Universidad Central “Marta Abreu” de Las Villas

Facultad de Matemática, Física y Computación

Sistema de Bases de Datos en Tiempo Real para el Servidor de Topología de la Red Eléctrica del SIGERE

Tesis presentada en opción al título académico de Máster en Ciencia de la Computación

Autor: Lic. Yusmaikel Cabrera Marín

Tutor: Dr. Abel Rodríguez Morffi

Santa Clara

2009

"Año del 50 Aniversario del Triunfo de la Revolución"

PENSAMIENTO

"Sólo es posible avanzar cuando se mira lejos. Solo cabe progresar cuando se piensa en grande."

José Ortega y Gasset

DEDICATORIA

A mi pequeño Allen Manuel, la persona más importante de mi vida.

A mis padres, Yolanda y Manuel, por guiarme por el camino seguro.

A mi hermano Yusnieskel, por su ejemplo.

A mi esposa Ailen por su apoyo y comprensión.

AGRADECIMIENTOS

Agradezco muy especialmente a mi hermano Denis Hernández Pacheco, sin su empuje, apoyo y ayuda la culminación de este trabajo no hubiera sido posible.

A mi tutor Abel, por su guía y apoyo.

A mis compañeros del grupo SIGERE, especialmente a Morgan, Raúl Fernández y Nolisbel Lizano por su colaboración.

A todos los que de alguna forma contribuyeron al desarrollo de este trabajo.

RESUMEN

Durante los últimos años, las investigaciones y el desarrollo de los Sistemas de Tiempo Real y las Bases de Datos en Tiempo Real, se ha incrementado drásticamente, así como también su complejidad, y el volumen de datos que manejan.

Los sistemas de tiempo real controlan un ambiente que tiene restricciones de tiempo bien definidas, es por ello, que se vuelven más complejos y por lo tanto demandan una alta confiabilidad, con resultados correctos, predecibles y a tiempo.

El módulo Servidor de Topología de la Red Eléctrica, contenido en el subsistema de operación del Sistema Integral de Gestión de Redes Eléctricas, tiene como funciones principales facilitar la operación, supervisión y adquisición de datos en tiempo real del estado de la red eléctrica, para modificar el estado de la misma actuando mediante telecomandos o directamente sobre sus interruptores; monitorear la evolución de sus parámetros; y recoger, procesar y almacenar la información recibida.

Una parte esencial del desarrollo de este módulo es la implementación de una Base de Datos en Tiempo Real que refleje el estado de la red eléctrica en cada momento, este informe describe la modelación e implementación de la Base de datos en Tiempo Real del Servidor de Topología de la Red Eléctrica.

