza por tener los tres campos siguientes diferentes de nulo:

Folio *Tipo_Pago*

Fecha_Status *Status*

Por otro lado, las formas impresas se caracterizan por contar con información en los campos siguientes:

Folio	*Apoyo*
Status	*Evidencia*
Fecha_status	*Ciclo*
Pago	*Pasadas*
Archivo	*Beneficiado*
Transmisión	*Regional*
Fecha_Pago	

Dentro de las formas impresas existe dos grupos, a saber:

- **No Conciliadas.** Las que se caracterizan por tener el campo de *Cinta* en nulo.
- **Conciliadas**. Las que se caracterizan por tener información de conciliación en el campo de *Cinta*. A su vez las formas conciliadas se clasifican en dos grupos:
  - Ratificadas.
  - No Ratificadas.

Cada forma va entrando a cada grupo de acuerdo al punto en que se encuentra de su ciclo de procesamiento Para modelar como pasa a cada etapa de una forma se determinan los posibles estados que adquiere en transcurso de su procesamiento. Por cuestiones prácticas se asigna una abreviación de tres letras para cada estado tal como se muestra en la tabla 13.

Abreviación	Descripción de cada estado.
CEA	Entrada de formas al almacén.
CSI	Salida de formas del almacén al área de impresión.
CEI	Cheque entra al almacén impreso para pago.
PAG	El banco confirmó el cobro del monto de la forma.
CSA	Cheque sale del almacén para entregarse al productor.
CCA	La forma en blanco se inutiliza por daño físico sobre la misma.
CRG	El cheque se canceló para regenerar otro.
CAN	El banco confirmó la cancelación de la forma y ASERCA había ratificado la cancelación.
CSR	ASERCA no había cancelado explícitamente la forma pero el banco avisa que la forma fue cancelada.
FSA	La forma sale en blanco del almacén.

tabla 13 Estados que adquiere una forma valorada en su procesamiento.

Cada estado que adquiere la forma valorada está dentro de uno de los grupos generales mencionados anteriormente. Esta correspondencia se precisa en la tabla 14. Los grupos están definidos por que campos están actualizados en ese momento y estado nos indica en que etapa de procesamiento se encuentra la forma.

Grupo.		Estado de procesamiento.
En blanco		CEA, FSA, CSI, CCA
Impresas	No Conciliadas	CSA, CEI, CRG, CAN
	Conciliadas	CSR, PAG, CAN

tabla 14 Correspondencia entre grupo general y el estado general de procesamiento de un forma valorada.

Dado que la matriz solo indica el curso que sigue la forma ante un evento, se hace necesario complementarla con las reglas que restringen el cambio entre los diferentes estados.

Del proceso operativo específico al manejo de formas valoradas se sabe que el proceso es el siguiente:

1) Una forma se registra en blanco en el almacén (CEA).
2) Cuando se han calculado los apoyos se deben emitir los pagos así que las formas salen a impresión (CSI).
3) Por el manejo físico de las formas, algunas de ellas se dañan aún estando en blanco así que se inutilizan (CCE).

4) Cuando se imprime información a una forma valorada se convierte en un cheque que se almacena un tiempo (CEI) y por consiguiente no puede salir otra vez a impresión (CSI).

5) Debido a que la asignación del número de folio de forma al número de control de emisión del apoyo depende de lo registrado por el operador, además la impresión por su naturaleza mecánica esta sujeta a errores como desfasamientos, es necesario confirmar la asignación de formas a apoyos. Cuando se ha confirmado tal asignación se dice que el cheque esta ratificado (CRA).

6) Si se determina que un apoyo no debía emitirse, pero se imprimió el cheque correspondiente, éste es retenido por ASERCA quien cancela el cheque en cuestión y posteriormente se envía al Banco de México (CAN).

Estado Actual	**Estado Siguiente**									
	CEA	FSA	CCA	CSI	CEI	CRG	CSA	CSR	PAG	CAN
CEA		Salida	Invalida-ción	Salida Impresa						
FSA	Entrada Almacén									
CCA	Correc-ción			Correc-ción						
CSI	Devolu-ción		Invali-dación		Impreso					
CEI				Corre-cción		Cancela y reexpide	Salida	Banco Cancela	Banco Paga	Cancela Regional
CRG										Banco Cancela. Cancelación Manual
CSA						CancelaRedic-tamen		Banco Cancela	Banco Paga	Cancela Regional
CSR					Corre-cción					Cancela Redictamen
PAG					Corre-cción					
CAN					Corre-cción		Corre-cción			Banco Cancela

figura 3.23 Matriz de transición de estados para formas valoradas.

1) Banco de México confirma a ASERCA que recibió el cheque cancelado (CAN).

2) Sólo salen cheques ratificados para ser entregados al productor. Una vez entregado se registra que el cheque esta en tránsito (CSA).

3) Si un cheque tiene datos erróneos entonces se cancela por parte del regional para que se reexpida (CRG) otro con los datos correctos.

4) Los cheques que están en tránsito finalmente se confirman por el banco quien indica si se cobró (PAG) o se rechazó y canceló (CSR).

Este proceso se muestra en forma sintetizada en la matriz de transición de datos mostrada en la figura 3.23.

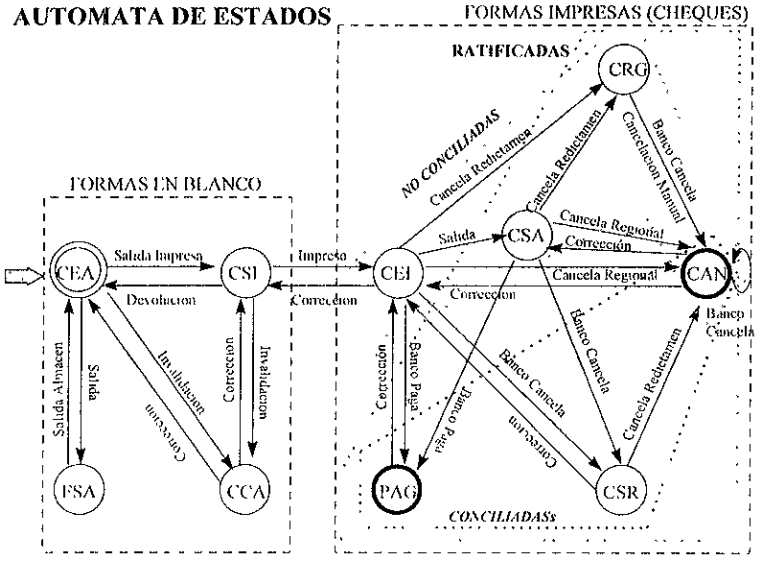


figura 3.24 Diagrama de estados para el procesamiento de formas valoradas.

Con base en la matriz de transición se determina el diagrama de estados mostrado en la figura 3.24 donde se muestra en forma precisa los cambios validos en el procesamiento de la formas valoradas En este diagrama además se muestra la correspondencia entre estados y grupos de formas.

La validación de la transición de estados y de la consistencia con el grupo de formas requiere de la programación de un *trigger* que se active ante cualquier operación de modificación de tuplas El código correspondiente se muestra en el Apéndice D

El *trigger* además de validar la reglas de operación, controla la generación de las operaciones y las almacena en la tabla *Movimientos*. No todos los cambios se envían a ASERCA Central ya que solo son de interés aquellas formas que son conciliables correspondientes al grupo de ratificadas

3.3.2 Módulos

El componente lógico denominado como *Manejador de operaciones* está constituido por dos partes principales: la primera conformada por los datos y la segunda por los métodos o funciones del manejador, ambas partes se presentan en la plantilla siguiente:

Nombre:	Manejador de operaciones.
Cardinalidad:	Una instancia por cada nodo.
Interfaz Pública:	
Métodos:	Consultar en otro nodo.
	Aplicar operaciones usando la red.
	Aplicar operaciones a través de archivos.
	Confirmar en otro medio distinto de la red.
Implementación:	
(Componentes privados)	
Datos:	Los campos del *Manejador de operaciones* están repartidos entre las tablas que forman el modelo de información o parte estática del objeto. Estas tablas son: *Movimientos* y *Tipos Movimientos Trans.*
Métodos:	Consultar operaciones.
	Aplicar operaciones.
	Confirmar.
	Compresión de datos.
	Recuperar ante error.

Dado que las herramientas no forman un entorno orientado a objetos no es directa la transformación de las clases. Así el método consultar del *Manejador de operaciones* se elabora en un programa separado del programa de compresión correspondiente a los métodos expandir y comprimir. Los métodos *aplicar operaciones*, *confirmar y recuperar ante error* se realizan en ejecutables separados. La ventaja de crear los ejecutables por separado es que una vez probado el programa del método ya no es necesario compilarlo otra vez cuando se está programando otro método. Los programas son coordinados mediante el programa de enlace a los nodos y la comunicación entre programas se efectúa a través de *pipes*, cuando los procesos se ubican en el mismo nodo de otra forma *sockets*.

Los métodos *aplicar operaciones desde archivo* y *aplicar confirmaciones desde archivo* se llevan cabo en un mismo programa. Similarmente los métodos *generar operaciones en archivo* y *generar confirmaciones en archivo* se realizan en el mismo programa.

A continuación se presenta el desarrollo de cada uno de los métodos correspondientes al *Manejador de operaciones*.

3.3.2.1 Módulo de consulta

El método de consulta de la clase *Manejador de operaciones* se desarrolla en un programa ejecutable separado, el cual tiene que acceder a los datos a través del RDBMS de Oracle.

Oracle proporciona cuatro métodos de acceso a través de sus precompiladores que pueden ser para COBOL, FORTRAN, Pascal y C. En este caso se utilizó el método 4 para el precompilador[xi] de C y se adecuó para su uso en C++. Primero se creó el código en Pro*C, posteriormente se precompiló, resultando un archivo fuente en C, el cual se fragmentó en métodos y se integraron a las clases correspondientes[xii].

Para obtener un mayor desempeño se disminuye el número de accesos al RDBMS[xiii], lo cual se consigue solicitando un número significativo de registros en cada acceso en lugar de un solo registro, para esto se crea un arreglo con un tamaño especificado por la variable de ambiente TAM_ARREGLO. El arreglo se lleva a cabo mediante un bloque de memoria obtenido dinámicamente y controlado a través de la clase *blk*, mismo que se presenta a continuación:

```
class blk {
    char  *base;              // Dirección base del bloque
    char  *sp;                // Cursor para recorrer el bloque
    int    lon;               // Longitud del bloque. Tamaño en bytes
    char   dec;               // Formato en decimales par conversión a reales
    int    dec_10;            // Diez elevado al numero de decimales
    double doble;             // Variable doble para cambio de tipo a doble
    float  flotante;          // Variable de punto flotante para conversiones
    int    entero;            // Variable entera para conversion
    char   buf[TAMANO_BUF]; // Variable entera para conversión
 public:
    blk(){base=NULL;sp=NULL;lon=0;dec=INI_DEC;dec_10=pow(10,INI_DEC); //constructor 1
          doble=0.0;flotante=0.0;entero=0;};
    blk(char *ibase,int ilon){base=ibase;sp=ibase;lon=ilon;// constructor 2
                              dec=INI_DEC;dec_10=pow(10,INI_DEC);};
    blk set(char *ibase, int ilon); // Inicializa el bloque
    blk set(char *ibase, int ilon, char decimales);
    char    deci(char idec); // Especifica el número de decimales en conversiones
    int     deci();          //Obtiene el numero de decimales
    blk     quita_char(char c);  //Elimina caracteres
    blk     operator<<(char *valor); //Escribe el contenido de valor en el bloque
    char *  operator>>(char *valor); //Escribe del bloque a valor como cadena de caracteres
    blk     operator<<(char c),      //Escribe el caracter c en el bloque
    char    operator>>(char& c);     //Escribe el primer caracter del bloque en c
    blk     operator<<(int),         //Escribe un entero en el bloque
    int     operator>>(int &),       //Escribe el valor entero del bloque en un entero
    blk     operator<<(float);       //Escribe el valor flotante en el bloque
    float   operator>>(float&);      //Escribe el contenido del bloque como flotante
    blk     operator<<(double);      //Escribe el valor doble en el bloque
    double  operator>>(double&),     //Escribe el contenido del bloque como doble
    char    operator==(char *str);   //Compara el contenido del bloque contra una cadena
    operator float();   //Ante un cast float convierte el bloque a su valor flotante
    operator double();  //Ante un cast double convierte el bloque a su valor doble
    operator int (),    //Ante un cast int convierte el bloque a su valor entero
    operator char *();  //Ante un cast char * convierte el bloque como cadena con terminador
    operator char();    //Ante un cast char regresa el valor ascii del primer byte del bloque
    friend   ostream& operator <<(ostream &salida, blk bloque), //Envía el bloque como salida
};
```

Dado que en la consulta dinámica se obtienen todos los datos como cadenas de caracteres, en los casos en que las columnas deban tratarse como valores numéricos flotante o entero, se debe realizar la conversión de tipos correspondiente tanto para asignar a alguna otra variable (operadores >>) como para recibir datos en el bloque (operadores <<). Estas conversiones no solo aplican para paso de valores con variables numéricas sino también a cadenas de caracteres terminadas por nulo.

Para obtener el resultado de una operación solicitada al RDBMS se requiere de una estructura donde el RDBMS informa al programa del resultado de su operación. La estructura proporcionada por Oracle *sqlca*[xiv] tiene ese propósito y se define como parte de la clase *cursor* descrita más adelante.

```
struct   sqlCA{
   char     sqlcaid[8];     /* Nombre de la estructura "SQLCA"*/
   int      sqlabc;         /* Longitud en bytes de la estructura sqlca */
   int      sqlcode;        /* Código de error de Oracle */
   struct{ /* Estructura de mensajes de error */
      unsigned short sqlerrml;  /* Longitud del mensaje de error */
      char           sqlerrmc[70];  /*Texto del mensaje de error */
    } sqlerrm;
   char     sqlerrp[8];     /* Reservada por Oracle para uso futuro */
   int      sqlerrd[6];     /* Contiene los  seis campos siguientes:
                               Elemento            Descripcion
                                  0:       Reservado por Oracle
                                  1:       Reservado por Oracle
                                  2:       Numero de registros procesados
                                  3.       Reservado por Oracle
                                  4:       Reservado por Oracle
                                  5:       Reservado por Oracle  */
   char     sqlwarn[8];    /* Contiene los ocho campos de advertencia siguientes:
                              Elemento
                                  0:       Se presenta una advertencia no determinada
                                  1:       Se truncó un campo al recibir datos
                                  2:       No tiene uso actualmente
                                  3:       El numero de columnas en una consulta es mayor al
                                           numero de variables de C que reciben los datos
                                  4:       Sentencia DELETE o UPDATE sin clausula WHERE
                                  5:       Reservada por Oracle
                                  6        Sin uso alguno
                                  7.       No se utiliza  */
   char     sqlext[8];     /* Reservada por Oracle  */
}/* struct sqlca */;
```

Para el manejo de datos de entrada o salida con el RDBMS, Oracle proporciona la estructura *sqlda*, misma que se muestra a continuación:

```
struct SQLDA {
   int     N;  /* Numero maximo de columnas */
   char  **V;  /* Arreglo de apuntadores a los bloques en memoria de cada columna  */
   int    *L,  /* Apunta al bloque que contiene las longitudes de cada columna */
        short  *T,  /* Apunta al bloque que contiene el tipo de datos de cada columna */
        short **I,  /* Arreglo de apuntadores al bloque de los indicadores de cada dato */
        int     F;  /* Numero de variables encontradas */
        char  **S;  /* Arreglo de apuntadores a los nombres de las columnas */
        short  *M;  /* Apunta al bloque de tamanos maximos de los nombres de las columnas */
        short  *C,  /* Apunta al bloque de tamanos de las columnas  */
        char  **X,  /* Apuntador al bloque de los nombres de la variables indicadoras */
        short  *Y,  /* Apuntador al bloque de tamaños maximos de las variables indicadoras  */
        short  *Z;  /* Apuntador al bloque de tamaños de las variables indicadoras */
};
typedef struct SQLDA SQLDA;
```

Mientras la estructura *sqlca* sólo recibe mensajes del RDBMS, la estructura *sqlexd* se utiliza para el envío de peticiones del programa al RDBMS. Estas peticiones son:

1) Realizar análisis sintáctico de la consulta (*Parse*).
2) Describir los campos de la consulta.
3) Abrir el cursor de la consulta.
4) Obtener tuplas (registros) de un cursor abierto (FETCH).
5) Cerrar el cursor de la consulta.
6) Ejecutar una sentencia de manipulación de tuplas (DML).

Estas peticiones podrían ser vistas como métodos definidos dentro de la estructura *sqlexd*, sin embargo como la biblioteca de funciones de Pro*C de Oracle ha sido creada considerando a *sqlexd* como formada solo por sus datos, se utiliza la definición de la estructura dada por la biblioteca del precompilador, tal como se muestra en seguida:

```
static struct sqlexd {
   unsigned long    sqlvsn;  //Usada internamente por Oracle
   unsigned short   arrsiz;  //Usada internamente por Oracle
   unsigned short   iters;   //Número de tuplas a procesar
   unsigned short   offset;  //Usada internamente por Oracle
   unsigned short   selerr;  //Usada internamente por Oracle
   unsigned short   sqlety;  //Usada internamente por Oracle
   unsigned short   unused;  //Usada internamente por Oracle
            short  *cud;     //Apunta a un arreglo de datos de valores predeterminado usados
                             //internamente por el RDBMS
   unsigned char   *sqlest;  //Apunta a la estructura sqlca donde se regresa el resultado de
                             //la operación.
            char   *stmt;    //Apunta al texto de la sentencia DML o de consulta
   unsigned char  **sqphsv; //Usada internamente por Oracle
   unsigned long   *sqphsl; //Usada internamente por Oracle
            short **sqpind; //Usada internamente por Oracle
   unsigned long   *sqparm; //Usada internamente por Oracle
   unsigned long  **sqparc; //Usada internamente por Oracle
   unsigned char   *sqhstv[1]; //Apunta a la estructura sqlda en caso de consulta
   unsigned long    sqhstl[1]; //Usada internamente por Oracle
            short  *sqindv[1]; //Usada internamente por Oracle
   unsigned long    sqharm[1]; //Usada internamente por Oracle
   unsigned long   *sqharc[1]; //Usada internamente por Oracle
} sqlstm = {4,1};
```

La estructura *sqlexd* se auxilia de las estructuras *sqlca* y *sqlda*, en la primera se resume el resultado de la petición indicando si hubo o no error, en tanto en la segunda se utiliza para especificar los datos de los registros de entrada o salida de la operación.

Dado que las funciones incluidas para acceder al RDBMS de Oracle fueron compiladas en C se debe indicar en el prototipo de cada función que son segmentos de código en C para lo cual se agrega el modificador *extern* "C" tal como se muestra a continuación:

```
extern "C" char  *sqlald(int max_vars,int max_name, int max_ind_name ); //Reserva memoria
                 // para la descripción de las columnas de la consulta
extern "C" sqlcxt( unsigned long *, struct sqlexd *, struct sqlcxp * ); //Ejecuta petición
extern "C" sqlcx2(/*_ unsigned long *, struct sqlexd *, struct sqlcxp * _*/);
extern "C" sqlcle(/*_ unsigned long *, struct sqlexd *, struct sqlcxp * _*/);
extern "C" sqlbuf(/*_ char *  _*/);
extern "C" sqlora(/*_ long *, void * _*/);
extern "C" sqlnul(short int *value_type, short int *type_code, int *null_status);
extern "C" sqlprc(long *length,int *precision,int *escale),  //Obtiene la precisión y la
                 // escala
extern "C" sqlclu(SQLDA *descriptor);   //Libera la memoria ocupada por el descriptor
```

Con base en la clase *blk*, las estructuras *sqlda* y *sqlca*, las funciones se crea una clase cuya función es la aplicación y control de las operaciones sobre las tuplas de una consulta realizada al RDBMS. Esta clase denominada *cursor* tiene como plantilla la siguiente:

```
class cursor{
    SQLDA *sd;           // Descriptor usado para las variables select
    char   *orabuff;     // Buffer para ejecución de sentencias
    blk    bcol;         // Control a los datos de los bloques
    int    row_fetched;  // Variable que recibe el código de Oracle al obtener los registros
    int    leidos,       // Total de registros proporcionados por Oracle.
    int    arraysize;    // Número máximo de registros almacenables en un bloque
    int    procesados;   // Acumulado de registros accesados a través del operador ++
    int    error_code;   // Código de error de la ultima operación realizada
    int    current_row;  // Número de renglón actual dentro del arreglo
    int    current_col,  // Número de columna accesada recientemente
    int    status;       // Estado de la consulta.
    struct sqlca sqlca;  // Estructura de interfaz con Oracle
 public:
    cursor();                  // Constructor de la clase, asigna valores iniciales
    int query(char *dinbuff);  // Envía la consulta a Oracle
    int oraexecim(char *buf);  // Ejecuta una sentencia DML
```

```
    int fetch();                // Obtiene n registros
    int found();                // Pregunta si hay más registros
    int close();                // Cierra la consulta
    int code();                 // Obtiene el código de error o éxito actual
    cursor& operator++(int);    // Operador ++ para avanzar registro por registro
    blk operator->*(char *columna); //Obtiene el contenido de la columna del registro actual
    char   *col(int i);         // Regresa el nombre de la i-ésima columna
    int    lon(int i);          // Regresa la longitud del valor de la i-ésima columna
    friend ostream& operator<<(ostream&, cursor&);// Envía al buffer de salida
                                                  // el registro actual
    int    lonC(int i);         // Regresa la longitud del nombre de la i-esima columna
    ~cursor() { close(); }      // Destructor
 private: // Métodos internos para realizar la consulta
    int  parse();               // Envía la consulta al PARSER de Oracle
    int  open();                // Solicita al RDBMS prepare los registros de la consulta
    int  is_open();             // Pregunta si el RDBMS tiene listos los registros
    int  allocdescriptors();    // Reserva memoria para el descriptor sd
    void allocselvars();        // Reserva memoria para los datos del select
    void cleanup();             // Libera el espacio de memoria reservado para el descriptor
    void descsel();             // Realiza el describe de las variables select
    void sqlcainit();           // Llena los valores iniciales de la interfaz con el RDBMS
    inline char *getadd(int col, int row) { return sd->V[col]+row*sd->L[col]; }
                                // Obtiene la dirección en memoria de acuerdo al número de
                                // columna col y el número de renglón row.
}; // clase cursor
```

No se incluye la estructura *sqlexd* debido a que las operaciones internas que realizan las funciones de Oracle requieren que se defina una variable estática fuera del bloque correspondiente al *main()* del programa.

El diagrama de clases presentado en la figura 3.23 muestra como están relacionados los elementos que permiten llevar a cabo la consulta y como a partir de la clase cursor se derivan las clases para consultas específicas como las consultas a las tablas *Movimientos*, *Tipos Movimientos_Trans* y *Formas_Valoradas*.

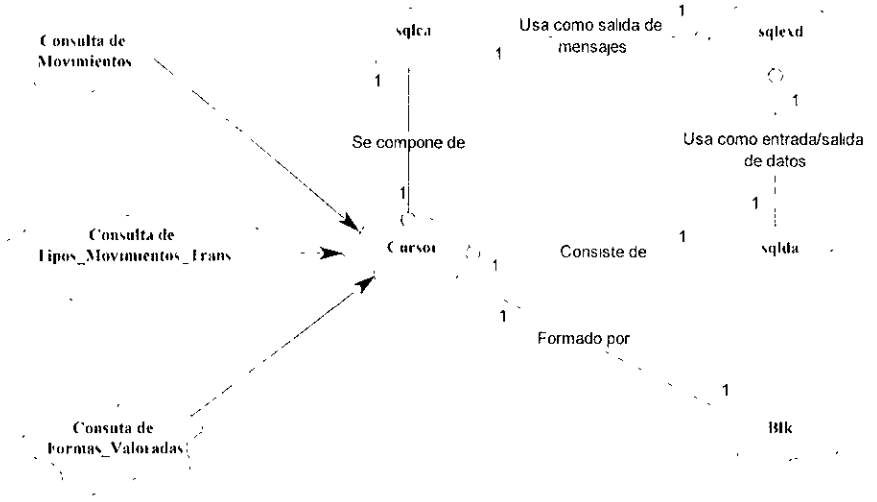


figura 3.25 Diagrama de clases del programa de consulta al RDBMS de Oracle

El código de definición de las clases se separa del código de sus métodos en el archivo de encabezado *desa.h* el cual se incluirá en cada programa que use la biblioteca de acceso al RDBMS. Los métodos principales de la clase cursor se describen en el Apéndice D.

La clase para manejo de bloques *blk* tienen como propósito hacer lo más transparente posible las conversiones a los tipos de datos básicos de C. Los códigos de los métodos correspondientes se muestran en el Apéndice D.

