Universidad Tecnológica De La Mixteca

"Modelos de cómputo distribuido y concurrente para el estudio de flujos de carga en sistemas eléctricos de potencia"

INSTITUTO DE ELECTRÓNICA Y COMPUTACIÓN

Tesis
Que para obtener el titulo de:

INGENIERO EN COMPUTACIÓN

Presenta:
Erik Germán Ramos Pérez

Asesor:
M.C. Francisco de Asís López Fuentes

Huajuapan de León, Oax. Agosto del 2001

DEDICATORIA

Dedico esta tesis con todo respeto y cariño a mi madre y mis hermanos:

Mama gracias por tu cariño, tus sabios consejos, por apoyarme en todo momento, sin ti no hubiero llegado hasta aquí.

Paty, Claudia y Fernando gracias por su compresión y por ser parte fundamental en mi vida.

AGRADECIMIENTOS

M.C. Francisco de Asis López Fuentes
Por su valiosa ayuda como asesor en la realizacion de mi tesis

A mis amigos
Ivor, Carlos, Noe, Pedro, Jessica, Eduardo, Luis, Efrain, Elizabeth, Hugo, Miriam, Irving, etc.
Por su ayuda dentro y fuera de la escuela

A mis sinodales
M.C. Gerardo Cruz González
M.C. Calos Alberto Fernández y Fernández
Dr. Yuri Paramonov

A la Dra. Viginia Berrón Lara y Dra. Laura Mendoza Santos
por su gran ayuda y apoyo mostrado durante la realización de esta tesis

RESUMEN

Actualmente los sistemas de cómputo distribuido y concurrente tienen un gran desarrollo en la investigación para su implementación a diversos modelos de ingeniería, el presente trabajo lleva a integrar modelos de cómputo distribuido y concurrente para solucionar un problema de la ingeniería eléctrica como son los flujos de potencia, valiéndose para esto de bases de datos distribuidas para poder fragmentar y asignar los datos de generación y consumo en diferentes sitios. Así como el uso de una interfaz gráfica concurrente que actualiza las interfaces de los diferentes sitios activos en ese momento a través de un servidor concurrente. Cabe destacar que el resultado obtenido al termino de este trabajo no requiere de equipo dedicado, por lo tanto puede ser utilizado equipo existente, así como los sistemas manejadores de bases de datos que se emplean en el trabajo existen bajo Windows y Linux logrando con esto una heterogeneidad entre sistemas operativos.

CONTENIDO

LISTA DE FIGURAS

CAPITULO 1

INTRODUCCIÓN

Los estudios de flujo de potencia son de gran importancia en la planeación y diseño de la expansión futura de los sistemas eléctricos de potencia, así como también en la determinación de las mejores condiciones de operación de los sistemas existentes. En los últimos años los sistemas de potencia han crecido enormemente y geográficamente se han expandido aun más, y hay muchas interconexiones entre sistemas vecinos. La planeación apropiada, la operación y el control de estos sistemas a gran escala, requieren técnicas computacionales avanzadas, como la programación de métodos numéricos.

Los sistemas de cómputo han contribuido al desarrollo del estudio de flujos de potencia, un sistema se puede dividir en áreas (sitios) o un estudio puede incluir los sistemas de varias compañías con lo cual se propone el uso de bases de datos distribuidas para lograr resolver el problema de flujo de potencias de todas las compañías sin necesidad de realizar un estudio a cada uno de los sitios de la compañía que lo requiera.

El ingeniero que planea la transmisión puede descubrir debilidades en el sistema, como el caso de voltajes bajos, sobrecargas en líneas o condiciones de carga que juzgue excesivas. Estas debilidades pueden ser removidas al hacer estudios de diseño que incluyan los cambios y/o adiciones al sistema. Entonces el modelo del sistema se sujeta a una prueba de contingencia a través de los sistemas de cómputo para descubrir si las debilidades surgen bajo estas condiciones, involucrando la programación de generación o de niveles de carga anormales. La interacción entre el diseñador del sistema y el programa de estudio de flujos de potencia que se tiene en la computadora continúa hasta que el comportamiento satisface la planeación local y regional o el criterio de operación.

1.1 ANTECEDENTES

Anteriormente el estudio de flujos de potencia se ejecutaba en analizadores de redes de corriente alterna, los cuales suministraban una reproducción a pequeña escala y monofásica de la red real al interconectar los elementos del circuito y fuente de voltaje. Efectuar las conexiones, hacer los ajustes y leer los datos era tedioso y requería de mucho tiempo.

Ahora con el desarrollo de las computadoras se puede lograr hallar las soluciones del estudio de flujos de potencia de sistemas complejos, se pueden manejar sistemas de más de 2000 barras, 3000 líneas y 500 transformadores. Los resultados completos son obtenidos de manera rápida y económicamente, simplemente con la impresión de algunas hojas.

1.2 OBJETIVO

El propósito de esta tesis es desarrollar un sistema de cómputo que realice un estudio de flujo de potencias apoyado de modelos de cómputo como bases de datos distribuidas, programación concurrente e interfaces graficas para la representación de diagramas eléctricos, haciendo énfasis que con ayuda de tales herramientas el estudio de flujos de potencia se puede realizar en menor tiempo y costo posible y de una manera más exacta que como se realizaba anteriormente.

1.3 ESTADO DEL ARTE

Existen compañías que realizan cálculos de flujos de potencias por medio de software tales como CYME, NEPLAN, CAPE, SIDAC, estas compañías realizan un profundo análisis en el estudio de flujos de potencia, algunas características importantes del software realizado por estas compañías son:

CAPE [1]

- Primer programa de flujo de potencia en 1975.
- Empleo de rápidos y exactos métodos de solución Newton y de desacoplamiento.
- Distintos modelos de dispositivos y de control.
- Impresión del diagrama en blanco y negro o a colores.
- Generación de informes para una o todas las barras.

NEPLAN [2]

- Desarrollado por primera vez en 1989.
- Analiza, planea , optimiza y almacena redes de potencia.
- Búsqueda de datos SQL para MS-Acces, Oracle.
- Conexión para bases de datos ODBC (Conectividad abierta a bases de datos).
- Almacenamiento de diferentes de redes.
- Técnicas multilíneas.
- Elección flexible para el despliegue de resultados.

PSAF-FLOW [3]

- Utilización del método numérico de Newton-Raphson.
- Algoritmo de solución Gauss-Seidel.
- Métodos vectoriales que usan matrices y vectores huecos.
- Análisis de sistemas con miles de barras y nodos.
- Entrada directa de datos desde el diagrama unifilar mediante el mouse.
- Entrada de datos tabular por medio de hojas de cálculo.
- Acceso simultáneo a bases de datos de equipos.
- Generación automática de las bases de datos de equipos y el diagrama unifilar de la red.
- Importación de datos desde archivos CYMFLOW bajo DOS.
- Unidades de medida inglesa o métricas.
- Visualización en el diagrama unifilar de los resultados del cálculo de flujo de potencia.
- Visualización de reportes específicos a cada componente de red en el diagrama unifilar para comparar resultados entre distintas simulaciones.
- Reportes tabulares detallados para describir los voltajes en las barras, ángulos y estados de carga asociados.
- Reportes tabulares detallados para describir flujos de potencia y pérdidas asociadas.
- Posibilidad de transferir todos los reportes tabulares a procesadores de texto mediante el portapapeles.

SIDAC [4]

- Orientado a estudios eléctricos.
- Soporta redes muy extensas de hasta 10000 o más nodos con rápida, precisa y robusta solución en base a un método propio desarrollado para cualquier tipo de redes.
- La dinámica SIDACFC permite crear a partir de un caso padre, todo una genealogía de casos en distinto orden jerárquico para preservar distintos casos analizados de interés del usuario.
- Estimación de demanda de potencia para flujo de potencia y otras aplicaciones relacionadas.
- Los resultados de flujo de potencia son automáticamente vaciados sobre el diagrama unifilar y en tablas muy detalladas.

Se han analizado algunas de las características más importantes de estos sistemas, pero ninguno de los sistemas analizados anteriormente presenta una alternativa de cómputo distribuido como el propuesto en esta tesis.

1.4 JUSTIFICACIÓN

La idea de este trabajo surge debido a los enormes avances que existen hoy en día en los sistemas de cómputo, los cuales pueden contribuir a realizar los estudios de flujos de potencia para que estos estudios sean menos tediosos y que requieran menor cantidad de tiempo. Esto debido principalmente al enorme crecimiento que han tenido los sistemas de potencia, los cuales se han ido expandiendo cada vez más. Por lo tanto en muchas ocasiones el estudio del flujo de potencias es hecho desde diferentes áreas (sitios) y el cómputo distribuido y la programación concurrente proveen herramientas útiles que pueden ser de gran ayuda para realizar estos estudios de manera más rápida y eficaz.

La propuesta de usar manejadores de bases de datos en Windows surge principalmente por ser el sistema operativos más usado y por la facilidad de manejo, la idea de usar Linux surge por la facilidad con que puede manejarse el cómputo distribuido en este sistema, además de que muchas empresas grandes están intentando migrar sus aplicaciones a este sistema operativo.

Una de las herramientas del cómputo distribuido son las bases de datos distribuidas, las cuales se emplean en este trabajo debido a sus numerosas ventajas que ofrecen, entre las cuales destaca la de poder colocar físicamente los datos en el lugar donde se accesan más frecuentemente por la aplicación para lograr un acceso más rápido, la posibilidad de que cuando un nodo falle no afecte a todo el sistema.

Este proyecto es visto desde la perspectiva de las bases de datos distribuidas, por lo tanto el balance de carga de trabajo queda pendiente para el campo de sistemas distribuidos.

CAPITULO 2

SISTEMAS DE POTENCIA

La ingeniería eléctrica se encarga del estudio de varios temas de investigación: estudio de flujos de potencia, estabilidad de sistemas de potencia, estimación de estado de potencia, operación económica de sistemas de potencia, etc. El presente trabajo se enfoca hacia el campo del estudio de flujos de potencia.

2.1 SISTEMAS ELÉCTRICOS DE POTENCIA

Las estaciones generadoras de potencia de todo el país se encuentran interconectadas entre si. Dada una interconexión de generadores, transformadores, líneas de transmisión y otros componentes constituyen un sistema eléctrico de potencia, comúnmente conocido como sistema de potencia.

GENERADORES

Generalmente son generadores síncronos [5] y [6]. Las máquinas síncronas operan a velocidad y frecuencia constante bajo condiciones uniformes. Las máquinas síncronas son capaces de operar como motores o generadores. Como generador estas máquinas síncronas son impulsadas por una turbina para convertir la energía mecánica en eléctrica, además los generadores son la principal fuente de generación de energia eléctrica en el mundo.

TRANSFORMADORES

Los transformadores [5] y [7] son los enlaces entre los generadores del sistema de potencia y las líneas de transmisión y entre líneas de diferentes niveles de voltaje. El transformador baja los voltajes a los niveles requeridos para uso residencial 240/120 V.

LÍNEAS DE TRANSMISIÓN

Una línea de transmisión [5] y [6] constituye el corredor principal de energía en un sistema de potencia. Generalmente son de acero o aluminio y de tamaño pequeño (aproximadamente 1 centímetro de diámetro).

2.2 FLUJOS DE POTENCIA

Un estudio de flujos de potencia [8] y [9] es la determinación del voltaje, la corriente, potencia y factor de potencia o potencia reactiva en varios puntos de una red eléctrica, en condiciones normales de funcionamiento. Los estudios de cargas son fundamentales en la programación del futuro desarrollo del sistema, puesto que su funcionamiento satisfactorio depende del conocimiento de los efectos de la interconexión con otras redes, de las nuevas cargas, de las nuevas centrales generadoras y de las nuevas líneas de transmisión, antes de que se instalen.

Con un estudio de flujos se puede investigar lo siguiente[9]:

- Flujo en KW o KVAR en las ramas de una red.
- Voltaje en los buses.
- Efecto del rearreglo de circuitos e incorporación de nuevos circuitos de carga.
- Efectos de perdidas temporales de generación o de circuitos de transmisión sobre las cargas del circuito.
- Condiciones óptimas de operación del sistema y de distribución de cargas.
- Influencia del cambio de tamaño en los conductores.
- Posición óptima del cambiador de derivaciones de los transformadores.

2.2.1 ESTUDIO DE FLUJOS DE POTENCIA

El estudio de flujos de potencia se puede hacer en diferentes sistemas, y el grado de complicación varía de acuerdo con el número de elementos del circuito. Los sistemas más simples son los que se conocen como *sistemas radiales*. Un sistema *radial* puede ser un sistema de distribución con varias cargas. Los cálculos para un estudio de flujos en sistemas no radiales, aún en sistemas pequeños, son demasiado laboriosos para ser hechos a mano.

Los métodos de estudio de flujos en sistemas radiales no son los que se emplean normalmente en sistemas grandes, ya que en éstos el trabajo resultaría en exceso tedioso y prácticamente imposible de realizar. La solución es representar el modelo de la red a escala y resolverlo con un analizador de redes o con una micro-red [10].

Con el desarrollo de las computadoras a partir de 1950, la atención de los ingenieros en potencia se ha concentrado en el uso de métodos numéricos para el análisis. El uso de la computadora tiene indudables ventajas sobre el analizador de redes o sobre la micro-red, ya que puede resolver sistemas más grandes y complicados en un tiempo menor a causa de su rapidez en la realización de operaciones aritméticas. Los métodos para el estudio de flujos aparecieron desde 1954, y en la actualidad los sistemas computacionales en algunos casos han hecho obsoleto al analizador de redes y a la micro-red, debido a su precisión, velocidad y aplicaciones que se le pueden dar.

2.2.2 ELEMENTOS QUE INTERVIENEN EN EL ESTUDIO DE FLUJOS

El inicio de un estudio de flujo de potencias debe ser un diagrama unifilar del sistema en que a cada bus o nodo se asocian cuatro cantidades [8], que son:

Potencia activa.	P
Potencia reactiva.	Q
Voltaje.	V
Ángulo de fase.	δ

El estudio de flujo se hace normalmente con base en buses o nodos, y se representan tres tipos de estos buses o nodos:

- Nodo flotante o compensador.
- Nodo de generación.
- Nodo de carga.

Nodo flotante o compensador: El nodo flotante [8] y [9] es un nodo en el cual se conocen la magnitud del voltaje | V | y su ángulo de fase (δ); en este nodo se desconocen la potencia activa P y la potencia reactiva Q. El generador conectado a este nodo se encarga de suministrar las pérdidas a la red. El ángulo del voltaje en la barra de compensación sirve como referencia para los ángulos de todos los demás voltajes de barra. El ángulo particular que se asigne al voltaje de la barra de compensación no es de importancia porque las diferencias voltaje-ángulo determinan los valores calculados de P_i y Q_i. Generalmente se le asigna un ángulo de fase de cero grados ($\delta=0$).

Nodo de generación o de voltaje controlado: En el nodo de generación [8] y [9] se conocen la magnitud del voltaje de operación | V | y la potencia activa P, ya que estas cantidades se pueden controlar físicamente en la planta generadora. En este nodo se desconocen la potencia reactiva Q y el ángulo del voltaje de generación δ. En las barras en las que hay un generador conectado se puede controlar la generación de megawatts por medio del ajuste de la fuente de energía mecánica y la magnitud del voltaje puede ser controlada al ajustar la excitación del generador. Después de que se ha resuelto el problema de flujos de potencia, se puede calcular la potencia reactiva.

Nodo de carga: El nodo de carga [8] y [9] es aquel en el cual hay demanda de energía, y en el que se conocen las potencias activas P y reactiva Q, y se desconocen la magnitud del voltaje | V | y ángulo de fase δ.

Se puede observar que se tienen cuatro variables por nodo, de las cuales dos son conocidas, es decir que para un sistema de N nodos se requiere resolver un sistema de 2N ecuaciones simultaneas.

Para el cálculo de los voltajes en los diferentes nodos se debe tomar un nodo de referencia, que usualmente es el neutro del sistema, al que se le fija un valor de cero. Para ángulos de voltajes la referencia es el ángulo del nodo compensador, que como ya se menciono anteriormente es cero (δ=0).

Nótese que en los nodos flotantes, de generación y de carga, las dos variables que se conocen en uno no son las mismas en los otros dos. Debido a esto no hay un número suficiente de ecuaciones en base de nodos o en base de mallas que permita definir el sistema, ya que por cualquiera de los métodos se tienen más incógnitas que ecuaciones.

Para resolver el problema, se recurre a la adición de ecuaciones que tengan cantidades conocidas como P y Q y entonces se transforma en un problema no lineal. El problema no lineal consiste en un conjunto de ecuaciones no lineales que describe al sistema y cuya solución se puede obtener en computadora por medio del análisis numérico.

En la solución del problema de flujos el análisis numérico representa uno de los pasos intermedios necesarios, pero en realidad es solo una herramienta matemática, ya que el problema tiene solución fundamentalmente con base en la solución de redes.

Uno de los métodos que más se emplea para la solución de flujos si se acompaña desde luego de un método numérico es el método de nodos [8] y [9] (método de admitancia Y_{BUS})

2.2.3 MÉTODO DE NODOS (MÉTODO DE ADMITANCIA YBUS)

El método de admitancia [8] y [9] se basa en el empleo de las ecuaciones de nodo para corriente y de la potencia compleja S para un nodo. El proceso de solución es iterativo y normalmente no requiere del uso de matrices inversas.

Las ecuaciones fundamentales que se usan en este método son las siguientes:

$$I_j = \sum_{K=1}^{N} Y_{jk} V_k \qquad \text{para } j = 1,2, \ldots, N \tag{2.1}$$

Donde

I_j = Suma de corrientes que entran al nodo j.
Y_{jk} = Suma de las admitancias conectadas al nodo j cuando j = k
Y_{jk} = Negativa de la suma de las admitancias conectadas entre j y k cuando $j \neq k$
V_k = Voltaje del nodo k.
N = Número de ecuaciones de nodo independientes.

La potencia compleja en un nodo es:

$$S_j = P_j + jQ_j = V_j I_j^* \tag{2.2}$$

Por lo tanto tomando (2.1) podemos definir

$$I^* = \sum_{K=1}^{N} Y^*_k V^*_k \tag{2.3}$$

Substituyendo (2.3) en (2.2)

$$S_j = V_j \sum_{K=1}^{N} Y^*_{jk} V^*_k \qquad \text{para } j = 1,2, \ldots, N \tag{2.4}$$

En el estudio de flujos, la ecuación (2.1) es en realidad matricial y tiene la forma

$$[I]_{bus} = [Y]_{bus} [V]_{bus} \tag{2.5}$$

$[I]_{bus}$ = vector de corrientes impresas en los buses.

$$[I]_{bus} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ \vdots \\ I_r \end{bmatrix} \tag{2.6}$$

$[V]_{bus}$ = vector de voltajes de los buses medidos con respecto al bus de referencia.

$$[V]_{bus} = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_r \end{bmatrix} \tag{2.7}$$

$[Y]_{bus}$ = matriz de admitancias de bus.

$$[Y]_{bus} = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{ik} & \cdots & Y_{1n} \\ Y_{21} & Y_{22} & \cdots & Y_{2k} & \cdots & Y_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ Y_{j1} & Y_{j2} & \cdots & Y_{jk} & \cdots & Y_{jn} \\ \vdots & \vdots & & \vdots & & \vdots \\ Y_{n1} & Y_{n2} & \cdots & Y_{nk} & \cdots & Y_{nn} \end{bmatrix} \tag{2.8}$$

Las ecuaciones (2.1) y (2.4) dan lugar a 2N ecuaciones, con lo cual se resuelve el problema de las 2N incógnitas.

En la ecuación (2.4) se puede observar que existen productos de voltajes y entonces las ecuaciones que se obtienen son no lineales; es necesario el empleo de métodos numéricos para la solución.