TABLA DE CONTENIDOS

INTRODUCCION 1

Antecedentes 1

Objetivo General. 3

Objetivos específicos 3

Organización del informe 4

CAPÍTULO 1. EL SERVIDOR DE TOPOLOGÍA DE LA RED ELÉCTRICA EN EL SIGERE 5

1.1 Visión general del SIGERE 5

1.2 Alcance y composición del SIGERE 5

1.3 Arquitectura del SIGERE 6

1.3.1 Enlaces con SIGE-UNE 6

1.3.2 Programas externos y auxiliares 7

1.3.3 Subsistema de instalaciones 7

1.3.4 Subsistema de operación 7

1.3.5 Subsistema de explotación 8

1.3.6 Subsistema de planificación 9

1.3.7 Subsistema de gestión y control 10

1.4 La necesidad del módulo Servidor de Topología de la Red Eléctrica 10

1.4.1 Propuesta de estructura del STRE 12

1.4.2 Tareas fundamentales del STRE 13

1.4.3 La necesidad de desarrollar una BDTR en el STRE 14

CAPÍTULO 2. LOS SISTEMAS DE BASES DE DATOS EN TIEMPO REAL 16

2.1 Introducción a las BDTR 16

2.1.1. Base de Datos Tradicionales y SBDTR. Consideraciones 17

2.2 Representación de los datos en BDTR. 19

2.3 Transacciones en las BDTR. 20

2.3.1 La necesidad de la predecibilidad 22

2.3.2 Tratando con Plazos Críticos. 23

2.3.3 Tratando con Plazos Suaves 24

2.3.4 Asignación de prioridades 24

2.3.5 Control de concurrencia 25

2.3.5.1 Protocolo de bloqueo en dos fases 26

2.3.5.2 Manejo de DeadLock 27

2.3.5.3 Métodos que evitan el DeadLock 28

2.4 Relación entre las BDTR y las Bases de Datos Activas. Los desencadenadores en las BDTR 29

2.5 Consultas basadas en atributos en las BDTR 31

CAPÍTULO 3. IMPLEMENTACIÓN DE LA BDTR EN EL SERVIDOR DE TOPOLOGÍA DE LA RED ELÉCTRICA 33

3.1 Descripción actual del STRE 33

3.3.1 Características de la tecnología .Net Remoting 38

3.3.1 Ficheros de configuración para establecer comunicación entre el STRE y las aplicaciones clientes. 39

3.2 Modelo de representación de los datos en la DBTR del STRE. 39

3.3 Implementación de las transacciones y del control de la concurrencia 48

3.3.1 Descripción de métodos que se consideran transacciones en la BDTR 48

3.3.2 El manejador de transacciones en el STRE 51
3.3.2 Implementación de bloqueos basándose en 2PL estricto 57
3.4 Implementación de desencadenadores (Triggers) 58
3.5 Implementación de consultas a la BDTR. 60
CONCLUSIONES Y RECOMENDACIONES 64
Conclusiones 64
Recomendaciones 65
BIBLIOGRAFÍA 66
ANEXOS 69
Anexo I Planilla Switching 69
Anexo II Vista gráfica del módulo de control de la red, Switching. 70
Anexo III Vista gráfica del módulo Supervision. 71
Anexo IV Contenido del fichero de configuración del servidor, Server.config 72
Anexo V Contenido del fichero de configuración del cliente, Client.config 73

INTRODUCCION

Muchas aplicaciones del mundo real involucran restricciones de tiempo en el acceso a los datos, que además pueden requerir de validez temporal. Considérese, por ejemplo, sistemas telefónicos, de administración de redes, programas de manejo de inventarios, fábricas para la elaboración automatizada de productos, y sistemas de comando y control. Todas estas áreas de aplicaciones involucran la obtención de datos del entorno, el procesamiento y la generación de nueva información a partir de los datos obtenidos en procesos anteriores, para a partir de ellos proporcionar una respuesta oportuna. Las aplicaciones con estas características introducen la necesidad de utilizar: *1)* los Sistemas de Tiempo Real (STR), que son sistemas informáticos que interaccionan repetidamente con su entorno y responden a los estímulos que reciben del mismo dentro de un plazo de tiempo determinado (Puente, 2003), y *2)* las Bases de Datos en Tiempo Real (BDTR), que son bases de datos de acceso rápido residentes en memoria RAM (Random Access Memory), con soporte para distribución y que proporcionan al STR una vista uniforme de los datos, asegurando su consistencia lógica y temporal, y además permitiendo que se puedan realizar consultas en tiempo de ejecución, lo que es muy útil para los sistemas de administración y monitoreo de los STR.

Antecedentes

Desde sus inicios, las Organizaciones Básicas Eléctricas (OBE) de nuestro país, han tenido como misión fundamental la distribución y comercialización de la energía eléctrica, y como primicia fundamental el tratar de brindar a sus clientes un servicio de calidad, apoyándose para ello en la correcta operación de la red del Sistema Electroenergético Nacional (SEN). La red eléctrica está compuesta por una gran diversidad de elementos eléctricos distribuidos

por todo el territorio que están agrupados en circuitos. Actualmente en el proceso de operación de la red eléctrica es el despachador el encargado de gestionar y regir todas las operaciones que se realizan sobre las diferentes sesiones de la red de su territorio, apoyándose para ello en las brigadas de trabajo, que son las encargadas de la manipulación de sus elementos en el terreno.

Cada vez que se requiere realizar alguna operación en la red, el despachador, de forma manual, siguiendo los procedimientos establecidos, tiene que dejar plasmadas todas las acciones acometidas en una planilla, que recibe el nombre de Switching, y que constituye la única constancia de las tareas realizadas, y por tanto del estado de la red en cada instante. Cuando varios despachadores trabajan de forma concurrente sobre la red el proceso enfrenta los inconvenientes asociados a errores humanos, inconsistencia de la información y redundancia en las operaciones, lo que deriva en una ineficiente gestión de los recursos disponibles, todo ello debido a que se dificulta mucho tener una vista completa del estado en tiempo real de red; y conocer el estado de los elementos de la red eléctrica en cada momento es lo que permite su correcta operación.

El Sistema Integral de Gestión de Redes Eléctricas (SIGERE) forma parte del Sistema de Gestión Empresarial de la Unión Nacional Eléctrica (SIGE-UNE), está integrado por todos los equipos, instalaciones, infraestructura y acciones que forman la red de Transmisión y Distribución de electricidad de nuestro país y recoge los datos técnicos, económicos y de gestión que facilitan la dirección, operación, explotación y planificación de las redes.