Al incluir la biblioteca de las clases *cursor* y *blk* se pueden crear programas directamente en C++ se evitando con ello el uso del precompilador de C++ proporcionado por Oracle. Un ejemplo que utiliza la biblioteca es el siguiente:

```
#include <desa.h>
class movimientos: cursor{
      strstream consulta;
   public:
      movimientos(){
        consulta<<   "SELECT * "
                     "FROM MOVIMIENTOS "
                     "WHERE STATUS='GEN'"<<'\0';
        *this.query(consulta.str()),
        *this++;
      }
}// class movimientos
void main(){
  if(conecta("/"){ // Abre una sesion con el RDBMS
   movimientos mov; // Crea el objeto de consulta a movimientos
   for(;mov.found;mov++) //Revisa si existen todavia registros y avanza al siguiente
     cout<<mov<<endl;     //Imprime a salida estándar las columnas del registro actual
  }
}//main
```

3.3.2.2 Módulo de compresión

Para la compresión de datos se utiliza el algoritmo LWZ (Lempel-Ziv-Welch). Se eligió este algoritmo por que al comprimir y expandir no existe perdidas en la información y por qué tiene una razón de compresión alta para texto con redundancia como es el caso de los registros de emisión de apoyos de ASERCA donde los folios de los cheques comienzan con una misma serie de números, presentan la misma fecha de pago, etc.

Para comprimir un bloque de datos se parte de una cadena S formada por el primer carácter del texto, esta cadena es el primer elemento de una tabla de códigos que se va creando de acuerdo al algoritmo presentado en apéndice A, donde se explica la compresión de datos.

3.3.2.3 Módulo servidor

El módulo servidor es un programa creado como demonio de Unix que está pendiente de las peticiones siguientes:

- Consultas remotas.
- Transmisión de Archivos.
- Aplicación de réplicas.

En el archivo *services* del directorio */etc* de Unix se agrega la línea que define el nombre de servicio *filetrans* el número de puerto 5000 y protocolo TCP para comunicarse con el módulo servidor:

```
filetrans  5000/tcp #Servidor: Transferencia de archivos, Consulta Remota y Efectua replicas
```

Para llevar a cabo el módulo servidor se utilizó la biblioteca de rutinas para programación en red de Berkeley[xv] las cuales se agregan con la línea para el precompilador de C siguiente:

```
#include <netdb.h>
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

El algoritmo que sigue el módulo servidor es el siguiente:

1. Al ejecutarse por primera vez se separa de la sesión del usuario y abre los descriptores de archivo para la salida estándar, entrada estándar y la salida de error.
2. Obtiene el número de puerto para el servicio *filetrans*.

```
struct servent *serv;
serv=getservbyname("filetrans",NULL); /* El segundo parametro se deja nulo por que solo hay una definición de filetrans */
```

3. Se obtiene un número de *socket* de tipo *stream* (SOCK_STREAM) perteneciente al familia de protocolos internet (AF_INET) y número de protocolo 0.

```
int sockfd,
sockfd=socket(AF_INET, SOCK_STREAM, 0);
```

4. Se obtiene la información del nodo local.

```
char host[256];
struct hosten* *hp;
gethostname(host, sizeof(host));      /* Obtiene nombre del host */
hp=gethostbyname(host);       /* Con base en el nombre del host recupera informacion */
```

5. Al socket obtenido en el punto 3 se asigna el número de puerto del servicio obtenido en el punto 2 y la información del nodo obtenida en el punto 4:

```
struct sockaddr_in saddr;      /* Estructura para recopilar puerto y datos del    host */
memset(saddr,0, sizeof(saddr));     /* Inicializa la estructura a  ceros */
saddr.sin_port= serv->s_port;      /* Asigna de puerto del servicio filetrans */
saddr.sin_family=AF_INET;        /* Familia de protocolos internet */
memcpy(&saddr.sin_addr, hp->h_addr, hp->length),   /* Asigna datos del host */
bind(sockfd, saddr, sizeof(saddr))  /* Se enlazan puerto, socket y datos del host */
```

6. Un proceso hijo no finaliza totalmente si su proceso padre no termina también, así que para evitar este comportamiento se hace ignorar al proceso si es hijo de otro proceso activo para terminar:

```
signal(SIGCHLD,SIG_IGN);
```

7. Se crea un bucle infinito que realiza lo siguiente:

   7.1 Se esperan las peticiones de conexión al programa.

```
int sockserv;     /* socket de servicio */
sockserv=accept(sockfd, saddr, sizeof(saddr));
```

   7.2 Si la conexión se establece entonces el proceso se divide en un proceso padre y un proceso hijo. El proceso padre sigue en espera de nuevas peticiones y el proceso hijo atiende la conexión. Esto se hace con el objeto de permitir múltiples conexiones a un mismo programa servidor.

8. Se recibe el tipo de servicio.

```
n=recv(sockfd, servicio,BUFSIZE, 0);
```

9. Se recibe la opción de procesamiento antes de enviar (Como compresión de datos).

```
n=recv(sockfd, servicio,BUFSIZE, 0);
```

10. Realiza el servicio solicitado, para lo cual se ejecuta otro programa con la entrada estándar direccionada a la salida estándar del servidor y la entrada estándar del servidor direccionada a la salida estándar del programa.
11. Mientras existan datos derivados del servicio solicitado por el programa cliente.

```
while(n_bytes=read(stdin, buf, SIZEBUF)
```

11.1 Si existe petición de compresión se comprime el bloque de datos.

11.2 Se envía el tamaño del bloque.

```
send(sockfd, &n_bytes, sizeof(long), 0)
```

11.3 Se envían los bytes del bloque.

```
send(sockfd, buf, n_bytes, 0);
```

11.4 Para cerrar la comunicación se envía una longitud cero para bloque.

```
n_bytes=0,

send(sockfd, &n_bytes, sizeof(long), 0)
```

13. Se guarda en bitácora el tiempo requerido para llevar a cabo el servicio.

3.3.2.4 Módulo cliente

Para solicitar cualquiera de los servicios que presta el programa servidor se ha de entablar la conexión al puerto correspondiente al servicio *filetrans*. En general los pasos a seguir son los siguientes:

1. Obtiene el número de puerto para el servicio *filetrans*.

```
struct servent *serv;
serv=getservbyname("filetrans",NULL); /* El segundo parametro se deja nulo por que solo  hay  una
definición de filetrans */
```

2. Se obtiene un número de *socket* de tipo *stream* (SOCK_STREAM) perteneciente a la familia de protocolos internet (AF_INET) y número de protocolo 0.

```
int sockfd;

sockfd= socket(AF_INET, SOCK_STREAM, 0);
```

3. Se obtiene la información del host remoto

```
char host[256];
struct hostent *hp;
gets("%s", host);       /* Obtiene nombre del host remoto */
hp=gethostbyname(host);       /* Con base en el nombre del host recupera informacion */
```

4. Se realiza la conexión al programa servidor:

```
struct sockaddr_in saddr,       /* Estructura para recopilar puerto y datos del     host */
memset(saddr,0, sizeof(saddr));      /* Inicializa la estructura a ceros */
saddr.sin_port= serv->s_port;       /* Asigna de puerto del servicio filetrans */
saddr.sin_family=AF_INET;         /* Familia de protocolos internet */
memcpy(&saddr sin_addr, hp->h_addr, hp->length);    /* Asigna datos del host */
connect(sockfd, saddr, sizeof(saddr))  /* Se enlazan puerto, socket y datos del host */
```

5. Se especifica el tipo de servicio solicitado al programa servidor:

```
char buf[BUFSIZE]
sprintf(buf,"%s", Servicio); /* Servicio: Transmision de archivos, Consulta remota,
                                Aplicar Réplicas */
send(sockfd, buf, strlen(buf)+1, 0);
```

6. Se indica si el resultado se procesa (en particular se comprime) antes de enviarse.

```
sprintf(buf,"%s", Preproceso); /* Preproceso: Operación sobre los datos antes de
                                  enviarse como compresion de datos */
send(sockfd, buf, strlen(buf)+1, 0);
```

7. Si la información se transmite comprimida se redirecciona la salida estándar al programa que expande la información.

8. Mientras se reciban datos.

```
long     n,          /* Bytes obtenidos por medio de la función recv() */
         bytes_t,    /* Total de bytes que va enviar el servidor */
         bytes_a;    /* Total de bytes acumulados despues de recibir bytes_t */
while(n=recv(sockfd, buf, BUFSIZE, 0))
```

8.1 Se obtiene el total de bytes a recibir, esto es necesario por que bloques de datos mayores a 256 kilobytes a través de un enlace X.25 llegan en más de una sola llamada a *recv*. Si el tamaño del siguiente bloque es cero se finaliza el proceso.

```
bytes_t=*(long*)buf
```

8.2 Mientras el número de bytes leídos sea inferior al número de bytes a recibir

```
while(bytes_a < bytes_t){
```

8.2.1 Recibe el próximo bloque de datos.

```
n=recv(sockid, buf, BUFSIZE,0);
```

8.2.2 Escribe a la salida estándar.

```
write(stdout, buf, n);
```

9. Se guarda en bitácora el tiempo requerido para atender la petición de servicio

3.3.2.5 Módulo de aplicación y confirmación de movimientos

La aplicación de los movimientos y confirmaciones operan básicamente de igual forma en ambos, la información se obtienen tanto desde un archivo proporcionado por el operador como a través de una petición remota al módulo servidor.

Para la aplicación de movimientos y de confirmaciones a través de archivo se realiza el proceso siguiente.

1. Se revisa que el encabezado corresponda a archivos generados con movimientos para el nodo donde se está corriendo la aplicación.
2. Mientras existan registros en el archivo.

   2.1 Si la sentencia actual pertenece a otra transacción se hacen permanentes los cambios.

   2.2 Si es aplicación de movimientos.

2.2.1 Se arma la sentencia y se ejecuta.

2.3 Si es confirmación de movimientos.

2.3.1 Actualiza cada movimiento con el código que resulta de aplicar dicho movimiento.

2.4 Si hay error al aplicar la transacción.

2.4.1 Revocar la transacción.

2.4.2 Guardar el error para regresarlo al nodo de origen de los movimientos.

2.5 Si se aplica exitosamente.

2.5.1 Almacenar confirmación de sentencia aplicada exitosamente.

La aplicación y confirmación de registros de movimientos provenientes de la entrada estándar sigue los puntos expuestos a continuación:

1. Prepara entrada y salida estándares.
2. Envía a la salida estándar la identificación del módulo.
3. Recibe de la entrada estándar si los datos están comprimidos.
4. Mientras existan datos en la entrada estándar.

4.1 Recibe de la entrada estándar el tamaño del bloque de datos siguiente.

4.2 Si vienen comprimidos también recibe el tamaño del bloque ya expandido.

4.3 Lee el bloque y verifica que el tamaño sea el especificado.

4.4 Si está comprimido el bloque lo expande y revisa que el tamaño expandido sea igual al proporcionado en la entrada estándar.

4.5 Si esta aplicando movimientos.

4.5.1 Arma la sentencia y la aplica.

4.6 Si esta aplicando confirmaciones.

4.6.1 Actualiza el registro de movimiento indicando el resultado de aplicarlo.

4.7 Si la sentencia pertenece a otra transacción diferente a la actual y no es la primera.

4.7.1 Hacer los cambios permanentes.

4.8 Aplica la sentencia.

4.9 Si hay error se almacena para regresarlo al nodo donde se originaron los movimientos.

4.10 Si fue exitosa la aplicación entonces se almacena la confirmación dirigida al nodo donde provienen las sentencias.

4.11 Se guarda en bitácora la razón de compresión del bloque y el tiempo consumido.

3.3.2.6 Módulo de bitácora y recuperación ante errores

Aunque el manejador de la base de datos proporciona varios mecanismos de recuperación de la operación ante errores, puede ocurrir que por el volumen de movimientos operados no sean suficientes

tales mecanismos. El módulo cliente ante errores almacenan el tipo de error y el punto de avance donde ocurrió dicho error y lo envían a la bitácora, éste módulo procede como sigue:

1. Prepara entrada y salida estándares.
2. Envía a la salida estándar la identificación del módulo.
3. Recibe petición para registrar en bitácora.
4. Mientras existan datos en la entrada estándar.
   4.1 Recibe de la entrada estándar el tamaño del bloque de datos siguiente.
   4.2 Lee el bloque y verifica que el tamaño sea el especificado.
   4.3 Registra en la base de datos la situación de cada movimiento del bloque.
   4.4 Si el RDBMS envía un error.
      4.4.1 Graba en un archivo los datos de la situación de cada movimiento del bloque.

Cuando se ha modificado las condiciones para aplicar los movimientos el módulo procede con los puntos siguientes:

1. Prepara entrada y salida estándares.
2. Envía a la salida estándar la identificación del módulo.
3. Recibe petición para recuperar movimientos.
4. Abre el archivo de bitácora.
5. Por cada registro en el archivo.
   5.1 Si es confirmación.
      5.1.1 Afecta sólo el registro en movimientos.
   5.1 Si no.
      5.2.1 Aplica la sentencia.
6. Cierra y borra el archivo.

3.3.2.7 Integración de los módulos

La integración de los componentes que participan en el mecanismo de réplica comprende los paquetes de PL-SQL en su forma de *triggers*, los programas ejecutables que fungen como manejadores de las operaciones y programas que sirven de interfaz con el usuario.

Los paquetes de PL-SQL se crean dentro de la base de datos y se asocian en forma de *trigger* a cada tabla que compartirá información a través del esquema de réplica. El comportamiento resultante es que, cada operación aplicada en la tabla es atrapada por el *trigger* correspondiente, en él se evalúa si la operación se sujeta a las reglas de operación y si se crea un movimiento para enviar la operación a otros nodos. Este comportamiento se muestra en la figura 3.26 donde el programa local puede ser una sesión en SQL*PLUS, un ejecutable en C, una aplicación creada con SQL*FORMS, etcétera; ya que de manera interna el RDBMS envía el control al paquete de código de la base de datos, el cual se representa en la figura como los rectángulos etiquetados con los métodos que pueden aplicarse a la

tabla: inserción, actualización, consulta y borrado. Una vez que en el método correspondiente se validan las reglas de operación y no existe error alguno, se inserta el movimiento, si el método recibe que se pudo insertar entonces afecta los datos.

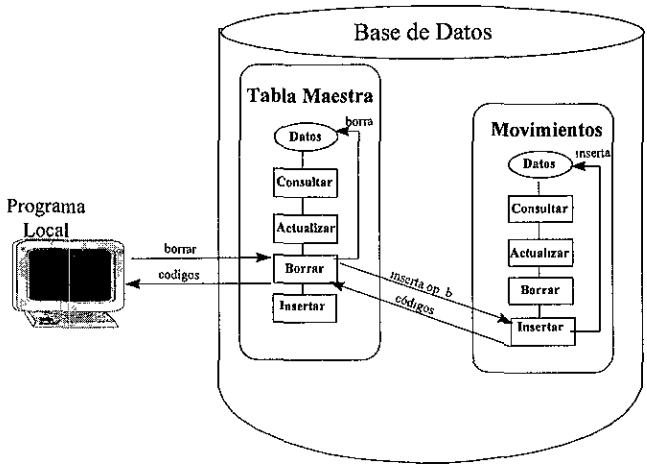


figura 3.26 Diagrama de módulos de los paquetes PL-SQL utilizados en los *triggers* de las tablas.

Una vez que se han acumulado los movimientos derivados de operaciones sobre las tablas que comparten información, se activa el proceso de transferencia de información el cual puede operar concurrentemente con aplicaciones que modifique las tablas. Para realizar la transferencia los módulos utilizan dos tipos de comunicación entre procesos: *pipes* y *sockets*. La comunicación a través de *pipes* se utiliza para procesos que se encuentran en el mismo nodo, cuando la comunicación se establece entre procesos en distintos nodos se hace a través de *sockets*. En la figura 3.27 se muestra la interacción de los módulos que participan en el mecanismo de réplica cuando se establece la comunicación a través de la red. Las líneas continuas indican que la comunicación se establece a través de *pipes* y las líneas punteadas indican que la comunicación se entable mediante *sockets*. El orden de las acciones se siguen de acuerdo al diagrama de sucesos presentados en el diseño del mecanismo de réplica. En la figura 3.27 el módulo servidor está "escuchando" constantemente las peticiones hechas al puerto correspondiente al servicio *filetrans*. El módulo cliente solicita el servicio de aplicación de actualización de un fragmento réplica y módulo servidor consulta la base de datos y obtiene las operaciones pendientes para aplicar. Envía las operaciones al módulo cliente el cual las aplica y regresa las confirmaciones correspondientes. El avance y estadísticas tanto del módulo cliente como el módulo servidor se almacenan en una sola bitácora ubicada en el nodo del módulo cliente.

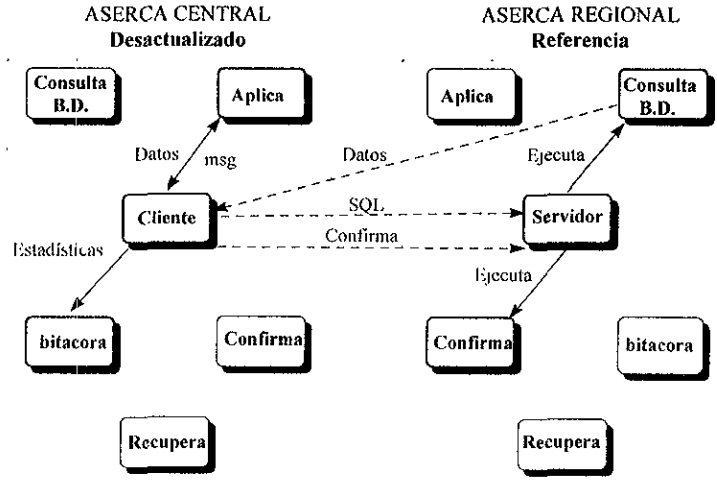


figura 3.27 Diagrama de modulos para aplicar réplicas.

3.3.3 Interfaz

Las interacciones con el esquema de replica se llevan a dos niveles:

- Programación.
- Operación.

A nivel programación al establecer el ambiente del esquema de replica simplifica posteriormente incluir otras tablas para compartir información entre distintos nodos. El ambiente del esquema de replica se conforma por las tablas que almacenan las transacciones y por los procesos que permiten aplicar dichas transacciones en distintos nodos. Se simplifica al proporcionar el espacio necesario para almacenar las transacciones diarias y al proporcionar un esquema de transferencia de información común. Sin embargo, se deben determinar las plantillas del conjunto de transacciones y programar el *trigger* para cada tabla, con el fin de automatizar este punto la aplicación mostrada en la figura 3.28 crea las plantillas de réplica con base en el nombre de la tabla proporcionado por el programador y con la información del RDBMS de las columnas, tipo de datos y llave primaria correspondientes.

Para crear de manera automática las plantillas de las transacciones se consultan las vistas donde el RDBMS guarda la información de la definición de los campos de las tablas *ALL_TAB_COLUMNS*, *ALL_IND_COLUMNS* y *ALL_CONSTRAINTS*. Como la plantilla se utiliza dentro de un programa en C se utilizan los formatos para imprimir valores en cadenas de caracteres terminados en nulo.

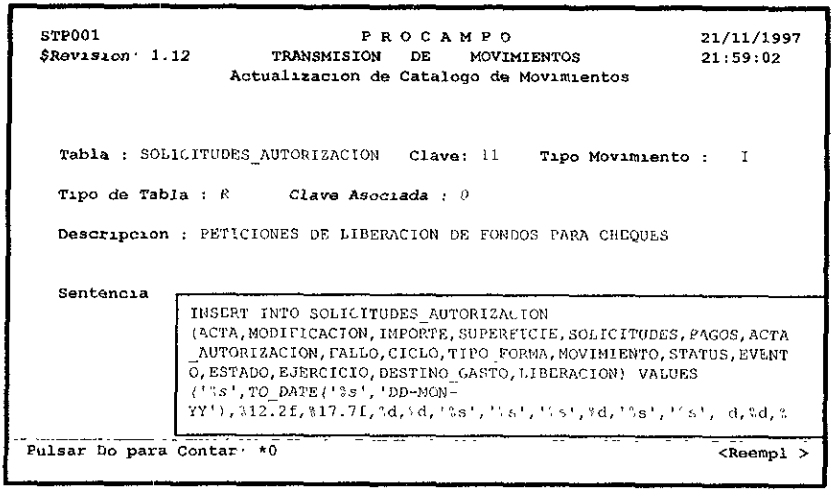


figura 3.28 Pantalla para creación de plantillas de transacciones.

Por otra parte a nivel operación se proporciona un conjunto de aplicaciones que permiten administrar la aplicación de los movimientos entre los nodos. Estas aplicaciones se presentan al operador a través del menú mostrado en la figura 3.29.

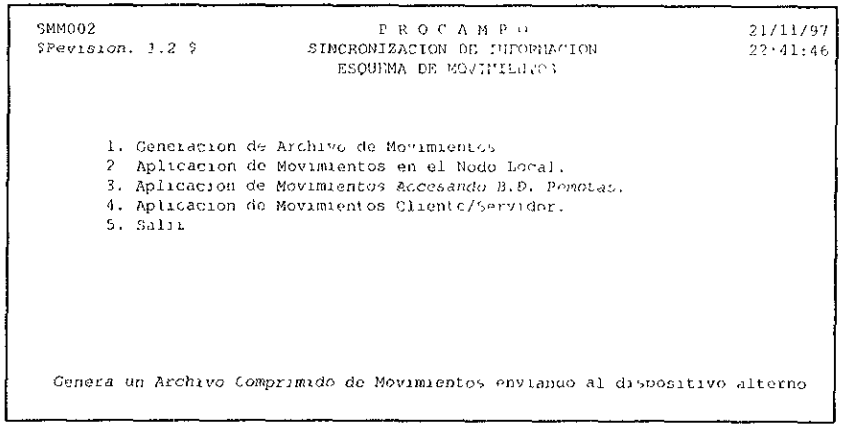


figura 3.29 Menú de aplicaciones para operación.

Una de estas aplicaciones es la mostrada en ECCC, en ella se especifican los nodos, donde se tomarán los movimientos acumulados, bajo el concepto de referencia y en otra columna se listan en que nodos se han de aplicar los movimientos leídos bajo el concepto de desactualizados. Permite indicar el tipo de transacciones a aplicar, lo cual permite dar prioridad por el tipo de información, si corresponde a cheques emitidos, conciliación de cheques cobrados, etcétera. Además se permite un nivel más

específico de aplicación de transacciones a través del número de evento, con lo cual se da preferencia a conjuntos de transacciones que hayan sido aplicadas en una sesión específica.

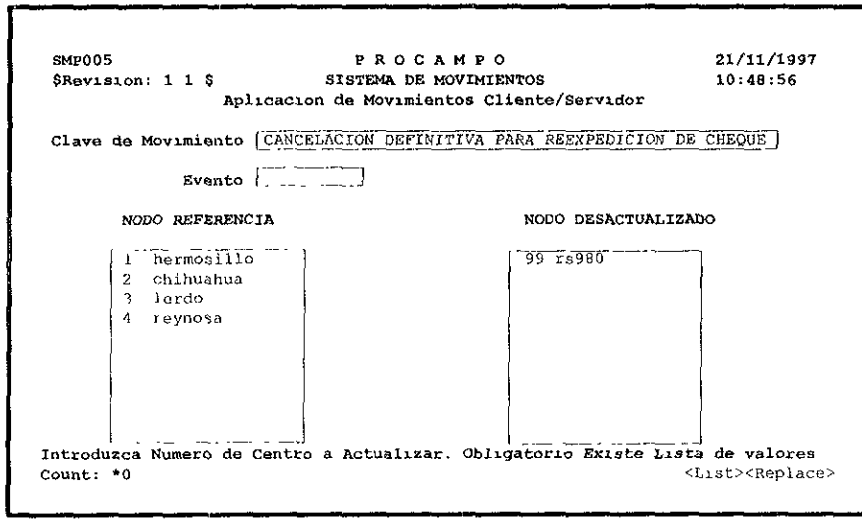


figura 3.30 Aplicación de movimientos entre nodos.

Con el fin de dar seguimiento a la aplicación de movimientos y revisar el estado de las conexiones se utiliza el shell *quien* que da formato a la salida del comando *netstat* de Unix, resultando la información siguiente:

```
Hrs570.enrique.REG]/u/enrique>quien
Desde la rs980pb:
Local Address          Foreign Address        (state)        C.R.
---------------------  ---------------------  -----------    ---------

Desde la rs980.
Local Address          Foreign Address        (state)        C R.
---------------------  ---------------------  -----------    ---------
128 100 0.1.3547             170 170.1.16.filetran   ESTABLISHED    xalapa
128.100.0.1 3542             170.170.1.2 filetran    ESTABLISHED    celaya
128.100.0 1.3541             170.170 1.17 filetran   ESTABLISHED    zacatecas
128.100.0 1.2681             170.170.1.16.filetran   ESTABLISHED    xalapa
128.100 0.1 1905             170.170.1.3.orasrv      ESTABLISHED    chihuahua
128.100.0.1 3656             170.170 1.15 orasrv     ESTABLISHED    tuxtla
128.100.0 1.4750             170 170.1.12.ftp        CLOSE_WAIT     puebla