Los métodos numéricos más comunes que se emplean en la solución de problemas de flujos son:

- Método de Jacobi
- Método de Gauss-Seidel
- Método de Newton-Raphson

2.2.4 MÉTODO DE JACOBI

El método de Jacobi [8] y [9], que se aplica a la solución de flujos, supone los voltajes como variables.

Considerando el voltaje de un bus k, de un sistema de N buses se tiene:

La potencia compleja para el bus k

$$V^*_k I_k = S_k \tag{2.9}$$

$$V^*_k I_k = P_k - jQ_k \tag{2.10}$$

Donde

$$I_k = Y_{k1}V_1 + Y_{k2}V_2 + \ldots + Y_{kk}V_k + \ldots + Y_{kj}V_j \tag{2.11}$$

Substituyendo (2.7) en (2.5) obtenemos

$$V^*_k\left(Y_{kk}V_k + \sum_{j=1}^{N} Y_{kj} V_j \right) = P_k - jQ_k \qquad j \neq k \qquad (2.12)$$

Despejando V_k

$$V_k = \frac{1}{Y_{kk}} \left[\frac{P_k - jQ_k}{V^*_k} - \sum_{j=1}^{N} Y_{kj} V_j \right] \qquad (2.13)$$

Es necesario considerar al nodo k como nodo de carga.

Cuando la barra que se analiza es de generación, entonces se desconoce la potencia reactiva, las componentes real e imaginaria del voltaje para cada iteración se encuentran calculando primeramente un valor para la potencia reactiva.

$$Q_i = - \text{Im} \left\{ Vi \sum_{j=1}^{N} Y_{ij} V_j \right\} \qquad (2.14)$$

Que tiene la expresión algorítmica equivalente

$$Q^{(k)}_i = - \text{Im} \left\{ V^{(k+1)}_i \left[\sum_{j=1}^{N} Y_{ij} V^{(k)}_j + \sum_{j=i}^{N} Y_{ij} V^{(k-1)}_j \right] \right\} \qquad (2.15)$$

Donde Im quiere decir "parte imaginaria de"

Si se supone que existe un solo nodo compensador, los pasos para encontrar la solución al problema son los siguientes [8]:

1. En la primera aproximación, se supone que se conocen los voltajes en todos los nodos del sistema, excepto el nodo compensador, cuyo valor se establece previamente y es $1.0 \angle 0°$.

2. Con la ecuación (2.13) y conociendo todos los valores de voltajes, si se considera al nodo K como nodo de carga, se obtiene el valor de V_k mejorado.

3. Con los valores mejorados de V_k calculados en el paso anterior, se recalculan los voltajes de cada nodo, hasta que las diferencias entre dos grupos de ellos sea muy pequeña; Cuando esto se logra, se obtiene la solución del problema.

2.2.5 MÉTODO DE GAUSS-SEIDEL

El método de Gauss-Seidel [8] y [9] es una variación del método de Jacobi; para cada paso de iteración se emplea la solución anterior y con esto se logra que la solución converja más rápidamente.

Es decir que, por este método, en las ecuaciones que se emplean por el método de Jacobi el nuevo voltaje calculado $V^{i+1}{}_k$ substituye a $V^1{}_k$ y se usa en la solución de las subsecuentes ecuaciones.
El superíndice i es el número de iteración.

$$V^{(k)}{}_i = \frac{1}{Y_{ii}} \left[\frac{P_i - jQ_i}{V^{(k-1)*}{}_i} - \sum_{j=1} Y_{ij} V^{(k)}{}_j - \sum_{j=i+1} Y_{ij} V^{(k-1)}{}_j \right] \qquad (2.16)$$

Con los valores finales de nodo y conociendo los valores de las admitancias de las líneas se calculan los flujos en las líneas con la ecuación

$$P_{jk} - jQ_{jk} = V^*{}_j (V_j - V_k) Y_{jk} \qquad (2.17)$$

2.2.6 MÉTODO DE NEWTON-RAPHSON

El método de Newton-Raphson [4] y [5] requiere que se exprese una relación entre los cambios de potencia activa y reactiva, y los cambios en magnitud y ángulo de voltaje de bus.

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} H = \dfrac{\partial P}{\partial \delta} & N = \dfrac{\partial P}{\partial |E|} \\ J = \dfrac{\partial Q}{\partial \delta} & L = \dfrac{\partial Q}{\partial |E|} \end{bmatrix} \begin{bmatrix} \Delta \delta \\ \dfrac{\Delta E}{|E|} \end{bmatrix} \qquad (2.18)$$

En esta matriz tiene un orden 2n – 1, ya que el nodo compensador no se incluye y está comúnmente como un jacobiano.

Por este método, los cambios en magnitud y ángulo de los voltajes de bus se calculan de las potencias residuales de bus.

Se puede observar que para resolver la ecuación (2.18) se tiene que pasar el jacobiano al lado izquierdo de la ecuación; esto se puede hacer por inversión del jacobiano, o triangulado y resolviendo por substituciones.

La ecuación para la potencia compleja se puede expresar como:

$$P_j - jQ_j = V^*_j \sum_{k=1}^{N} Y_{jk} V_k \qquad (2.19)$$

Y_{jk} se define en la misma forma que para (2.1); además,

$$V_j = | V | e^{j\delta j} \tag{2.20}$$

$$Y_{jk} = | Y_{jk} | e^{j\delta j} = G_{jk} + jB_{jk} \tag{2.21}$$

La ecuación (2.19) se puede expresar como:

$$P_j - jQ_j = \sum_{k=1}^{N} | E_j E_k Y_{jk} | e^{j(\theta jk + \delta j - \delta k)} \tag{2.22}$$

Separando la ecuación (2.22) en sus partes reales e imaginarias

$$P_j = \sum_{k=1}^{N} | E_j E_k Y_{jk} | \cos (\theta jk + \delta j - \delta k) + |E_j|^2 Y_{jj} \cos \theta_{jj} \quad \text{para } k \neq j \tag{2.23}$$

$$Q_j = - \sum_{k=1}^{N} | E_j E_k Y_{jk} | \operatorname{sen} (\theta jk + \delta j - \delta k) - |E_j|^2 Y_{jj} \operatorname{sen} \theta_{jj} \quad \text{para } k \neq j \tag{2.24}$$

De las ecuaciones (2.22), (2.23) y (2.24) se obtienen los elementos del jacobiano, como sigue:

$$H_{jk} = \frac{\partial P_j}{\partial \delta_k} = \operatorname{Im} [V^*_j V_k Y_{jk}] \tag{2.25}$$

$$J_{jk} = \frac{\partial Q_j}{\partial \delta_k} = - \operatorname{Re} [V^*_j V_k Y_{jk}] \tag{2.26}$$

$$N_{jk} = \frac{\partial P_j}{\partial |E_k|} |E_k| = - J_{jk} \qquad (2.27)$$

$$L_{jk} = \frac{\partial Q_j}{\partial |E_k|} |E_k| = H_{jk} \qquad (2.28)$$

$$H_{jj} = \frac{\partial P_j}{\partial \delta_j} = - \sum_{k=J}^{N} \text{Im} [V^*_j V_k Y_{jk}] \quad \text{para} \quad k \neq j \qquad (2.29)$$

$$N_{jj} = \frac{\partial P_j}{\partial |E_j|_j} |E_p| = \sum_{k=1}^{N} \text{Re}[V^*_j V_k Y_{jk}] + 2G_{jj}|E_j|^2 \quad \text{para } k \neq j \qquad (2.30)$$

$$J_{jj} = \frac{\partial Q_j}{\partial \delta_j} = \sum_{k=1}^{N} \text{Re} [V^*_j V_k Y_{jk}] \quad \text{para } k \neq j \qquad (2.31)$$

$$L_{jj} = \frac{\partial Q_j}{\partial |E_j|} |E_p| = \sum_{k=1}^{N} \text{Im} [V^*_j V_k Y_{jk}] - 2G_{jj}|E_j|^2 \quad k \neq j \qquad (2.32)$$

.

Los elementos del jacobiano se obtienen de la ecuación (2.19) y desde luego se calculan primero los elementos fuera de la diagonal.

Los pasos a seguir para la solución por este método son los siguientes:

1. Por el método de admitancia de BUS y empleando la solución por Gauss-Seidel, ejecutar una iteración para obtener la primera estimación de voltajes de bus.

2. Calcular los elementos del jacobiano.

3. Calcular las potencias residuales:

$$\Delta P^{i}_{k} = P_k \text{ (deseada)} - P^{i}_{k} \text{ (calculada)} \qquad (2.33)$$

$$\Delta Q^{i}_{k} = Q_k \text{ (deseada)} - Q^{i}_{k} \text{ (calculada)} \qquad (2.34)$$

4. Resolver los cambios de voltaje en magnitud y ángulo.

5. Calcular los nuevos voltajes en los buses.

$$| V_k |^{i+1} = | V_k |^{i} + \Delta | V_k | \qquad (2.35)$$

$$\delta^{i+1} = \delta^{i}_{k} + \delta_k \qquad (2.36)$$

6. Verificar las potencias residuales para ver si hay una tolerancia que se desea. Si la hay, existe una solución; si no , ir al paso 2.

CAPITULO 3

CÓMPUTO DISTRIBUIDO

3.1 INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS

La capacidad de conectar a los usuarios con los recursos computacionales por medio de comunicaciones es esencial para las organizaciones, ya que una inversión razonable en hardware y software hace que el acceso a las computadoras este disponible a todas las personas que lo necesitan.

Los costos de comunicación están aumentando mientras que el costo de las computadoras esta disminuyendo. Ahora es posible colocar computadoras en cada lugar donde se recolecten los datos o se necesite la información.

3.1.1 CONCEPTO

Un sistema distribuido [11] interconecta los lugares que tienen recursos computacionales para capturar y almacenar datos, procesarlos y enviar datos e información a otros sistemas, tales como un sistema central. El rango de recursos computacionales varía. Algunos lugares utilizan terminales, otros microcomputadoras, otros incluso grandes sistemas de cómputo. No existe el requisito de que todo el equipo sea del mismo fabricante. De hecho se espera que estén

implicadas varias marcas de hardware. Esto permite al usuario tener el tipo más adecuado a sus necesidades.

3.1.2 COMUNICACIONES

Para cumplir con las condiciones impuestas de portabilidad y heterogeneidad la solución ideal es el uso de protocolos de comunicaciones disponibles universalmente. La familia de Internet [12] es la más extendida de todas, en sus dos formas: TCP orientado a conexión y UDP mediante datagramas.

La portabilidad está asegurada ya que es posible encontrar implementaciones de IP en las arquitecturas más comunes: Solaris, SunOS, Linux, Iris, HP-UX, DEC Unix, WindowsNT/95, etc.. en cuanto a la heterogeneidad, la propia existencia de Internet (que cuenta con el mayor número de maquinas diferentes en subredes) nos garantiza la posibilidad de que sistemas totalmente distintos se comuniquen entre sí.

Un caso específico de los sistemas distribuidos son las *bases de datos distribuidas,* las cuales ofrecen diversas ventajas, entre las cuales destaca la de poder colocar físicamente los datos en el lugar donde se accesan más frecuentemente por la aplicación para lograr un acceso más rápido, la posibilidad de que cuando un nodo falle no afecte a todo el sistema. Debido a las razones mencionadas anteriormente el presente trabajo se enfoca al empleo de las bases de datos distribuidas.

3.2 BASES DE DATOS DISTRIBUIDAS

3.2.1 CONCEPTO

Una base de datos distribuida [13] y [14] es un conjunto de múltiples bases de datos lógicamente interrelacionadas que se encuentran distribuidas en una red de computo.

3.2.2 DIFERENCIA ENTRE BASES DE DATOS DISTRIBUIDAS Y CENTRALIZADAS

La principal diferencia [13] y [15] entre las bases de datos distribuidas y bases de datos centralizadas radica en que en las bases de datos centralizadas los datos se localizan en *una sola computadora*, no así en las bases de datos distribuidas, en las cuales los datos se localizan en *diferentes computadoras*.

3.2.3 ESTRUCTURA DE LAS BASES DE DATOS DISTRIBUIDAS

Como ya se mencionó anteriormente un sistema distribuido de base de datos consiste en un conjunto de recursos computacionales, cada uno de las cuales mantiene un sistema de base de datos local.

Existen condiciones para que se trate de una base de datos distribuida:

- Cada computadora está conciente de la existencia de las demás.
- Cada computadora permite ejecutar transacciones tanto locales como globales.

3.2.4 VENTAJAS DE LAS BASES DE DATOS DISTRIBUIDAS

Las principales ventajas del uso de las bases de datos distribuidas son las siguientes:

1. Los datos están localizados en un lugar más cercano [13], por lo tanto el acceso es más rápido.

2. El procesamiento es rápido [13] debido a que varios nodos intervienen en el procesamiento de una carga de trabajo, nuevos nodos se pueden agregar fácil y rápidamente.

3. La probabilidad de que una falla en un solo nodo afecte al sistema es baja, es decir un sitio puede fallar mientras los demás siguen funcionando correctamente [14] y existe una autonomía e independencia entre los nodos.

4. Posibilidad de compartir [14] los datos en el lugar donde se accesan más frecuentemente al tiempo que se mantiene un control local de los datos.

5. Mediante la replicación de información [13], las bases de datos distribuidas pueden presentar ciento grado de tolerancia a fallas haciendo que el funcionamiento del sistema no dependa de un solo lugar como en el caso de las bases de datos centralizadas.

3.2.5 DESVENTAJAS DE LAS BASES DE DATOS DISTRIBUIDAS

Las principales ventajas del uso de las bases de datos distribuidas son las siguientes:

1. La posibilidad de violaciones de seguridad [13] es creciente si no se toman las precauciones debidas.

2. La habilidad para asegurar la integridad de la información en presencia de fallas no predecibles tanto de componentes de hardware como de software es compleja [13]. La integridad se refiere a la consistencia, validez y exactitud de la información.

3. Menor rendimiento [14] si la carga de trabajo necesita un gran número de actualizaciones concurrentes sobre duplicados que están demasiado distribuidos.

Observemos que la carga de trabajo es distribuida en los sitios logrando que el procesamiento sea más rápido, este punto es más importante para nuestra investigación que la seguridad que pueda presentar el sistema, debido a que la informacion que manejamos no es confidencial. Nos importa más la posibilidad de compartir los datos donde se accesan más frecuentemente, que la seguridad de la integración de la información, debido a la naturaleza de nuestra aplicación, en donde la integración de los datos es controlada directamente por la interfaz.

3.2.6 ENFOQUES AL PROBLEMA DE DISEÑO DE BASES DE DATOS DISTRIBUIDAS

Para abordar el diseño de una base de datos distribuida existen dos enfoques [13]:

Enfoque descendente

Es más apropiado para aplicaciones nuevas y para sistemas homogéneos. Consiste en partir desde el análisis de requerimientos para definir el diseño conceptual y las vistas de usuario. A partir de ellas se define un esquema conceptual global y los esquemas externos necesarios. Se prosigue con el diseño de la fragmentación de la base de datos, y de aquí se continua con la localización de los fragmentos en los sitios, creando las imágenes físicas. Este enfoque se completa ejecutando, en cada sitio, "el diseño físico" de los datos, que se localizan en este.

Enfoque ascendente

Se utiliza particularmente a partir de bases de datos existentes, generando con esto bases de datos distribuidas. En esta forma resumida, el enfoque ascendente de una base de datos distribuida requiere de la selección de un modelo de bases de datos común para describir el esquema

3.2.7 REQUERIMIENTOS DE INFORMACIÓN

Es necesario conocer información cuantitativa relativa a la base de datos, las aplicaciones que se utilizarán, la red de comunicaciones, las capacidades de procesamiento y de almacenamiento de cada nodo en la red [13].

Información sobre la base de datos: es necesario conocer la selectividad de un fragmento R_j con respecto a una consulta Q_i, esto es, el numero de tuplas de R_j que será necesario accesar para procesar Q_i. Este valor se denota como sel (R_j). Así también, es necesario conocer el tamaño de cada fragmento, el cual esta dado por:

Tamaño (R_j) = cardinalidad (R_j) * longitud (R_j) (3.8)

Información sobre las aplicaciones

Es necesario distinguir el número de lecturas que una consulta Q_j hace a un fragmento R_j durante su ejecución, del número de escrituras. Se requiere de una matriz que indique que consultas actualizan cuales fragmentos. Una matriz similar se necesita para indicar las lecturas de consultas a fragmentos. Finalmente se necesita saber cual es el sitio de la red que origina cada consulta.

Información sobre cada sitio de la red

Las medidas utilizadas son el costo unitario de almacenamiento de datos en un sitio y el costo unitario de procesamiento de datos en un sitio.

Información sobre la red de comunicaciones

Las medidas a considerar son la velocidad de comunicación, el tiempo de latencia en la comunicación y la cantidad de trabajo adicional a realizar por una comunicación.

3.2.8 DISEÑO DE BASES DE DATOS DISTRIBUIDAS

Considérese una relación r la cual se va a almacenar en la base de datos, para diseñar una base de datos distribuidas se debe tomar en cuenta lo siguiente factores [13] y [14]:

Replicación

El sistema mantiene varias copias idénticas de la relación. Cada una de las copias se almacena en un sitio diferentes. Lo que da como resultado una replicación de la información.

Fragmentación

Se determina por la forma en que las relaciones globales se subdividen en fragmentos horizontales, verticales o mixtos los cuales se almacenan en sitios diferentes.

3.2.8.1 REPLICACIÓN DE LOS DATOS

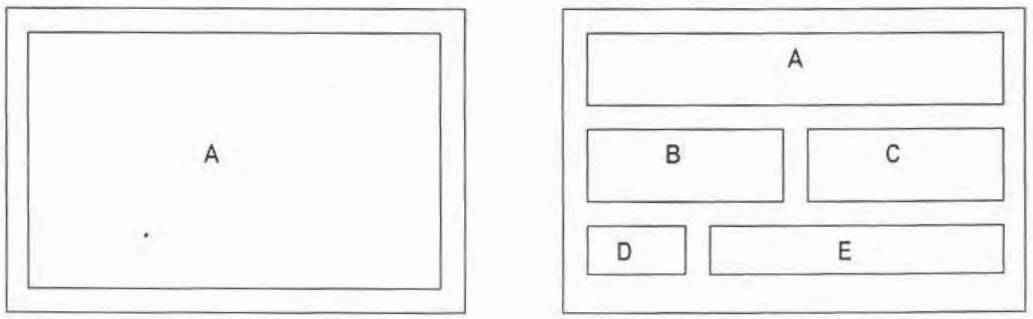
Si la relación r se encuentra repetida, se almacena una copia en las computadoras que así lo requieran. En el caso extremo se tiene replicación total, que consiste en almacenar una copia de la relación r en todas las computadoras en donde se involucre la base de datos distribuida. También se pueden tener replicas parciales solo en algunos sitios donde la aplicación lo requiera.

La replicación mejora el rendimiento de las operaciones de lectura aumentando la disponibilidad de datos para las transacciones de lectura.

3.2.8.2 FRAGMENTACIÓN DE LOS DATOS

Si la relación r está fragmentada, r se dividirá en varios fragmentos r_1, r_2, ..., r_n. Estos fragmentos contiene información suficiente para reconstruir la relación r original. Esta reconstrucción puede llevarse a cabo ya sea aplicando la operación de unión o un tipo especial de operación de unión sobre los diversos fragmentos.

El objetivo de la fragmentación es encontrar un nivel de particionamiento adecuado en el rango que va desde tuplas o atributos hasta relaciones completas.