El módulo Servidor de Topología de la Red Eléctrica (STRE), contenido en el subsistema de operación del SIGERE, tiene como funciones principales la operación, supervisión y adquisición de datos en tiempo real del estado de la red eléctrica, para modificar el estado de la misma actuando mediante telecomandos o directamente sobre sus interruptores; monitorear la evolución de sus parámetros; y recoger, procesar y almacenar la información recibida.

El funcionamiento del STRE y la introducción en el SEN de automáticas conectadas a los diferentes componentes de la red, así como interruptores inteligentes, deben permitir que los despachos cuenten con un servicio que garantice el conocimiento del estado en tiempo real de la red eléctrica de su territorio y la constante actualización de la misma basada en la

operación de los despachadores sobre un módulo informatizado concebido para este propósito, denominado Módulo de Operación de la Red, Switching, donde se plasman sus operaciones, que se ejecutan sobre la configuración de la red que está en el servidor, así como obtener información de los componentes eléctricos que poseen automáticas en el campo, lo que reduciría sustancialmente el tiempo en la operación de la red, así como los posibles conflictos que se pudieran originar en esta, además de poder telecomandar los interruptores principales, dejando atrás la operación manual de los mismos por intermedio de brigadas de trabajo.

En la actualidad se requiere de una BDTR como el componente más importante del STRE, donde se distribuyan en tiempo real los valores de las variables de la red eléctrica que se deben monitorear y/u operar. Por lo anterior se formula el siguiente **problema de investigación**: No existe una BDTR que refleje el estado de la red eléctrica en cada momento.

Dado el problema anterior se define la siguiente **hipótesis de investigación**: Una Base de Datos en Tiempo Real (BDTR) que se integre al STRE del subsistema de operación del SIGERE permitirá abordar el proceso de operación en tiempo real de los datos de la red eléctrica soportando la manipulación de altas cargas de trabajo.

Objetivo General.

Modelar e implementar una BDTR en el STRE del subsistema de operación del SIGERE, usando la teoría de las BDTR y la tecnología de distribución de aplicaciones .Net Remoting, para crear y publicar en tiempo real la topología de la red eléctrica y así conocer su estado, proporcionando a los despachos del país una herramienta para la operación eficiente de la red.

Objetivos específicos

- Modelar la BDTR del STRE del subsistema de operación del SIGERE.
- Describir las técnicas para el control de la concurrencia de las transacciones en la BDTR del STRE.

- Implementar la BDTR, usando la tecnología .Net Remoting para distribuir los objetos de la misma y facilitar su interacción con los módulos clientes.

Organización del informe

Este documento consta de los capítulos siguientes:

- Capítulo 1, “El Servidor de Topología de la Red Eléctrica en el SIGERE”, donde se brinda una panorámica del SIGERE: su visión, alcance y composición, arquitectura, enlaces con SIGE-UNE, los programas externos y auxiliares, los subsistemas; y del módulo STRE contenido en el Subsistema de Operaciones, la necesidad de este módulo, propuesta de estructura, sus tareas fundamentales, y la necesidad de desarrollar una BDTR en el STRE.

- Capítulo 2, “Los Sistemas de Bases de Datos en Tiempo Real”, se brinda una introducción a las DBTR y sus diferencias con las bases de datos tradicionales, se trata la representación de los datos en las BDTR, las transacciones, la necesidad de predecibilidad, el tratamiento de los plazos críticos, la asignación de prioridades, el control de la concurrencia, la relación entre las BDTR y las bases de datos activas, y las consultas basadas en atributos.
- Capítulo 3, “Implementación de la BDTR en el Servidor de Topología de la Red Eléctrica”, se describe el estado actual del STRE y se informa acerca de las particularidades de los componentes en su entorno, se caracteriza la tecnología .Net Remoting usada en la implementación del STRE, se muestran los ficheros de configuración usados para lograr las comunicaciones entre el STRE y las aplicaciones clientes que se conectan con este, se detalla el modelo usado para la representación de los datos en la BDTR, se define y ejemplifica como se implementaron las transacciones y el control de la concurrencia entre estas, se muestra como se implementaron los desencadenadores y el mecanismo de consultas en la BDTR.