Desde la rs570:
Local Address          Foreign Address        (state)        C R.
---------------------  ---------------------  -----------    ---------
x25mexico.1023         170.170.1 7.login      ESTABLISHED    hermosillo
```

La salida de este shell muestra que conexiones se han realizado desde los nodos centrales a los nodos regionales, los tipos de enlace y estado actual. Dentro de los tipos se observan en esta salida *ftp* para transferencia de archivos, *login* para sesiones remotas, *orasrv* para conexiones a través de sql*net de Oracle y *filetrans* para el servicio de replica.

4. Discusión

4.1 Desempeño

Con la finalidad de determinar las oportunidades para mejorar la eficiencia y flexibilidad de un proceso, es necesario clasificarlo o evaluarlo. Es el caso del proceso de transmisión de información que se da de los diferentes nodos de la red institucional de ASERCA a las oficinas centrales.

ASERCA cuenta con 17 nodos en 16 centros regionales distribuidos en el territorio nacional estos nodos son los de: Hermosillo, Chihuahua, Lerdo, Reynosa, Culiacán, Zacatecas, Puebla, Chilpancingo, Oaxaca, Tuxtla, Mérida e Hidalgo. El enlace de estos nodos con ASERCA central no es del mismo tipo, ya que los nodos de Guadalajara y Mérida cuenta con RDI y el resto de ellos se encuentran trabajando con X25.

La información que se requiere tener sincronizado y lo más completa posible es la que se refiere al programa institucional llamado "Entrega de apoyos".

En entrega de apoyos se requiere conocer y registrar los pagos hechos a los productores de acuerdo a la superficie cultivada en cada ciclo agrícola.

Los pagos efectuados, el número del cheque en que se entrego, el nombre del beneficiario, la superficie que se pago o que recibió un apoyo, son algunos de los campos que se requieren transmitir a las oficinas centrales de ASERCA.

Para que los pagos se puedan efectuar y se puedan entregar los cheques en cada uno de los centros regionales, es necesario que en ASERCA central se autoricen los pagos o movimientos presupuestales. Estos movimientos pueden ser el pago, la cancelación para una reexpedición o pago parcial de un apoyo.

Es necesario que la información que existe en la base de datos de cada centro regional sea la misma que la que existe en la base de datos de ASERCA Central; ya que en base a esta información se realizan diferentes auditorías, conciliaciones bancarias y se entregan reportes a TESOFE del presupuesto ejercido.

Debido a que en ASERCA Central se requiere dar la autorización a los movimientos que se realizan en los centros regionales y reflejar esta autorización en las bases de datos de los *regionales sin interrumpir su operación, se requiere un esquema de replicas asincrono, que* sea flexible para poder ajustarse a los cambios ya sean tecnológicos o de rendimiento y a las limitantes internas, una mejor eficacia evaluándola por medio de mediciones externas e internas y lograr una eficiencia optima que refleje la productividad de las operaciones internas y de los recursos empleados en el proceso.

Antes del desarrollo de este sistema la transmisión de información entre los Centros Regionales y ASERCA Central se utilizaba dos esquemas: la Transmisión de Archivos la cual consistía en extraer en archivos las operaciones diarias de las tablas a replicar, estos archivos se comprimían y se transmitían vía FTP al nodo de replicas o base de datos de

ASERCA Central, la Transmisión vía SQL-NET consistía en utilizar la herramienta de conectividad que proporciona la base de datos ORACLE, se transmiten las operaciones diarias al nodo replica vía SQL-NET. La transmisión de la información se realiza consultando las operaciones de la tabla de movimientos en los nodos fuente, se construyen las sentencias correspondientes y se aplican en el nodo replica; el RDBMS controla la transmisión de datos sin la posibilidad de ser comprimidos antes de ser enviados.

Debido a bajo desempeño de estos dos esquemas se desarrollo la Transmisión vía el Sistema Cliente/Servidor. Mediante un programa servidor (creado como *daemos*) se atienden las peticiones de consulta y transmisión de archivos que solicitan los nodos con programas cliente; para el esquema de replicas se consulta la base de datos mediante el programa servidor, el resultado obtenido se comprime y transmite al cliente en bloques de 600 KB. El programa cliente recibe estos bloques, los descomprime y comienza a construir las sentencias. Si las sentencias transmitidas en cada bloque están completas, se aplican en la base de datos mientras se están transmitiendo los bloques restantes. Por cada bloque recibido y aplicado se envía una confirmación al programa servidor, indicando que se recibió correctamente el paquete.

Para lograr una clasificación adecuada de los procesos y poder medir el desempeño se requiere hacer una "evaluación del proceso". Aplicando la evaluación a los tres procesos mencionados anteriormente, se pueden clasificar e identificar el nivel para cada uno.

En los tres esquemas de transmisión de información se tiene definida con claridad la titularidad del proceso; es el centro regional, sea el personal o el equipo, el responsable del rendimiento, costo y calidad del mismo. Por otra parte también se tienen establecidas relaciones con los usuarios para atender los requerimientos tanto internos como externos.

Para cada uno de los procesos, la documentación es importante para conocer el flujo de trabajo, esta documentación proporciona un registro permanente de la transformación física que ocurre en un proceso de producción.

Otros parámetros que se considerado en la evaluación fueron la eficacia, la eficiencia, la identificación de puntos de control con el propósito de conocer las fallas, trabajo adicional, problemas con usuarios y redundancia. Tanto el proceso de Transmisión de Archivos como el de Transmisión vía sql*net Presentaron fallas considerables ya que los procesos eran excesivamente lentos, generaban problemas de control para los usuarios, no estaban diseñados para permitir replicas asíncronas y no eran lo suficientemente flexibles para poder adaptarse a los cambios de estrategia y de dirección.

Considerando lo anterior, estos procesos requieren de cierta reingeniería importante para que tuvieran una eficiencia, eficacia y flexibilidad de alto nivel.

Para los esquemas de Transmisión de Archivos y Transmisión vía SQL-NET se obtuvieron estadísticas de la eficiencia sobre el manejo de registros y se tomaron muestras de algunos Centros Regionales.

Centro Regional	Proceso	Registros	Tiempo Ideal (hh:mm)	No. de registros por hora ideal	Tiempo Real (hh:mm	No. de registros por hora Real
Chihuahua	Crear archivo	8,340	0:50	4,954.45	12:28	668.98
	Aplicar		0:16			
	Crear archivo		0:16			
	Confirmar		0:19			
	TOTAL		1:41			
Zacatecas	Crear archivo	15,417	0:23	6,166.80	4:18	3,585.84
	Aplicar		0:28			
	Crear archivo		0:51			
	Confirmar		0:48			
	TOTAL		2:30			
Celaya	Crear archivo	8,340	1:11	4,132.23	6:35	3,797.46
	Aplicar		1:22			
	Crear archivo		0:18			
	Confirmar		3:12			
	TOTAL		3:03			

En el cuadro anterior se puede observar que el número total de registros en tiempo real es mucho menor que el número de registros en tiempo ideal. Este sistema utiliza mucho tiempo para procesar pocos registros. La siguiente gráfica muestra la diferencia entre el proceso real y el proceso requerido para el manejo de registros.

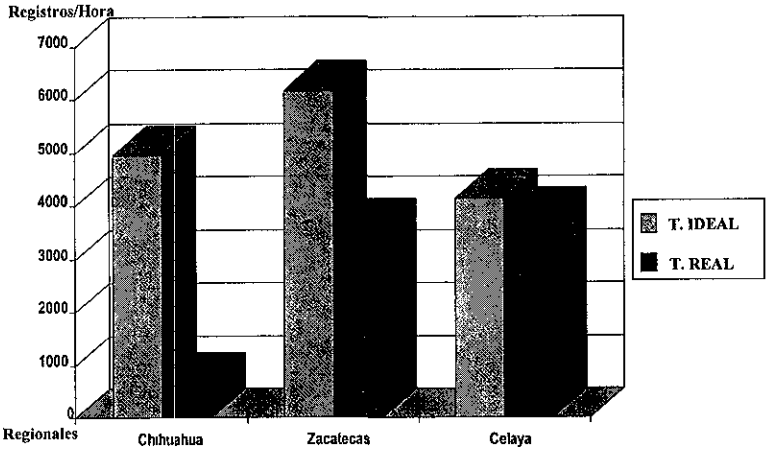


figura 4.1 Estadísticas del sistema de archivos.

El siguiente cuadro muestra la eficiencia del sistema de Transmisión vía SQL-NET.

CENTRO REGIONAL	NÚMERO DE REGISTROS	TIEMPO (HH:MM)	REGISTROS HORA
Chihuahua	1,000	12:28	8,178
Zacatecas	10,000	4:18	5,700
Celaya	4,202	6:35	9,404

Aquí podemos observar que el número de registros procesados por hora es mayor que en el sistema de Transmisión de Archivos.

La gráfica siguiente muestra un comparativo de los registros procesados entre los Centros Regionales de mayor carga de información.

Este sistema le deja a sql*net el envío de la información, transmite registros y no transacciones sin la posibilidad de compactarlos y de enviarlos por otro medio alterno si las líneas de comunicación no se encuentran disponibles.

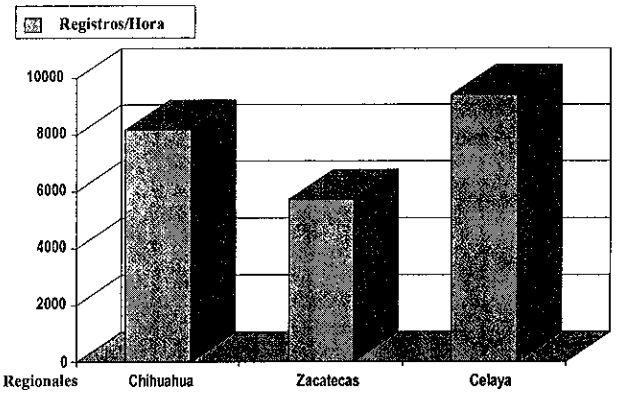


figura 4.2 Estadísticas del sistema basado en el producto sql*net de Oracle.

Para el Sistema Cliente/Servidor se obtuvieron las gráficas de eficiencia considerando también el número de registros procesados por hora de una muestra de los Centros Regionales, en el cuadro siguiente se muestra el comparativo entre regionales:

REGIONAL	NÚMERO DE REGISTROS	TIEMPO (HH:MM)	REGISTROS / HORA
Chihuahua	7320	2:00	14,936.00
Zacatecas	7,320	0:28	15,685.71
Celaya	124,008	9:11	13,503.59

En este cuadro, los tiempos son pequeños y el número de transacciones que envía son muy grandes. Para tiempos como los que se utilizaron en los dos sistemas anteriores, este proceso no pudo evaluarse ya que no hay tantas transacciones en los centros regionales en la producción diaria.

La gráfica que muestra la relación de registros por hora en los Centros Regionales se muestra en la figura 4.3.

Haciendo un comparativo entre los tres esquemas podemos observar que el número de registros procesados son mucho mayores en el esquema Cliente/Servidor que en los otros dos esquemas.

El cuadro estadístico de los tres esquemas se presenta a continuación, indicando el tiempo utilizado y los registros procesados por hora.

CENTRO REGIONAL	ARCHIVOS		SQL-NET		C / S	
	TIEMPO HH:MM	REG / HORA	TIEMPO HH:MM	REG / HORA	TIEMPO	REG / HORA
Zacatecas.	4:18	3,585.84	4:18	5,700	0:28	15,678
Chihuahua	12:28	668.98	12:28	8,178	2:00	14,936
Celaya	6:35	3,797.46	6:35	9,404	9:11	13,503

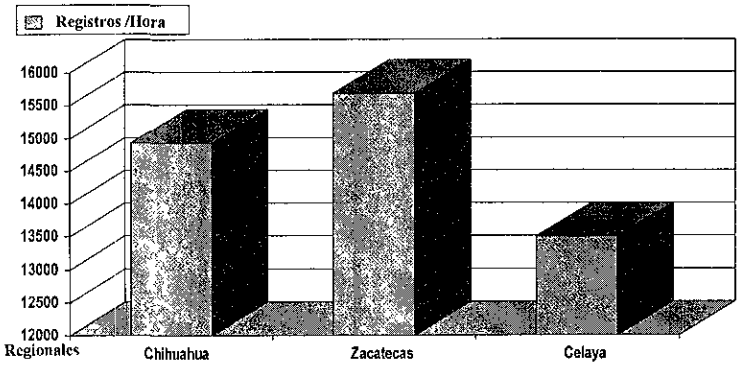


figura 4.3 Estadísticas del sistema Cliente/Servidor para el esquema de replica propio.

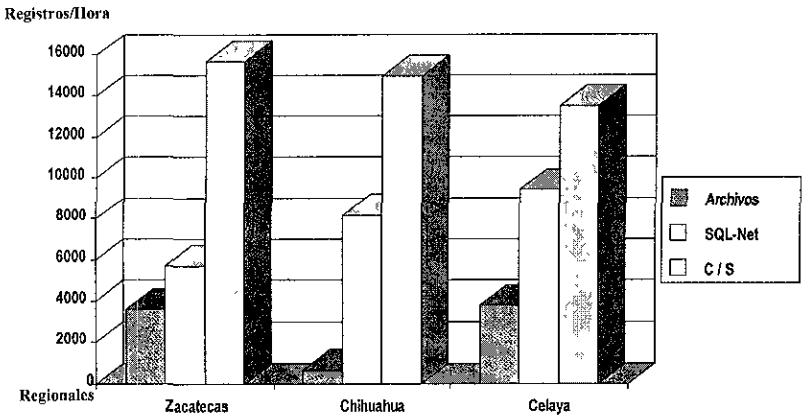


figura 4.4 Comparativo de los sistemas.

El sistema Cliente/Servidor, de acuerdo a la clasificación que da el sistema de evaluación de procesos, tiene un nivel alto de desempeño ya que el proceso cumple con lo siguiente:

- El proceso cumple con los requerimientos del usuario.
- El proceso se encuentra definido y documentado.
- Se cuenta con indicadores de eficiencia y eficacia que el mismo sistema deja en las bitácoras de trabajo donde viene el número de registros por tiempo.
- No existen desviaciones que no permitan llevar un adecuado control.
- Las mediciones de eficiencia marcan un considerable avance en la reducción de tiempo de proceso.
- Las cargas de trabajo se distribuyen en los nodos disponibles.

- El proceso es competitivo frente a procesos similares tanto en eficacia como en eficiencia.
- El proceso se adapta a los constantes cambios operativos del mismo programa institucional.

Mediante la identificación y utilización de una adecuada arquitectura, como es la arquitectura Cliente/Servidor, es posible desarrollar procesos que cumplan con los requerimientos de los usuarios, de la organización o institución y pueda adaptarse a los continuos cambios en el modelo operativo.

La arquitectura Cliente/Servidor es la primera solución en tecnología informática que satisface las presiones de costo rendimiento de una empresa actual de alto nivel. Es por esto que se observa una clara tendencia de migración hacia este modelo.

Los principales motivos por los cuales el modelo Cliente/Servidor fue utilizado para la replicación de las bases de datos en ASERCA son: La flexibilidad para satisfacer las necesidades del programa, la reducción de costos, el poder identificar la solución que se adecue a las necesidades de PROCAMPO y que crezca a la par de esas necesidades, incrementar la productividad del personal tanto de oficinas centrales como en los centros regionales, aprovechar la infraestructura con que se cuenta la cual no se utiliza al 100% de su capacidad y hacer que los procesos sean más confiables.

La infraestructura de computo en ASERCA está formada por equipos multiusuarios RS6000, modems y terminales donde se ejecutan las aplicaciones, así como equipos PC's conectados a una red de computadoras, la cual se utiliza para la automatización de oficinas. Con la arquitectura Cliente/Servidor se pueden integrar diferentes tipos de equipos y sistemas operativos en un ambiente único de procesamiento. Con el empleo de este modelo la capacidad de cómputo la constituyen todos los equipos interconectados, en los que se incluyen las PC's.

Otro aspecto importante a considerar, en donde se observa un mejor desempeño, es la reducción en el trafico de la red, ya que las operaciones de procesamiento se jerarquizan y se manejan prioridades de acuerdo a las necesidades de los usuarios y sólo le transmite información realmente útil.

Mediante este modelo, en ASERCA se pudo estructurar una estrategia que hizo que los recursos de cómputo se utilicen de la mejor manera posible, con lo cual se reducen los costos de manera considerable.

Las medidas que se toman para mantener un nivel alto de desempeño son básicamente las que se listan a continuación:

- ***Separación de Tareas:*** *Las funciones de cómputo se sincronizan con los recursos de* tal manera que el servidor procesa sólo las que son más complejas o que requieren operaciones intensas para liberar al cliente de esa labor. De esta manera, el cliente tiene la oportunidad de ejecutar otras tareas simultáneas. Con esta configuración puede integrarse una estrategia que equilibre la combinación y costo de los dispositivos de cómputo.

- **Comportamiento de Periféricos:** La combinación de esos dispositivo, también puede realizarse al hacer que sus características coincidan con las funciones que se piensan ejecutar y como la arquitectura Cliente/Servidor es abierta, permite la incorporación de un dispositivo especializado para realizar tareas muy específicas.
- **Comportamiento de herramientas:** En ASERCA los clientes comparten diversas aplicaciones con la gran ventaja de que éstas están controladas de manera centralizada por un mismo administrador. Con esta ventaja se elimina la necesidad de tener que instalar, configurar y dar de alta la aplicación en cada cliente cada vez que surja una nueva versión de las aplicaciones que son ejecutadas en los Centros Regionales.
- **Acceso a la información:** A pesar de que ASERCA se encuentra geográficamente separada, la arquitectura Cliente/Servidor permite el acceso a la información a nivel del servidor de manera transparente y esto se logra teniendo estándares comunes.

4.2 Comparativo con productos comerciales

Actualmente la productividad en las empresas se ha redefinido en función del tiempo, la velocidad, la calidad y la conectividad que los sistemas de trabajo ofrecen a los usuarios. El sistema Cliente/Servidor propuesto cumple con los objetivos y metas que se definieron al momento de su planeación.

El sistema Cliente/Servidor, para su funcionamiento, incorporó muchas características que eran necesarias debido al cambio continuo en el modelo operativo, a la capacidad de respuesta y a la misma operación del programa PROCAMPO.

Se busco que el sistema tuviera compresión de datos para una velocidad mayor al momento de transmitir información, que fuera simétrico de tal forma que si se modificaba la información en cualquier nodo no se detuviera la operación, que contara con transmisión de archivos en caso de que no hubiera líneas de la red así también que contara con un control por transacción y no nada mas por registro.

Estas características se buscaron también en los productos comerciales que existían en el mercado, encontrando que no cumplían en su totalidad con los requerimientos solicitados. En la tabla se presenta un cuadro comparativo entre los productos comerciales que existían y el sistema propuesto:

El cuadro nos muestra que el sistema Cliente/Servidor tiene las características necesarias para la operación del programa que los otros sistemas, para esas versiones, no habían incorporado.

PRODUCTOS DE REPLICACIÓN	TOLERANTE A ERRORES	SIMETRICO	COMPRESIÓN DE DATOS	INTERFASE GRÁFICA CON EL USUARIO	CONTROL POR TRANSACCIÓN	CONEXIÓN A OTRAS BASES DE DATOS	ENVIO TRAVES DE ARCHIVOS	MULTI PLATAFORMA
ORACLE VERSION 7.3	√			√		√		√
DB2 VERSION 2.0	√	√		√		√		√
INFORMIX VERSION 5.0	√			√		√		√
SISTEMA CLIENTE/SERVIDOR *PROPUESTO*	√	√	√		√		√	

tabla 15 Cuadro comparativo entre productos comerciales disponibles en el mercado y el sistema Cliente/Servidor propuesto.

5. Conclusiones

El objetivo de mantener sincronizada la información entre los distintos nodos de ASERCA con la disminución en el esfuerzo, tiempo y costos requeridos anteriormente fue alcanzado. Además se cumplió con las características planteadas para el sistema inicialmente las cuales son :

- Mantendrá copias de cualquier información de la base de datos de ASERCA Regional en ASERCA Central.
- Permitirá la operación con los datos tanto en ASERCA Central como en ASERCA Regional.
- Deberá contar con mecanismos alternos a las líneas de la red con el objeto de mantener los datos sincronizados a pesar de que las líneas no funcionen.
- Tolerante a caídas temporales de las líneas de comunicación, deberá continuar en el punto donde ocurrió la interrupción cuando las líneas se restablecen.
- El uso de la línea de comunicación deberá ser óptimo con el fin de no degradar por tiempos prolongados las comunicaciones.

Cada una de estas características están soportadas por los módulos que conforman el sistema. Así existe un módulo que maneja archivos con operaciones para los casos en que no exista comunicación en línea. Además se cuenta con un módulo para recuperación ante errores Para hacer uso óptimo de la línea se comprime la información y solo se envía la información que ha cambiado.

El sistema cumplió con las expectativas de desempeño planteadas en un inicio donde se esperaba que la transferencia de información fuera de 0.39 segundos por transacción y como se ve en los cuadros de la discusión se alcanza hasta 15.678 registros por hora en Zacatecas lo cual equivale a 0 23 segundos por transacción. A nivel operativo ha redundado en la disminución del tiempo requerido para mantener la información sincronizada. En términos de tiempo lo que requería de 7 horas de proceso para transferir las expediciones de cheques en un día se ha reducido hasta una hora.

A nivel programación el sistema ha permitido el desarrollo de esquemas con información compartida en menos tiempo. Ahorrando el diseño de las estructuras de datos que temporalmente almacenan la información de transferencia y permitiendo a los diseñadores mayor concentración en el dominio del problema que han de implantar. Así se han desarrollado tres esquemas: de autorización de emisiones, conciliación de cheques y depósitos en cuentas.

5.1 Aplicaciones

El sistema de réplica propio se ha creado flexible lo cual permite su uso en diferentes esquemas de bases de datos que requieran compartir información tales como aplicaciones financieras, nóminas, facturación etc. Sin embargo el código obtenido opera con el manejador de bases de datos comercial Oracle y bajo ambiente Unix, lo cual reduce su campo de utilización.

5.2 Crecimiento futuro

El sistema de réplica propio fue codificado con base en sockets se podría actualizar para que utilice un protocolo de mayor nivel como RCP (*Remote Call Procedure*). Por otro lado el sistema carece de una interfaz amigable para el seguimiento de la aplicación de las transacciones diarias, así que como crecimiento futuro se considera la programación de una interfaz gráfica en la que se represente la información que está sujeta al esquema de réplica y su estado. Además para la incorporación de entidades al esquema de réplica sería conveniente disponer de una interfaz que utilizará la notación propuesta en el capítulo 3.2.3.

En cuanto al desempeño se puede reducir el tiempo de transferencia si se sustituye el módulo de compresión de datos por otro con mayor razón de compresión. La sustitución de este módulo es sencilla dado que sólo se basa en la salida y entrada estándares. La razón de compresión se puede aumentar si se combina el algoritmo LZW aquí presentado con un algoritmo aritmético.

Apéndice A. Compresión de Datos

A.1 Introducción

La compresión en el presente con todas sus variantes, es ampliamente aceptada por la mayoría de los administradores de sistemas en aquellos recursos que tienen más demanda como lo son los dispositivos de almacenamiento y los de transmisión de información. Existen una gran cantidad de técnicas que van desde algoritmos sencillos a complejos, pasando por técnicas que recuperan fielmente la información y otras que no recuperan toda pero entran en un margen de aceptación. Sin duda alguna en el envío de datos dentro de una red con una base de datos distribuida y con procesos de actualización en volúmenes considerables, es perfectamente aplicable alguna técnica de compresión de datos como se detalla a continuación.

A.2 Razones para usar la compresión

a) *Disminuye la necesidad de almacenamiento y de Acceso.*- Hoy en día que la información se maneja en grandes volúmenes, y a pesar de haberse reducido costos en los dispositivos de almacenamiento, todavía es necesario disminuir el espacio, sobre todo cuando el acceso a la información a discos y a cintas es relativamente alto por su naturaleza mecánica en comparación con la velocidad a la que se leen los datos lógicamente, es decir entre más pequeña o comprimida sea la información, más rápido será su acceso.

b) *Permite una mejor razón de transmisión.*- Cuando la información requiere ser transmitida y replicada en diversos nodos de una red, es lógico considerar que el canal de transmisión puede soportar un mayor ancho de banda cuando los datos viajan comprimidos que cuando viajan en un estado normal, además el flujo de dicha información mejora en forma considerable reduciendo otros costos como lo puede ser el costo de tiempo de conexión en la línea, luz, electricidad, horas extras, etc.

A.3 Definición

La compresión es la técnica de tomar una cadena de información y operar de acuerdo a un algoritmo particular para producir una cadena de datos comprimidos o codificados. Esta operación recibe el nombre de *proceso de codificación.* La operación inversa, es decir, tomar la cadena de datos comprimidos y reproducir la cadena original de datos o cadena descodificada se le denomina *proceso de descodificación* (ver figura A.1).

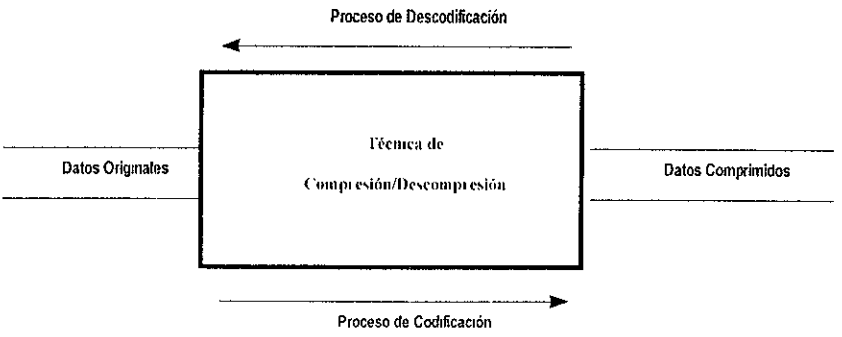


figura A.1 Caja negra que representa las técnicas de Compresión/Descompresión.

El grado en el cual se reduce la cadena original se le denomina *Razón de Compresión* y se expresa como:

$$RC = \frac{\text{Longitud de la cadena original}}{\text{Longitud de la cadena comprimida}}$$

La *Entropía* es el recíproco de la razón de compresión, es decir:

$$E = \frac{\text{Longitud de la cadena comprimida}}{\text{Longitud de la cadena original}}$$

De esto se deduce que entre más grande es la razón de compresión, más eficiente es la técnica de compresión empleada, y la entropía siempre debe de ser menor a la unidad para que el proceso de compresión sea efectivo.

La compresión en general se divide en dos grandes tipos:

a) *Compresión Lógica.-* Trabaja principalmente con las estructuras de datos, es decir actúa sobre un diseño particular de una aplicación o base de datos, permitiendo asignar el menor espacio en cuanto a capacidad de almacenamiento de variables, campos, objetos, etc.; y

b) *Compresión Física.-* Trabaja para disminuir de forma efectiva la cantidad de bits requeridos para almacenar o transmitir la información, esta compresión se realiza a través de la correlación de los datos, la eliminación superflua, las probabilidades de ocurrencias, etc. haciendo que la redundancia en la información desaparezca.

Los algoritmos de compresión se clasifican por su naturaleza respecto al manejo de la información como.

a) *Con pérdida de información.-* Estos algoritmos descodifican la información con un rango aceptable de pérdidas en los datos, debido a las técnicas empleadas al codificar. El rango es aceptable solo cuando la información no se degrada de forma tal que pierda características importantes y cuando la información no se requiera a un 100% de la información original, esta técnica es muy usada en compresión de imágenes continuas que serán digitalizadas y las pérdidas son insignificante para la vista humana. Para nuestro caso este tipo de técnica no es útil.

b) *Sin pérdida de información.*- Son también conocidos como codificación sin ruido y permiten que la información original al ser descodificada sea recuperable en un 100%. Entre las técnicas más populares se encuentran:
   - *código de Huffman y su variante, código de Huffman modificado*
   - *el algoritmo Lempel-Ziv (LZ) y su variante, el algoritmo Lempel-Ziv-Welch (LZW)*
   - *el algoritmo RLE (Run Length Encoding)*

Por último, los algoritmos o métodos de compresión existentes también se clasifican por su naturaleza en cuanto al manejo de sus códigos como:

a) *Estáticos* - Se dan cuando el proceso de codificación crea previamente los códigos o palabras claves antes de empezar la transmisión, así una cadena x será representada por el mismo código cada vez que aparece en el mensaje de datos a codificar. El método clásico de este tipo de algoritmo es el *Método de Huffman*, Este algoritmo debe utilizar técnicas de códigos de longitud variable, de manera que las cadenas poco encontradas en la información se les asignen palabras largas de código y las cadenas más frecuentemente encontradas se les asignen palabras cortas de código. Los códigos se diseñan de tal forma que el número de bits para cada unidad de dato codificado sea el mínimo posible. Todas las probabilidades se realizan antes de que el proceso de codificación inicie, es decir requieren dos "pasadas".

b) *Dinámicos* - Estos métodos también conocidos como *adaptables,* son de una sola "pasada" y los códigos se van generando en *tablas de traducción* o *diccionarios*, conforme se analizan las cadenas. El algoritmo clásico para este método es el Ziv-Lempel (LZ) y el Ziv-Lempel-Welch (LZW).

Todas estas técnicas de compresión presentan una razón de compresión regular que varía desde 1.7·1 hasta 4:1.

La técnica empleada en el proyecto fue la LZW que es una técnica de compresión física, sin pérdida de información y de tipo Dinámica, a continuación se darán sus características.

A.4 Compresión LZW[xvi]

El algoritmo de compresión LZW fue publicado en un artículo llamado "A Technique for High-Performance Data Compression" en la revista IEEE COMPUTER, de Junio de 1984 , su autor Terry A. Welch describe un nuevo sistema de compresión basado en el algoritmo Ziv-Lempel (LZ). Este algoritmo es ampliamente utilizado hasta hoy por sus siguientes características:

a) Utiliza una tasa de memoria baja en datos temporales (0 Bytes en algunos casos), en comparación con sus antecesores.

b) Es una herramienta eficiente en términos de tiempo de respuesta tanto en la compresión como en la descompresión.

c) Lleva a cabo una aceptable razón de compresión.

d) Es un algoritmo genérico y de fácil implementación en cualquier plataforma hardware.

Si bien es cierto que este algoritmo presenta estas ventajas es necesario decir que sufre de algunas desventajas, que son:

a) Por tratarse de un algoritmo dinámico, la razón de compresión depende en gran medida de los datos que se comprimirán y del tamaño de la "ventana"[3] del algoritmo, ya que no es lo mismo comprimir datos que imágenes debido a las diferencias de información secuencial que presentan; sin embargo esto suele ser una ventaja para datos que contienen demasiada redundancia o presentan información aleatoria.

b) Es perfectamente factible que el archivo comprimido ocupe más espacio que el archivo original, sin embargo esto sucede en raras ocasiones y es debido a compresiones sobre datos que no requieren tal proceso.

c) En los primeros bytes a analizar, no hay suficiente información sobre las características de los datos para poder comprimirlos, por lo que estos pueden sufrir una expansión en lugar de una reducción (desventaja anterior). Como consecuencia de esta razón, la técnica se recomienda solo para archivos de un tamaño que supere un mínimo. Es más eficiente, por lo tanto la compresión de un archivo resultante de unir varios, que comprimir cada uno por separado.

El algoritmo LZW está organizado en torno a una tabla de traducción, que relaciona cadenas de caracteres del mensaje de datos en códigos de longitud fija, siendo 12 bits su tamaño normal; es decir tendremos hasta $2^{12} = 4096$ diferentes entradas de códigos en nuestra tabla de traducción. Esta tabla cumple con una propiedad denominada *prefijo*, la que requiere que, para cada cadena de la tabla de traducción, su cadena prefijo también aparezca en ella. Es decir, que si la cadena 'xk', compuesta por la cadena 'x' y el carácter 'k', se encuentra en la tabla de traducción, entonces la cadena 'x' también se encuentra en ella; 'k' es el carácter de extensión de la cadena 'x'. En esta descripción la tabla de cadenas es inicializada a cadenas de un solo carácter

A.4.1 Algoritmo de Compresión LZW

La tabla de traducción del sistema LZW contiene las cadenas que han sido encontradas previamente en el mensaje que esta siendo comprimido y se les ha asignado como ya vimos un único código con la propiedad del prefijo. Además estas cadenas en determinado momento reflejan datos estadísticos del mensaje. Los datos de entrada son examinados byte a byte, buscando la mayor cadena reconocida. La mayor cadena reconocida es aquella que ya existe en la tabla de traducción. Las nuevas cadenas que se añadirán a la tabla de traducción son determinadas por el siguiente Algoritmo de Compresión. Cada cadena reconocida es extendida con el siguiente carácter (es decir una cadena 'x' pasa a ser 'xk' donde 'k' es el carácter de extensión) y esta nueva cadena se añade a la tabla. A cada nueva cadena se le va asignando un código único.

Algoritmo de Compresión LZW

Inicializar la tabla de traducción a un solo carácter

Lee el primer carácter de la entrada, siendo este 'k'

Inicializar la cadena con prefijo 'x' al valor 'k' (x=k)

Bucle: *Lee el siguiente carácter, siendo este 'k'*

[3] Se denomina ventana de desplazamiento a la cantidad de datos que se analizarán en determinado momento

Si no existe 'k' (final de archivo)

Envía el código de 'x' al archivo de salida

Fin

Si existe la cadena 'xk' en la tabla de cadenas

Actualiza la cadena prefijo (x=xk)

Ir a Bucle

Si no existe la cadena 'xk' en la tabla de traducción

Envía el código de 'x' al archivo de salida

Añade la cadena 'xk' a la tabla de traducción

Inicializa la cadena prefijo (x=k)

Ir a Bucle

Cabe hacer notar que el algoritmo no hace ningún intento de seleccionar una cadena óptima de la tabla de traducción y a pesar de eso su razón de compresión, aunque inferior al óptimo, es eficiente; además su implementación es bastante simple y rápida.

A.4.2 Algoritmo de descompresión LZW

El descompresor LZW, es conceptualmente más complejo pero en realidad es muy sencilla su implementación en programa. Lógicamente utiliza una tabla de traducción que se irá creando con base en la tabla de traducción del compresor y es lo único que se necesita conocer para iniciar el algoritmo; para esto necesitamos además definir dos términos: el 'código actual' y el 'código anterior' el primer código de entrada se asignara a 'código actual'. Este código será el primero de nuestra tabla de traducción, y se manda el carácter al archivo de salida. Posteriormente este código será el 'código anterior' Se toma el siguiente código y se hace al 'código actual'. Es muy probable que este código no se encuentre en la tabla de traducción, pero por ahora se asume que si existe; se manda su carácter correspondiente al archivo de salida. A partir de este momento se genera un bucle que analiza todo el archivo comprimido con los siguientes tres pasos:

1. Se toma el primer carácter que se tradujo y se le denomina 'k'
2. Se le adiciona el prefijo 'x' generado por el 'código anterior' formando una nueva cadena 'xk'.
3. Se coloca en la tabla de traducción y se fija 'código anterior' como 'código actual'.

Esto pasos serían todo el proceso de descompresión, sin embargo consideremos ahora que 'código actual' no esta en la tabla de traducción y supongamos que viene una cadena como 'kxkxkq'. Vamos a suponer que 'kx' ya se encuentra en la tabla de traducción, pero 'kxk' no. El compresor analizará 'kx' y sacara este cadena. pero encontrará que 'kxk' no está en la tabla de traducción. El descompresor esta siempre un 'paso' más adelante que el compresor. Cuando el descompresor vea el código para 'kxk', no tendrá adicionado este código a la tabla de traducción todavía porque necesita el carácter de inicio k para formar el código para 'kxk'. Sin embargo cuando el descompresor encuentra un código que no conoce, este será adicionado en el siguiente bucle en la tabla de traducción. Así se puede adelantar que cadena para el código será y de hecho esto siempre será correcto. El algoritmo se vería como está a continuación:

```
Algoritmo de descompresion LZW
Inicializar la tabla de traducción
Asigna a 'codigo actual' el código asociado al caracter 'k'
Envia 'k' al archivo de salida
Hacer 'código anterior' = 'codigo actual'
Bucle  Asignar a 'codigo actual' el siguiente código de la cadena
              Si no hay mas caracteres termina
Existe codigo actual' en la tabla de traduccion?
si existe entonces
              sacar el caracter asociado a 'codigo actual' al archivo de salida
              asignar como prefijo 'x' la traduccion de 'codigo anterior'
                     asigna como carácter 'k' el primer caracter de la tabla de traduccion
       para 'codigo actual'
                     adiciona 'xk' a la tabla de traducción
                     asignar a 'codigo anterior' el 'codigo actual'
       si no existe (caso especial)
                     asignar al prefijo 'x' la traduccion de 'codigo anterior'
                     asignar al caracter 'k' el primer caracter de prefijo 'x'
       enviar   al   archivo   de   salida   a   'x'   y   adicionarlo   a   la   tabla   de
              traducción
              asignar 'código actual a 'codigo anterior'
                     ir a Bucle
```

Resulta normal poder ver que este método accede muchas veces la tabla de traducción por lo que es necesario poner énfasis en el acceso eficiente de dicha estructura, la mejor forma de hacerlo es trabajando con buffers circulares o directamente con apuntadores en localidades de memoria. Dicha implementación logrará además de aumentar la rapidez, generar las cadenas en orden inverso en que se extraen para lograr la descompresión ideal.

Este método está implementado como una "utilería" UNIX con el nombre de "*compress*", pero para propósitos del proyecto fue mejor crear un procedimiento propio.

El procedimiento en lenguaje C es el siguiente.

```
/* Programa de demostración del metodo LZW para compresion/descompresion */
#include <stdio.h>
#define BITS 12                     /* tamaño del codigo en bits, 12, 13, o 14*/
#define HASHING_SHIFT BITS-8
#define MAX_VALUE (1 << BITS) - 1
#define MAX_CODE MAX_VALUE - 1

#if BITS == 14
  #define TABLE_SIZE 18041          /* tamaños de la tabla*/
#endif

#if BITS == 13
  #define TABLE_SIZE 9029
#endif

#if BITS <= 12
  #define TABLE_SIZE 5021
#endif

void *malloc();
int *code_value,                    /* arreglo de los valores de codigo*/
unsigned int *prefix_code,          /* arreglo que guarda los prefijos */
unsigned char *append_character;    /* arreglo que guarda el caracter de entrada */
unsigned char decode_stack[4000]; /* arreglo que guarda la cadena codificada */

/*** Dado un nombre de archivo, el programa lo comprime generando
**   un archivo test.lzw  y luego genera un archivo que descomprime
**   llamado test.out. Test.out deberá ser del mismo tamaño que el
**   archivo comprimido */

main(int argc, char *argv[]){
  FILE *input_file,
  FILE *output_file;
  FILE *lzw_file;
  char input_file_name[81],
/* Los tres buffers se requieren para la fase de compresion */
```

```
  code_value=malloc(TABLE_SIZE*sizeof(unsigned int));
  prefix_code=malloc(TABLE_SIZE*sizeof(unsigned int));
  append_character=malloc(TABLE_SIZE*sizeof(unsigned char));
  if (code_value==NULL || prefix_code==NULL || append_character==NULL)  {
    printf(",Grave error localizando el espacio para la tabla!\n"),
    exit();
  }

/* El usuario da el nombre del archivo, se abre, y se genera el archivo test.lzw */
  if (argc>1)
    strcpy(input_file_name,argv[1]);
  else  {
    printf("¿Nombre del Archivo a comprimir? ");
    scanf("%s",input_file_name);
  }
  input_file=fopen(input_file_name,"rb");
  lzw_file=fopen("test.lzw","wb");
  if (input_file==NULL || lzw_file==NULL)  {
    printf("Grave error al quere abrir el archivo \n");
    exit();
  };

/* Procedimiento de compresión */
  compress(input_file,lzw_file);
  fclose(input_file);
  fclose(lzw_file);
  free(code_value),

/* Ahora abre el archivo resultante para descomprimirlo */
  lzw_file=fopen("test.lzw","rb");
  output_file=fopen("test.out","wb");
  if (lzw_file==NULL || output_file==NULL)  {
    printf("Grave error al abrir el archivo.\n");
    exit();
  },

/* Descomprime el archivo */
  expand(lzw_file,output_file);
  fclose(lzw_file);
  fclose(output_file),
  free(prefix_code);
  free(append_character);
}

/* Rutinas de compresion */
compress(FILE *input,FILE *output){
  unsigned int next_code,
  unsigned int character;
  unsigned int string_code;
  unsigned int index,
  int i;

  next_code=256;              /* el código siguiente es el 256, 0-255  */
/* reservados                                */
  for (i=0,i<TABLE_SIZE;i++)  /* Limpia la tabla antes de empezar */
    code_value[i]=-1;
  i=0;
  printf("Comprimiendo.  \n");
  string_code=getc(input);    /* Asigna el primer codigo              */

/* Esta es la rutina del Loop donde pasa todo el archivo  Este loop corre todas
** las entradas de caracteres.  Note que el loop parara adicionando códigoa a la
** tabla después de que todos los posibles códigos se han definido */
  while ((character=getc(input)) != (unsigned)EOF)  {
    if (++i==1000){                          /* Imprime un * cada 1000        */
                                             /* caractees de entrada. Esto es */
      i=0;                                  /* solo un indicador             */
      printf("*"),
    }
    index=find_match(string_code,character);/* Analiza si la cadena esta en  */
    if (code_value[index] != -1)             /* la  tabla.  Si es asi        */
      string_code=code_value[index];         /* lo da el valor del código  Si */
    else {                                    /* no aparece                    */
                                             /* intentará adicionarla.        */
      if (next_code <= MAX_CODE)  {
        code_value[index]=next_code++;
```

```
        prefix_code[index]=string_code;
        append_character[index]=character;
      }
      output_code(output,string_code);  /* Cuando una cadena que no esta       */
      string_code=character;            /* en la tabla es encontrada           */
    }                                   /* Se saca la ultima cadena de la tabla*/
  }                                     /* y despues se adiciona la nueva      */
/** Fin del Loop principal.*/
  output_code(output,string_code);
  output_code(output,MAX_VALUE);
  output_code(output,0);
  printf("\n");
}

/** Rutina que intenta encontrar el prefijo+caracter optimo
** en la tabla de traduccion.  Si este es encontrado, el
** indice es regresado. Si no es encontrado, el primer
** indice disponible en la tabla se regresa */
find_match(int hash_prefix,unsigned int hash_character)
  int index;
  int offset;

  index = (hash_character << HASHING_SHIFT)  hash_prefix;
  if (index == 0)
    offset = 1;
  else
    offset = TABLE_SIZE - index;
  while (1)  {
    if (code_value[index] == -1)
      return(index);
    if (prefix_code[index] == hash_prefix &&
        append_character[index] == hash_character)
      return(index);
    index -= offset;
    if (index < 0)
      index += TABLE_SIZE;
  }
}

/**  Rutina de Descompresión. Toma el archivo test.out
**  y lo descomprime en test.out. */
expand(FILE *input,FILE *output)
  unsigned int next_code;
  unsigned int new_code;
  unsigned int old_code;
  int character;
  int counter;
  unsigned char *string;
  char *decode_string(unsigned char *buffer,unsigned int code);
  next_code=256;            /* Este es el siguiente codigo disponible a definir */
  counter=0;                * Este contador sirve para el tratamiento *         */
  printf("Descomprimiendo. .\n");
  old_code=input_code(input);  /* Lee el primer código, inicializa la       */
  character=old_code;          /* variable caracter, y la envia al primer   */
  putc(old_code,output);       /* codigo para la salida a archivo test.out */
/**  Esta es la rutina principal de Descompresion, nuevamente es un loop.
**  Lee el caracter del archivo comprimido hasta que encuentra el código
** especial usado para indicar el fin de los datos */
  while ((new_code=input_code(input)) != (MAX_VALUE)) {
    if (!(counter==1000))    /* Esta seccion imprime un *          */
    {                        /* cada 1000 caracteres               */
      counter=0;             /* Es solo un indicador               */
      printf("*");
    }
/** Este es el codigo que checa la entrada del carácter
** especial bxkjq o bien  STRING+CHARACTER+STRING+CHARACTER+STRING
** Se maneja decodificando el último codigo y adicionando un simple caracter al final de la
** cadena decodificada */
    if (new_code>=next_code) {
      *decode_stack=character;
      string=decode_string(decode_stack+1,old_code);
    }
/** Si no aparece esta cadena especial se asigna un nuevo codigo (case normal) **/
    else
      string=decode_string(decode_stack,new_code);
/** Ahora se sacará la cadena decodificada en orden inverso **/
```

```
    character=*string;
    while (string >= decode_stack)
      putc(*string--,output);
/** Finalmente, si es posible, adicionar un nuevo código a la tabla de traducción. */
    if (next_code <= MAX_CODE){
      prefix_code[next_code]=old_code;
      append_character[next_code]=character;
      next_code++;
    }
    old_code=new_code;
  }
  printf("\n");
}

/** Esta rutina simplemente decodifica una cadena de la tabla de traducción,
** almacenándola en un buffer. El buffer puede entonces ser sacado en orden inverso
** para la descompresión */
char *decode_string(unsigned char *buffer,unsigned int code){
  int i;
  i=0;
  while (code > 255) {
    *buffer++ = append_character[code];
    code=prefix_code[code];
    if (i++>=4094){
      printf("Fatal error during code expansion.\n");
      exit();
    }
  }
  *buffer=code;
  return(buffer);
}

/** Las siguientes dos rutinas se usan para sacar la longitud de la variable del código.
**  Estas se escriben estrictamente para la claridad del metodo y no para mejorar
**  la eficiencia del mismo */
input_code(FILE *input){
  unsigned int return_value;
  static int input_bit_count=0;
  static unsigned long input_bit_buffer=0L;

  while (input_bit_count <= 24){
    input_bit_buffer |=
        (unsigned long) getc(input) << (24-input_bit_count);
    input_bit_count += 8;
  }
  return_value=input_bit_buffer >> (32-BITS);
  input_bit_buffer <<= BITS;
  input_bit_count -= BITS;
  return(return_value);
}