(a) (b)

Figura 3.1 Relación (a) no fragmentada, (b) fragmentada

Existen tres tipos de fragmentación:

- Fragmentación horizontal
- Fragmentación vertical
- Fragmentación híbrida

El presente trabajo se centrara solamente en la fragmentación horizontal

Fragmentación horizontal

Un fragmento puede definirse como una selección en la relación global r. Es decir, se utiliza un predicado P_i para construir el fragmento r_i de la siguiente manera:

$$r_i = \sigma_{Pi} (r) \qquad (3.1)$$

La reconstrucción de la relación r puede obtenerse al calcular la unión de todos los fragmentos, es decir:

$$r = r_1 \cup r_2 \cup \ldots \cup r_n \qquad (3.2)$$

La fragmentación horizontal es definida a través de una operación de selección en las relaciones propietarias de un esquema de la base de datos.

$$R_j = \sigma_{Fj} (R),\ 1 \leq j \leq w \qquad (3.3)$$

Donde F_j es una fórmula de selección, la cual es preferiblemente un predicado mintermino. Por lo tanto, un fragmento horizontal R_i de una relación R consiste de todas las tuplas de R que satisfacen un predicado mintermino m_i, lo anterior implica que dado un conjunto de predicados mintermino M, existen tantos fragmentos horizontales de r como minterminos existan. Es importante que el conjunto de predicados sea ***completo*** y ***mínimo***.

Un conjunto de predicados simples P_r se dice que es completo si y solo si los accesos a las tuplas de los fragmentos minterminos definidos en P_r requieren que dos tuplas del mismo fragmentos tengan la misma probabilidad de ser accesados por cualquier aplicación.

Un conjunto de predicados simples se dice que es mínimo si y solo si todos los predicados de un conjunto P_r son relevantes, un predicado es relevante si un predicado P_i que pertenezca a P_r no puede ser fragmentado otra vez.

Existe un algoritmo llamado COM_MIN [13] que genera un conjunto completo y mínimo de predicados P_r' dado un conjunto de predicados simples P_r.

Algoritmo 3.1 COM_MIN

Regla:

Una relación o fragmento es particionado en al menos dos partes las cuales se accesan en forma diferente por al menos una consulta de usuario.

Entrada:

Una relación R y un conjunto de predicados simples P_r

Salida:

Un conjunto completo y mínimo de predicados simples P_r' para P_r.

INICIO:

- Encontrar un $p_i \in P_r$ tal que p_i particiona a R de acuerdo a regla
- $P_r' \leftarrow p_i$
- $P_r \leftarrow P_r - p_i$
- $F \leftarrow f_i$

Inicio Ciclo

- Agregar predicados a P_r' hasta que sea completo
- Encontrar un $p_i \in P_r$ tal que p_i particiona algún f_k de P_r' de acuerdo a regla
- $P_r' \leftarrow P_r' \cup p_i$

- $P_r \leftarrow P_r - p_i$
- $F \leftarrow F \cup f_i$
- Si $\exists\ p_k \in P_r'$ el cual no es relevante, entonces:
  - $P_r' \leftarrow P_r' - p_k$
  - $F \leftarrow F - f_k$

Fin ciclo

FIN

Finalmente, la fragmentación es realizada por el algoritmo PHORIZONTAL [13] que toma como entrada una relación R y el conjunto de predicados simples Pr y proporciona como resultado el conjunto de fragmentos de $R = \{ R_1, R_2, ,... R_m \}$ el cual obedecerá las reglas de fragmentación.

Algoritmo 3.2 PHORIZONTAL

Entrada:

Una relación R y un conjunto de predicados simples P_r

Salida:

Un conjunto de predicados minterminos M de acuerdo a los cuales la relación será fragmentada.

INICIO

- $P_r' \leftarrow$ COM_MIN (R, P_r)
- Determinar el conjunto M de predicados minterminos
- Determinar el conjunto I de implicaciones entre $p_i \in P_r$
- Eliminar minterminos contradictorios a partir de M.

FIN

Comprobación de una fragmentación

Al realizar la fragmentación de una relación se deben satisfacer condiciones [13] para garantizar la comprobación de la misma:

1.- Condición de completicidad. La descomposición de una relación r en los fragmentos $r_1, r_2, ..., r_n$ es completa si y solamente si cada elemento de datos en r se encuentra en algún de los r_i.

2.- Condición de reconstrucción. Si la relación r se descompone en los fragmentos r_1, r_2, ..., r_n, entonces debe existir algún operador relacional ∇,tal que:

$$r = \nabla_{1 \le i \le n} r_i \qquad (3.7)$$

3.- condición de disjunción. Si la relación r se descompone en los fragmentos r_1, r_2, ..., r_n, y el dato d_i está en r_j, entonces d_i no debe estar en ningún otro fragmento r_k tal que $k \neq j$.

3.2.9 ASPECTOS DE DISEÑO A CONSIDERAR EN LA DISTRIBUCIÓN DE LOS DATOS

Procesamiento local: la distribución de los datos, para maximizar el procesamiento local corresponde al principio simple de colocar los datos tan cerca como sea posible de las aplicaciones que los utilizan.

Distribución de la carga de trabajo: la distribución de la carga de trabajo sobre los sitios, es una característica importante de los sistemas de cómputo distribuidos. Esta distribución de la carga se realiza para tomar ventaja de las diferentes características (potenciales) o frecuencia de uso de las computadoras de cada sitio, y maximizar el grado de ejecución de paralelismo de las aplicaciones. Sin embargo, la mala distribución de la carga de trabajo podría afectar negativamente el procesamiento local deseado.

Costo de almacenamiento y disponibilidad: la distribución de la base de datos refleja el costo y disponibilidad del almacenamiento en diferentes sitios. Para esto, es posible tener sitios especializados en la red para el almacenamiento de datos. Sin embargo el costo de

almacenamiento de datos no es tan relevante si éste se compara con el CPU, I/O y los costos de comunicación de las aplicaciones.

3.3 PROCESOS CONCURRENTES

3.3.1 SERVIDOR CONCURRENTE [12] y [16]

Un servidor concurrente puede atender a múltiples solicitudes al "mismo tiempo", como podemos ver en el siguiente pseudocódigo:

Maestro:	**Esclavo:**
Ciclo infinito	
{ esperar conexión	
crear esclavo --->	leer solicitud
}	realizar acción
	mandar respuesta
	morir

No se crea un número ilimitado de esclavos, sino hasta una cuota máxima. La acción de crear esclavos depende de las herramientas disponibles en el sistema operativo. En Windows una alternativa es usar *threads*, aunque esta alternativa requiere mucho trabajo de sincronización por parte del programador.

El estilo de comunicación entre cliente y servidor como se indica en la figura 3.2, es muy importante. Un servicio *orientado a conexión* ("connection-oriented") trata las solicitudes de cada cliente como secuencia ordenada. Si esto importa o no, depende de la aplicación, pero se puede asumir que toda la comunicación es libre de errores y los datos llegan en orden. En este proyecto usaremos una conexión TCP para dar estas garantías.

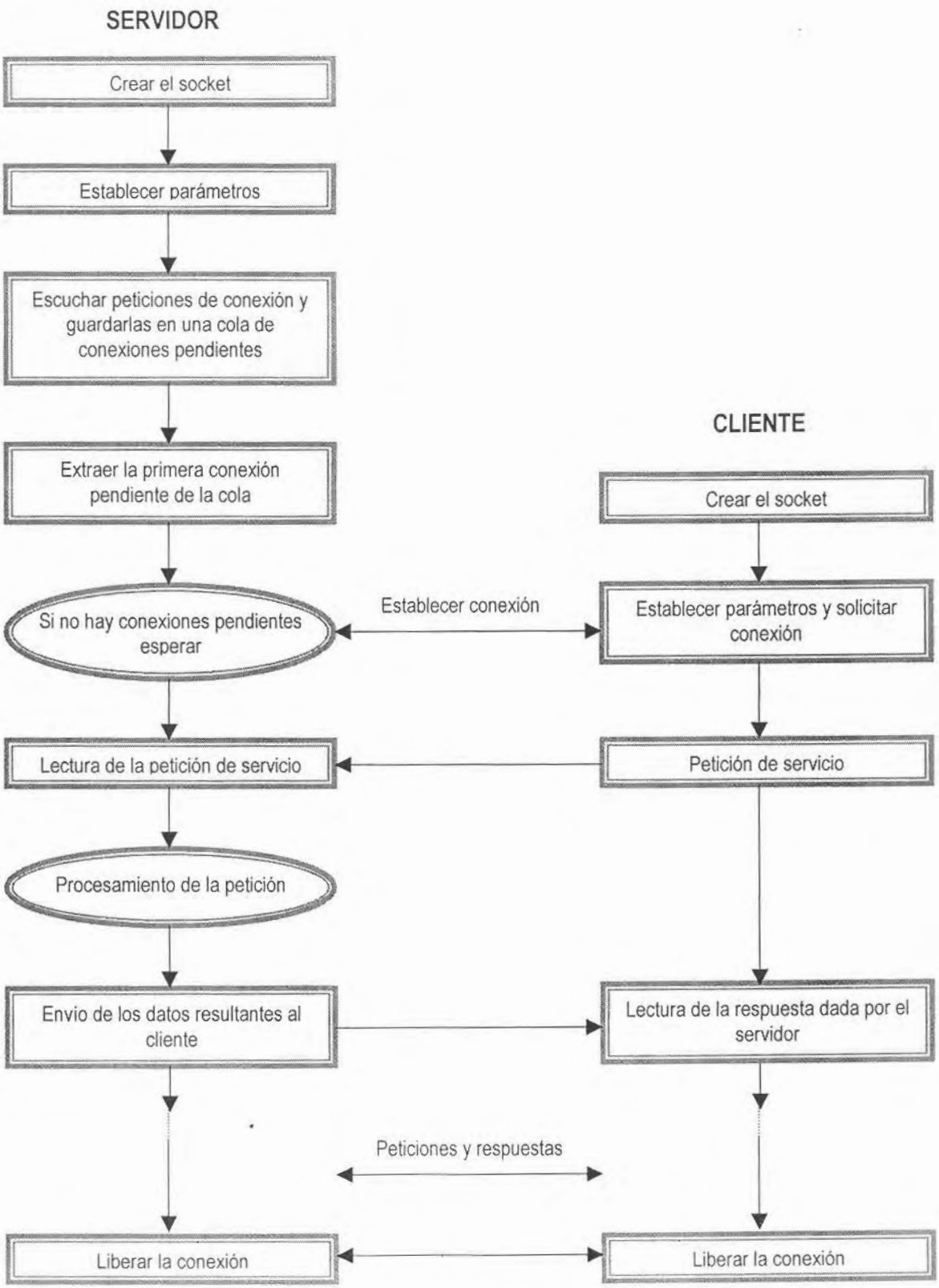


Figura 3.2 Cliente / Servidor

Para la conexión de un cliente con un servidor concurrente [12], primero se intenta una negociación de la conexión entre el cliente y el servidor como se indica en la figura 3.3 (a), después el servidor concurrente pasa la conexión a un proceso hijo como se puede ver en la figura 3.3 (b), Por ultimo el proceso servidor queda en su estado inicial de escucha así, si un segundo proceso cliente negocia una conexión, el servidor concurrente la pasa a un segundo proceso hijo, esta idea se muestra en la figura 3.3 (c).

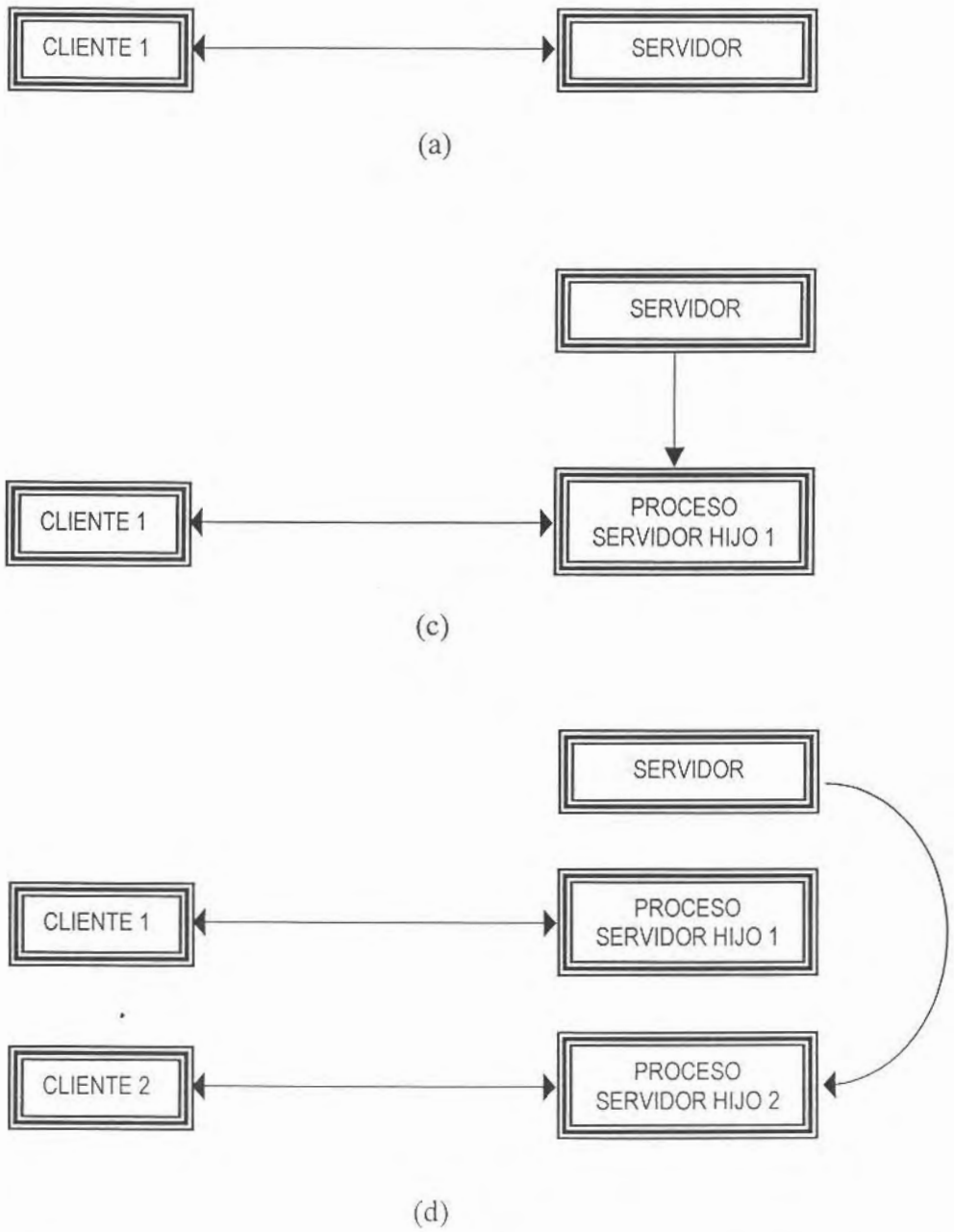


Figura 3.3 Servidor concurrente (a) negociación entre cliente y servidor, (b) conexión de un proceso, (c) nueva conexión de un segundo proceso

3.3.2 CONTROL DE CONCURRENCIA

Cuando se ejecutan varias transacciones de manera simultanea en distintos procesos [16], se necesita cierto mecanismo para mantener a cada uno lejos del camino del otro. Este mecanismo se llama control de concurrencia.

Cerradura

Es el algoritmo más antiguo y de más amplio uso.

Cuando un proceso necesita leer o escribir en un archivo objeto (registro) como parte de una transacción, primero cierra el objeto.

La cerradura [16] se puede hacer mediante un controlador centralizado de cerraduras, o bien con un controlador local de cerraduras en cada maquina, que maneje los objeto locales. En ambos casos, el control de cerraduras mantiene una lista de los objeto cerrados y rechaza todos los intentos de otro proceso por cerrarlos. Puesto que los procesos bien comportados no intentan tener acceso a un objeto antes de ser cerrado, el establecimiento de una cerradura en un archivo mantiene a todos lejos de este, lo cual garantiza que no será modificado durante la transacción. El sistema de transacciones es el que por lo general adquiere y libera las cerraduras y no necesita acción alguna por parte del programador.

CAPITULO 4

DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

El sistema que se plantea para la solución del estudio de flujos de potencia de manera distribuido consta de dos subsistemas como se muestra en la figura 4.1, Interfaz Concurrente y el Servidor, estos dos subsistemas se comunican entre sí a través del protocolo de comunicación TCP/IP.

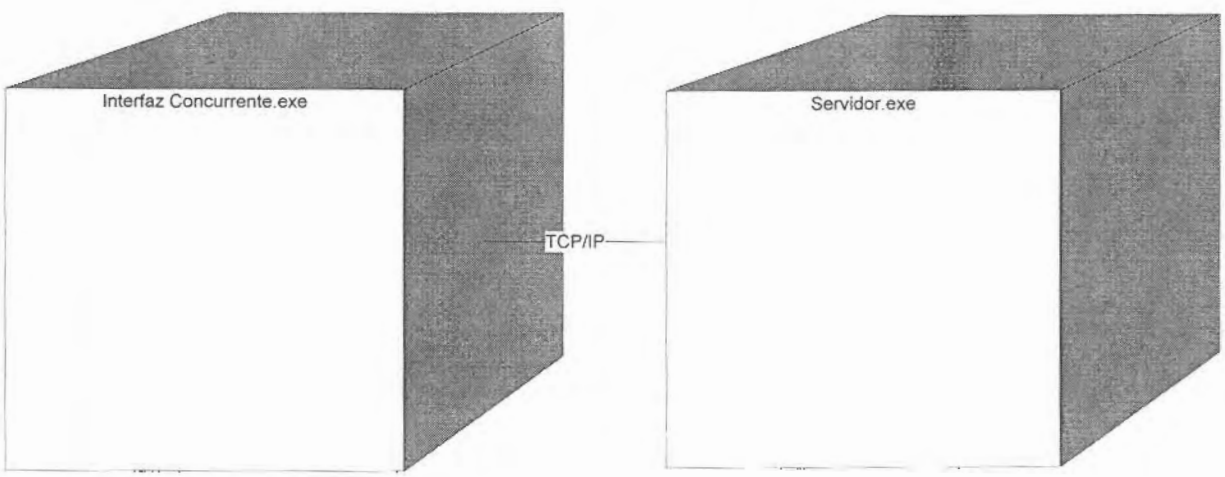


Figura 4.1 Diagrama de despliegue

4.1 DESCRIPCIÓN DEL SISTEMA

La aplicación que se plantea en este trabajo consta de una base de datos distribuida en tres sitios diferentes S1, S2, S3 como se muestra en la figura 4.2, estos sitios tendrán cada uno un fragmento diferente de la relación NODOS la cual fue creada con ***Microsoft Access (bajo Windows 98)***. También consta de otra relación denominada ARCOS la cual no se encuentra fragmentada y fue creada con ***PostgreSQL (bajo Linux)*** y montada en el servidor ndikandi.utm.mx.

Al momento que se quiera insertar, eliminar o modificar un NODO o un ARCO la base de datos será actualizada la interfaz concurrente por medio del servidor concurrente.