output_code(FILE *output,unsigned int code){
  static int output_bit_count=0;
  static unsigned long output_bit_buffer=0L;
  output_bit_buffer |= (unsigned long) code << (32-BITS-output_bit_count);
  output_bit_count += BITS;
  while (output_bit_count >= 8)  {
    putc(output_bit_buffer >> 24,output);
    output_bit_buffer <<= 8;
    output_bit_count -= 8;
  }
}
```

A.5 Pruebas, Resultados y Conclusiones

Para ver el comportamiento del algoritmo LZW, se hicieron 2 pruebas diferentes: la primera usando una base de datos con información poco redundante, es decir no repetida, más que en ciertos campos y de manera poco frecuente, la segunda con diferentes tipos de archivos como son textos, binarios, imágenes, etc. Los resultados se muestran a continuación:

Prueba No. 1

Se cuenta con una base de datos con información no redundante, que es frecuentemente actualizada para ser enviada a diferente estaciones de trabajo. Las características de la base de datos son tales que presentan poca repetición en sus registros y solo coinciden con poca frecuencia en algunos campos. Para realizar la prueba, se tomaron 16 muestras de dicha base de datos, conteniendo un número de registros variante de 0 a 5000; esto es con el fin de que al variar el número de registros varía también el tamaño del archivo. La tabla A.1 muestra el número de registros contenidos en la muestra, el tamaño del archivo en bytes antes y después de aplicar el algoritmo y su razón de compresión que nos indica si el método es eficiente o no para determinada muestra.

No. de Registros	Tamaño Normal (en Bytes)	Tamaño comprimido (en Bytes)	Razón de compresión
0	98,304	378,64	2 60
487	229,376	154,464	1 48
981	294,912	260,277	1 13
1,700	393,216	364,197	1.08
2,140	491,520	494,854	0 99
2,498	589,824	583,216	1 01
3,225	655,360	611,766	1 07
3,700	753,664	737,515	1 02
4,100	819,200	700,629	1 17
4,225	851,968	766,489	1 11
4,380	917,504	785,274	1 17
4,480	950,272	851,301	1 12
4,680	1,015,808	919,888	1 10
4,898	1048,576	943,678	1 11
4,950	1,081,344	978,057	1 11
5,000	1,114,112	1,098,040	1 01

tabla A.1 Muestras de una base de datos con diferentes números de registro y su comportamiento al comprimir.

El promedio resultante de la razón de compresión fue de *1 205*, lo que nos indica que el método es aceptable para esta base de datos.

La quinta fila de la tabla es la única menor a uno en su razón de compresión, indicándonos que el archivo comprimido es mayor que el archivo sin comprimir, esto sucede porque los datos en la tabla de *traducción* se *incrementan* a medida que nuevos caracteres son añadidos a ella y no encuentra repeticiones que pueda sustituir por las claves previamente asignadas mientras se analizaba el archivo, sin embargo este dato no afecta la eficiencia del método durante las siguientes pruebas de la muestra.

En la figura A.2 se gráfica el número de registros contra la razón de compresión, observando el comportamiento del método; la curvatura que al principio tiende a ser exponencial, se va estabilizando conforme aumenta el número de registros, pasando en algún momento por debajo de la eficiencia del método (menor que 1), sin embargo, pese a que los archivos presentan poca repetición en la información, el algoritmo es aceptable; esto se nota al sacar el promedio de la razón de compresión (1.205), *también graficado en una recta, el cual está* por encima del valor de la razón de compresión obtenido de manera individual para cada archivo. Si bien se puede observar que a mayor número de registros es menor la razón de compresión esto resulta explicable por el tipo de datos que contienen los archivos de la muestra. Si se tiene una base de datos redundante, como se verá en el ejemplo siguiente, el comportamiento es diferente.

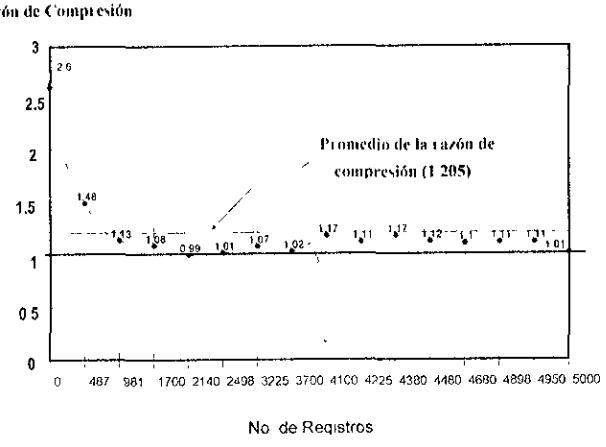


figura A.2 Razón de compresión para muestras en un base de datos no redundante.

Prueba No. 2

También se analizaron 6 diferentes tipos de archivos con coincidencia en tamaño, con una variación de ± 100 Bytes, para homogeneizar los resultados. Dicha prueba sirve para validar la eficiencia del método no solo en bases de datos o archivos de información, sino en imágenes y aun en archivos binarios. La tabla A.2 muestra el tipo de archivo que sirvió de muestra, su tamaño en bytes antes y después de ser comprimido y su razón de compresión que es el indicador de eficiencia del método.

Tipo de Archivo	Tamaño Normal (en Bytes)	Tamaño comprimido (en Bytes)	Razón de compresión
Gráfico (BMP)	1,030,086	366,798	2.81
Documento (DOC)	1,030,656	456,739	2.26
Aplicación (EXE)	1,089,300	1,045,380	1.04
Binario (BIN)	1,022,612	1,307,394	0.78
Base de Datos redundante (DBF)	1,013,521	172,224	5.88
Base de Datos no redundante (DBF)	1,079,648	621,156	2.56

tabla A.2 Muestras de diferentes archivos y su comportamiento al comprimir.

Como se puede observar, el algoritmo LZW sólo falla en el tipo de archivo binario, no quiere decir con esto que no sea aceptable, pero si hace resaltar que presenta limitantes para cierto tipo de información. Sin embargo en los archivos de interés que son los que contienen información, resultan muy eficientes. Esto es muy notorio en la base de datos, (similar en estructura a la de la prueba anterior pero redundante en su contenido) la que dio la mayor razón de compresión de todos los archivos analizados. Otros archivo que presentaron buenos resultados fueron el tipo imagen BMP y el documento DOC. La figura A.3 ilustra mejor estos resultados.

Como se Puede observar es realmente complicado caracterizar los resultados de esta técnica de compresión de datos. El nivel de compresión depende en gran medida de los datos que contiene el archivo a analizar. El método es muy ventajoso cuando se comprimen datos que tienen un alto grado de cadenas repetidas (redundancia), es en este punto donde podemos hablar de la eficiencia del método, es decir a mayor número de cadenas repetidas, mayor eficiencia; con este antecedente podemos decir que este método de compresión es útil para comprimir archivos de texto y de datos aun cuando no sean redundantes y el promedio de la razón de compresión se puede esperar del orden de 2 o más. Igualmente en archivos de imágenes, estos pueden ser generados con buenos resultados. Intentar comprimir archivos binarios o de aplicación es más arriesgado. Dependiendo de los datos, la compresión puede o no rendir buenos resultados. En algunos casos, los archivos se comprimen aun con más tamaño (razón de compresión menor que 1). Un poco de experiencia generalmente dará la norma sobre que archivos son aptos o no para aplicar la técnica

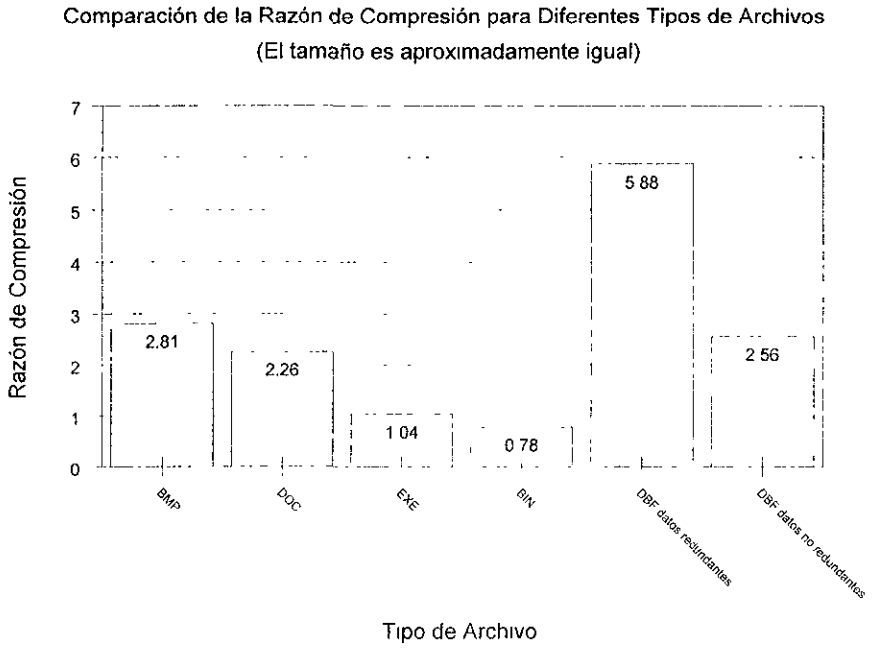


figura A.3 Gráfica de la razón de compresión para los diferentes archivos analizados.

Apéndice B. Sistema Operativo UNIX[xvii]

B.1 Historia y Desarrollo

Aunque son diversos los distribuidores de UNIX, el sistema operativo se puede clasificar en dos grandes categorías:

- El System V.- Esta versión es la desarrollada por AT&T, los verdaderos creadores de UNIX.
- UNIX BSD.- Relacionada con el trabajo de los estudiantes de la Universidad de Berkeley en California

A continuación, se explica a grandes rasgos el surgimiento de este Sistema Operativo

En 1969 Ken Thompson, un programador empleado en AT&T Bell Laboratories coordinando a un equipo de trabajo, desarrollaron MULTICS, un sistema operativo multiusuario, y a manera de broma, lo bautizó como UNIX, ya que era una ridiculización de lo que el sistema pretendía no ser. Es en ese año donde inicia la historia de UNIX.

Berkeley Software Distribution de la Universidad de Berkeley en California, en 1978, casi 10 años después, da a conocer su primera versión de UNIX, basada en la versión 7.0 de AT&T. Esta versión es una mejora hecha por estudiantes que previamente habían adoptado a este sistema operativo como su favorito. Hasta hoy en día el BSD es uno de los sistemas más aceptados, y aunque no es cien por ciento compatible con UNIX, si ha incluido mejoras en su interfaz que lo hace ser más accesible para el usuario común. La tendencia es utilizar la plataforma Intel para abarcar un mercado mucho mayor ya sea con el FreeBSD o con el Linux, - otro clon de UNIX - que son completamente gratuitos.

Otros intentos de crear clones de UNIX son por ejemplo el Coherent de Mark William's Company, el XENIX de Microsoft que tuvo su auge y su caída a finales de los 70 y principios de los 80, aunque resurge posteriormente con ayuda de AT&T en una fusión que se denominó System V/386 para procesadores Intel. IBM introduce al mercado el AIX, pero comercialmente hablando dejó mucho que desear.

System V es el estándar a adoptar para estaciones de trabajo más potentes que el 386 y surge de la unión de AT&T con Sun Microsystem, que liberan la versión 4.0 y posteriormente la 4.2 con interfase gráfica. Hoy en día es la que más predomina en los ambientes multiusuario y multitareas.

Para concluir con esta breve historia cabe señalar que el futuro de UNIX ahora esta ligado con Novell, Inc., ya que en 1993 AT&T, le vende los derechos sobre UNIX tanto en código como en marca registrada.

B.2 El Núcleo, el Shell y el Sistema de Archivos

B.2.1 El Núcleo

Todas las operaciones referentes a los procesos, están presentes en el núcleo, UNIX contiene rutinas en el núcleo que entre otras cosas se encargan de:

- Creación y destrucción de procesos
- Sincronización de procesos
- Comunicación entre procesos
- Manipulación de los bloques de control de procesos (PCB's)
- Administración de la memoria de los procesos
- Administración de las Entradas/Salidas
- Administración del reloj para el quantum de tiempo de procesos

En pocas palabras el núcleo es de vital importancia para el desarrollo del proyecto, el cual encuentra apoyo en las funciones referentes a los procesos, el núcleo suele ser un programa pequeño que radica completamente en memoria todo el tiempo que el sistema está activo. Como ya se mencionó, la mayor parte del trabajo del núcleo en UNIX radica en la administración de procesos concurrentes. El núcleo se encarga de ejecutar las funciones solicitadas por estos procesos, denominadas llamadas al sistema, estas llamadas pueden o no ser privilegiadas, es decir pueden o no apropiarse del sistema, por lo general esto se verifica en los derechos con los que el usuario haya entrado a la sesión.

B.2.2 El Shell

El shell en UNIX es un mecanismo de interfase entre el usuario y el sistema. Este es un interprete de comandos que lee las líneas escritas por el usuario y realiza la ejecución de las mismas.

El shell es una aplicación como cualquier otra y no es una función del núcleo. Los shell personalizados son frecuentemente escritos o creados. Los sistemas UNIX por lo regular soportan varios diferentes shell. El shell no reside permanentemente en memoria principal como el núcleo este puede intercambiar segmentos de memoria (swapping) si es necesario.

Tres de los más populares son: el shell Buorne (en el archivo sh), el shell Berkeley c (en el archivo csh) y el shell korn (en el archivos ksh).

B.2.3 El Sistema de Archivos

Una de las partes más importantes con la que el usuario tiene mayor contacto, es el sistema de archivos, el sistema de archivos muestra y manipula la información de un usuario de acuerdo con sus requerimientos. Los archivos son el modo abstracto del sistema para manipular la información del usuario, son una manera de almacenar grandes cantidades de información en un disco para que luego puedan ser utilizadas. Estas operaciones de creación y recuperación deben de hacerse de una manera transparente de tal manera que el usuario quede protegido contra los detalles de forma y lugar de almacenamiento de la información, así como del funcionamiento real de los discos.

Cuando un proceso crea un archivo se le da un nombre. Cuando el proceso concluye el archivo sigue existiendo. UNIX distingue entre las letras mayúsculas y minúsculas.

UNIX utiliza el tipo de estructura denominado *Series de Bytes*, dicha disposición es una secuencia no estructurada de bytes, en este caso al sistema operativo no le interesa el contenido del archivo. Lo único visible son los bytes; el significado se lo dan los programas o procesos que utilizan al archivo. El hecho de que UNIX considere a los archivos como una secuencia de bytes proporciona una flexibilidad absoluta. Los programas de usuario pueden colocar lo que sea dentro de los archivos y nombrarlos de la manera que crean más conveniente. UNIX no realiza algún tipo de manipulación sobre los archivos, por lo cual el usuario puede realizar las operaciones que quiera sobre esta estructura de archivo.

UNIX soporta varios tipos de archivos: archivos regulares, archivos especiales de caracteres y archivos especiales de bloques. Los directorios son archivos del sistema que se utilizan para dar seguimiento a la manera como se encuentran realmente colocados todos los archivos dentro del disco (Regulares, directorios, especiales de caracteres y de bloques). Los archivos regulares son aquellos que contienen información del usuario y son generalmente archivos ASCII (o en otro código de caracteres como EBCDIC) o binarios, estos últimos no son comprensibles o interpretables a simple vista. Los archivos especiales de caracteres tienen relación con las operaciones de entrada/salida y se utilizan para modelar dispositivos seriales de entrada/salida, tales como las terminales o las impresoras. Los archivos especiales de bloques se utilizan para modelar básicamente discos.

Aunque desde el punto de vista técnico, el archivo solo es una secuencia de bytes, el sistema operativo sólo ejecuta un archivo que contenga el formato adecuado. En UNIX el formato tiene 5 secciones:

1. Encabezado
2. Texto
3. Datos
4. Bits de reasignación
5. Tabla de símbolos

El encabezado inicia con un número especial denominado número mágico, el cual identifica el archivo como archivo ejecutable, esto es para evitar la ejecución o el intento de ejecución de un archivo que no se encuentre en ese formato. Después vienen varios enteros de 32 bits con los tamaños de las distintas partes de los archivos, la dirección de inicio del programa ejecutable y algunos bits de estado. Después del encabezado vienen el texto y los datos del mismo programa. Estos se cargan en memoria y se reasignan mediante los bits de reasignación. La tabla de símbolos se utiliza durante la depuración.

Otra de las características del sistema de archivos UNIX es que son archivos de acceso aleatorio es decir sus bytes o registros pueden leerse en cualquier orden. Esto permite que muchas aplicaciones modernas lo utilicen para fines de búsquedas de grandes volúmenes de información (Aplicaciones de misión crítica); por ejemplo en los grandes sistemas de bases de datos.

Se utilizan dos operaciones para realizar una lectura en un archivo de acceso aleatorio, primeramente una operación de búsqueda (SEEK) coloca el apuntador de registro en el registro deseado, posteriormente una operación de lectura (READ) obtiene la información del archivo.

B.2.4 Atributos de un archivo

Los archivos en UNIX tienen varias características entre las cuales las más conocidas son su nombre y tipo de datos. Además, asocia información adicional a cada archivo; por ejemplo, su tamaño, la fecha y hora de su creación. La lista de atributos de un archivo varía de un sistema operativo UNIX a otro también UNIX. Aquí mostramos una lista de los atributos considerados como estándares dentro de UNIX.

Campo	Significado
Contraseña	Clave de acceso al archivo
Creador	Identificador de la persona que creó el archivo
Propietario	Propietario actual
Bandera Solo lectura	0 Lectura/escritura, 1 solo lectura
Bandera Archivo escondido	0 Visible, 1 invisible
Bandera binario	0 ASCII, 1 binario
longitud del registro	En caso de sistemas de archivos de bases de datos
longitud de la llave	En caso de sistemas de archivos de bases de datos
posición de la llave	En caso de sistemas de archivos de bases de datos
Creación	fecha de creación del archivo
Ultima modificación	fecha más actual en que fue accedido el archivo
Ultima referencia	fecha en que se hizo la más actual referencia a ese archivo

tabla B.1 Atributos de archivo en el entorno UNIX.

Las banderas son campos de 1 bit que controlan o permiten ciertas propiedades determinadas.

Los campos de la longitud del registro, posición de la llave, y longitud de la llave están presentes en los archivos en cuyos registros se pueden hacer búsquedas mediante una llave. Estos campos proporcionan la información necesaria para encontrar las llaves de manera eficiente.

Los campos de creación, última modificación y última referencia son útiles por varias razones. Por ejemplo, un archivo fuente que ha sido modificado después de crear el ejecutable debe volverse a compilar.

En la actualidad existen además del sistema de archivos tradicional de UNIX otros como el NFS o sistema de archivos de red para control de archivos a través de una red, manteniendo las propiedades descritas en el sistema de archivos tradicional pero con la potencia de enlaces a través de una red no importando la plataforma en la cual se esté trabajando. Otro de los sistemas que ha dado más vida a UNIX es el sistema de archivos remoto, muy útil en los sistemas distribuidos y su característica principal es que es independiente del protocolo o protocolos que maneje el sistema distribuido.

B.2.5 Administración de Procesos

El término proceso, por cierto muy ligado a la creación de UNIX, encierra varias definiciones:

Un proceso es una secuencia temporal de ejecuciones de instrucciones que corresponden en su totalidad a la ejecución de un programa secuencial o modular[XVIII], también puede definirse como una entidad dinámica que nace cuando es cargada a la memoria física, y muere al finalizar la ejecución del programa[XIX]; otra definición común de proceso es la de una actividad asíncrona, el centro de control de un procedimiento.[XX]

De estas definiciones la más adecuada (y más repetida) es la de programa en ejecución. Cuando existen varios procesos en ejecución dentro de una máquina, dichos procesos se encuentran en diferentes estados dependiendo de la carga del procesador y de las actividades en sí que realice el proceso durante su ejecución, estos estados son los siguientes:

- En ejecución: las instrucciones se están ejecutando.
- Bloqueado: el proceso está esperando a que ocurra algún evento.
- Listo: el proceso está esperando a que se le asigne a un procesador.

La transición de dichos estados es generada por el despachador que entre otras cosas realiza:

- el cambio de contexto del proceso que no es otra cosa más que la transición de un estado a otro por medio de una interrupción
- el cambio del procesador o procesadores de modo núcleo o privilegiado a modo usuario
- el reinicio del proceso en el mismo lugar donde se le expropió el procesador o procesadores

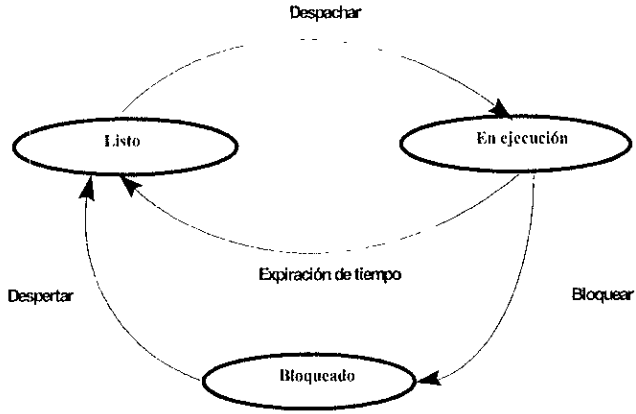


figura B.1 Transición de estados. de los procesos

Debido a que las tareas del despachador son críticas este debe ejecutarlas lo más rápido posible para que no exista una baja en el rendimiento.

Para poder ejecutar varios procesos dentro de una misma computadora UNIX se debe llevar el control estricto de cada una de las actividades que realizan los mismos, por lo cual se debe establecer una estructura de datos que conserve el estado de todas las actividades de un proceso para que cuando sea cambiado el control del procesador a otro proceso no se pierda el avance del proceso al que se le expropia el procesador. La estructura que contiene los datos necesarios para restablecer la ejecución de un proceso cuando se le vuelve a asignar el procesador se conoce como bloque de control del proceso, PCB (siglas de Process Control Block) o descriptor de procesos.

El PCB contiene fundamentalmente la siguiente información.

1. El estado del proceso.- Listo, Bloqueado y En ejecución
2. Identificador o número del proceso.- es un número entero que le indica al sistema operativo cual es el proceso en cuestión, es decir, es el nombre del proceso.
3. Apuntador hacia el proceso creador.- Es decir, al proceso que lo lanzó; dentro de un ambiente donde se permite la ejecución de múltiples procesos existen mecanismos especiales para la creación de los mismos, en general, un proceso es creado por otro proceso, al creador se le denomina padre, y al proceso creado se le denomina hijo, por lo cual es importante conservar la liga entre un proceso y su proceso padre.
4. Apuntador hacia los procesos derivados.- Es decir, procesos lanzados por él ; lo mismo que aplica a los padres aplica a los hijos, esta liga se conserva para mantener un tipo de supervisión sobre los hijos.
5. Prioridad del proceso.- Dentro de un sistema pueden existir niveles donde cada nivel tiene asignada una cantidad determinada de recursos, estos niveles son formados por las prioridades.
6. Registros del procesador incluyendo el contador del programa.- Se deben conservar los registros del procesador para que cuando se le devuelva el control al proceso este continúe con su ejecución exactamente en el lugar donde quedó anteriormente.
7. Información de administración de memoria (registros de borde o frontera, tablas de paginación, etc.).- Para poder restablecer su ejecución también es necesario conocer que áreas de memoria tenía asignado el proceso antes de que se le expropiara el procesador, en la lista de límites de memoria generalmente se incluye también los niveles de autorización a las mencionadas áreas.
8. Lista de recursos asignados (tiempos, recursos, etc.).- Se debe proporcionar la lista de recursos asignados para poder continuar con la ejecución, ya que aunque se le expropie el procesador no se deben expropiar los recursos dedicados tales como unidades de respaldo, u otro tipo de recursos que solo pueden ser utilizados por un proceso a la vez.
9. Información del estado de la E/S (archivos abiertos, dispositivos asignados, operaciones pendientes, etc.).- Se conserva una lista de archivos abiertos para conservar todas las referencias de actualización o de lectura de los mismos.
10. Información contable.- En los sistemas donde se realiza algún tipo de cobro o se lleva un control estricto de los recursos asignados se incluyen campos para la información contable, estos campos registran la cantidad de tiempo de procesador utilizado, cuenta utilizada, límites de tiempo, etc.

El algoritmo comúnmente utilizado en UNIX para la planificación de los procesos concurrentes, es el de *Asignación en Base a Prioridades*, aunque cabe mencionar, no es el único. Este algoritmo tiene las siguientes características:

A cada proceso se le asigna un número de prioridad que va de -20 a 20 donde -20 es la prioridad más alta y 20 la más baja.

Se planifica el CPU para aquel proceso con la prioridad más alta.

Las prioridades se van ajustado con base en cálculos hechos por medio de las condiciones que va presentando el problema, estos cambios pueden ser peticiones de servicios de impresión, de discos, de memoria, de apropiación del procesador o procesadores. Estos ajustes se van sumando a las prioridades para generar una prioridad actual. Como se puede ver, este algoritmo beneficia a procesos que utilizan muy poco el procesador o no han generado peticiones, sin embargo para evitar esto, UNIX aumenta la prioridad en todos los proceso a medida que pasa el tiempo. La planificación está enfocada a lograr una producción máxima reduciendo al mínimo el costo extra.

Otros componentes importantes dentro del sistema operativo para el manejo de procesos son el reloj o relojes y el quantum o porción de tiempo que el sistema asigna a estos, con el fin de evitar que ciertos procesos monopolicen el uso del procesador, o bien se caiga en bloqueos mutuos, también para salvaguardar que dichos procesos utilicen la exclusión mutua cuando caen en secciones críticas.

Una forma de aprovechar la utilización del procesador para el despacho de procesos sobre todo los independientes es por medio de hilos (threads) o procesos ligeros. Un hilo es una unidad básica de utilización de CPU, con esto se pretende tener varias unidades de procesamiento en el mismo espacio virtual de direcciones, que es mucho más eficiente que usar espacios de direcciones individuales, todos los hilos que se generan pertenecen al mismo proceso y en condiciones normales de trabajo no representan ningún riesgo para el sistema. El entorno de ejecución de un hilo se le llama tarea.

La tarea consiste de un contador de programa, el conjunto de registros y el espacio de pila.

La diferencia entre un proceso ligero y uno pesado es que el proceso pesado o tradicional es una tarea con un solo hilo.

En una tarea que contenga múltiples hilos, estos comparten la sección de código, la sección de datos y los recursos del sistema operativo, además mientras un hilo servidor está bloqueado y esperando, otro hilo en la misma tarea puede ejecutarse. El uso de múltiples hilos cooperantes, confiere mayor productividad y mejora las prestaciones, y genera un cambio de contexto más rápido.

Otra forma de aprovechar y explotar el manejo de la administración es usar los procesos especiales que contiene UNIX, existe un tipo de proceso que es muy útil por sus características particulares, conocido como *daemon* o demonio y tiene la propiedad de que siempre permanece en el sistema en un estado bloqueado o listo y se ejecuta automáticamente por medio de algún señalamiento sin intervención directa del usuario, este señalamiento puede provenir del reloj interno del sistema o bien de la ejecución de algún programa del núcleo. Si bien el usuario no interviene directamente como se mencionó si puede hacerlo indirectamente desde el shell con un proceso *daemon* bastante conocido denominado *cron* para la planeación de ejecución de comandos, este proceso mantiene 6 parámetros los primeros cinco especificando minuto (0-59), hora (0-23), día del mes (0-31), mes del año (0-12), día de la semana (1-7) y el último parámetro especificando simplemente el comando a ejecutar. Con esta forma de planeación se pueden ejecutar comandos controlados por períodos de tiempo y es útil en aquellos sistemas donde las horas pico del día no son las adecuadas y es preferible hacerlo en la noche

o madrugada dejando que el *cron* sea quien ejecute dichos comandos. Los seis parámetros de cron se guardan en un archivo denominado *crontab* que puede ser editado conforme el administrador lo requiera. Veamos un ejemplo de un archivo *crontab*:

```
30  16  *  *  1,2,3,4,5,   /bin/su/       root -c "sbin/cleanup > /dev/null"
```

Este archivo ejecuta el comando *cleanup* a la raíz, con su salida estándar a nulo a las 16:30 hrs. de lunes a viernes.

B.3 Comunicación entre procesos

La ejecución de procesos concurrentes en UNIX tiene las siguientes características:

- Los procesos comparten recursos físicos y lógicos
- Permiten acelerar los cálculos.
- Presentan características de modularidad.

Estas características deben de ser bien cuidadas para mantener la estabilidad del sistema y sobre todo para lograr que UNIX responda de manera eficiente, sobre todo en aquellas acciones que involucren el traspaso de información

Un buen comienzo para lograr la estabilidad es conocer como se relacionan los procesos y las diferentes formas de comunicación que existen entre ellos. En general se tienen 2 clases diferentes de procesos: Proceso independiente y proceso cooperativo:

El proceso independiente tiene las siguientes características:

- *No puede verse afectado por la ejecución de otro proceso.*
- *Su ejecución es determinista.*
- *Su estado es privado.*
- *Su ejecución es reproducible.*
- *Su ejecución puede detenerse y reiniciarse*

El proceso cooperativo tiene las siguientes características:

- *Afecta o se ve afectado por otros procesos.*
- *Su estado es compartido.*
- *Su ejecución no es determunista*

Existen diversas formas de comunicación entre procesos; a continuación se hablará de dos muy necesarias en la elaboración del proyecto y que suelen ser las más aplicadas en términos de forma de comunicación en el entorno UNIX

B.3.1 Conductos sobre UNIX

Los conductos o *pipes* se utilizan para la comunicación entre procesos, mediante un flujo de información unidireccional. Cuando un conducto es creado se obtienen dos descriptores de archivos, uno para lectura y otro para escritura. .Un diagrama de una apariencia de un pipe como en un proceso único se muestra en la figura B 2 .

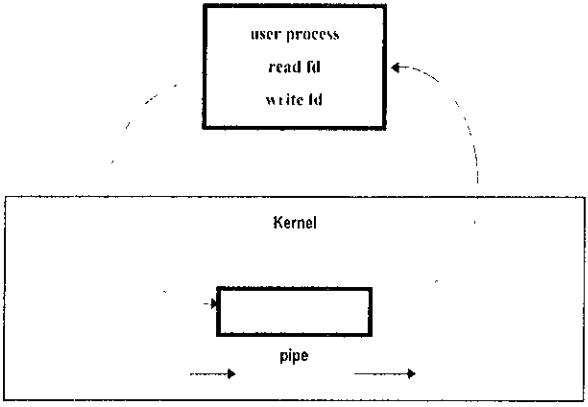


figura B.2 Diagrama del proceso de comunicación a través de *pipes*.

La creación de un conducto se realiza mediante el llamado a la función pipe, la cual recibe como parámetro los descriptores de archivos y retorna un código de error indicando el estado de la operación. Un ejemplo de creación de un conducto es el siguiente:

```
int pipe (int *fldes);
```

Con este llamado se obtienen dos descriptores de archivos:

fldes[0] abierto para lectura y

fldes[1] abierto para escritura

Mediante estos descriptores se puede llevar a cabo el intercambio de información.

Los conductos son utilizados para llevar a cabo la comunicación entre procesos de la siguiente manera: Inicialmente un proceso crea un conducto y luego crea uno o más procesos hijos (mediante la instrucción **fork**), luego de esto, el proceso padre cierra el descriptor de lectura del conducto mientras que el proceso hijo se encarga de cerrar el descriptor de escritura del mismo, con lo cual se obtiene una comunicación unidireccional entre los procesos.

La gran desventaja de utilizar conductos para la comunicación de procesos es que éstos deben tener un proceso padre en común. Lo anterior se debe a que un conducto es pasado de un proceso a otro mediante la utilización de la instrucción **fork**, la cual hace que el proceso hijo herede todos los archivos, del padre, abiertos antes del llamado a la función.

Otra de las desventajas de los conductos, es el hecho de no poder comunicarse con procesos que no tengan otro proceso en común. Un mecanismo para poder llevar a cabo la comunicación de procesos sin parientes en común, es el de dar nombres a los conductos para que cada uno de los procesos efectúe lecturas o escrituras. A estos conductos a los que se les asigna un nombre se les conoce como *named*

pipes o FIFOs. La sigla FIFO se debe a la frase en ingles *First In First Out*, la cual significa que el primer dato en entrar es el primer dato en salir.

Un FIFO puede ser creado con la instrucción *mknod* que recibe como parámetros el nombre del conducto y el modo de apertura, que indica los permisos de lectura y escritura sobre el recurso. Una vez creado un FIFO este debe ser abierto para lectura o escritura utilizando cualquiera de las funciones estándar de E/S: *fopen*, *freopen* u *open*.

Tanto los conductos sin nombre como los FIFOs siguen las siguientes reglas para lectura o escritura:

- Un requerimiento de lectura hace un pedido de datos inferior al que existe en el conducto, obteniendo la cantidad pedida. Los datos restantes permanecen disponibles para siguientes requerimientos de lectura.
- Si un proceso trata de leer más datos de los disponibles en el dispositivo, sólo la cantidad de datos disponibles es retornada.
- Si no hay datos disponibles en el dispositivo y ningún proceso tiene abierto el conducto en modo escritura, produce un cero en el momento de ejecutarse una operación de lectura, con lo cual se indica la terminación del archivo.
- Si un proceso escribe menos de la capacidad de un conducto (que por lo menos es de 4096 bytes) a este proceso se le garantiza que la escritura es atómica, con lo cual se quiere decir que si dos procesos tratan de escribir casi simultáneamente sobre el recurso, se les garantiza que los datos que cada uno escriba no se mezclarán, esto es no se intercalarán datos de uno y otro sino que por el contrario los datos de cada uno permanecerán contiguos. Si un proceso trata de escribir una cantidad de datos mayor a la capacidad del conducto no se le garantiza que la operación sea atómica.
- Si un proceso escribe en un conducto y no hay procesos que hayan abierto el recurso para lectura, se retorna un código de error y puede suceder que el proceso sea terminado.
- Cuando se ejecuta un proceso de lectura o escritura sobre un conducto y no existe un proceso del otro lado que haya abierto el recurso en el modo complementario, el proceso que ejecuta la operación de entrada/salida permanece bloqueado hasta tanto pueda completar la operación. Para evitar este comportamiento es posible abrir el conducto con la opción **ON_DELAY**, la cual hace que el proceso no permanezca bloqueado hasta completar la operación

B.3.2 Memoria Compartida

Considere los pasos normales que involucran la copia de un archivo en un modelo cliente - servidor.

- El servidor lee desde el archivo de entrada. Típicamente los datos son leídos por el buffer interno del cliente y copiado al buffer del servidor
- El servidor escribe estos datos en un mensaje.
- El cliente lee los datos desde el canal IPC, nuevamente requiriendo para que los datos se copien desde el kernel IPC al buffer del cliente.
- Finalmente los datos se copia desde el buffer del cliente, el segundo argumento hace la llamada al sistema write, al archivo de salida.

- Un total de cuatro copias de los datos son requeridos. Estas cuatro de copias se hacen entre el kernel y el proceso del usuario

El problema con estas formas de IPC --pipes, FIFOS, y trenzas de mensaje-- es para dos procesos que intercambian información, la información tiene que ir mediante el kernel. La memoria compartida provee una manera alrededor esto por dejar que dos o más procesos compartan un segmento de memoria. Existe un problema que involucra que procesos múltiples compartan un pedazo de memoria: los procesos tienen que coordinar el uso de la memoria entre sí mismos. Si un proceso lee en alguna porción de memoria compartida, por ejemplo, los otros procesos deben esperar a que finalice la lectura antes de procesar los datos. Afortunadamente este es un problema fácil para resolver, usando semáforos para la sincronización.

Los pasos para el ejemplo del modelo cliente - servidor de la copia de un archivo son:

- El servidor consigue acceso al segmento compartido de memoria que usa un semáforo.
- El servidor lee desde el archivo de entrada dentro del segmento compartido de memoria. La dirección para leer en el segundo argumento de la instrucción read, puntos en la memoria compartida.
- Cuando la lectura se completa el servidor notifica el cliente, nuevamente usando un semáforo. El cliente escribe los datos desde el segmento de la memoria compartida al archivo de salida.

B.3.3 Conectores o Sockets en UNIX

A continuación se describirán los APIs (Application Program Interfaces) más utilizados en UNIX para los protocolos de comunicación : los conectores o sockets.

El sistema de entrada/salida de UNIX desarrollado a finales de los años sesenta y principio de los setenta se basa en el paradigma "open-read-write-close". Antes de que un proceso usuario pueda ejecutar operaciones de E/S, llama a un "open" para especificar el archivo o dispositivo que se va a utilizar y obtiene el permiso. La llamada al open devuelve un descriptor de archivo (un entero) que el proceso utiliza cuando ejecuta las diferentes operaciones de E/S. Una vez el archivo ha sido abierto, se utilizan "read" o "write" para hacer la transferencia de datos entre el archivo y el proceso. Luego de haber realizado todas las transferencias de datos el proceso usuario llama a "close" para cerrar el archivo. Esto indica al sistema operativo que ya no se realizarán más operaciones sobre este archivo.

El esquema de E/S incluye el manejo de dispositivos en UNIX orientados a caracteres como el teclado y el manejo de dispositivos por bloques, como los discos.

Para describir los conectores, vamos a comparar el sistema de E/S para archivos en UNIX con el mecanismo de E/S para red también en UNIX. Para ello se tendrán en consideración algunos puntos como son :

- *La relación cliente-servidor no es simétrica.*
- *Las comunicaciones en redes pueden ser orientadas a conexión o no orientadas a conexión*
- *Las interfaces para las comunicaciones en redes pueden utilizar diferentes protocolos de red.*

Estas diferencias influyeron en la realización del API. Por ello, los diseñadores decidieron añadir algunas funciones que permitieran manejar estas diferencias, tales como el esquema cliente-servidor. En este esquema, el servidor es un proceso pasivo que espera requerimientos de un proceso cliente, y

cuando estos llegan los acepta y procesa. Por otro lado, el proceso cliente, es un proceso activo que realiza requerimientos al servidor.

B.3.4 El concepto de conector

La base de la E/S sobre redes en BSD UNIX son los conectores o sockets. Estos se consideran simplemente como una generalización del mecanismo de E/S sobre archivos en UNIX. Al igual que como con los archivos, los procesos usuario requieren crear un conector para realizar comunicaciones. Al llamar la función de creación, el sistema devuelve un descriptor (un entero), el cual será utilizado cuando se ejecuten operaciones que envuelvan el uso del conector. En algunos casos las operaciones de E/S sobre conectores son iguales a las realizadas sobre archivos (en el esquema orientado a conexión). Estos será explicado más adelante.

Para establecer las comunicaciones entre procesos utilizando conectores, es necesario definir primero el dominio. Este, especifica el formato de las direcciones que se podrán dar a los conectores y los diferentes protocolos soportados por la comunicación por medio de los conectores.

El dominio Internet sirve para establecer comunicaciones locales. Las direcciones de los conectores tienen la estructura *sockaddr_in* que se muestra a continuación:

```
struct sockaddr_in {
    short           sin_family;     /* Dominio Internet  AF_INET */
    u_short         sin_ port;      /* el numero del puerto.  */
    struct in_addr  sin_addr;       /* la direccion internet  */
    char            sin_zero[8];    /* un campo de ocho ceros. */
}
```

Por otro lado, los tipos de conectores pueden ser clasificados en .

SOCK_DGRAM.- Hace referencia a conectores destinados a la comunicación en modo no conectado para el envío de datagramas de tamaño limitado; cuando se envía un mensaje desde un conector con destino a otro conector, el que emite el mensaje no obtiene ninguna información sobre la llegada del mensaje a su destino. En el dominio *Internet* está asociado al protocolo *UDP*.

SOCK_STREAM.- Sirve para establecer comunicaciones confiables en modo conectado (ningún dato transmitido se pierde, los datos llegan en el orden que han sido transmitidos, sólo llega al destino una única instancia de un dato emitido y se establece una conexión entre dos puntos antes de empezar la comunicación.). En el dominio *Internet* está asociado al protocolo *TCP*.

SOCK_RAW.- Su uso está reservado al superusuario, permite la implantación de nuevos protocolos.

SOCK_SEQPACET.- Corresponde a las comunicaciones que se encuentran circunscritas en el dominio *XEROX NS*.

B.3.3.1 Creación de un conector

Para la creación de un conector se utiliza la primitiva :

```
descriptor = socket(familia, tipo, protocolo)
```

El parámetro *familia* especifica la familia de protocolo que se va a utilizar en la conexión, por ejemplo. TCP/IP. Los parámetros *tipo* y *protocolo* especifican el tipo de comunicación que se desea utilizar "orientado a conexión" o "datagramas".

Especificación de una dirección local

Inicialmente el conector se crea sin ninguna asociación hacia direcciones locales o de destino. Para los protocolos TCP/IP, esto significa que ningún puerto de protocolo local ha sido asociado al conector y que ningún puerto IP se ha especificado. En muchos casos, los programas de aplicación no se preocupan por las direcciones locales que utilizan. Sin embargo los procesos del servidor que operan en un puerto bien conocido deben ser capaces de especificar dicho puerto para el sistema, por ejemplo ftp. Una vez que ha sido creado un conector, el proceso servidor utiliza la primitiva *bind* para establecer una dirección local para ello. El llamado de la primitiva es :

```
bind(socket, dir-local, long-dir)
```

El parámetro *socket* es el descriptor del conector que se quiere asociar a una dirección local. El parámetro *dir-local* es una estructura que especifica la dirección local a la que el conector deberá asociarse y el tercer parámetro indica la longitud de las direcciones medidas en bytes.

Conexión de un conector con direcciones de destino

Inicialmente un conector se crea en el estado no conectado, lo que significa que el conector no esta asociado con ninguna dirección externa. La llamada a la primitiva *connect* crea la conexión con un destino permanente. El llamado a la primitiva es :

```
connect(socket, destino, long-dir)
```

A partir del momento que se realiza este llamado, se establece un canal de comunicación entre los procesos cliente y servidor.

B.3.3.2 Comunicación entre procesos utilizando conectores

Una vez establecido el canal de comunicación entre el cliente y el servidor, hay cinco llamados posibles a primitivas del sistema operativo para enviar datos a través de dicho canal : *send*, *sendto*, *sendmsg*, *write* y *writev*. Las primitivas *send*, *write* y *writev* sólo pueden ser utilizadas si el conector esta asociado con una dirección destino, en términos de TCP/IP sería si el conector es orientado a conexión como se logra apreciar en la figura 2. Las diferencias entres estas tres primitivas son muy pocas.

A continuación se describe la primitiva *write*.

```
write(socket, buffer, longitud)
```

Esta primitiva es la misma que se utiliza para el manejo de archivos en UNIX. Es decir, desde el punto de vista del proceso usuario no hay ninguna diferencia entre enviar datos por un canal de comunicaciones y una escritura a un archivo.

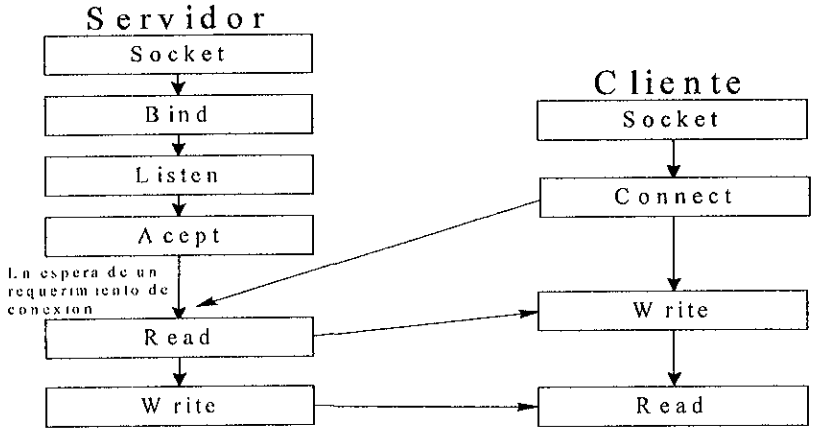


figura B.3 Esquema "Orientado a conexión" utilizando conectores.

Las primitivas *sendto* y *sendmsg* permiten la comunicación entre procesos que utilizan conectores no orientados a conexión, por ejemplo los datagramas, en términos de TCP/IP, como se logra apreciar en la figura B.3. Estas dos primitivas requieren que quien va a mandar un mensaje, especifique el destino.

El llamado a la primitiva *sendto* es :

```
sendto(socket, mensaje, longitud, flags, destino, long-dir)
```

El primer parámetro especifica el conector que se va a utilizar, el segundo especifica el mensaje a ser enviado, longitud especifica el número de bytes enviados, el campo *flags* se usa para controlar la transmisión, el destino especifica la dirección de destino del mensaje y longitud, la longitud de dicha dirección.

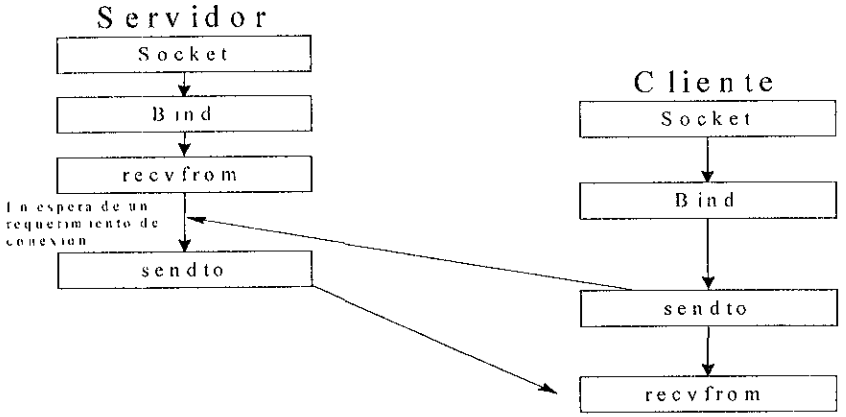


figura B.4 Esquema "No orientado a conexión - datagramas". utilizando conectores.

Hasta ahora hemos presentado la parte que tiene que ver con el envío de mensajes. A continuación presentaremos las primitivas relacionadas con la recepción de datos a través de un conector.

Para esto, los conectores ofrecen cinco primitivas (en forma análoga al envío) que son : *read*, *ready*, *recv*, *recvfrom* y *recvmsg*. La primitiva convencional de UNIX *read* sólo puede ser utilizada cuando se refieren a conectores orientados a conexión. El llamado es :

```
read(socket, buffer, longitud)
```

Finalmente, explicaremos la primitiva utilizada en conectores para admitir requerimientos de conexión en el caso orientado a conexión. Una vez que se ha establecido el conector, el servidor llega a un estado pasivo de espera (espera un requerimiento de conexión). Para hacer esto se utiliza la primitiva *accept*. Una llamada a esta función bloquea al proceso hasta que llegue la petición de conexión. La sintaxis de la primitiva es :

```
newsocket = accept(socket,direccion,long-dir)
```

El parámetro *socket* especifica el descriptor del conector en el que se va a esperar. El segundo parámetro es la dirección de una estructura donde están especificados información relacionada con el conector y el último argumento es la longitud de dicha estructura. Cuando llega una petición, el sistema llena el argumento dirección, con la dirección del cliente. Por último, el sistema crea un nuevo conector que tiene su destino conectado hacia el cliente que requirió dicha conexión y devuelve el nuevo descriptor al proceso que acepta la conexión.

B.3.4 Archivos y bloqueo de registros

Hay ocasiones que múltiples procesos quieren acceder el mismo recurso, para esto es esencial proveer de alguna forma de exclusión para que un único proceso pueda acceder el recurso.

B.3.4.1 Bloqueo Anunciado contra Bloqueo Obligado

Bloqueo anunciado, por medio del sistema operativo se tiene un conocimiento de aquellos archivos que son bloqueados y los procesos que los bloqueo. El sistema no prevé a otros procesos que desean escribir en el archivo que esta bloqueado por otro proceso. Un proceso puede ignorar un bloqueo anunciado y escribir en el archivo solo si tiene los permisos adecuados. La programación de daemons empleada para la programación en red es un ejemplo de procesos cooperativos, los programas que acceden a un recurso compartido esta bajo el control del administrados del sistema.

Bloqueo obligado, por medio del sistema operativo siempre se verifican los requerimientos de lectura y escritura sobre un archivo con el objetivo de verificar que la operación no interfiera con un bloqueo realizado por un proceso.

B.3.4.2 Bloqueo de Archivo versus el Bloqueo de Registro

El bloqueo por archivos bloquea un archivo entero, mientras que un bloqueo de registros es un proceso que permite bloquear un porción especifica de un archivo.

Apéndice C. Lenguaje Relacional Estándar

Las operaciones del álgebra relacional son muy importantes para comprender los tipos de peticiones que se pueden hacer a una base de datos relacional.

Por si mismas estas operaciones son generalmente clasificadas como un lenguaje de alto nivel, ya que se aplican a relaciones completas. El lenguaje más conocido del álgebra relacional es el SQL (Structured Query Language) que en sus inicios se llamó SEQUEL (Structured English Query Language) y fue diseñado e implementado en IBM como una interfase para un sistema de bases de datos experimental llamado SYSTEM R. Hoy en día se pueden encontrar diferentes versiones comerciales del mismo y dadas las características de los modernos sistemas, es frecuente mezclar este lenguaje con cualquier otro como C, C++, Pascal, etc.

C.1 Definición de Datos en SQL

SQL utiliza los términos tabla, renglón y columna para relación, tupla y atributo respectivamente. Los comandos para definir datos en este lenguaje son CREATE, TABLE, ALTER, TABLE y DROP TABLE los que a continuación se describirán:

- CREATE TABLE :Se usa para definir una nueva relación, dándole un nombre y definiendo sus atributos o columnas, cada columna recibe un nombre, el tipo de datos que recibirá, es decir su dominio y algunas características de condición para integridad referencial necesarias. Los tipos de datos son generalmente numéricos y cadenas de caracteres. De los primeros se pueden definir enteros (Integer), de punto flotante (Float), con número decimal o no. De los segundos se pueden tener de longitud fija o variable. Existen también los tipos Fecha (Date), Memo (Memo) para atributos más especiales.

  Ya que SQL permite el uso de valores NULOS (NULLS), una condición NO NULA (NOT NULL) se puede especificar a los atributos sobre todo si estos son llave primaria.

  A continuación dos ejemplos:

  Ejemplo 1

```
CREATE TABLE Empleados
(id_empleado INTEGER NOT NULL,
a_paterno VARCHAR (15) NOT NULL,
a_materno VARCHAR (15),
nombre VARCHAR (20),
sexo CHAR (1),
salario INTEGER,
seguro_social CHAR (9) NOT NULL
proyecto_asignado INTEGER)
Ejemplo 2
CREATE TABLE Proyectos
(nombre_del_proyecto VARCHAR (15) NOT NULL,
id_proyecto INTEGER NOT NULL,
localidad VARCHAR (15),
num_de_empleados INTEGER NOT NULL)
```

  Las relaciones creadas con la sentencia CREATE TABLE se denominan **tablas base** (o relaciones base) en la terminología SQL; esto significa que la relación y sus tuplas se crearán y almacenarán en un archivo por el DBMS. Las relaciones base se distinguen de las **relaciones virtuales**, creadas con la sentencia CREATE VIEW que puede o no coincidir con el archivo físico. En el lenguaje SQL los atributos en una tabla base se consideran ordenados en la secuencia en la cual fueron

especificados en la sentencia CREATE TABLE. Sin embargo los renglones o tuplas no necesariamente se consideran en algún orden.

- *DROP TABLE: Si se decide eliminar una relación en una* base de datos esto se logra con la sentencia DROP TABLE la cual borrará la definición de dicha relación.

  Ejemplo 3

  Si deseamos eliminar la relación "Proyectos" del ejemplo anterior sólo basta escribir:

```
DROPTABLE Proyectos
```

- ALTER TABLE: Si se decide agregar un atributo a una de las tablas base, se utiliza la sentencia ALTER TABLE. Este nuevo atributo contendrá NULLS en todas las tuplas existentes de la relación, por lo que la condición NOT NULL no se podrá aplicar en ese momento.

  Ejemplo 4

  Si deseamos incluir un atributo denominado "puesto" en la tabla "Empleados":

```
ALTER TABLE Empleados ADD puesto Varchar(15),
```

C.2 Consultas en SQL

SQL tiene una sentencia básica para recuperar información de una base de datos: SELECT; hay una gran diferencia entre el SELECT del álgebra relacional y el SELECT de SQL, éste último permite a una tabla tener dos o más tuplas con valores idénticos en sus atributos, sin embargo en el álgebra relacional un conjunto de tuplas no debe tener elementos iguales. Por lo tanto una relación o tabla SQL no es un conjunto de tuplas a menos que tenga la característica de multiconjunto de tuplas. Algunas relaciones SQL se condicionan usando el comando CREATE UNIQUE INDEX o bien la sentencia DISTINCT acompañada del SELECT.

Consultas básicas SQL

La forma básica de la sentencia SELECT es:

SELECT <Lista de atributos>

FROM <Lista de Tablas>

WHERE <Condición>

donde

<Lista de atributos>.- Es una lista de nombres de atributos cuyos valores serán recuperados por la consulta.

<Lista de Tablas>.- Es una lista de nombres de relación requeridos para procesar la consulta.

<Condición>.- Es una expresión condicional (booleana) que identifica las tuplas que serán recuperadas por la consulta.

Para ilustrar veamos los siguientes ejemplos:

Ejemplo 5:

Recupera el salario y el No. de seguro social de los empleados que se apelliden "García Solis":

```
SELECT salario, seguro_social
FROM   Empleados
WHERE  a_paterno='Garcia' AND a_materno='Solis'
```

Ejemplo 6:

Recupera el Apellido Paterno, Apellido Materno y Nombre de los Empleados que trabajan en el proyecto "Rescate Ecológico"

```
SELECT a_paterno, a_materno, nombre
FROM   Empleados, Proyectos
WHERE  nombre_del_proyecto='Rescate Ecológico' AND id_proyecto=proyecto_asignado
```

C.2.1 Especificando Nombres de Atributos iguales.

En SQL se puede usar el mismo nombre para dos o más atributos en diferentes relaciones, solo hay que *calificar* el nombre del atributo con el nombre de la relación para prevenir la ambigüedad, de manera que éste quede prefijo como lo ilustra el siguiente ejemplo.

Ejemplo 7:

Supóngase que el atributo proyecto_asignado de la relación Empleados recibiera el nombre id_proyecto, similar a la relación Proyectos; y de la misma manera el atributo nombre_del_proyecto de la Relación Proyectos recibiera el nombre, similar al atributo de la Relación Empleados; tendríamos del ejemplo 6 la siguiente consulta:

```
SELECT a_paterno, a_Materno, Empleados nombre
FROM   Empleados, Proyectos
WHERE  Proyecto.nombre='Rescate Ecologico' AND     Proyecto.id_proyecto=Empleados.id_proyecto
```

C.2.2 Tablas como conjuntos en SQL.

Como se menciono anteriormente, SQL no trata una Relación como un conjunto, es decir, las tuplas duplicadas pueden aparecer más de una vez en una Relación o en el resultado de una Consulta. SQL no elimina automáticamente estas tuplas. Para que se pueda realizar esto, se utiliza la palabra reservada DISTINCT, así solo tuplas diferentes se mantendrán en el resultado de la consulta.

Ejemplo 8:

```
SELECT DISTINCT       salario
FROM   Empleados
```

Sin la palabra DISTINCT, si varios empleados tienen el mismo salario, la consulta presentaría muchas tuplas presentando los salarios iguales, sin embargo al aplicar la palabra reservada, solo cada valor distinto aparecerá solo una vez, causando que el resultado sea una relación o tabla como un conjunto de tuplas.

C.2.2.3 Consultas Anidadas y Comparación de Conjuntos.

Algunas consultas requieren de valores previamente consultados y usarlos en una condición. A este tipo de consultas se les denomina consultas anidadas que no son otra cosa que consultas completas dentro de la cláusula WHERE de otra consulta que se denomina consulta de salida.

Ejemplo 9:

Generar una lista de todos los proyectos que involucran a empleados con apellido paterno o apellido materno "Pérez".

```
SELECT DISTINCT       nombre_del_proyecto
FROM   Proyectos
WHERE  id_proyecto IN        (SELECT id_proyecto
       FROM Proyectos, Empleados
       WHERE a_paterno='Pérez')
```

```
OR
id_proyecto IN        (SELECT proyecto_asignado
FROM Empleados
WHERE a_materno='Pérez')
```

El operador de comparación IN compara un valor *v* con un conjunto o multiconjunto de valores *V* y evalúa con TRUE si *v* es uno de los elementos en *V*.

C.2.4 Función EXISTS.

Esta función es usada para verificar si el resultado de una consulta contiene tuplas o no; es comúnmente utilizada en consultas anidadas, su función complemento es NOT EXISTS. El resultado de la función EXISTS es TRUE si al menos hay una tupla en el resultado y FALSE si no existe alguna. NOT EXISTS regresa TRUE si no hay tuplas en el resultado de la consulta y FALSE si al menos existe una

Ejemplo 10:

Seleccionar el nombre, apellido paterno y materno de los empleados que no tengan asignado un proyecto

```
SELECT DISTINCT       a_paterno, a_materno, nombre
FROM   Empleados
WHERE  NOT EXISTS     (SELECT *
       FROM Proyectos
       WHERE id_proyecto=proyecto_asignado)
```

C.2.5 Conjuntos Explícitos y Valores Nulos.

Es posible en ocasiones usar un conjunto explícito de valores entre paréntesis en la cláusula WHERE en vez de consultas anidadas

Ejemplo 11:

Consultar a los empleados que están asignados a los proyectos 1,2 y 3:

```
SELECT DISTINCT       a_paterno, a_materno, nombre
FROM   Empleados
WHERE  proyecto_asignado IN (1,2,3)
```

SQL además permite verificar si un valor es nulo: esta perdido, no definido o no aplicable. Sin embargo en vez de usar los operadores = o <> para comparar un atributo con nulo, SQL usa las palabras reservadas IS e IS NOT. Esto se debe a que SQL considera cada valor nulo distinto de otros valores, así la comparación de igualdad no es apropiada. De aquí se desprende la idea de que cuando se especifica una operación JOIN, las tuplas con valores nulos no se incluirán en el resultado.

Ejemplo 12:

Consultar los nombres de todos los empleados que no tienen seguro social

```
SELECT a_paterno, a_materno, nombre
FROM   Empleados
WHERE  seguro_social IS NULL
```

C.2.6 Funciones Agregadas y de Grupo.

Las funciones agregadas requeridas por la mayoría de las bases de datos son: COUNT, SUM, MAX, MIN y AVG. La función COUNT regresa el número de tuplas o valores especificados en una consulta.

Las funciones SUM, MAX, MIN y AVG se aplican a multiconjuntos de valores numéricos y regresan la suma, el valor máximo, el valor mínimo y el promedio de dichos números respectivamente.

Ejemplo 13:

Encontrar la suma de los salarios de todos los Empleados, el salario máximo, el salario mínimo y el salario promedio.

```
SELECT SUM(salario), MAX(salario), MIN(salario), AVG (salario)
FROM   Empleados
```

Ejemplo 14:

Cuenta el número total de empleados que trabajan en el proyecto 5.

```
SELECT COUNT (*)
FROM   Empleados
WHERE  proyecto_asignado=5
```

En muchos casos se desea aplicar funciones agregadas a un subconjunto de tuplas, en una relación basada en algunos valores de atributos. Por ejemplo, se desea encontrar el promedio del salario de los empleados en cada proyecto asignado y el número de empleados que trabajan en cada proyecto. En este caso se desea agrupar las tuplas que tienen el mismo valor de algún o algunos atributos, llamado *atributo(s) de grupo*, y aplicar la función para cada grupo independientemente. SQL tiene la cláusula GROUP BY que especifica los atributos de grupo, los cuales deberán aparecer también en la cláusula SELECT.

Ejemplo 15:

Cuenta el número total de empleados que trabajan en el proyecto 5.

```
SELECT proyecto_asignado, COUNT (*), AVG(salario)
FROM   Empleados
GROUP BY       proyecto_asignado
```

Además de esto SQL provee otra palabra reservada HAVING para condicionar el grupo de tuplas asociadas con la orden GROUP BY. Por ejemplo si se quiere condicionar el número de empleados trabajando a mayor de 2 del ejemplo 15:

Ejemplo 16:

```
SELECT proyecto_asignado, COUNT (*), AVG(salario)
FROM   Empleados
GROUP BY       proyecto_asignado
HAVING COUNT(*)>2
```

Finalmente se tiene la cláusula ORDER BY que permite al usuario ordenar las tuplas, resultado de la consulta, en base a uno o más atributos con el fin de generar un reporte más comprensible.

Ejemplo 17:

Realizar una lista con los empleados ordenados por apellido paterno, apellido materno y nombre que trabajen en el proyecto 3.

```
SELECT a_paterno, a_materno, nombre
FROM   Empleados
WHERE  proyecto_asignado=3
ORDER BY       a_paterno, a_materno, nombre
```

C.3 Declaraciones de Actualización

En SQL hay tres comandos para modificar la base de datos, a continuación de discuten cada uno de ellos

C.3.1 Comando INSERT.

En su forma más simple este comando se utiliza para adicionar una simple tupla a una relación, los valores deberán ser listados en el mismo orden en que se presentan los atributos de la relación cuando se crean con el comando CREATE.

Ejemplo 18

Para adicionar una nueva tupla a la tabla Empleados, viendo el orden de los atributos del Ejemplo 1 se tiene que realizar lo siguiente:

```
INSERT INTO    Empleados
VALUES (133,          'Fernández',          'Hurtado',          'Victor',          'M',6500,
       '2345978565',5)
```

Otra forma del comando, es dando los nombres de los atributos de manera específica, sobre aquellos que conocemos y sabemos que no se restringen por el parámetro NOT NULL, los atributos que quedan en blanco son fijados a NULL, y sus valores son listados en el mismo orden que los atributos de la relación.

Ejemplo 19

Se conoce solo el apellido paterno, apellido materno, el nombre y el número del seguro social de un empleado

```
INSERT INTO    Empleados        (a_paterno,        id_empleado,        a_materno,        nombre,
       seguro_social)
VALUES (134, 'Guerrero', 'Jimenez', 'Leticia','29966748S6')
```

Los atributos que quedan vacíos se fijan como NULL, y los valores son listados en el mismo orden en el que se especificaron en el comando INSERT.

Si en algún momento no se introduce el valor a un atributo con la condición NOT NULL, el manejador debe verificar la integridad referencial y evitar que la nueva tupla se integre a la relación con ese error.

Finalmente una variación del comando INSERT es permitir insertar múltiples tuplas dentro de una relación. Esto se usa principalmente en conjunción al crear una relación temporal y llenarla de datos con el resultado de una consulta.

Ejemplo 20

Se desea crear una tabla temporal que tenga el proyecto 5 con el total de empleados en ese proyecto y total del salario que se destina a los empleados de dicho proyecto.

```
CREATE TABLE   InfoSalarioProyectos
(id_proy       INTEGER,
nom_proy       VARCHAR(15)
total_empl     INTEGER,
total_salario  INTEGER)
INSERT INTO    InfoSalarioProyectos(id_proy,                nom_proy,                total_empl,
       total_salario)
```

```
SELECT  (id_proyecto,                                          nombre_del_proyecto,
        COUNT(*),
        SUM(salario)
FROM    Proyectos, Empleados
WHERE   id_proyecto=5
GROUP BY        id_proyecto)
```

C.3.2 Comando DELETE.

Este comando elimina cero, una o varias tuplas de una relación. Incluye la cláusula WHERE para condicionar el borrado de las tuplas que satisfagan el parámetro. Las tuplas son eliminadas solo de una tabla cada vez que el comando se aplica. Si no existe condición WHERE se entiende que se desean eliminar todas las tuplas de la relación dejando una tabla existente en la base de datos, pero sin ningún elemento. A continuación se presentan tres ejemplos:

Ejemplo 21

Eliminar la tupla del empleado con identificador de empleado igual a 8:

```
DELETE FROM    Empleados
WHERE   id_empleado=8
```

Ejemplo 22

Eliminar la tupla o tuplas de los empleados que participan en el proyecto "Rescate Ecológico":

```
DELETE FROM    Empleados
WHERE   proyecto_asignado in  (SELECT id_proyecto
                FROM    Proyectos
                WHERE   nombre_del_proyecto=
                        'Rescate Ecológico')
```

Ejemplo 23

Eliminar todas las tuplas de la tabla InfosalarioProyectos

```
DELETE FROM    InfosalarioProyectos
```

C.3.3 Comando UPDATE.

Se usa para modificar valores de atributos de una o más tuplas seleccionadas. Así como el comando DELETE, este también se realiza sobre una relación a la vez y se apoya de la condición WHERE para seleccionar los registros a actualizar. Además se utiliza una sentencia SET que especifica los atributos que serán modificados con sus nuevos valores.

Ejemplo 24

Cambiar el proyecto 10 con el nuevo nombre "Reciclaje de Plástico"

```
UPDATE  Proyectos
SET     nombre_del_proyecto='Reciclaje de Plástico'
WHERE   id_proyecto=10
```

Es también posible modificar varias tuplas a la vez.

Ejemplo 25

Dar a los empleados del proyecto 'Reciclaje de Plástico un 10% de aumento a su salario.

```
UPDATE  Empleados
SET     Salario=Salario * 1.1
WHERE   proyecto_asignado IN  (SELECT id_proyecto
```

```
FROM    Proyectos
WHERE   nombre_del_proyecto                                    =
        'Reciclaje de Plastico')
```

C.4 Vistas SQL

C.4.1 Concepto de una Vista.

Una vista en la terminología SQL es una simple tabla que deriva de otras tablas que a su vez pudieron o no ser definidas previamente por otra vista. Una vista no necesariamente existe de manera física o literal como sucede con las relaciones definidas con el comando CREATE TABLE y se les denomina "*tablas virtuales*". Se puede pensar en una vista como una manera de especificar una tabla que requiere ser referenciada frecuentemente, pero que físicamente no existe, pues sus atributos forman parte de varias tablas.

C.4.2 Especificaciones de Vistas.

El comando para especificar una vista es el CREATE VIEW, dando como parámetros el nombre de una tabla, que recordemos será virtual, la lista de atributos y una consulta especificando el contenido de dicha vista. A continuación se creará una vista que requiere el nombre, apellido paterno y materno del empleado, identificador del proyecto, nombre del proyecto y salario total de los que trabajan en el proyecto 5.

Ejemplo 26

```
CREATE VIEW    Vista_Virtual
AS SELECT      nombre,      a_paterno,      a_materno,      id_proyecto,      nombre_del_proyecto,
       total_salario
FROM   Empleados, Proyectos, InfosalarioProyectos
WHERE  id_proyecto = 5 AND id_proyecto = proyecto_asignado
```

Como se puede observar una vista se compone de la misma estructura que las relaciones pero de una manera dinámica y con posibilidades de acrecentar la velocidad de la presentación porqué se realizan dentro de la memoria del sistema y dando opción a actualizarse de relaciones de las cuales se creó dicha vista cuando los datos son modificados.

Si ya no se desea conservar una vista, se dispone del comando DROP VIEW para eliminarla de memoria.

Ejemplo 27

Eliminar la vista Vista_Virtual del ejemplo anterior:

```
DROP VIEW Vista_Virtual
```

C.4.3 Actualización de Vistas.

La actualización de las vistas puede ser complicada en la mayoría de los casos; cuando se desea actualizar una vista para modificar los datos para una sola relación, solo se tiene que mapear a la relación original y realizar la actualización con el comando UPDATE, sin embargo si existen varios atributos de diferentes relaciones la actualización se puede hacer de muy diversas maneras existiendo posibles discrepancias al direccionar a las relaciones originales. De hecho este tema hoy en día es un área de frecuente investigación entre los desarrolladores de sistemas de bases de datos, pero en términos generales se pueden sacar las siguientes conclusiones:

1. *No se puede garantizar que una vista sea actualizable debido a existen muchas posibilidades de realizarse.*
2. *Una vista con una simple definición de una tabla es actualizable si los atributos en la vista contienen llaves primarias o algún otra llave candidata de la relación base, ya que esta mapeará cada tupla de la vista como si fuera una sola relación.*
3. *Las vistas son definidas de múltiples tablas usando uniones generalmente no actualizables.*
4. *Las vistas son definidas usando funciones de agregación y de grupo no actualizables.*

En forma general estos puntos darán la experiencia para realizar vistas que mantengan integridad referencial para actualizar los datos de una manera coherente.

C.5 Indices

Un índice dentro del lenguaje SQL es una estructura que da acceso real o físico a uno o más atributos de una relación o tabla. Ya que una tabla representa en la mayoría de las veces un archivo, los índices se especifican en dicho archivo. El o los atributos para los cuales se crea un índice se denominan atributos indexados. Un índice hace más eficiente el acceso a los datos en las consultas, esto significa que se llevará menos tiempo consultar atributos que previamente han sido indexados a consultar atributos que no lo están. En SQL los índices pueden ser creados y eliminados dinámicamente. Los comandos respectivos son CREATE INDEX y DROP INDEX.

Ejemplo 28

Crear un índice para el atributo a_paterno de la tabla Empleados se haría de la siguiente manera:

```
CREATE INDEX   paterno_index
ON      Empleados (a_paterno)
```

Ejemplo 29

Eliminar el índice del ejemplo anterior:

```
DROP INDEX     paterno_index
```

Apéndice D. Fuentes del sistema de réplica.

D.1 Trigger de la tabla *FORMAS_VALORADAS*

Código de creación del *trigger* para la tabla *Formas_Valoradas*.

```
/**    SCRIPT DE CREACION DE TRIGGER: tval$_formas_valoradas
**    A.S.E.R C.A.
**    $Revision: 1.5 $                          $Date: 1997/09/08 20.04:43*/
SET SERVEROUT ON
DROP TRIGGER tval$_formas_valoradas
/
CREATE OR REPLACE TRIGGER tval$_formas_valoradas
BEFORE INSERT OR UPDATE OR DELETE ON folios_cheques
REFERENCING OLD AS old NEW AS new
FOR EACH ROW
DECLARE
   réplica       CONSTANT CHAR(1)       := 'R';
   c_destino     CONSTANT NUMBER(2)     := 99;
   CUENTA_CORRIENTE   CONSTANT NUMBER := 8,
   nom_tabla     CONSTANT VARCHAR2(20) := 'FORMAS_VALORADAS',
   RATIFICADA    CONSTANT VARCHAR2(20) := 'RATIFICADA';
   ratificacion VARCHAR2(30)           := 'NO RATIFICADA';
   c_local       NUMBER(2);
clave         NUMBER(4);
   evento        NUMBER;
   v_codigo      NUMBER;
   tipo_forma    VARCHAR2(30);
   tipo_dml      CHAR(1)                := 'N';
   v_status      CHAR(3);
   datos         VARCHAR2(2000)         := null ;
   condicion     VARCHAR2(600)          := :old.folio||''||''||:old.regional||''||''||
:old.tipo_pago;
   fecha  VARCHAR2(9)    :=to_char(sysdate,'DD-MON-YY',
                              'nls_date_language=spanish');
   actualizacion_invalida EXCEPTION;
   insercion_invalida     EXCEPTION;
   borrado_invalido       EXCEPTION;
   regional_invalido      EXCEPTION;
   datos_inconsistentes   EXCEPTION;
   no_ratificado          EXCEPTION;
   evento_ant_aun_act     EXCEPTION;
   evento_no_val          EXCEPTION;

BEGIN
  IF :new.pago IS NOT NULL AND :new.apoyo IS NOT NULL THEN
     IF pkg_eventos_imp.ratificada(new.pago, new.ciclo,
                          new.tipo_apoyo) THEN
       ratificacion := RATIFICADA;
       IF :new.cinta IS NULL THEN
            tipo_forma := 'NO CONCILIADA';
       ELSE
            tipo_forma := 'CONCILIADA';
       END IF;

     ELSE
       ratificacion := 'NO RATIFICADA';
     END IF;
ELSIF :new.pago IS NULL AND :new.apoyo IS NULL AND :new.cinta IS NULL
    AND :new.evidencia IS NULL THEN
      tipo_forma := 'EN BLANCO';
END IF;
  IF UPDATING THEN
     :new.fecha_status := sysdate;
     IF :new.status NOT IN ('CEA', 'FSA', 'CSI', 'CCA', 'CCE', 'CEI',
        'PAG', 'CRA','CSA', 'CRG', 'CSR', 'CAN') THEN
           RAISE datos_inconsistentes;
     ELSIF :new.status='CEA' AND ( :old.status NOT IN ('CEA','FSA','CCA',
        'CSI') OR  tipo_forma != 'EN BLANCO' ) THEN
           RAISE actualizacion_invalida;
     ELSIF :new.status='FSA' AND ( :old.status NOT IN ('FSA','CEA') OR
```

```
        tipo_forma != 'EN BLANCO' ) THEN
            RAISE actualizacion_invalida;
     ELSIF :new.status='CCA' AND ( :old.status NOT IN ('CCA','CEA','CSI')
        OR  tipo_forma != 'EN BLANCO' ) THEN
            RAISE actualizacion_invalida;
     ELSIF :new status='CSI' AND ( :old.status NOT IN ('CSI','CEA','CCA')
        OR tipo_forma != 'EN BLANCO' ) THEN
            RAISE actualizacion_invalida;
     ELSIF :new.status='CCE' AND :old.status NOT IN ('CCE','CEI') THEN
            RAISE actualizacion_invalida;
     ELSIF :new.status='CEI' AND ( :old.status NOT IN ('CEI', 'CSI',
        'CCE', 'CRA') OR tipo_forma != 'NO CONCILIADA' ) THEN
            RAISE actualizacion_invalida;
     ELSIF :new.status='CRA' AND ( :old.status NOT IN ('CRA', 'CEI',
       'PAG', 'CSR','CAN','CSA') OR tipo_forma != 'NO CONCILIADA'
OR ratificacion != RATIFICADA ) THEN
            RAISE actualizacion_invalida;
     ELSIF :new.status='CSA' AND ( .old.status NOT IN ('CSA','CRA','CAN')
                                OR tipo_forma != 'NO CONCILIADA'
                                OR ratificacion != RATIFICADA
                                OR .new.evidencia is null ) THEN
            RAISE actualizacion_invalida;
     ELSIF .new.status='CRG' AND ( :old.status NOT IN ('CRG','CRA','CSA')
                                OR tipo_forma != 'NO CONCILIADA'
                                OR ratificacion != RATIFICADA ) THEN
            RAISE actualizacion_invalida;
     ELSIF :new.status='CSR' AND ( :old.status NOT IN ('CSR','CRA','CSA')
                                OR tipo_forma != 'CONCILIADA'
                                OR ratificacion != RATIFICADA ) THEN
            RAISE actualizacion_invalida;
     ELSIF 'new status ='PAG' AND ( :old.status NOT IN ('PAG','CRA','CSA')
                                OR tipo_forma != 'CONCILIADA'
                                OR ratificacion != RATIFICADA ) THEN
            RAISE actualizacion_invalida;
     ELSIF :new.status='CAN' AND (:old.status NOT IN ('CAN', 'CRA', 'CSA',
       'CRG', 'CSR','PEX') OR ratificacion != RATIFICADA ) THEN
            RAISE actualizacion_invalida;
     ELSIF :new.status = 'CRA' AND :old.status = 'CEI' THEN
            tipo_dml  := 'I';
            condicion :- null;
     ELSIF :new.status = 'CRA' AND :old.status in ('PAG', 'CSR', 'CAN',
       'CRA') THEN
            tipo_dml  :- 'U',
     ELSIF .new.status in ('PAG','CSA','CRG','CSR','CAN') THEN
            tipo_dml  := 'U',
     ELSIF .new status   'CVI' AND :old.status = 'CEI' THEN
            tipo_dml  := 'I';
            condicion .- null;
     ELSIF .new.status - 'CEI' AND .old.status = 'CRA' THEN
            tipo_dml  :- 'D';
            datos     := '';
     END IF;
  ELSIF INSERTING THEN
     IF  new.status NOT IN ('CEA','CSI') THEN
            RAISE insercion_invalida;
     ELSIF :new.status='CRA' AND tipo_forma '='EN BLANCO' THEN
            RAISE datos_inconsistentes;
     ELSIF :new.status='CSI' AND tipo_forma !='EN BLANCO' THEN
             RAISE datos_inconsistentes;
     END IF;
  ELSE /* La operación es de borrado */
     IF  old status in ('CEI','CRA','CSA','CRG','CSR','PAG','CAN') THEN
            RAISE borrado_invalido;
     END IF;
   END IF;

BEGIN /* Bloque de control y generacion de las operaciones de réplica */
   IF 'new tipo_pago - CUENTA_CORRIENTE THEN
     tipo_dml := 'N';
   END IF;
IF tipo_dml '- 'N' THEN
       IF :new.transmision IS NULL THEN
          evento '= obten_evento,
       ELSE
          evento := :new.transmision,
       END IF;
```

```
      SELECT clave_mov
        INTO clave
        FROM TIPOS_MOVIMIENTOS_TRANS
       WHERE tabla = nom_tabla
         AND tipo_mov = tipo_dml
         AND tipo_tabla = réplica;
      IF tipo_dml != 'D' THEN
       datos := 'new.folio||'|'||:new.apoyo||'|'||:new.status||'|'||
        :new.evidencia||'|'||:new.fecha_status||'|'||:new.ciclo||'|'||
        :new.tipo_pago||'|'||:new.pago||'|'||:new.pasadas||'|'||
        translate(:new.archivo,chr(124),chr(32))||'|'||
        translate(:new.beneficiado,chr(124),chr(32))||'|'||
        'new.transmision||'|'||
        :new.cinta||'|'||:new.regional||'|'||:new.fecha_pago||'|'||
        :new.tipo_apoyo||'|'||:new.evento||'|'||:new.estado;
      END IF;
      INSERT INTO MOVIMIENTOS VALUES
                  (clave,SEC_MOVIMIENTOS.NEXTVAL,0,1,datos,'GEN','0',
                   SYSDATE,NULL,c_local,c_destino,evento,condicion);
   END IF;
   EXCEPTION WHEN OTHERS THEN
           dbms_output.put_line('ERROR.INS MOV '||sqlerrm);
   END   ;
EXCEPTION
     WHEN insercion_invalida THEN
          raise_application_error(-20101,'imposible insertar una forma '||
           lower(tipo_forma)||' con status '||:new.status);
     WHEN actualizacion_invalida THEN
          raise_application_error(-20102,'imposible actualizar una forma
        '|| lower(tipo_forma)||' y '||lower(ratificacion)||' de '||
         :old.status ||' a '||'new.status),
     WHEN borrado_invalido THEN
          raise_application_error(-20103,'imposible borrar una forma
          impresa con status '||:old.status);
     WHEN datos_inconsistentes THEN
          raise_application_error(-20104,'datos inconsistentes al efectuar
          transaccion'),
     WHEN no_ratificado THEN
          raise_application_error(-20105,'se debe registrar un evento para
          cada emision de pagos');
     WHEN regional_invalido THEN
          raise_application_error(-20106,'imposible tener una forma del
          regional'||:new.regional||' en este centro');
     WHEN evento_ant_aun_act THEN
          raise_application_error(-20107,'El folio aun esta registrado con
          el ' ||'evento de entrega-recepcion' '||to_char(:old.evidencia)
          || '. Debe utilizarse Corrección de Actas de Entrega-Recepcion
          para alterar el valor. ');
     WHEN evento_no_val THEN
          raise_application_error(-20108,'El evento: '||
          to_char('new.evidencia)||'no esta registrado como
          entrega-recepcion  Utilice Correccion  o Captura de Actas de
           Entrega-Recepcion '),
     WHEN OTHERS THEN
          dbms_output.put_line(sqlerrm),
END;
/
```

D.2 Métodos de la clase *cursor*

La clase *cursor* tiene definido un constructor en el cual se asignan los valores iniciales: las variables de conteo *leidos* y *procesados* se fijan a cero, el apuntador a la variable que describe la consulta *sd* y el buffer del texto de la consulta *orabuff* se apuntan a la dirección nula, la variable de interfaz con el RDBMS se asigna a una estructura *sqlcanulo* que contiene los valores de inicio. Y se obtiene de la variable de ambiente *TAMANO_ARREGLO* el número de registros que obtendrá del RDBMS en cada acceso. Si esta variable no está definida se asigna un número de registros definido por la constante *TAMANO* definida internamente con un valor diferente de cero. El código correspondiente es el siguiente:

```
cursor.:cursor() // Constructor de la clase cursor
```

```
{
   sd = NULL;
   orabuff = NULL;
   row_fetched = FALSE;
   leidos = 0;
   procesados = 0;
   current_row = -1;
   error_code = SQL_OK;
   arraysize=((getenv("TAMANO_ARREGLO"))==(char *)0)?TAMANO:atoi(getenv("TAMANO_ARREGLO"));
   sqlca=::sqlcanulo;
}
```

El método *query()* de la clase *cursor* contiene en forma general los pasos para que el RDBMS prepare los registros que ha de proporcionar al hacer la consulta, el código correspondiente es:

```
int cursor::query(char *dinbuff)
{

   orabuff = dinbuff;                  // Apuntamos al buffer de la consulta
   allocdescriptors();                 // Reserva espacio para el descriptor
   if(parse() < 0) goto ERROR_4;       // Si el análisis sintáctico es erróneo libera
   if(open() < 0) goto ERROR_4;        // Si no se puede abrir el cursor
   descsel();                          // Describe las variables select en sd
   if( sd->F != 0 )                    // Si existe al menos una columna
      allocselvars();                      //Reserva memoria para los datos
   return SQL_CODE;

ERROR_4:
   cleanup();
   return SQL_CODE;

}/**** query ****/
```

El parámetro de entrada al método *query* es el texto de la sentencia de consulta, el cual se analiza sintácticamente y se revisa la existencia de los objetos referenciados. Posteriormente se abre el cursor y se obtiene la descripción de las columnas de la consulta para con base en esta reservar espacio para los datos.

Al nivel siguiente de detalle se ubica el método *allocdescriptor()* donde la variable *sd* se prepara reservando el espacio correspondiente donde se almacenará la descripción de la consulta. El código de este método es el siguiente:

```
int cursor::allocdescriptors(){
   sd = (SQLDA *)sqlald(NO_SVARS,LEN_SVARS,0);
   if (sd == NULL)
      return (error_code=MEMORIA_AGOTADA);
   sd->N = NO_SVARS; /* Inicializa el total de elementos del arreglo */
   return SQL_OK;
}
```

El análisis sintáctico se solicita a través de la variable *sqlstm*, la cual no forma parte de la clase cursor porque su definición debe ser *static* de forma global ya que es referenciada así internamente en las funciones de Oracle. El código para llenar *sqlstm* presentado aquí fue creado automáticamente al precompilar la sentencia encerrada en comentarios identificada como *SQL stmt #* <Número de sentencia > y podría variar si se precompila en otra plataforma distinta.