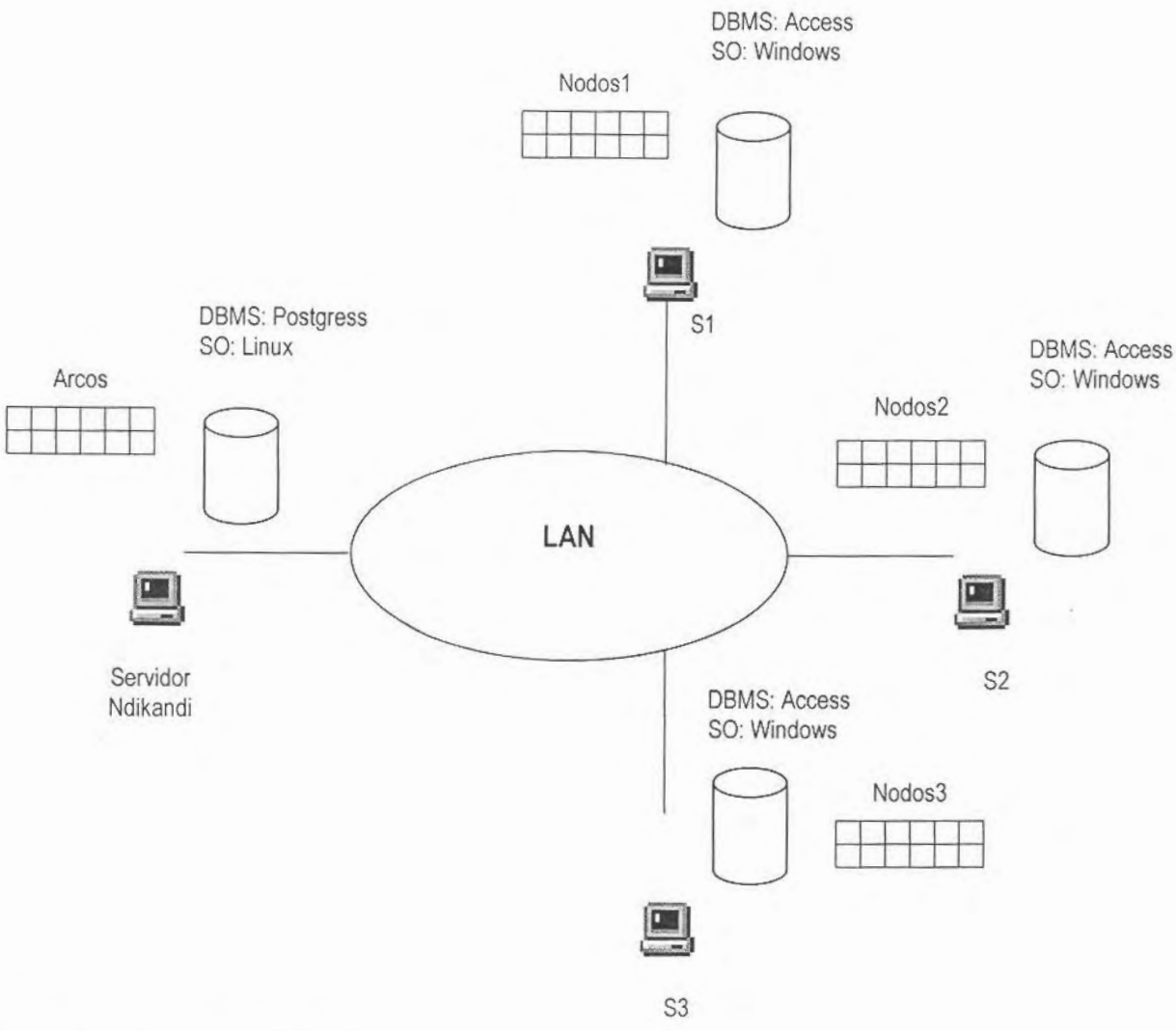


Figura 4.2 Diagrama general de la aplicación

Además es necesario la creación de un servidor concurrente el cual se encargara de actualizar las interfaces graficas de todos los sistemas activos en ese momento cuando ocurra una modificación en el sistema. También se va a encargar de sincronizar a todos los sistemas activos, por último se encarga de verificar la consistencia cuando ocurren modificaciones de forma concurrentes en los sistemas activos en ese momento. La conceptualización de este servidor concurrente se detalla en la figura 4.3, para la implementación de este servidor concurrente se utilizaron hilos y sockets[18]. Como se detallará más adelante.

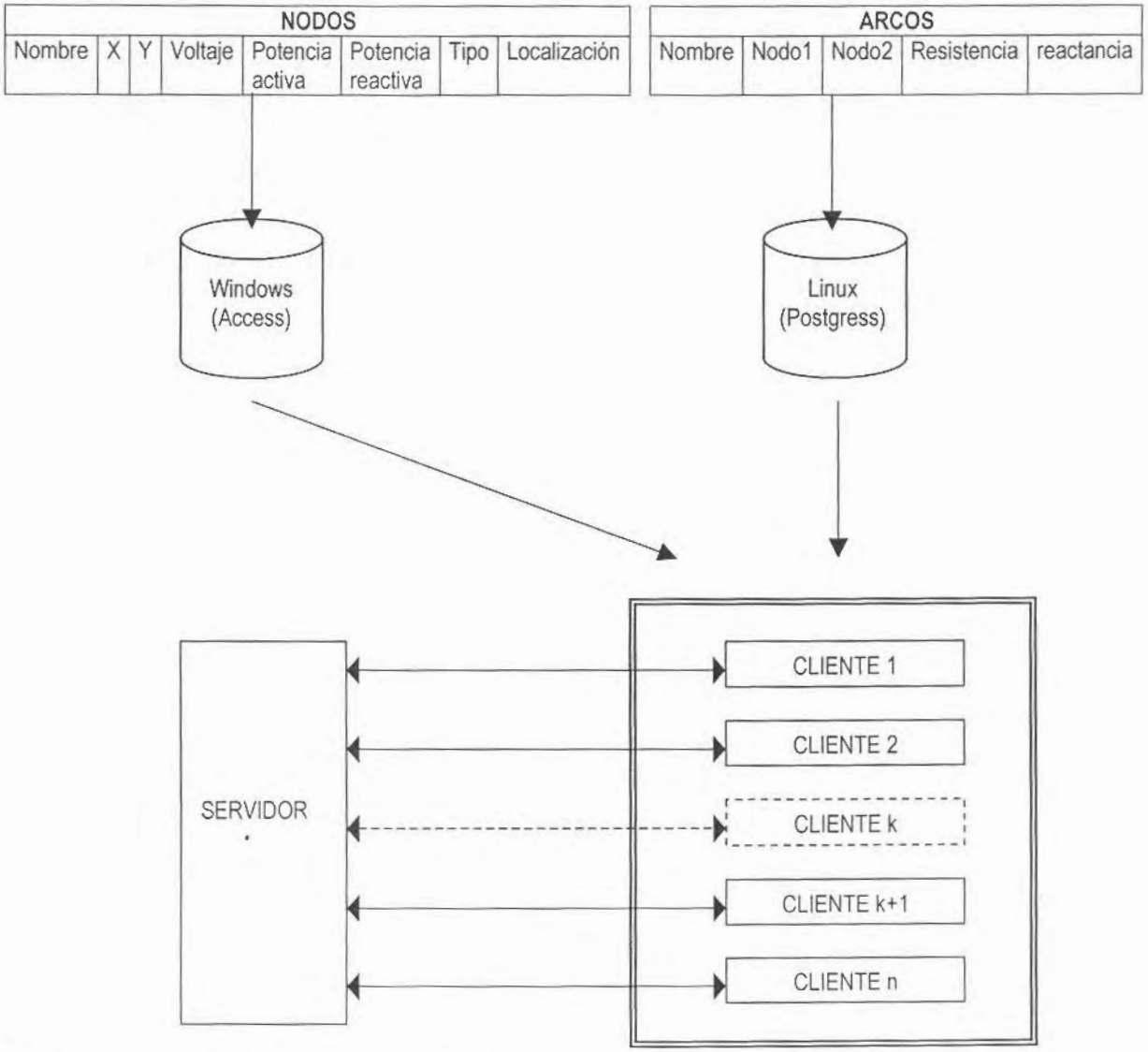


Figura 4.3 Diagrama detallado de la aplicación

4.2 DISEÑO DE LA BASE DE DATOS DISTRIBUIDA

4.2.1 MODELANDO EN E-R

Para el diseño de este sistema se planteo el desarrollo de una base de datos distribuida, la cual será el punto central. Como definimos en la sección 4.1, la base de datos distribuida estará compuesta por dos relaciones NODOS y ARCOS, las cuales pueden relacionarse entre si y modelarse en base a un diagrama E-R como se indica en la figura 4.4 toda la información que se necesita para el estudio de flujo de carga quedara registrada en estas relaciones.

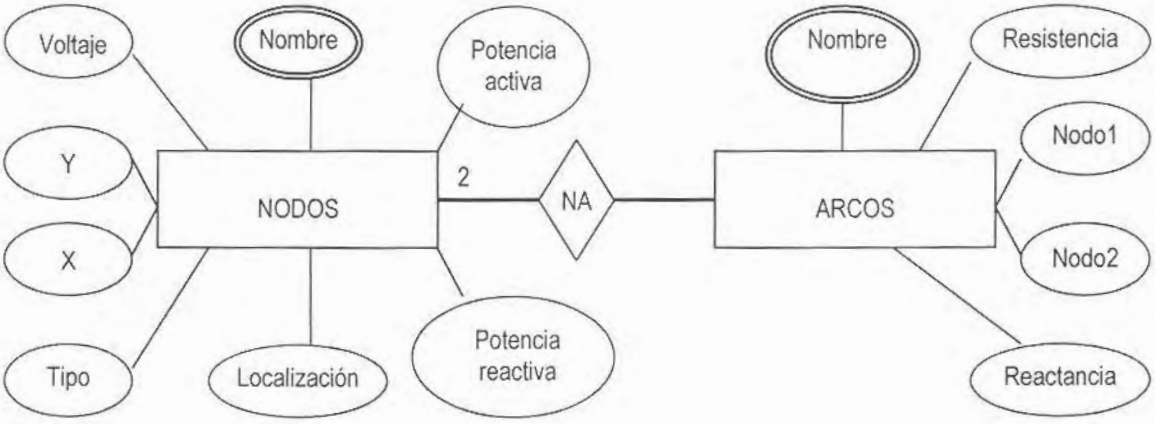


Figura 4.4 Diagrama entidad relación de la base de datos

Definición de claves

La tabla NODOS contiene los siguientes atributos:

Nombre, X, Y, Voltaje, Potencia activa, Potencia reactiva, tipo, Localización

Siendo el atributo *Nombre* la clave primaria.

La tabla ARCOS contiene los siguientes atributos:

Nombre, Nodo1, Nodo2, Resistencia, Reactancia

Siendo el atributo *Nombre* la clave primaria.

4.2.2 NORMALIZACIÓN DE LAS RELACIONES GLOBALES

Como se sigue un diseño en base al estándar SPARC/ANSI [17] definimos a las relaciones NODOS y ARCOS como relaciones globales.

Es importante indicar que aun en el diseño distribuido es necesario observar las reglas de normalización para evitar anomalías como las de eliminación ó actualización. Las relaciones NODO y ARCO cumplen con la cuarta formal normal:

Los atributos de las relaciones NODOS y ARCOS poseen valores simples, además todos los valores de cualquier atributo de la relación son del mismo tipo. El orden de las tuplas no es relevante, ninguna tupla es idéntica a otra y el orden de los atributos no es importante. Por lo tanto las relaciones NODOS y ARCOS cumplen con la **primera formal normal**.

La relación NODOS la clave es *nombre* y todos los demás atributos dependen de la clave. La relación ARCOS la clave es nombre y todos los demás atributos dependen de la clave. Por lo tanto las relaciones NODOS y ARCOS cumplen con la **segunda forma normal**.

Dado que las relaciones NODOS y ARCOS no tienen dependencias transitivas, se dice que tales relaciones cumplen con la **tercera formal normal**.

La relación NODOS contiene dos claves candidatas a) Nombre, b) X, Y y estas claves son los únicos determinantes.

La relación ARCO contiene un solo determinante, el cual a su vez es la única clave. Por lo tanto cumple con la **forma normal de Boyce-Cood**.

Las relaciones NODOS y ARCOS están en BCNF y no tienen dependencia de valores múltiples, por lo tanto cumplen con la **cuarta forma normal**.

4.2.3 FRAGMENTACIÓN

La relación NODOS se fragmenta en forma horizontal primaria (vea **CAPITULO 3**).

Para el sistema todas las consultas que involucren a la relación NODOS necesitan saber la *localización* de los nodos.

Los predicados simples que serán usados para fragmentar la relación NODOS son:

p_1 = Localización = "NORTE

p_2 = Localización = "SUR"

p_3 = Localización = "ESTE"

Donde $P_r = \{ p_1, p_2, p_3 \}$

Tales predicados deben cumplir con las propiedades de ser: completos y mínimos, para verificar que en realidad cumplan con estas propiedades fue necesario aplicar el algoritmo COM_MIN (Ver algoritmo 3.1)

Aplicación del algoritmo COM_MIN a los predicados simples:

Entrada:

NODOS (Nombre, X, Y, Potencia Activa, Potencia Reactiva, Tipo, Localización)

P_r = {Localización = "NORTE", Localización = "SUR", Localización = "ESTE"}

Salida:

P_r' = { }

Para poder aplicar el algoritmo tenemos que crear la variable F la cual va a tener un conjunto de minterminos fragmentados.

Primero se busca un elemento p_i que pertenece a P_r tal que pi particiona a NODOS de acuerdo a la regla definida para el algoritmo 3.1, es necesario eliminar ese elemento de P_r, y agregarlo a F y a P_r.

p_i = Localización = "NORTE"
P_r' = { Localización = "NORTE" }
P_r = { Localización = "SUR", Localización = "ESTE"}
F = { Localización = "NORTE" }

Ahora se busca un elemento p_j que pertenece a P_r tal que p_j particione algún elemento de F de acuerdo a la regla, este elemento es agregado a P_r' y a F, este mismo elemento es eliminado de P_r.

p_i = Localización = "SUR"
P_r' = { Localización = "NORTE", Localización = "SUR" }
P_r = { Localización = "ESTE" }
F = { Localización = "NORTE", Localización = "SUR" }

Como p_j no puede particionar a ningún elemento de F se dice que p_j es relevante.

Se prosigue tomando a otro elemento de P_r, este elemento es agregado a P_r' y a F, este mismo elemento es eliminado de P_r.

p_j = { Localización = "ESTE" }
P_r' = { Localización = "NORTE", Localización = "SUR", Localización = "ESTE" }
P_r = { }
F = { Localización = "NORTE" }

Como de nuevo p_j no puede particionar a ningún elemento de F se dice también que el nuevo p_j es relevante. Se vuelve a tomar a otro elemento de P_r, pero ahora P_r ya no tiene elementos, entonces aquí termina el algoritmo.

Finalmente P_r' queda de la siguiente forma:

P_r' = { Localización = "NORTE", Localización = "SUR", Localización = "ESTE" }

Después de aplicar el algoritmo COM_MIN se pudo verificar que $P_r = \{ p_1, p_2, p_3 \}$ es completo y minimal, $P_r' = P_r$.

Se procede a aplicar el algoritmo PHORIZONTAL (Ver algoritmo 3.1), para determinar los fragmentos necesarios en la relación NODOS.

Inicialmente se determina el conjunto M de predicados minterminos:

M_1 = (Localización = "NORTE") $\wedge$ NOT(Localización = "SUR") $\wedge$ (Localización = "ESTE")

M_2 = (Localización = "NORTE") $\wedge$ NOT(Localización = "SUR") $\wedge$ NOT(Localización = "ESTE")

M_3 = (Localización = "NORTE") $\wedge$ (Localización = "SUR") $\wedge$ (Localización = "ESTE")

M_4 = (Localización = "NORTE") $\wedge$ (Localización = "SUR") $\wedge$ NOT(Localización = "ESTE")

M_5 = NOT(Localización = "NORTE") $\wedge$ NOT(Localización = "SUR") $\wedge$ (Localización = "ESTE")

M_6 = NOT(Localización = "NORTE") $\wedge$ NOT(Localización = "SUR") $\wedge$ NOT(Localización = "ESTE")

M_7 = NOT(Localización = "NORTE") $\wedge$ (Localización = "SUR") $\wedge$ (Localización = "ESTE")

M_8 = NOT(Localización = "NORTE") $\wedge$ (Localización = "SUR") $\wedge$ NOT(Localización = "ESTE")

Donde $M = \{ m_1, m_2, m_3, m_4. m_5, m_6, m_7, m_8 \}$

Se prosigue a determinar el conjunto I de implicaciones

Dado que el atributo localización tan solo puede tomar uno de tres posibles valores, que son: "NORTE", "SUR", "ESTE", por lo que se tienen las siguientes implicaciones:

i_1 = (Localización = "NORTE") $\Rightarrow$ NOT(Localización = "SUR") $\wedge$ NOT(Localización = "ESTE")

i_2 = NOT(Localización = "NORTE") $\Rightarrow$ (Localización = "SUR") $\wedge$ NOT(Localización = "ESTE")

i_3 = NOT(Localización = "NORTE") $\Rightarrow$ NOT(Localización = "SUR") $\wedge$ (Localización = "ESTE")

i_4 = (Localización = "SUR") $\Rightarrow$ NOT(Localización = "NORTE") $\wedge$ NOT(Localización = "ESTE")

i_5 = NOT(Localización = "SUR") $\Rightarrow$ (Localización = "NORTE") $\wedge$ NOT(Localización = "ESTE")

i_6 = NOT(Localización = "SUR") $\Rightarrow$ NOT(Localización = "NORTE") $\wedge$ (Localización = "ESTE")

i_7 = (Localización = "ESTE") $\Rightarrow$ NOT(Localización = "NORTE") $\wedge$ NOT(Localización = "SUR")

i_8 = NOT(Localización = "ESTE") $\Rightarrow$ (Localización = "NORTE") $\wedge$ NOT(Localización = "SUR")

i_9 = NOT(Localización = "ESTE") $\Rightarrow$ NOT(Localización = "NORTE") $\wedge$ (Localización = "SUR")

Finalmente, eliminar minterminos contradictorios a partir de M

De acuerdo a i_1 m_1, m_3 y m_4 son contradictorios
De acuerdo a i_4 m_3, m_4 y m_7 son contradictorios
De acuerdo a i_7 m_1, m_3 y m_7 son contradictorios
De acuerdo a i_2, i_3, i_5, i_6, i_8, i_9 m_6 es contradictorio

Por lo tanto solo queda M = { m_2, m_5, m_8 }entonces se definen tres fragmentos de acuerdo a M.

NODOS1 = Localización == "Norte" NODOS
NODOS2 = Localización == "Sur" NODOS
NODOS3 = Localización == "Este" NODOS

Una vez definidos los fragmentos es necesario probar si son correctos, esto se logra mediante las reglas de completicidad, reconstrucción y disjunción definidas en el capitulo 3.

- Esta fragmentación satisface la condición de completicidad dado que "Norte", "Sur" y "Este" son solamente los únicos valores posible del atributo **Localización**.

- La condición de reconstrucción se logra mediante la unión de todos los fragmentos:

  NODOS = NODOS1 ∪ NODOS2 ∪ NODOS3

  Dado que:

  $$NODOS1 = \sigma_{Localización = "NORTE"} (NODOS)$$
  $$NODOS2 = \sigma_{Localización = "SUR"} (NODOS)$$
  $$NODOS3 = \sigma_{Localización = "ESTE"} (NODOS)$$

  La intersección de los fragmentos es vacía, por lo tanto cumple con esta condición.

- La condición de disjunción se cumple dado que a través del atributo Nombre (clave principal) no puede existir un nodo que tenga diferentes valores en el atributo Localización, además dado que m_2, m_5 y m_8 son mutuamente exclusivos, la fragmentación de NODOS es disjunta.

Elección del lenguaje de programación

Para propósitos de la programación del sistema se opto por usar el lenguaje Visual C++ [18] por las siguientes razones:

- Visual C++ es un lenguaje muy potente para la creación de aplicaciones cliente servidor.
- Visual C++ incluye interfaces que permiten el manejo de bases de datos.