```
int cursor::parse()
{
/* SQL stmt #5
   EXEC SQL WHENEVER SQLERROR GOTO ERROR_4;
   EXEC SQL PREPARE S1 FROM :orabuff;           /o Ejecuta el parse del query o/
*/
   sqlstm.iters = (unsigned short)1;
   sqlstm.offset = (unsigned short)2;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlstm.sqhstv[0] = (unsigned char *)orabuff;
```

```
    sqlstm.sqhstl[0] = (unsigned long)0;
    sqlstm.sqindv[0] = (short *)0;
    sqlstm.sqharm[0] = (unsigned long)0;
    sqlstm.sqphsv = sqlstm.sqhstv;
    sqlstm.sqphsl = sqlstm.sqhstl;
    sqlstm.sqpind = sqlstm.sqindv;
    sqlstm.sqparm = sqlstm.sqharm;
    sqlstm.sqparc = sqlstm.sqharc;
    sqlcex(&sqlctx, &sqlstm, &sqlfpn);
    return (error_code=SQL_CODE);
}
```

Para abrir la consulta también se hace la petición mediante la variable *sqlstm* tal como se muestra en seguida:

```
/* ************************************************************************** */
** open: libera el espacio de memoria reservado para los descriptores      **
** recibe : (SQLDA *sd) descriptores de variables.                          **
** ************************************************************************** */
int cursor::open()
{
/* SQL stmt #7
   EXEC SQL DECLARE C2 CURSOR FOR S1;          /o Declara el cursor o/
   EXEC SQL OPEN C2;                           /o Abrir el cursor   o/
*/
   sqlstm.stmt = "";
   sqlstm.iters = (unsigned short)1;
   sqlstm.offset = (unsigned short)17;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char *)&sqlca,
   sqlstm.sqlety = (unsigned short)0,
   sqlcex(&sqlctx, &sqlstm, &sqlfpn);
   status=(SQL_CODE==SQL_OK)?OPEN.CLOSED;
   return (error_code=SQL_CODE);
}
```

Una vez abierto el cursor, la descripción de los componentes de la consulta se obtienen en la variable *sd* a través de otra petición al RDBMS:

```
void cursor: descsel(){
   int NEW_SVARS=0;
/* SQL stmt #15  EXEC SQL DESCRIBE SELECT LIST FOR S1 INTO sd;*/
// La líneas identadas a este comentario fueron creadas
// automaticamente por Oracle y sustituyen la sentencia SQL precedente
              sqlstm.iters = (unsigned short)1;
              sqlstm.offset = (unsigned short)54;
              sqlstm.cud = sqlcud0,
              sqlstm.sqlest = (unsigned char *)&sqlca,
              sqlstm.sqlety = (unsigned short)0;
              sqlstm.sqhstv[0] = (unsigned char *)sd;
              sqlstm.sqhstl[0] = (unsigned long)0;
              sqlstm.sqindv[0] = (short *)0;
              sqlstm.sqharm[0] = (unsigned long)0,
              sqlstm.sqphsv = sqlstm.sqhstv;
              sqlstm.sqphsl = sqlstm.sqhstl;
              sqlstm.sqpind = sqlstm.sqindv;
              sqlstm.sqparm = sqlstm.sqharm;
              sqlstm.sqparc = sqlstm.sqharc;
              sqlcex(&sqlctx, &sqlstm, &sqlfpn),
   error_code = SQL_CODE;
   if( sd->F < 0 ){ /* si F es negativa, hubo mas elementos que los reservados */
      NEW_SVARS = -(sd->F);     /* Reasignamos el total del elementos  */
      sqlclu(sd);               /* Regresa el espacio reservado */
      sd = (SQLDA *)sqlald(NEW_SVARS,LEN_SVARS,0); /* Obtiene el tamaño correcto */
/* SQL stmt #15  EXEC SQL DESCRIBE SELECT LIST FOR S1 INTO sd;*/
// La líneas identadas a este comentario fueron creadas
// automáticamente por Oracle y sustituyen la sentencia SQL precedente
/* SQL stmt #16       EXEC SQL DESCRIBE SELECT LIST FOR S1 INTO sd;*/
                      sqlstm.iters = (unsigned short)1;
                      sqlstm.offset = (unsigned short)69;
                      sqlstm.cud = sqlcud0;
                      sqlstm.sqlest = (unsigned char *)&sqlca;
                      sqlstm.sqlety = (unsigned short)0;
```

```
                    sqlstm.sqhstv[0] = (unsigned char *)sd;
                    sqlstm.sqhstl[0] = (unsigned long)0;
                    sqlstm.sqindv[0] = (short *)0;
                    sqlstm.sqharm[0] = (unsigned long)0;
                    sqlstm.sqphsv = sqlstm.sqhstv;
                    sqlstm.sqphsl = sqlstm.sqhstl;
                    sqlstm.sqpind = sqlstm.sqindv;
                    sqlstm.sqparm = sqlstm.sqharm;
                    sqlstm.sqparc = sqlstm.sqharc;
                    sqlcex(&sqlctx, &sqlstm, &sqlfpn);
    error_code = SQL_CODE,
   }// if( sd->F < 0 )
  sd->N = sd->F; /* Reasignamos el total de elementos de la lista */
                 /* select encontrado por el describe */
}/* fin de descsel */
```

Así se obtiene en la variable *sd* los nombres, tamaños y tipos de datos de cada columna incluida en la consulta con base en la cual el método *allocselvars()* crea un bloque de memoria por columna de forma que se almacenen tantos registros como el tamaño del arreglo se haya definido, tal como se observa en el código siguiente:

```
void cursor::allocselvars(){
   int i, precision, escala, nullok,
   for( i = 0; i < sd->F; i++){
      sqlnul(&(sd->T[i]),&(sd->T[i]),&nullok); /* Obtiene tipo y nulo */
      switch(sd->T[i]){
          case 1:  break;  /* tipo CHAR no se necesita cambiar longitud */
          case 2:  /* tipo NUMBER obtenemos precision y escala */
                   sqlprc(&(sd->L[i]),&precision,&escala);
                       if( precision == 0 ) precision = 26;
                       sd->L[i] = precision + 2;
                       if( escala < 0 ) sd->L[i] += -escala;
                   break;
          case 12: /* tipo DATE */
                   sd->L[i] = 9;
                   break;
          case  8: /* tipos de datos LONG y LONG ROW respectivamente.
          case 24:    Se Asigno un valor arbitrario */
                   sd->L[i] = TAM_LONGS;
                   break;
      } /* fin del switch */
      sd->T[i] = 1;   /* forzamos todos los tipos a char */
      /* Se compara la longitud del campo con la longitud maxima de 2040 */
      sd->L[i] = (sd->L[i] < MAXLEN ? sd->L[i] : MAXLEN),

      /* Reserva memoria para la variable y variable indicadora */
      sd->V[i] =   (char *)calloc(arraysize,(int)sd->L[i]+1);
      sd->I[i] = (short *)calloc(arraysize,sizeof(short)); /* Variable indicadora */
      *sd->I[i] = nullok,
   } /* fin del for */
   return;
} /* fin de allocselvars */
```

Una vez que se ha reservado el espacio para almacenar los datos de los registros se solicita al RDBMS obtener el número de registros especificado por la variable *arraysize* tal como se muestra en el método *fetch()*:

```
int cursor::fetch(){
/* SQL stmt #11
   EXEC SQL FOR :arraysize FETCH C2 USING DESCRIPTOR sd;
*/
   sqlstm.iters = (unsigned short)arraysize;
   sqlstm.offset = (unsigned short)28;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlstm.sqhstv[0] = (unsigned char *)sd;
   sqlstm.sqhstl[0] = (unsigned long)0,
   sqlstm.sqindv[0] = (short *)0,
   sqlstm.sqharm[0] = (unsigned long)0;
   sqlstm.sqphsv = sqlstm.sqhstv,
```

```
    sqlstm.sqphsl = sqlstm.sqhstl;
    sqlstm.sqpind = sqlstm.sqindv;
    sqlstm.sqparm = sqlstm.sqharm;
    sqlstm.sqparc = sqlstm.sqharc;
    sqlcex(&sqlctx, &sqlstm, &sqlfpn);
    if(SQL_CODE == SQL_OK || SQL_CODE == SQL_NOT_FOUND) {
       if(SQL_COUNT > leidos ) {
          current_row = -1;
          leidos = SQL_COUNT;
          SQL_CODE = SQL_OK;
       }
    }
    row_fetched=(SQL_CODE==SQL_OK)?TRUE FALSE;
    return (error_code=SQL_CODE);
}
```

Para ocultar el control del bloque a los programadores que usen la clase *cursor*, el acceso a cada registro se hace solo a través del método *operator*++ el cual obtiene el siguiente registro. Existen dos niveles para recorrer la información: uno es registro por registro dentro del bloque de datos y el otro bloque por bloque hasta agotar los registros del cursor de Oracle. Cuando a nivel bloque se está en el último registro y se hace la petición del siguiente registro se debe llevar a cabo otro acceso al RDBMS con el método *fetch()* para obtener otro bloque de registros.

```
cursor& cursor::operator++(int){
   if(leidos == procesados) {
      if( (leidos == 0) || (current_row+1) == arraysize) {
         if((fetch() != SQL_OK) && (SQL_CODE != SQL_NOT_FOUND)) {
            close();
            return *this;
         }
      }
      else {
         row_fetched=FALSE;
         close();
         return *this;
      }
   }
   current_row++;
   procesados++;
   return *this;
}
```

Al ejecutar cada petición de obtención de registros (método *fetch()*) podría ocurrir que se llegue al final del cursor del RDBMS, cuando esto ocurre el número de registros obtenidos se hace cero. Para preguntar sobre este valor se hace uso del método *found()*

```
int cursor::found(){ // Proporciona el número de registros obtenidos en la peticion al RDBMS
 return row_fetched;
}
```

Cuando se alcanza el final de la consulta se envía la petición de cierre del cursor al RDBMS, esta petición se realiza mediante el método *close()* mostrado a continuación:

```
int cursor::close(){ //Cierra el cursor
/* SQL stmt #13
   EXEC SQL WHENEVER SQLERROR CONTINUE;
   EXEC SQL CLOSE C2,          */
   if(is_open()) {
   sqlstm.iters = (unsigned short)1;
   sqlstm.offset = (unsigned short)43;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char *)&sqlca,
   sqlstm.sqlety = (unsigned short)0;
   sqlcex(&sqlctx, &sqlstm, &sqlfpn);
   cleanup();
   }
   sd = NULL;
   orabuff = NULL;
   row_fetched = FALSE;
   leidos = 0;
   procesados = 0;
   current_row = -1;
   error_code = SQL_OK;
```

```
}// code()
```

Para manipular cada columna del registro actual se lleva a cabo el operador ->* el cual se basa en la estructura *blk*. En un bloque se almacenan los valores de una columna para todos los registros, cada valor no está delimitado por ningún carácter de control y tampoco son cadenas terminadas en nulo. Así para delimitar cada valor se cuenta con una dirección base y una longitud que es fija para todos los valores del bloque y cambia para cada bloque.

```
blk cursor::.operator->*(char *columna)
{
  bcol.set(NULL,0);
  for(current_col=0; current_col < sd->N; current_col++)
      if( bcol.set( sd->S[current_col],sd->C[current_col])==columna ){
         bcol.set(getadd(current_col,current_row), sd->L[current_col]);
         bcol.quita_char(ESPACIO);
         break;
      }
  return bcol;
}//cursor->*(char *)

int cursor::lonC(int i) // Regresa la longitud de la i-esima columna
{
  if (sd!=NULL && i<sd->N)
    return sd->C[i];
  else
    return FALSE;
}//operator->*
```

La dirección base de cada valor dentro del bloque está dada en función del número de registro, la longitud de la columna y la dirección base del bloque. Con base en esto se crea el método *getadd()* :

```
inline char *getadd(int col, int row) { return sd->V[col]+row*sd->L[col]; }
```

Para la impresión de los datos del *cursor* en la salida estándar se agrega el método *operator<<*

```
ostream& operator<<(ostream& salida, cursor& cur) {
  cur.current_col=0;
  salida<<endl;
  while(cur.col(cur.current_col)){
    cur.bcol.set(cur.col(cur.current_col), cur.lonC(cur.current_col));
    salida<<cur.bcol<<" ";
    ++cur.current_col;
  }
  salida<<endl;
  for(cur.current_col=0; cur.current_col < cur.sd->N; cur.current_col++) {
     cur.bcol.set( cur.getadd(cur.current_col,cur.current_row),
                   cur.sd->L[cur.current_col]);
     cur.bcol.quita_char(ESPACIO);
     salida<<cur.bcol<<"   ";
   }//for current_col
   salida<<endl;
   return salida;
}/* ostream operator << */

char * cursor::.col(int i) // Regresa el nombre de la i-esima columna
{
  if (sd!=NULL && i<sd->N)
    return sd->S[i];
  else
    return NULL;
} /* cursor.col */

int cursor::lon(int i) // Regresa la longitud de la i-esima columna
{
  if (sd!=NULL && i<sd->N)
    return sd->L[i];
  else
    return FALSE;
}
```

D.3 Métodos de la clase *blk*

Los códigos de conversión de tipos de la clase *blk* están basados inicialmente en los presentados por Kernighan y Ritchie[XXI] pero han sido modificados para controlar otras variantes y posibles errores de conversión.

```
/*****************************************************************************
*                Métodos de la clase blk para el manejo                      *
*                           de bloques                                       *
*****************************************************************************/

blk blk::set(char *ibase, int ilon) { // Asigna una dirección base y longitud
  if(ibase!=NULL && ilon!=0) {
    base=ibase;
    lon=ilon;
    sp=base;
  }
  return *this;
}/* blk::set */

blk blk::set(char *ibase, int ilon, char idec) {// Asigna una dirección base y longitud y el
  if(ibase!=NULL && ilon!=0) {        //número de decimales para conversión numérica
    base=ibase;
    lon=ilon;
    sp=base;
    dec_10=pow(10,idec);
    dec=idec;
   }
   return *this;
}/* blk::set */

char blk::deci(char idec) // Establece el número de decimales manejados al convertir
{  dec_10=pow(10,idec);
   dec=idec;
   return idec ;
} /*blk::deci */

int blk::deci() {
   return dec;
}/*blk::deci */

blk blk::quita_char(char c){ // Elimina al principio y al final del bloque el caracter c
  for(;*base==c&&lon!=0;base++,lon--); /* al inicio */
  for(;*(base+lon-1)==c&&lon>0;lon--); /* al final */
  return *this;
}// blk::quita_char

blk blk::operator<<(char *valor){
  if(lon >0&&valor!=NULL){
    for(sp=base; (sp-base)<lon&&*valor!='\0'; sp++, valor++)
            *sp= *valor;
    for(; (sp-base)<lon; sp++)
            *sp= ' ';
  }
  return *this;
}//blk::operator<<(char *)

char *blk::operator>>(char *valor) // valor debe apuntar a un bloque con
{                                  // memoria ya reservada y suficiente
   if(lon >0&&valor!=NULL)
      for(sp=base;(sp-base)<lon&&base!=NULL;sp++, valor++)
       *valor=*sp;
    *valor='\0';
   return buf;
}// blk::operator>>(char *)

blk blk::operator<<(char c){
  if(lon >0 ){  *base=c;
    for(sp=base+1; (sp-base)<lon; sp++)
    *sp=' ';
  }
```

```
  return *this;
}//blk::operator<<(char)

char blk::operator>>(char &c) {
    c=*base;
    return c;
}//blk.:operator>>(char)

blk blk::operator<<(int n) { // Conversión de un entero a una cadena ascii
   int sign;
   if((sign=n)<0) n=-n;
   sp=base+lon-1;
   do{
      *(sp--)=n%10+'0';
    }while((n/=10)>0&&(sp-base)>=0);
   if (sign < 0)
    *(sp--)='-';
   return *this;
}//blk::operator<<(int)

int blk:.operator>>(int &i) {          // Conversión de alfanumérico a entero
   int sign=1;
   i=0;
   for(sp=base;(sp-base)<lon&&base!=NULL,sp++)
      if(*sp>='0'&&*sp<= '9')
         i=(*sp-48)+i*10;
      else if(*sp=='-') sign=-1;
      i=sign*i;
    return i;
}//blk.:operator>>(int)

blk blk::operator<<(float f) {         // Conversion de número flotante a ascii
   return *this<<(double)f;
}//blk::operator<<(float)

float blk::operator>>(float& f){       // Conversión de ascii a flotante
   f = *this>>doble;
   return f;
}//blk::operator>>(float)

blk blk: operator<<(double d) {        // Conversión de número doble a ascii
   char sign;
   float n;
   if(d<0) d=-d, sign='N';
   else sign='P';
   d=d*dec_10;
   sp=base+lon-1;
   do{
      d/=10.0;
      n=(d - trunc(d))*10.0;
      *(sp--)= (char )(n  +'0');
      if((base+lon-1-sp)==dec) *(sp--)='.';
    }while((int)trunc(d)>0&&(sp-base)>0);
   if (sign == 'N') *(sp--)='-';
   for(;(sp-base)>=0; sp--)
       *sp=ESPACIO;
   return *this;
}//blk::operator<<(double)

double blk. operator>>(double &f){   // Conversión de ascii a doble
 enum sta{SG_D, DECN,DECP, FRAC, SG_E, EXPN, EXPP,ERR} s=SG_D;
 float ex=0,k=0.1;
 f=0.0;
 for(sp=base;(sp-base)<lon&&base!=NULL&&*sp!='\0'&&s!=ERR;)
     switch(s){
       case SG_D:
                if(*sp=='-' ) s=DECN, sp++, k=-0.1;
                   else if(*sp=='+') sp ++, s=DECP;
                      else if (isdigit(*sp) ) s=DECP;
                        else if (*sp=='.') sp++, s=FRAC;
                              else if (*sp!=' ') s=ERR;
                                 else sp++;
                break;
       case DECP:
              if(isdigit(*sp))
                   f=(float )(*sp-48)+f*10, sp++,
```

```
            else if (*sp=='.') s=FRAC, sp++;
                  else if(*sp=='e'||*sp=='E') s=SG_E, sp++,
                        else if(*sp!=' ') s=ERR;
                          else sp++;
             break;
      case DECN:
             if(isdigit(*sp))
                  f=-((float)(*sp-48))+f*10, sp++;
            else if (*sp=='.') s=FRAC, sp++;
                  else if(*sp=='e'||*sp=='E') s=SG_E, sp++,
                        else if(*sp!=' ') s=ERR;
                          else sp++;
             break;
      case FRAC:
             if(isdigit(*sp))
                  f=f+(*sp-48)*k, sp++, k=0.1*k;
               else if(*sp=='e'||*sp=='E') s=SG_E, sp++;
                        else if(*sp!=' ')s=ERR;
                          else sp++;
             break;
      case SG_E:
              if(*sp=='-' ) s=EXPN, sp++;
                else if(*sp=='+') s=EXPP,sp++;
                  else if(isdigit(*sp) ) s=EXPP;
                       else if (*sp!=' ') s=ERR;
                          else sp++;
                break;

      case EXPP:
             if(isdigit(*sp))
                  ex=(*sp-48)+ex*10, sp++,
                     else if(*sp!=' ') s=ERR;
                          else sp++;
             break;
      case EXPN:
             if(isdigit(*sp))
                  ex=-((float)(*sp-48))+ex*10, sp++;
                     else if(*sp!=' ') s=ERR;
                          else sp++;
             break;
    }//switch (s) */
    if (s!=ERR)
     f=f*pow(10,e );
    else
     f=0.0;
    return f;
}//blk::operator>>(double)

char blk::operator==(char *stringl)
{
  for(;lon && *stringl != '\0' && *stringl == *base ; lon--, stringl++, base++),
  if(lon == 0 && *stringl == '\0')
     return 1;
  else
     return 0;
}//blk:: operator==(char *)

blk:: operator float(){
   return *this>>flotante;
}// cast operator float

blk::operator double(){
   return *this>>doble;
}// cast operator double

blk::operator int(){
   return *this>>entero;
}//cast operator int

blk:: operator char *(){
   return *this>>buf;
}// cast operator char *

blk::operator char (){
   return *base;
}// cast operator char
```

```
ostream& operator<<(ostream& salida, blk col)
{
  for(;col.lon > 0;col.lon--,col.base++)
    salida<<*col.base;
  return salida;
}// friend ostream operator<<(ostream,blk)
```

Índice

C

D

E

N

O

P

Q

R

S

T

U

V

W

X

Bibliografía

[i] Elmasri/Navathe, Fundamentals of Database Systems, 1995

[ii] Alfons Kempler y Guido Moerkotte: Object Oriented Database Management, 1994, editorial Prentice-Hall, pág. 107.

[iii] Páginas web del *site* Dublin City University & School of Computer Applications, con URL: http://www.compapp.dcu.ie

[iv] Una Panorámica de la Ingeniería de Software. Soluciones Avanzadas. Marzo 1994. P 4-6

[v] Diccionario Larousse de la Lengua Española

[vi] Documento interno *Acciones y Necesidades de la Función Informática de la Coordinación General Operativa* publicado en noviembre de 1996.

[vii] ASERCA, documento oficial de la institución. *Autoevaluación del Año de 1996*, página 37, México D.F, publicado el 20 de febrero de 1997.

[viii] EL UNIVERSAL, Sección Finanzas pág. 4, publicado el lunes 4 de abril de 1997.

[ix] *ORACLE RDBMS Database Administrator's Guide*, 1992, pág. 4-10

[x] *ORACLE RDBMS Developer Release, vol I, 1992*, pág. 7-50.

[xi] *PRO*C Supplement to the Oracle Precompilers Guide*, 1992, pág. 4-1.

[xii] Stroustrup, B.: *The C++ Programming Language*, 1986

[xiii] Eyal Aronoff, Kevin Loney y Noorali Sonawalla: *Advanced Oracle Tuning and Administration*, 1997, editorial McGraw-Hill, pág. 261.

[xiv] *ORACLE Precompilers, Programer's* Guide Versión 1.3, 1991, pág. 6-4

[xv] W. Richard Stevens: *UNIX Network Programming*, 1990, editorial Prentice Hall.

[xvi] Dr. Dobb's Journal, Octubre, 1989.

[xvii] Mcgilton/Morgan, introducing the UNIX System, 1994

[xviii] Salas Parrilla Jesús, Sistemas Operativos y Compiladores, *1992*

[xix] Singhal/Shivaratri, Advanced concepts in operating systems 1994.

[xx] Milenkovic Milan, Sistemas operativos, conceptos y diseño, 1994.

[xxi] Kernigahn, B W. y Ritchie D.M.: *The C Programming Language*, 1988, segunda edición, editorial Prentice Hall