Visual C++ proporciona varias formas de trabajo con bases de datos como podemos ver en la figura 4.5, por ejemplo, utilizando la biblioteca de clases **MFC**, podemos recurrir a las clases **DAO** (Data Access Objects – Objetos de acceso a datos) o a las clases **ODBC** (Open DataBase Connectivity – Conectividad abierta a bases de datos). Otra opción que ofrece **Microsoft** es la de utilizar **OLE DB** como un proveedor de datos y objetos **ADO** (ActiveX Data Objects – objetos ActiveX para acceso a datos), como tecnología de acceso a datos, argumentando que el acceso a datos basado en **OLE DB** y **ADO** es adecuado para una gama de aplicaciones, desde pequeños procesos en estaciones de trabajo a aplicaciones Web a gran escala.

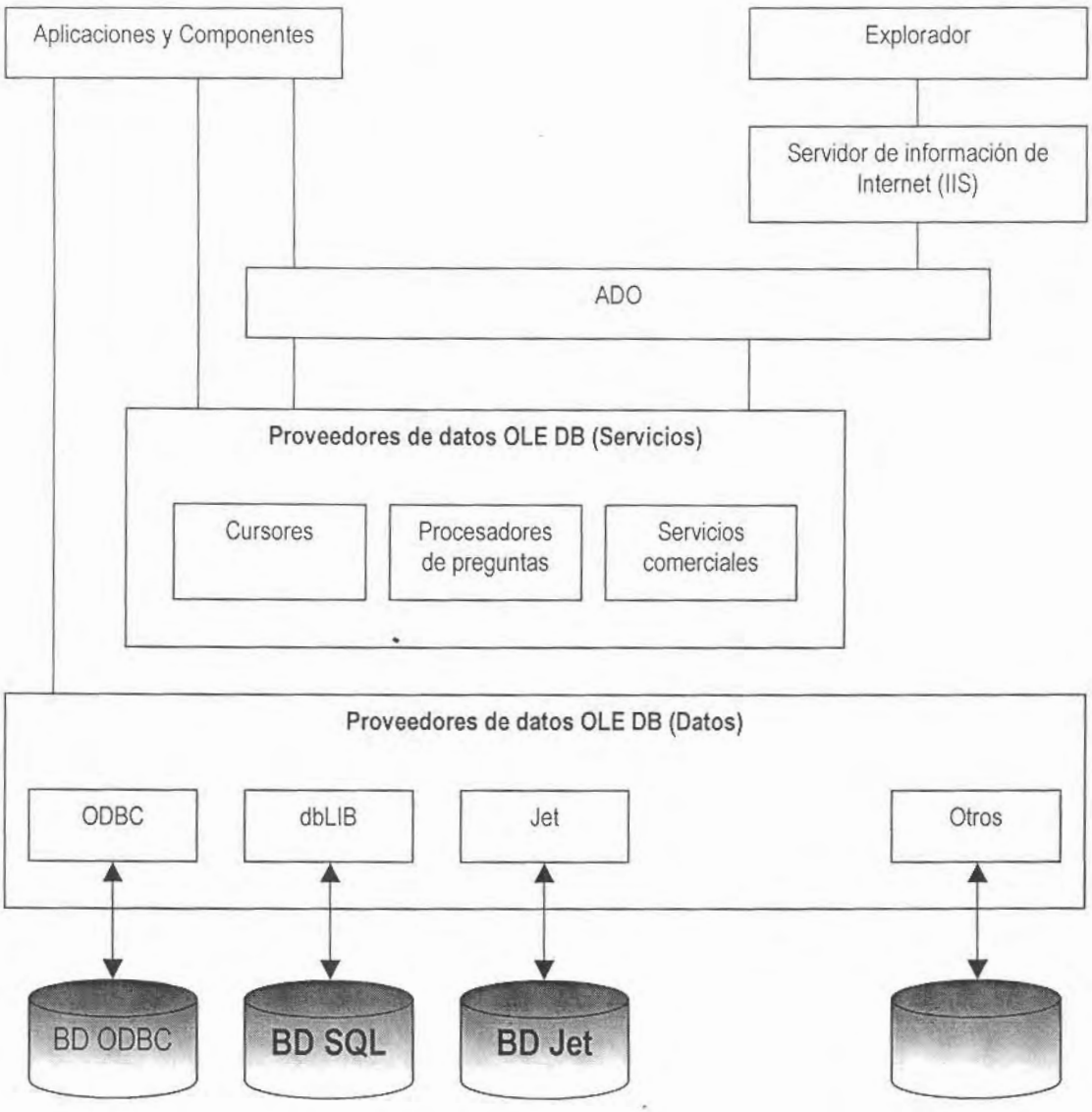


Figura 4.5 Biblioteca de clases MFC [17][1]

El entorno ODBC ayuda a manejar conexiones a diferentes orígenes de datos a través del administrador de controladores, que esta implementado en ODBC32.dll. La aplicación solo necesita vincularse a dicho administrador.

La aplicación se conecta a un origen de datos **ODBC**, en lugar de hacerlo directamente a una base de datos en particular. Los orígenes de datos **ODBC** disponibles para las aplicaciones en la maquina local se configuran con el Administrador de orígenes de datos **ODBC** de 32 bits.

[1] Imagen obtenida de Visual C++ 5.0 para desarrolladores

La ultima adición a la familia Microsoft de interfaces de bases de datos son lo objetos ADO. Debido a que los objetos que se utilizan en ADO son COM, se pueden utilizar fácilmente desde un rango muy amplio de entornos de programación.

Para incorporar la tecnología **ADO** en una aplicación de Visual C++ se debe utilizar la directriz **#import**. Esta directriz permite incorporar información de una biblioteca de tipos, que es convertida en clases C++ que describen las interfaces **COM** correspondientes.

```
#import "C:\ARCHIVOS DE PROGRAMA \ ARCHIVOS COMUNES \ SYSTEM \ ADO \ MSADO15.DLL" no_namespace
rename("EOF", "EndOfFile")
```

Esto se hace con el fin de poder importar la biblioteca de **ADO**.

Debido a que **ADO** es una interfaz **COM**, la aplicación necesita inicializar el entorno **COM** antes de hacer cualquier otra cosa con **ADO**. Esto se hace mediante una llamada a CoInitialize().

```
struct InitOle {
        InitOle()  { ::CoInitialize(NULL); }
        ~InitOle() { ::CoUninitialize();  }
} _init_InitOle_;
```

Una vez inicializado **COM** y **ADO** se puede proseguir a la conexión con la base de datos.

Aunque Microsoft Acces no ejecute SQL dinamico, ADO realiza las consultas a traves de filtros:

```
CString strFiltro;
strFiltro = "Nombre = '" + Nombre_Viejo + "'";
sitio->Filter = _bstr_t(strFiltro);
```

4.3 DISEÑO DE UN SERVIDOR CONCURRENTE

Para el desarrollo de un servidor concurrente usamos sockets de flujo debido a que estos utilizan protocolos basados en conexión, es decir se debe establecer la conexión y después leer los datos de flujo.

Los sockets requieren de dos extremos en una comunicación en la red, estos extremos definen las direcciones que utilizan los procesos. Tan pronto como cada proceso se conecte a un extremo destino diferente , además diferentes servicios pueden utilizar el mismo puerto de protocolo sin ninguna confusión.

El servidor concurrente (ver figura 4.6) crea un proceso para cada solicitud de servicio, es decir, se tiene un servidor maestro que enlaza a él una dirección y esperará por solicitudes de conexión de clientes, guardando el descriptor del socket y la dirección correspondiente en un arreglo y llamando a una función que comenzará un nuevo subproceso de ejecución para atender a cada nueva conexión. Cabe mencionar que el socket servidor solo puede aceptar conexiones nuevas.

Proceso servidor

Cuando llega una solicitud de servicio al socket que monitorea la función accept, esta devuelve al programa servidor que la llamo el descriptor de socket del socket recién creado.

Después el programa servidor crea un nuevo proceso para manejar la solicitud del servicio, este nuevo proceso recibe una copia del nuevo socket y maneja la solicitud de servicio. Entonces el programa servidor continuará escuchando nuevas solicitudes de conexión con la función accept.

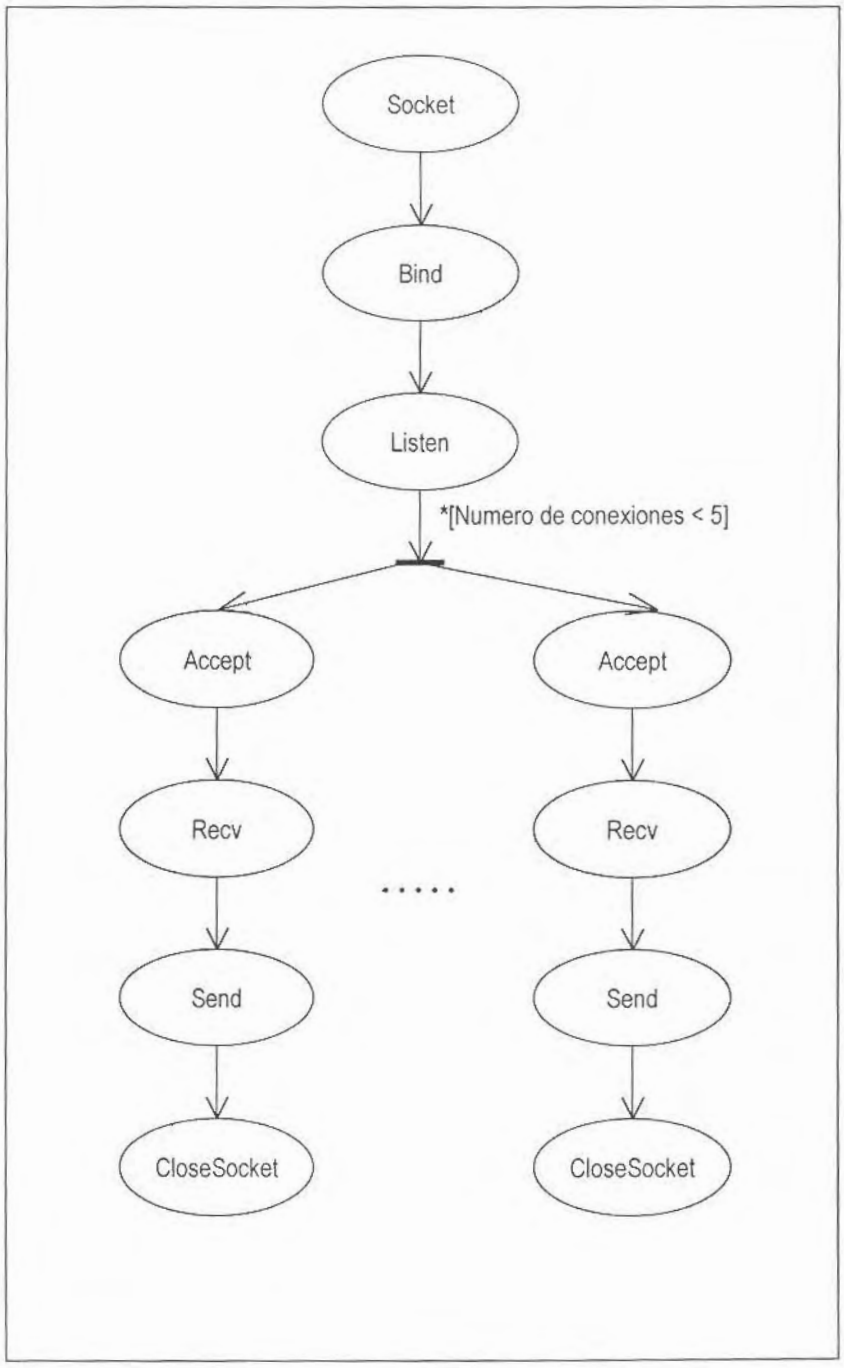


Figura 4.6 Diagrama de actividades de un Servidor Concurrente

La función **socket()** se utiliza para obtener un descriptor de socket

A una llamada a **bind()** se le pasa el descriptor del socket y un apuntador a una estructura de directorios, así como la longitud de dicha estructura. Con esto se le asigna al socket un puerto.

Ya que se creó el socket y fue enlazado con una dirección, se necesita tener una forma de establecer una conexión con un cliente, para hacer esto se utiliza la función **listen()**.

Una vez que el socket esta listo para escuchar las solicitudes de conexión, se aceptara alguna conexión mediante la función **accept()**.

Cuando la función **accept()** es aceptada, se genera un proceso hijo, llamado proceso servidor, este proceso servidor es simplemente un socket que se encargará de atender las peticiones de los clientes.

Por ultimo las funciones **send()** y **recv()** se encargan de mandar y recibir datos respectivamente a través del cliente y el socket servidor.

4.4 INTERFAZ GRAFICA CONCURRENTE

4.4.1 GENERALIDADES

La razón de crear la interfaz gráfica concurrente surge por la naturaleza del problema, este sistema calcula el flujo de potencia en un sistema eléctrico, por lo tanto se necesitaba de una interfaz totalmente gráfica y amigable para que al momento de la introducción de los datos no exista confusion y sea de más rápido acceso.

Debido que para el calculo de flujos de potencia es necesario un diagrama unifilar del sistema de potencia, surge la idea de convertir el diagrama unifilar a un grafo. Esto debido a que el diagrama unifilar consta de buses (Nodo flotante, Nodo generador, Nodo de carga) que pueden ser fácilmente representados por los Nodos de un Grafo, además también consta de líneas de transmisión que conectan a los buses, tales líneas de transmisión pueden ser fácilmente representadas por los Arcos de un grafo, puesto que también unen a los nodos del grafo.

4.4.2 DISEÑO

Para el desarrollo de la interfaz gráfica usamos la programación orientada a objetos debido principalmente a la facilidad de reutilización del código, es necesario el empleo de algún método de análisis y diseño. Se emplea UML (lenguaje unificado de modelado) tomando en cuenta que puede ser usado por una gama amplia de áreas de aplicación, también puede ser usado durante todo el ciclo de vida de desarrollo de software, y por ultimo los procesos de UML son iterativos.

Se modela lo siguiente:

- Casos de uso
- Diagramas de clases
- Diagramas de iteración

4.4.3 MODELADO

Un ingeniero eléctrico debe poder dibujar un diagrama eléctrico, posteriormente realizar el estudio de flujo de potencia. El sistema eléctrico posee tres tipos de nodos y un tipo de línea de transmisión ó arcos, los nodos pueden ser de tres tipos: carga, generación y compensación

El ingeniero puede realizar las siguientes operaciones:

- Insertar Nodo
- Eliminar Nodo
- Modifica Nodo
- Insertar Arco
- Eliminar Arco
- Modifica Arco
- Calculo de Flujos de Potencia

La base de datos distribuida es afectada por las siguientes operaciones:

- Insertar Nodo
- Eliminar Nodo
- Modifica Nodo
- Insertar Arco
- Eliminar Arco
- Modifica Arco

Esto puede observarse en la Figura 4.7

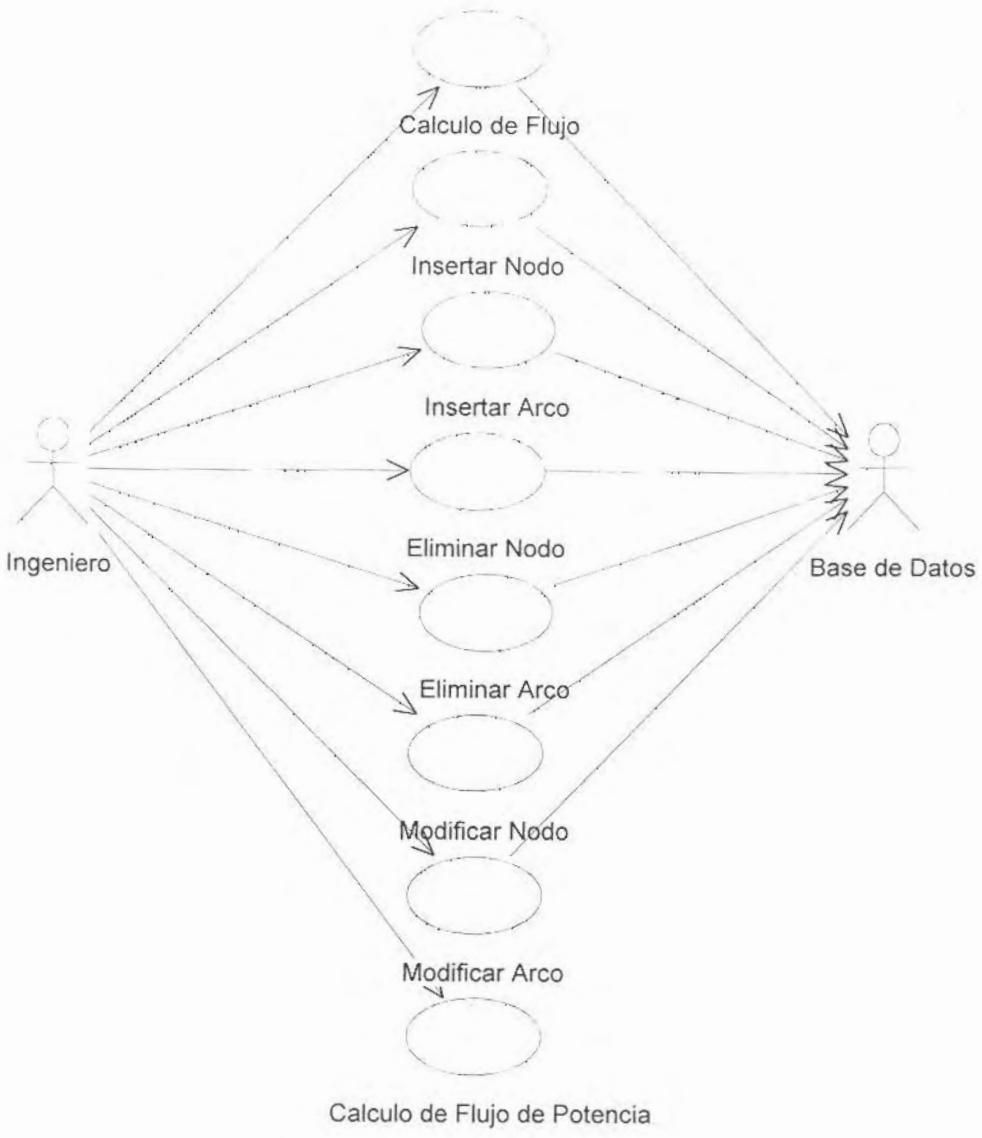


Figura 4.7 Diagrama de casos de usos

Por cada caso de uso se obtiene un diagrama de iteración. Para el caso de uso de Insertar Nodo vea figura 4.8

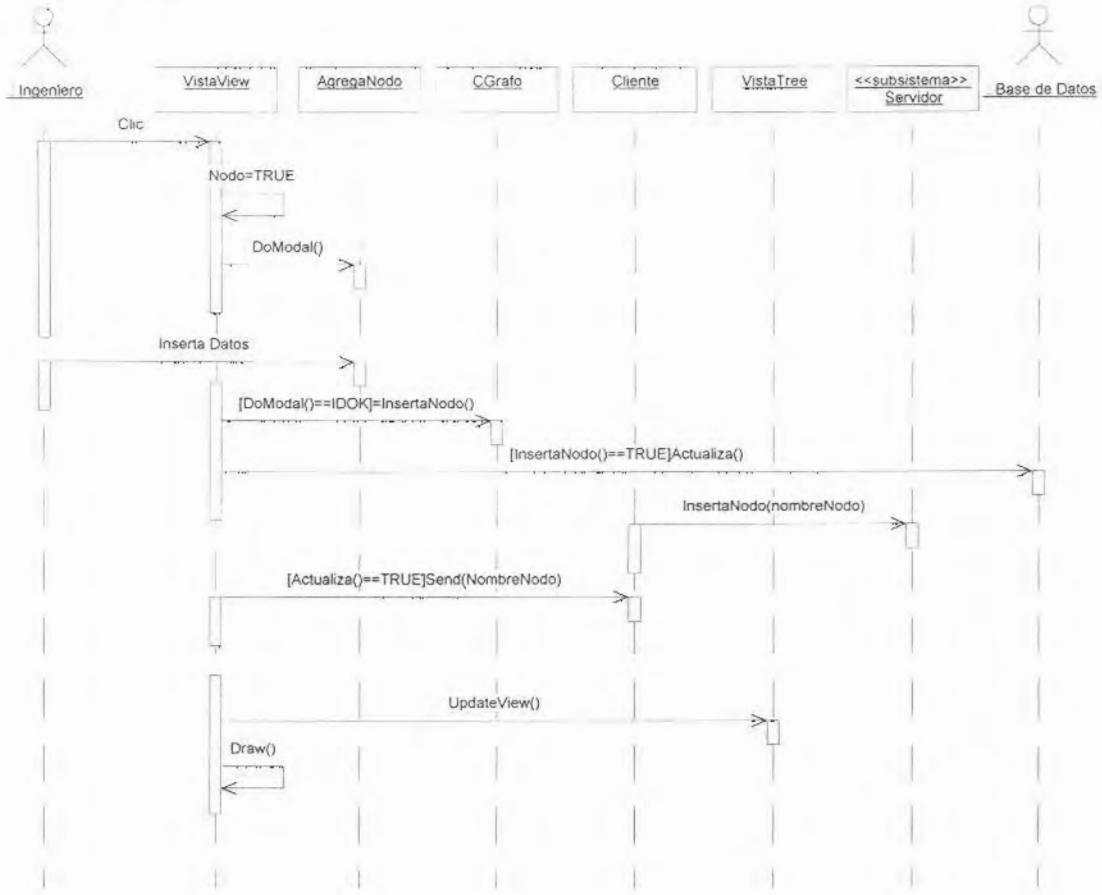


Figura 4.8 Diagrama de iteración del caso de uso Insertar Nodo

El ingeniero selecciona el ITEM NODO y da clic en la vista, la vista pide al ingeniero datos sobre el NODO a través del objeto AgregaNodo, el ingeniero otorga los datos, entonces la vista agrega los datos obtenidos a la base de datos, después la vista manda un mensaje por medio de un objeto cliente al servidor para que actualice las vistas de las demás aplicaciones que estén activas en ese momento, finalmente la vista agrega los datos del nodo al objeto grafo.

Flujos alternos

Si la base de datos falla al momento de tratar de insertar los datos en la base de datos, provoca una excepción que puede ser detectada a través del manejador de excepciones del compilador, impidiendo con esto que el sistema aborte.

Si por alguna razón no se pudieron actualizar las interfaces de las demás aplicaciones activas, entonces no se realiza el commit (comprometerse) a la base de datos. Entonces el NODO no es insertado en la base de datos.

Para el caso de uso Eliminar Arco vea figura 4.9

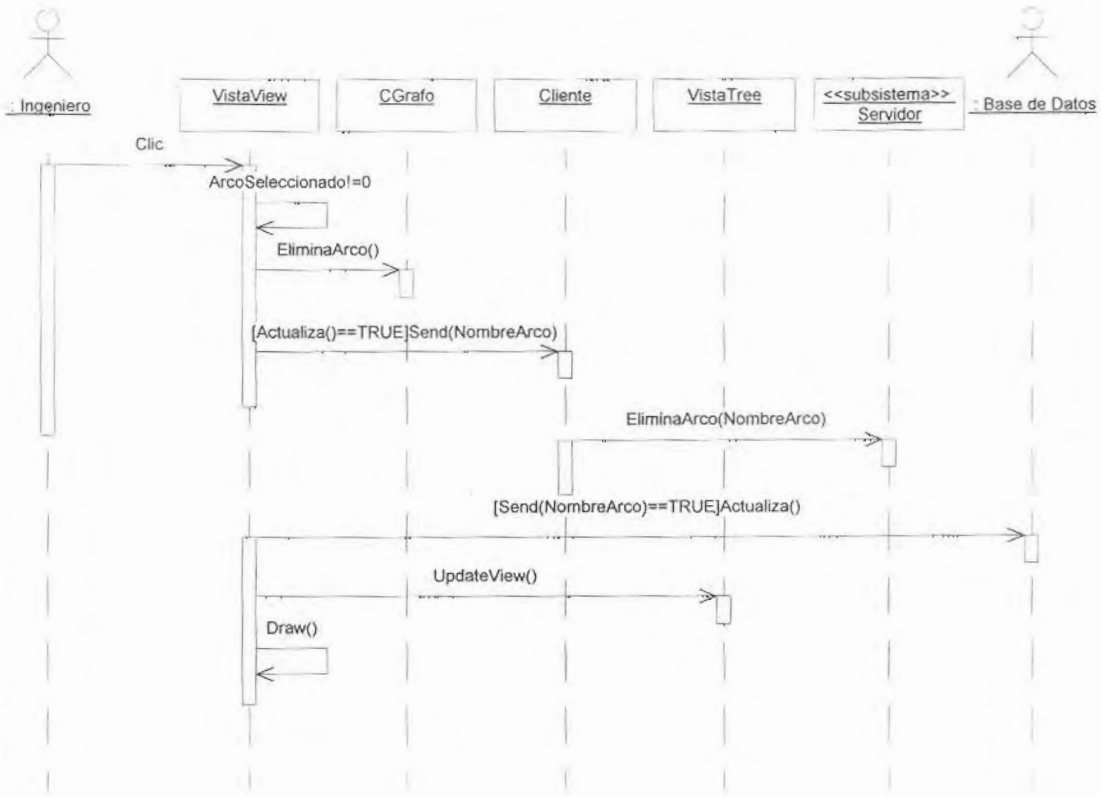


Figura 4.9 Diagrama de iteración del caso de uso Eliminar Arco

El ingeniero selecciona el ARCO a eliminar de la vista, la vista elimina al ARCO del objeto Grafo, después la vista elimina ese ARCO de la base de datos, posteriormente la vista manda un mensaje al servidor a través del objeto llamado cliente para que actualice las vistas de las demás aplicaciones activas en ese momento.

Flujos alternos

Si la base de datos falla al momento de tratar de insertar los datos en la base de datos, provoca una excepción que puede ser detectada a través del manejador de excepciones del compilador, impidiendo con esto que el sistema aborte.

Para el caso de uso Modificar Arco vea figura 4.10

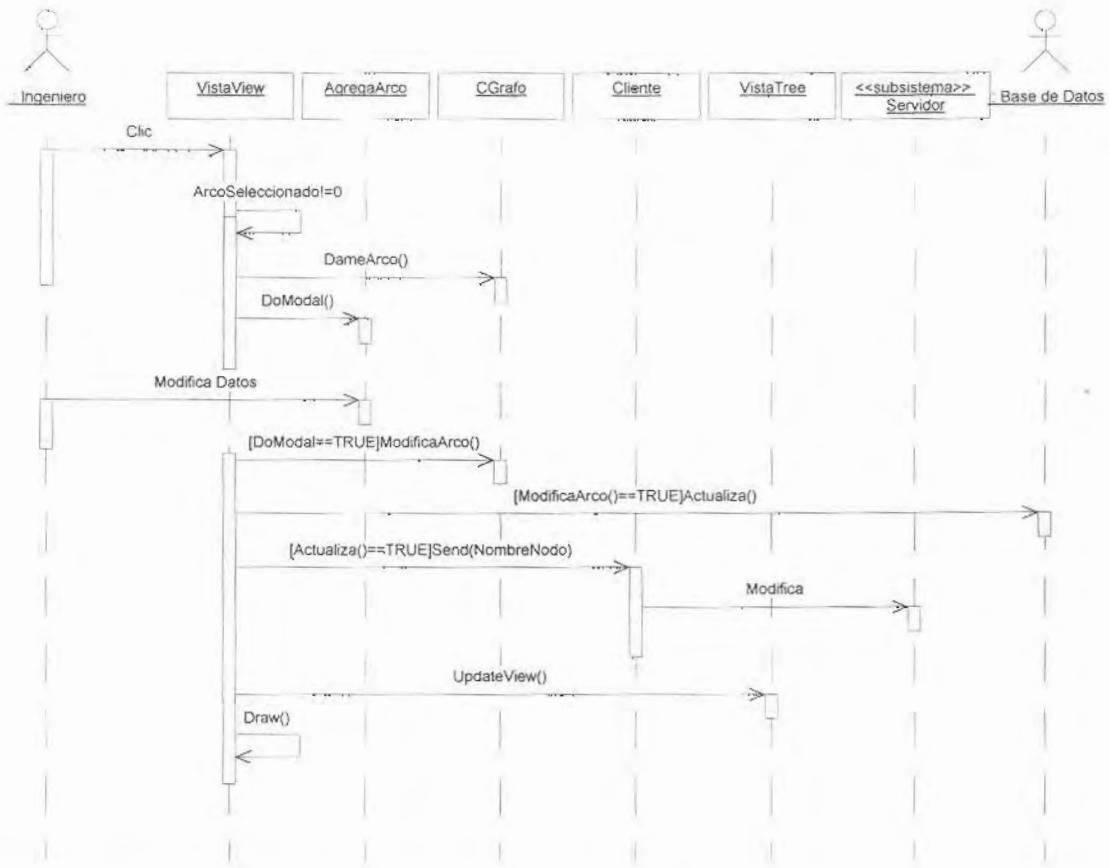


Figura 4.10 Diagrama de iteración del caso de uso Modificar Arco

El ingeniero selecciona el ARCO que desea modificar, la vista pide los nuevos datos del ARCO a través de un objeto llamado AgregaArco, este objeto obtiene los antiguos datos del ARCO a través del objeto CGrafo para que el ingeniero los pueda ver. Una vez que el ingeniero otorga los datos nuevos del ARCO la vista actualiza los datos de la base de datos, finalmente la vista manda un mensaje al servidor a través del objeto cliente para que modifique todas las aplicaciones activas en ese momento.

Flujos alternos

Si la base de datos falla al momento de tratar de insertar los datos en la base de datos, provoca una excepción que puede ser detectada a través del manejador de excepciones del compilador, impidiendo con esto que el sistema aborte.

Si por alguna razón no se pudieron actualizar las interfaces de las demás aplicaciones activas, entonces no se realiza el commit (comprometerse) a la base de datos, entonces el ARCO no es eliminado de la base de datos.

Después de analizar los casos de uso, los diagramas de iteración, se obtiene un conjunto de clases vea Figura 4.11.

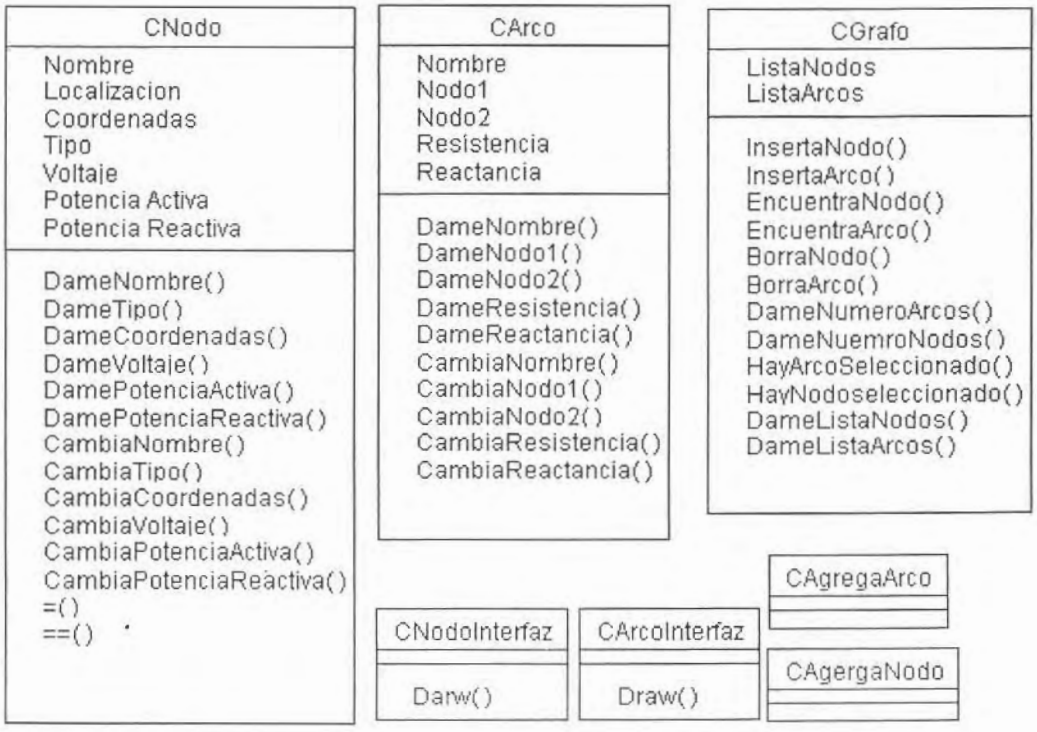


Figura 4.11 Clases principales

Clase CNodo

Esta clase es empleada para almacenar un Nodo del grafo.

Clase CNodo_Interfaz

La clase CNodo_Interfaz es derivada de la clase CNodo, esto se hace con el propósito de no repetir el código empleado en la clase CNodo y mantener la idea principal de la clase CNodo separada de las funciones de pintado.

Clase CArco

la clase CArco es empleada para almacenar un Arco del grafo

Clase CArco_Interfaz

La clase CArco_Interfaz es derivada de la clase CArco, esto se hace con el propósito de no repetir el código empleado en la clase CArco y mantener la idea principal de la clase CArco separada de las funciones de pintado.

Clase CGrafo

La clase CGrafo, es la más importante de todas las clases, ya que esta clase almacena los datos del sistema de potencia eléctrico.

Las clases CAgregaNodo y CAgregaArco

Sirven para presentar un cuadro de dialogo donde se introducirán los datos de los NODOS y ARCOS respectivamente al momento de insertar o modificar un NODO o ARCO

El diagrama de clases muestra la relación que existe entre todas las clases.

El diagrama de la figura 4.12 muestra que la clase Vista es la clase principal del sistema. Esta clase contiene a la clase CGrafo como un atributo.

La clase CGrafo contiene como atributo a las clases ListasNodos y ListaArcos, además contiene a las clases CAddNodo y CAddArco como asociación.

Las clases ListaNodos y ListaArco pueden tener 0 ó varios NODOS y 0 ó varios ARCOS respectivamente.

La clase CNodoInterfaz y CArcoInterfaz son derivadas de las clases CNodo y CArco respectivamente con un solo procedimiento Draw().

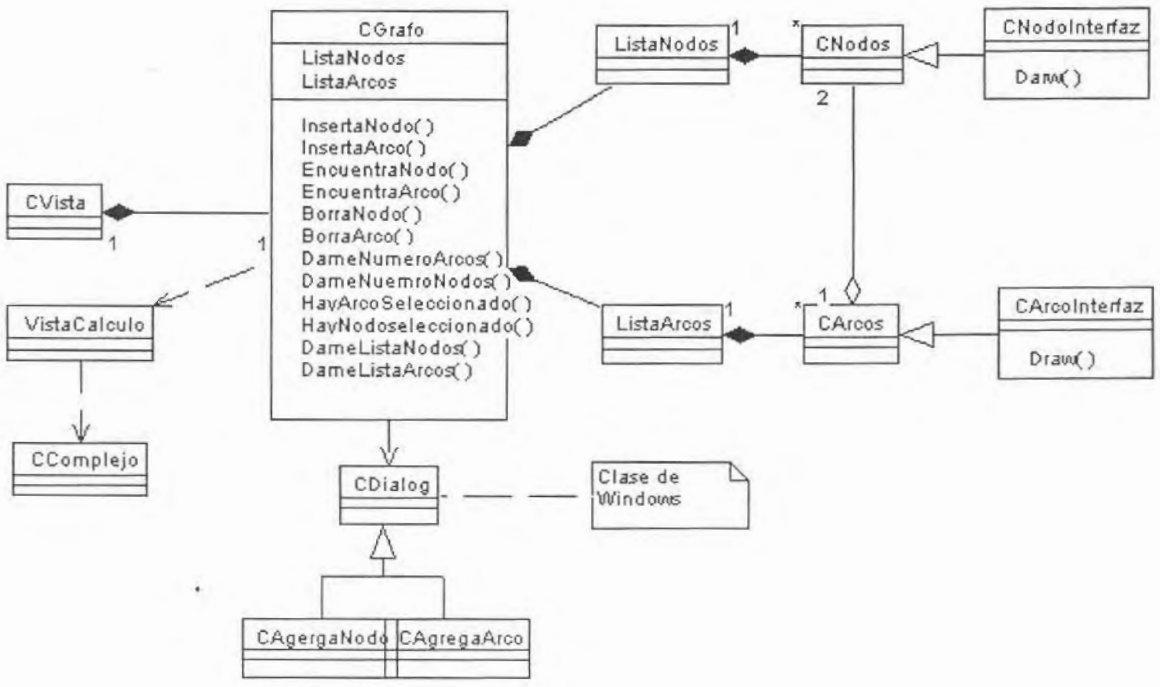


Figura 4.12 Diagrama de clases

4.5 IMPLEMENTACIÓN DE LOS ALGORITMOS PARA EL FLUJO DE POTENCIA

Aplicando paso a paso el método Gauss-Seidel visto en el CAPITULO 2 se realiza el estudio de flujos de potencia vea figura 4.13

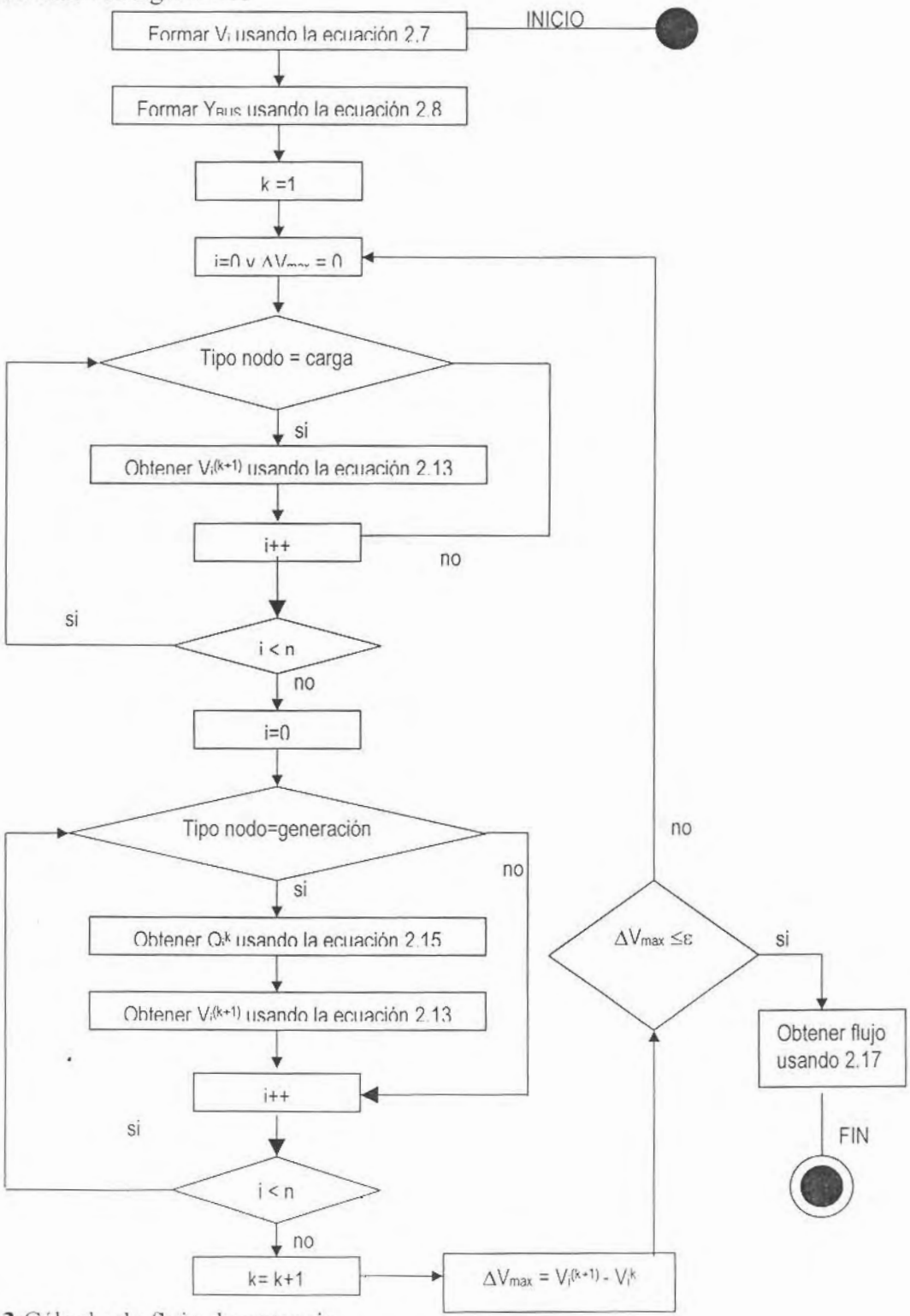


Figura 4.13 Cálculo de flujo de potencia

1. Primero se forma el vector Vi y se calcula la matriz Y_{BUS}
2. Se inicia el contador de iteraciones
3. Se iguala la diferencia del voltaje con 0.
4. Para todos los NODOS tipo carga
5. Se calcula el voltaje de todas las NODOS de carga
6. Si hay más NODOS volver al paso 5
7. Para todos los NODOS tipo generación
8. Se calcula la parte imaginaria de la potencia reactiva
9. Se calculan los voltajes de todas los NODOS de generación
10. Si hay más nodos volver al paso 8
11. Calcular la diferencia de voltajes
12. Si la diferencia de voltaje es no menor o igual que ε volver al paso 3

Para el cálculo de YBUS es necesaria crear una matriz de números complejos y con ayuda de la ecuación 2.8 llenar tal matriz, esto se logra obteniendo los datos de la clase CGrafo, la cual contiene el los NODOS y ARCOS del sistema eléctrico como se muestra en la figura 4.14

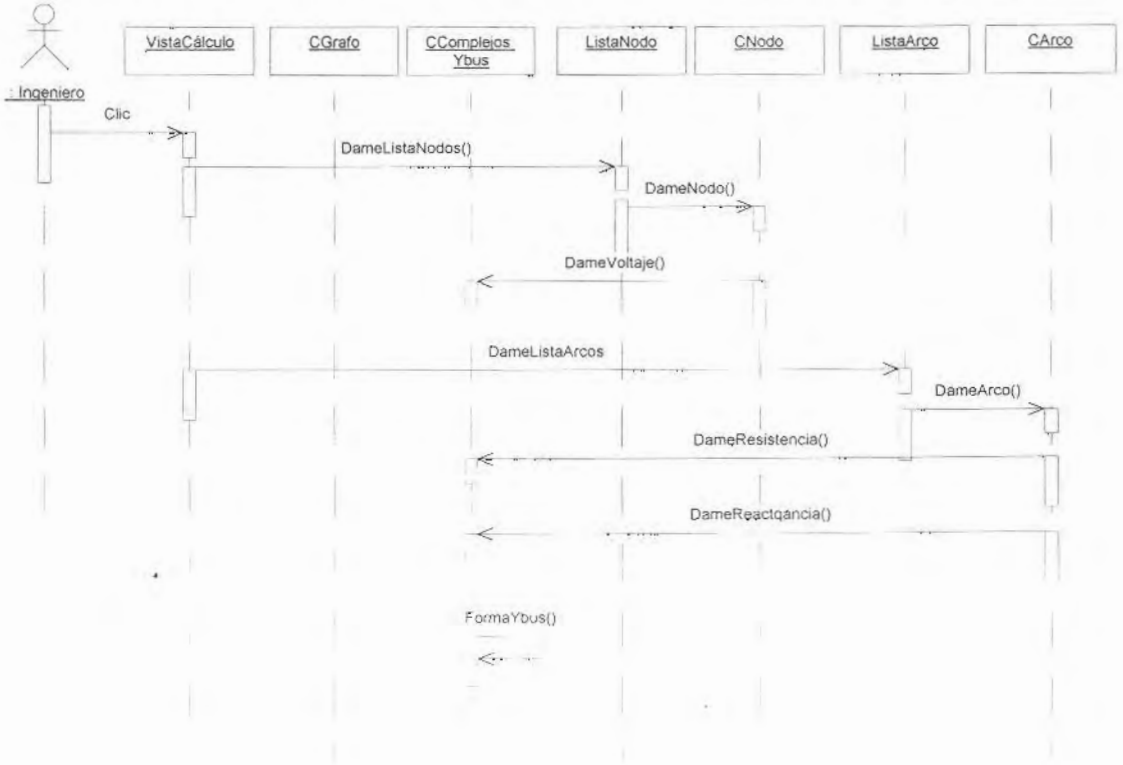


Figura 4.14 Cálculo de Y_{BUS}

CAPITULO 5

APORTACIONES, CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

5.1 APORTACIONES

Después de observar y analizar el estado del arte en que se encuentran los sistemas que realizan estudio de flujos de potencia, queda claro que aunque son sistemas muy robustos y complejos, el estudio de flujos de potencia se realiza solo de manera centralizada, por lo tanto surge la necesidad de desarrollar una aplicación que realice el estudio de flujos de potencia de manera distribuida

Resolver el estudio de flujos de potencia desde un enfoque distribuido usando la misma infraestructura en hardware existente, pero logrando que las aplicaciones estén más cerca de los datos, y un mejor uso de los sistemas de comunicación. Entonces el estudio de flujos de potencia pueda ser realizado desde cualquiera de esos sitios sin necesidad de tener que ir a realizar el estudio de flujos de potencia en cada uno de los sitios.

La aportación con respecto a los procesos concurrentes en este trabajo consiste en actualizar de manera concurrente las interfaces graficas de todas las áreas que se encuentren implicadas en el estudio de flujos de potencia de un sistema eléctrico de potencia; es decir, si en una de las áreas se realiza una modificación, todas las demás áreas serán modificadas inmediatamente, modificando el diagrama eléctrico existente. Aunque los métodos empleados para el control de concurrencia no son originales de esta tesis, si lo es la idea de implementarlos para el estudio de flujos de potencia.

5.2 CONCLUSIONES

El gran crecimiento geográfico de los sistemas de potencia ha contribuido a la interconexión entre sistemas vecinos ó a la división en áreas de sistemas muy grandes. Este crecimiento ha originado por lo tanto un gran desarrollo de aplicaciones computacionales para la ingeniería eléctrica, en particular al desarrollo de aplicaciones para el estudio de flujos de potencia.

De la realización de esta tesis puede extraerse diversas conclusiones importantes:

i. Las bases de datos distribuidas pueden estar dispersas en sistemas operativos heterogéneos y DBMS heterogéneos, logrando con esto una mayor robustez, e integración con los sistemas de computo existentes.

ii. La información del sistema puede ser consultada desde cualquier sitio involucrado, inclusive desde una pagina de Internet.

iii. No se requieren sistemas de computo nuevo para la ejecución de esta aplicación, ya que la aplicación se puede ejecutar en los sistemas de computo existente, por lo que otras tareas pueden ser realizadas en las computadoras donde esta aplicación se instale.

iv. Finalmente, podemos concluir que modelos de computo como son bases de datos, programación concurrente, sistemas distribuidos pueden ser integrados con la identificación de necesidades para solucionar problemas de ingeniería.

5.3 LÍNEAS FUTURAS DE TRABAJO

Con el propósito de mejorar continuamente el desarrollo de sistemas que apoyen a la ingeniería eléctrica, en particular el estudio de flujos es importante enfocar las nuevas aplicaciones que se desarrollen sobre este tópico al WEB.

El sistema obtenido al finalizar esta tesis puede ser migrado a Internet usando nuevas tecnologías como PHP, Java 2, MYSQL, lo que cambiaria seria la interfaz gráfica, puesto que la base de datos seguiría siendo la misma. Inclusive en base a experiencia hemos comprobado que usando un 80% del código del código del servidor puede obtenerse una aplicación que se ejecute en Windows y Linux. Tomando en cuenta que para llevarlo a Internet se tienen que implementar los mecanismos de seguridad correspondientes.

Otra línea futura importante de trabajo es la integración de este proyecto como una unidad en un sistema modular para la administración de los sistemas eléctricos de potencia.

BIBLIOGRAFÍA

[1] CAPE, *Flujo de potencia*, http://ic.net/~eii/cape /pf_sp.htm, ultima fecha de acceso: 9 de julio del 2001

[2] NEPLAN, *NEPLAN - Electricity Overview*, http://www.neplan.ch /eng /elec /elec_intro.htm, ultima fecha de acceso: 9 de julio del 2001

[3] CYME, *Análisis de flujo de potencia*, http://espanol.cyme.com/psafflow.shtml, ultima fecha de acceso: 9 de julio del 2001

[4] SIDAC, *Análisis de flujo de potencia*, http://www.interredes.com.ar/asinelsa /sidacfc.htm, ultima fecha de acceso: 9 de julio del 2001

[5] C.R. PAUL, S.A. NASAR, L. E. UNNEWEHR, *Introduction to electrical engineering*, 1986, McGraw-Hill

[6] OLLE I. ELGERD, *Electric energy systems theory: An introduction*, 1971, McGraw-Hill

[7] THEODORE WILDI, *Electrical power technology*, 1976, John Wiley & Sons

[8] WILLIAM D. STEVENSON, *Análisis de sistemas eléctricos de potencia*, Segunda edición, McGraw-Hill

[9] ENRÍQUEZ HARPER, *Introducción al análisis de los sistemas eléctricos de potencia*, Tercera Edición, Limusa

[10] JOHN J. GRAINGER, *Análisis de sistemas de potencia*, McGraw-Hill

[11] ANDREW S. TANENBAUM, *Distributed operating systems*, 1995, Prentice-Hall

[12] KRIS JAMÁS, KEN COPRE, *Programación en internet*, 1996, McGraw-Hill

[13] M. TAMER ÖZSU, ATRICK VALDURIEZ, *Principles of distributed database systems*, 1991, Prentice-Hall

[14] STEFANO CERI, GIUSEPPC PELAGATTI, *Distributed Databases principles & systems*, International Edition 1985, McGraw-Hill

[15] HENRY F. KORTH, ABRAHAM SILBERSCHATZ, *Fundamentos de bases de datos*, Tercera Edición 1998, McGraw-Hill

[16] ALAN BURNS, GEOFF DAVIES, *Concurrent Programming*, 1994, Addison-Wesley

[17] DAVID M. KROENKE, *Procesamiento de bases de datos*, quinta edición, 1996, Prentice-Hall

[18] BENNET, DAVID, *Visual C++ 5.0 Para desarrolladores*, Prentice-Hall

[19] GRADY BOOCH, JAMES RUMBAUGH, IVAR JACOBSON, *The Unified Modeling Language User Guide*, 1999, Addison-Wesley

ANEXO A

MÉTODOS NUMÉRICOS

A1 GAUSS-SEIDEL

Considérese un sistema de n ecuaciones con n incógnitas.

una vez que se haya resuelto cada x no conocida, se obtienen ecuaciones de la forma:

$x_i = f(x_1\ x_2 \ldots x_i \ldots x_n)$
donde $1 \leq i \leq n$

Se inicia con valores iniciales aproximados conocidos para las x, escritas como $x_1^0, x_2^0, \ldots x_n^0$

la primera aproximación para x_1 es:

$x_1^1 = f_1(x_1^0\ x_2^0 \ldots x_n^0)$
$x_2^1 = f_2(x_1^1\ x_2^0 \ldots x_n^0)$

para x_i^1 tenemos lo siguiente:
$x_i^1 = f_i(x_1^1, x_2^1 \ldots x_{i-1}^1\ x_i^0\ x_{i+1}^0 \ldots x_n^0)$

en general, la k-esima aproximación para x_i se calcula de la siguiente forma:

$x_i^k = f_i(x_1^k\ x_2^k \ldots x_{i-1}^k\ x_i^{k-1}\ x_{i+1}^{k-1} \ldots x_n^{k-1})$

se examina el cambio en cada variable:

$\Delta x_i = x_i^k - x_i^{k-1}$

Cuando $\Delta x_i < E$; para todas las i se dice que existe convergencia.

A partir de la experiencia con determinados problemas, se puede reducir él numero de iteraciones requerido para convergencia. si cambiamos los valores anteriores por algo mas que Δx_i usando

$x_i^k = x_i^{k-1} + \sigma\ \Delta x_i$

donde $\sigma \geq 1$

A2 NEWTON-RAPHSON

Considérese las ecuaciones:

$f(x, y) = 0$
$g(x, y) = 0$

el desarrollo de las funciones f(x, y) y g(x, y) por la serie de Taylor respecto a un punto x^k, y^k produce:

$$f(x, y) = f^k + \frac{1}{1!}\frac{\partial f(x^k, y^k)}{\partial x}(x-x^k) + \frac{1}{1!}\frac{\partial f(x^k, y^k)}{\partial y}(y-y^k) + \ldots$$

$$g(x, y) = g(x^k, y^k) + \frac{1}{1!}\frac{\partial g(x^k, y^k)}{\partial x}(x-x^k) + \frac{1}{1!}\frac{\partial g(x^k, y^k)}{\partial y}(y-y^k) + \ldots$$

Simplificando la notación como sigue:

$f(x^k, y^k) = f^k$
$g(x^k, y^k) = g^k$

$$\frac{\partial f(x^k, y^k)}{\partial x} = f_x^{\,k}$$

$$\frac{\partial f(x^k, y^k)}{\partial y} = f_y^{\,k}$$

$$\frac{\partial g(x^k, y^k)}{\partial x} = g_x^{\,k}$$

$$\frac{\partial g(x^k, y^k)}{\partial y} = g_y^{\,k}$$

$\Delta x = x\text{-}x^{\,k}$
$\Delta y = y\text{-}y^{\,k}$
$\Delta f = f\text{-}f^{\,k}$
$\Delta g = g\text{-}g^{\,k}$

$$\Delta f = f_x^{\,k}\,\Delta x + f_y^{\,k}\,\Delta y$$

$$\Delta g = g_x^{\,k}\,\Delta x + g_y^{\,k}\,\Delta y$$

en forma matricial:

$$\begin{bmatrix} \Delta f \\ \Delta g \end{bmatrix} = \begin{bmatrix} f_x^{\,k} & f_y^{\,k} \\ g_x^{\,k} & g_y^{\,k} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

donde

$$[j^{\,k}] = \begin{bmatrix} f_x^{\,k} & f_y^{\,k} \\ g_x^{\,k} & g_y^{\,k} \end{bmatrix}$$

por lo tanto

$$\begin{bmatrix} \Delta f \\ \Delta g \end{bmatrix} = [j^k] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

despejando tenemos:

$$\begin{bmatrix} \Delta x^k \\ \Delta y^k \end{bmatrix} = [j^k] \begin{bmatrix} \Delta f^k \\ \Delta g^k \end{bmatrix}$$

$$x^{k+1} = x^k + \Delta x^k$$
$$y^{k+1} = y^k + \Delta y^k \quad .$$

ANEXO B

FRAGMENTOS FÍSICOS E INTERFAZ GRAFICA

Fragmento1 : Base de datos

Nodos : Tabla

Nombre	X	Y	Voltaje	Potencia_Activa	Potencia_Reactiva	Tipo	Localizacion
1	100	100	1.05	0	0	0	Norte
2	300	100	1	-0.55	-0.13	1	Norte

Registro: 1 de 2

(a)

Fragmento2 : Base de datos

Nodos : Tabla

Nombre	X	Y	Voltaje	Potencia_Activa	Potencia_Reactiva	Tipo	Localizacion
3	100	300	1	-0.3	-0.18	1	Sur

Registro: 1 de 1

(b)

Fragmento3 : Base de datos

Nodos : Tabla

Nombre	X	Y	Voltaje	Potencia_Activa	Potencia_Reactiva	Tipo	Localizacion
4	300	300	1.1	0.5	0	2	Este

Registro: 1 de 1

(c)

Figura B.1 fragmentos de la base de datos distribuida (a) Localización = "NORTE", Localización = "SUR", Localización = "ESTE"

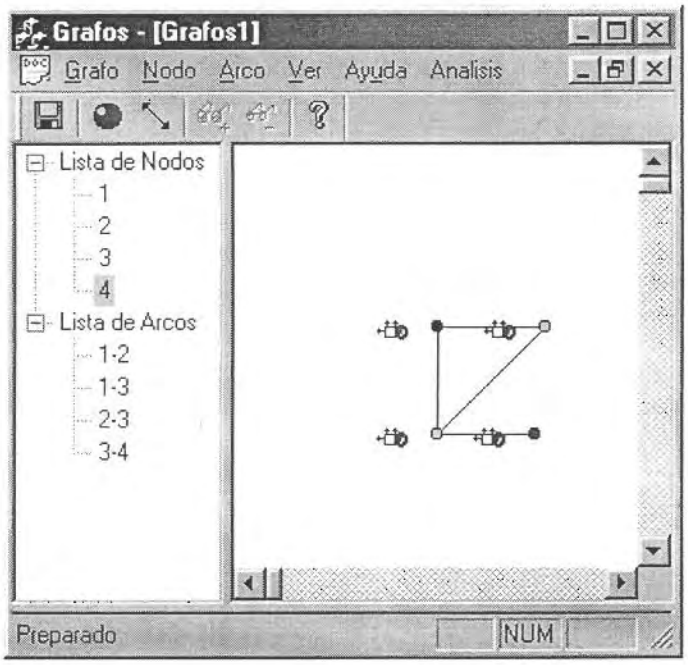


Figura B.2 Representación del sistema eléctrico de potencia como grafo

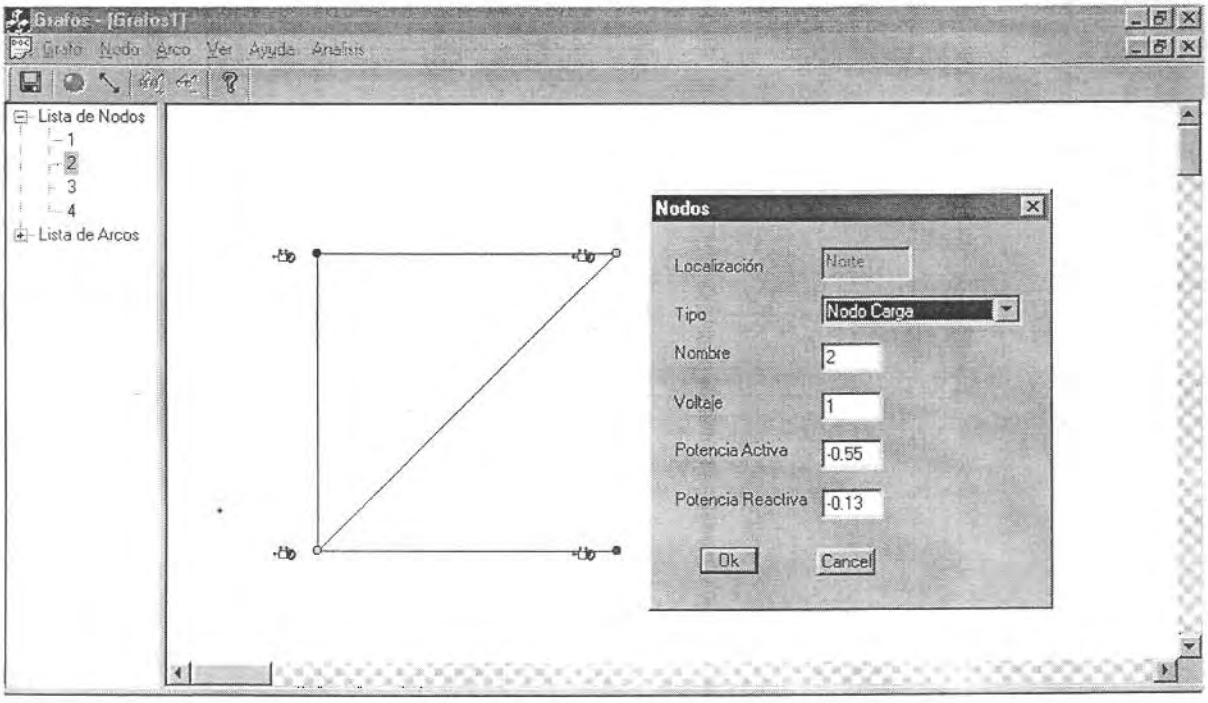


Figura B.3 Modificación de un NODO donde los colores: rojo indica generación, verde indica carga y negro indica compensación.

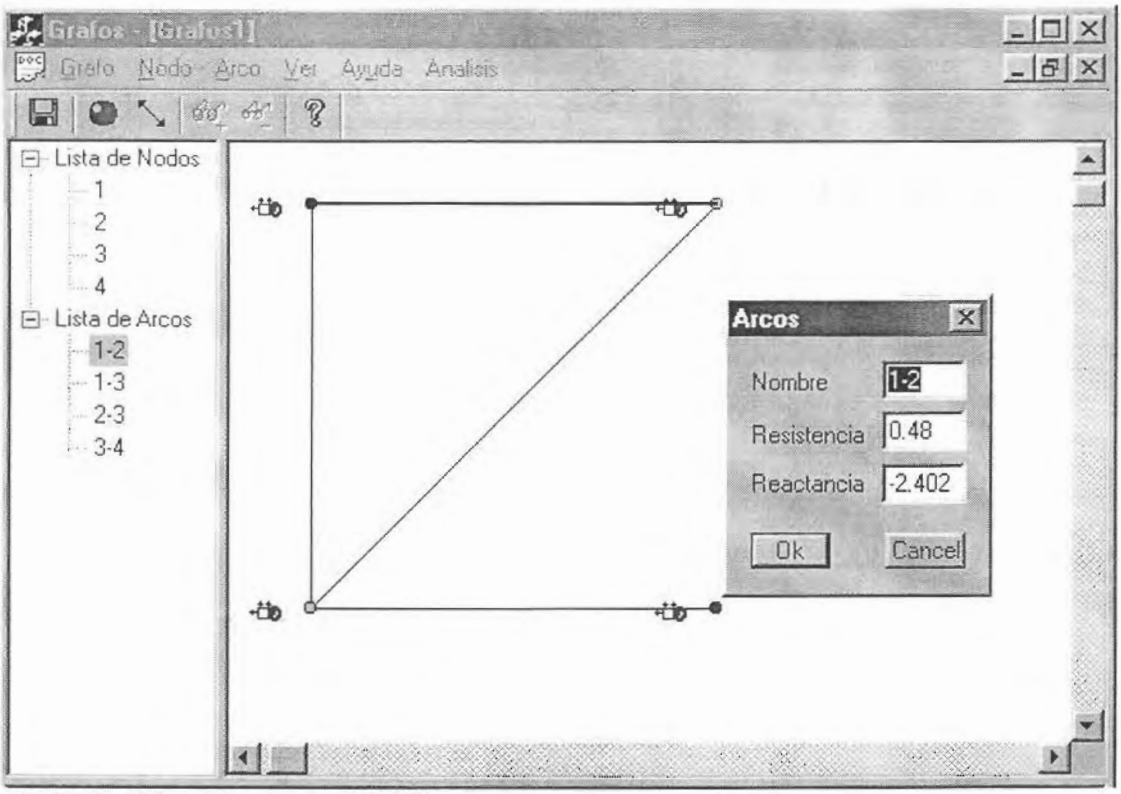


Figura B.4 Modificación de un ARCO

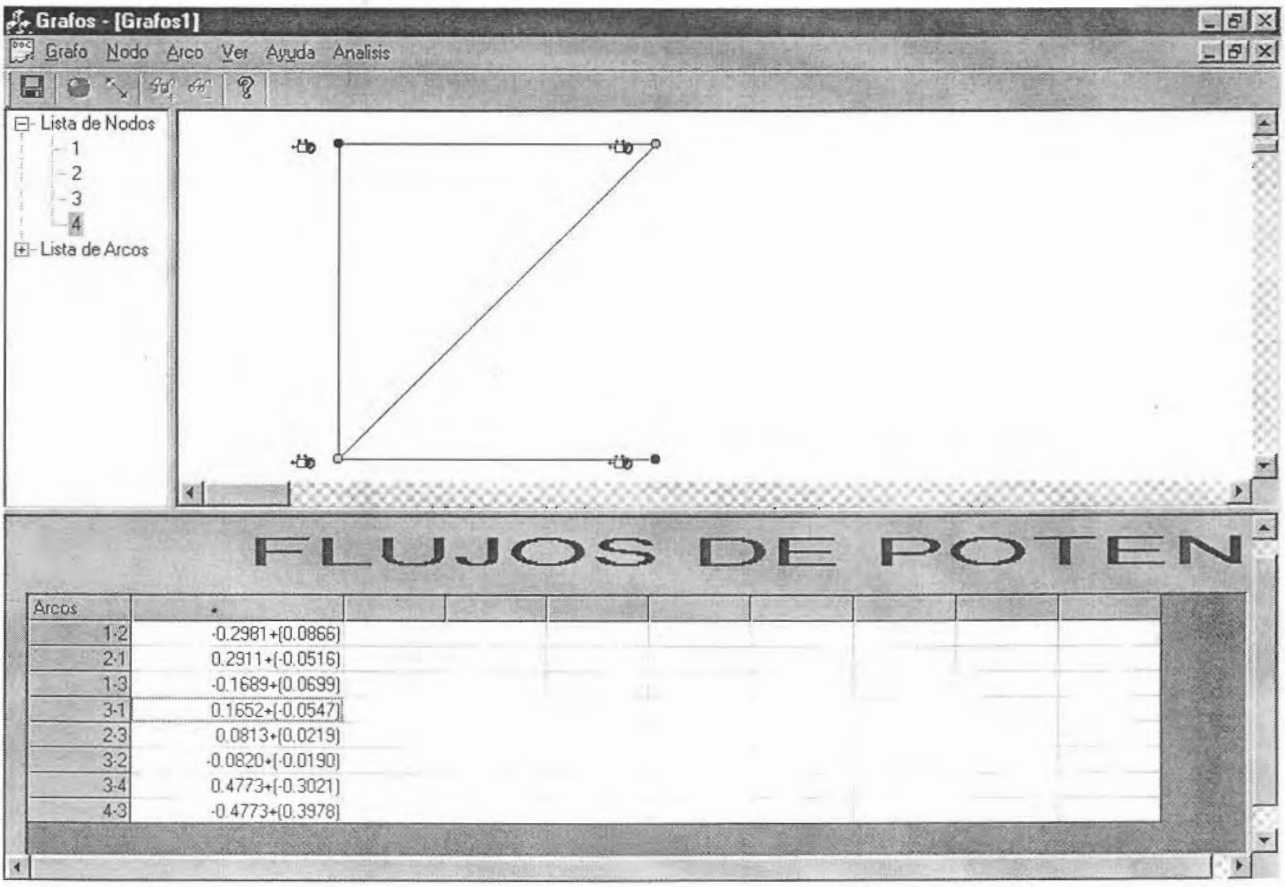


Figura B.5 Resultado del calculo de flujo de potencia

ANEXO C

CÓDIGO FUENTE

C1 SERVIDOR CONCURRENTE

```
UINT arranca(LPVOID parametro)
{
	sServSock = socket(AF_INET,SOCK_STREAM,0);
	addr.sin_family=AF_INET;
	addr.sin_port=htons(5050);
	addr.sin_addr.s_addr=htonl(INADDR_ANY);
	if(bind(sServSock,(LPSOCKADDR)&addr,sizeof(addr))==SOCKET_ERROR)
		MessageBox("Error bind");
	if(listen(sServSock ,2)==SOCKET_ERROR)
		MessageBox("Error listen");
	while(nNumConns< 5 )// se cambia el numero de clientes que se necesitan
	{
		sConns[nNumConns]=accept(sServSock,&ConnAddrs[nNumConns],&nAddrLen);
		if(sConns[nNumConns]==INVALID_SOCKET)
			MessageBox("Error sConns");
		else
			AfxBeginThread(hilo, parametro);
	}
	return 0;
}

UINT hilo(LPVOID parametro)
{
	numero=nNumConns-1;
	int num_id;
	SOCKET sock= sConns[numero];
	while(1)
	{
		.
		.
		// recv(sock,cadena,sizeof(cadena),0);   si es que recibe información
		// send(sConns[numero],cadena,sizeof(cadena),MSG_DONTROUTE);
		// si es que necesita mandar información
	}
	return 0;
}
```

C2 CLASES MAS IMPORTANTES

Clase CSet

```
template<class TYPE, class ARG_TYPE> class CSet : public CObject
{

protected:
        struct CNode {      CNode* pNext, * pPrev;  TYPE data;   };
public:
        CSet(int nBlockSize = 10);
        int GetCount() const;
        BOOL IsEmpty() const;
        void operator = (CSet& rNewSet);
        TYPE& GetHead();
        TYPE& GetTail();
        TYPE RemoveAtHead();
        TYPE RemoveAtTail();
        POSITION Add(ARG_TYPE newElement);
        void AddSet(CSet& rNewSet);
        bool ComparарSet(CSet& rNewSet);
        void RemoveAll();
        POSITION GetHeadPosition() const;
        POSITION GetTailPosition() const;
        TYPE& GetNext(POSITION& rPosition);
        TYPE& GetPrev(POSITION& rPosition);
        TYPE& GetAt(POSITION position);
        void RemoveAt(POSITION position);
        POSITION Find(ARG_TYPE searchValue) const;
        POSITION FindIndex(int nIndex) const;
protected:
        CNode* m_pNodeHead;
        CNode* m_pNodeTail;
        int m_nCount;
        CNode* m_pNodeFree;
        struct CPlex* m_pBlocks;
        int m_nBlockSize;
        CNode* NewNode(CNode*, CNode*);
        void FreeNode(CNode*);
public:
        ~CSet();
        void Serialize(CArchive&);
        void Dump(CDumpContext&) const;
        void AssertValid() const;
};
```

Clase CNodo

```
class CNodo
{
 protected:
            CString Nombre, Localizacion;
            CPoint Coordenadas;
            int Marca;
            double voltaje, pa, pr;
public:
           CNodo();
           CNodo(CString &,CString &,double,double,double,CPoint,int);
           CNodo(const CNodo &);
           virtual ~CNodo();
public:
            void operator = (const CNodo & );
            bool operator == (const CNodo & ) const;
            bool operator <  (const CNodo &) const;
            CNodo &GetDatos();
            bool Valido(CPoint&,int);
            CPoint& Dame_Coordenadas();
            void Cambia_Coordenadas(CPoint &);
            void Cambia_Nombre(CString &Nom);
            void Cambia_Marca(int);
            void Cambia_Voltaje(double);
            void Cambia_pa(double );
            void Cambia_pr(double);
            CString & Dame_Nombre();
            CString & Dame_Localizacion();
            int Dame_Marca();
            bool Dame_Posicion(CString &);
            double Dame_Voltaje();
            double Dame_Pa();
            double Dame_Pr();
};
```

Clase CNodo_Interfaz

```
class CNodo_Interfaz : public CNodo
{
public:
            CNodo_Interfaz();
            CNodo_Interfaz(CNodo &);
            virtual ~CNodo_Interfaz();
public:
            void Draw(CDC *,int);
protected:
            BOOL AddBitmapToImageList(UINT );
            CImageList m_imageList;
};
```

Clase CArco

```
class CArco
{
protected:
        CNodo Nodo1;
        CNodo Nodo2;
        CString Nombre;
        double Resistencia;
        double Reactancia;
public:
        CArco();
        virtual ~CArco();
public:
        CArco(const CNodo &,const CNodo &,CString &,double,double);
        CArco(CArco &);
        bool operator <(const CArco &)const;
        bool operator ==(const CArco &)const;
        void operator =(const CArco &);
        CNodo &GetNodo1();
        CNodo &GetNodo2();
        CArco &GetDatosA();
        CString & Dame_Nombre();
        void Cambia_Nombre(CString);
        void Cambia_Resistencia(double);
        void Cambia_Reactancia(double);
        double Dame_Resistencia();
        double Dame_Reactancia();
        bool Valido(CPoint &);
        bool Dame_Posicion(CString &);
};
```

Clase CArco_Interfaz

```
class CArco_Interfaz : public CArco
{
public:
        CArco_Interfaz(CArco &);
        void Draw(CDC *,int);
        virtual ~CArco_Interfaz();

};
```

Clase CGrafo

```
class CGrafo
{
public:
        CGrafo();
        //Grafo(CClientDC &);
        CGrafo(CSet <CNodo,CNodo &> &,CSet <CArco,CArco &> &,CString &);
        CGrafo(CGrafo &G);
        virtual ~CGrafo();
protected:
        CString Nombre;
        CSet <CNodo,CNodo &> SetNodo;
        CSet <CArco,CArco &> SetArco;
        //CClientDC dc;
public:
        void operator =(CGrafo &);
        bool operator ==(CGrafo &);
        void ChgNombre(CString &);
        CNodo & InsNodo(CNodo &);
        CArco & InsArco(CArco &);
        CNodo &FindNodo(CNodo &);
        CArco &FindArco(CArco &);
        bool DelNodo(CNodo &);
        bool DelArco(CArco &);
        void Imprime(CDC *,int);
        void DelGrafo();
        CString &GetNombre();
        int GetNArcos();
        int GetNNodos();
        bool Valido(CPoint&,int);
        int Nodo_Seleccionado(CPoint & N,int A);
        int Arco_Seleccionado(CPoint &);
        CNodo & FindNodo_Posicion(int i);
        CArco & FindArco_Posicion(int i);
        int Dame_Posicion_Nodo(CString &);
        int Dame_Posicion_Arco(CString &);
        void Suma_Arcos(CSet <CArco,CArco &> &);
        void Suma_Nodos(CSet <CNodo,CNodo &> &);
        CSet <CNodo,CNodo &> & Dame_Lista_Nodos(void);
        CSet <CArco,CArco &> & Dame_Lista_Arcos(void);
};
```

C3 CONEXIÓN A LA BASE DE DATOS

```
_RecordsetPtr Nodos;
CString Con;
CString Text;

Con = _T("Provider=MSDASQL;Persist Security Info=False;Data Source=Base");
Text = _T("select * from Nodos");
Nodos = NULL;
try
{
        Nodos.CreateInstance(__uuidof(Recordset));
        Nodos->CursorLocation = adUseClient;
        Nodos->Open((LPCTSTR)Text,(LPCTSTR)Con,adOpenForwardOnly,adLockOptimistic,adCmdUnknown);
}
catch (_com_error &e)
{
        GenerateError(e.Error(), e.Description());
}
```

C4 CÁLCULO DE FLUJO DE POTENCIA

```
for(i=0;i<numero_lineas;i++)  // RELLENAR YBUS
{
        complejo=CComplejos(Grafos->FindArco_Posicion(i).Dame_Resistencia(),Grafos-
        >FindArco_Posicion(i).Dame_Reactancia());
        nodo1=Grafos->Dame_Posicion_Nodo(Grafos->FindArco_Posicion(i).GetNodo1().Dame_Nombre());
        nodo2=Grafos->Dame_Posicion_Nodo(Grafos->FindArco_Posicion(i).GetNodo2().Dame_Nombre());
        Y[nodo1][nodo2]=(-1)*complejo;
        Y[nodo2][nodo1]=(-1)*complejo;
        Y[nodo1][nodo1]=Y[nodo1][nodo1]+complejo;
        Y[nodo2][nodo2]=Y[nodo2][nodo2]+complejo;
}



for(j=0;j<numero_buses;j++)
{
        if(Grafos->FindNodo_Posicion(j).Dame_Marca()==1)  // NODOS DE CARGA
        {
                voltaje=CComplejos(0,0);
                for(i=0;i<=(j-1);i++)
                {       volt=V[i][1];
                        voltaje=voltaje+volt*Y[j][i];
                }
                for(i=(j+1);i<numero_buses;i++)
                {       volt=V[i][0];
                        voltaje=voltaje+volt*Y[j][i];
                }
                voltaje=CComplejos(Grafos->FindNodo_Posicion(j).Dame_Pa(),-Grafos->
        FindNodo_Posicion(j).Dame_Pr())/V[j][0].conjugado()-voltaje;
                voltaje=voltaje/Y[j][j];
                V[j][1]=voltaje;
        }
}
```

```
for(j=0;j<numero_buses;j++)
{
    if(Grafos->FindNodo_Posicion(j).Dame_Marca()==2)  // NODOS DE COMPENSACIÓN
    {
        voltaje=CComplejos(0,0);
        for(i=0;i<=(j-1);i++)
        {
            volt=V[i][1];
            voltaje=voltaje+volt*Y[j][i];
        }
        for(i=j;i<numero_buses;i++)
        {
            volt=V[i][0];
            voltaje=voltaje+volt*Y[j][i];
        }
        voltaje=voltaje*V[j][0].conjugado();
        Qp=-voltaje.Dame_Imaginaria();
        voltaje=CComplejos(0,0);
        for(i=0;i<=(j-1);i++)
        {
            volt=V[i][1];
            voltaje=voltaje+volt*Y[j][i];
        }
        for(i=(j+1);i<numero_buses;i++)
        {
            volt=V[i][0];
            voltaje=voltaje+volt*Y[j][i];
        }
        voltaje=CComplejos(Grafos->FindNodo_Posicion(j).Dame_Pa(),-Qp)/V[j][0].conjugado()-voltaje;
        voltaje=voltaje/Y[j][j];
        V[j][1]=voltaje;
    }
    V[j][1]=(sqrt(pow(V[j][0].Dame_real(),2)+pow(V[j][0].Dame_Imaginaria(),2))/sqrt(pow(
    V[j][1].Dame_real(),2)+pow(V[j][1].Dame_Imaginaria(),2)))*V[j][1];
}
```

Por ultimo se calculan los flujos de potencia.

```
for(j=1,i=0;i<numero_lineas;i++)
{
        nombre1=Grafos->FindArco_Posicion(i).GetNodo1().Dame_Nombre();
        nombre2=Grafos->FindArco_Posicion(i).GetNodo2().Dame_Nombre();
        p=Grafos->Dame_Posicion_Nodo(nombre1);
        q=Grafos->Dame_Posicion_Nodo(nombre2);
        total=nombre1+"-"+nombre2;
        m_cuadricula.SetTextMatrix(i*2+1,0,LPCTSTR(total));
        potencia=V[p][1].conjugado()*(V[p][1]-V[q][1])*Y[p][q];
        total.Format("%.4f+(%.4f)",potencia.Dame_real(),potencia.Dame_Imaginaria());
        m_cuadricula.SetTextMatrix(i*2+1,1,LPCTSTR(total));
        total=nombre2+"-"+nombre1;
        m_cuadricula.SetTextMatrix(i*2+2,0,LPCTSTR(total));
        potencia=V[q][1].conjugado()*(V[q][1]-V[p][1])*Y[q][p];
        total.Format("%.4f+(%.4f)",potencia.Dame_real(),potencia.Dame_Imaginaria());
        m_cuadricula.SetTextMatrix(i*2+2,1,LPCTSTR(total));
}
```