CAPÍTULO 1. EL SERVIDOR DE TOPOLOGÍA DE LA RED ELÉCTRICA EN EL SIGERE

En este capítulo se brinda una panorámica del Sistema Integral de Gestión de Redes Eléctricas (SIGERE), y del módulo Servidor de Topología de la Red Eléctrica contenido en el Subsistema de Operaciones.

1.1 Visión general del SIGERE

El SIGERE forma parte del SIGE-UNE y está integrado por todos los equipos, instalaciones, infraestructura y acciones que forman la red de Transmisión y Distribución de electricidad de nuestro país. El sistema recoge datos técnicos, económicos y de gestión que facilitan la dirección, operación, explotación y planificación de las redes, y está orientado al cliente permitiendo la reducción de costos operativos y mejorando la calidad del suministro que se les brinda. (Fernández, 2007)

1.2 Alcance y composición del SIGERE

El SIGERE incluye las instalaciones desde la salida de las subestaciones de las centrales generadoras hasta las instalaciones de medición del cliente, donde limita con los Sistemas Informáticos de Generación y Comercial.

El SIGERE consta de dos modelos de redes que componen el SEN, *a)* uno de distribución, que cuyo esquema es radial y mallado, y que comprende básicamente las instalaciones concebidas dentro de la Red de Distribución de Energía y el otro *b)* que comprende todas las instalaciones y accesorios de la red de Transmisión, cuya representación es en malla, o sea, pudiendo existir lazos en la interconexión de los elementos que conforman la Red.

1.3 Arquitectura del SIGERE

Para un mejor control del desarrollo e implementación del SIGERE, el mismo se divide en subsistemas y módulos. Una división conveniente de estos subsistemas y módulos es hacerlos corresponder con los puestos de trabajo de las estructuras administrativas o entidades reales. En la figura 1.3.1 se muestra el organigrama de la estructura por niveles del SIGERE.

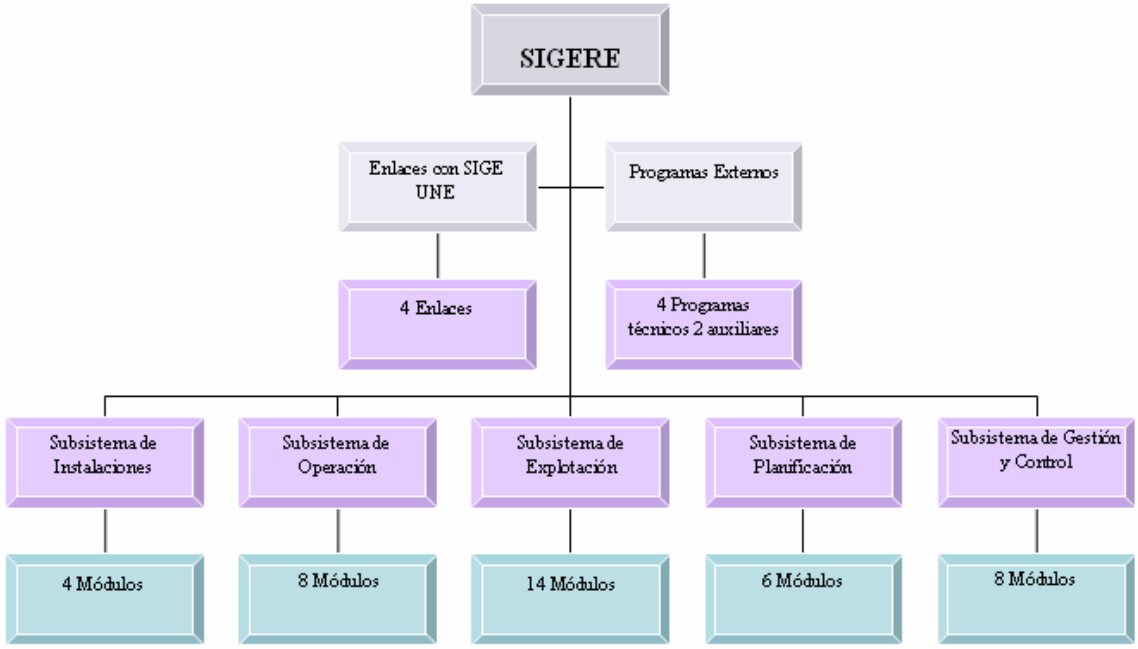


Figura 1.3.1 Organigrama del SIGERE.

1.3.1 Enlaces con SIGE-UNE

El enlace SIGE-UNE incluye los módulos que sirven de punto de enlace entre la información contenida en el SIGERE y el resto de los sistemas desarrollados, entre los que pueden mencionarse:

- Sistema de Gestión Comercial (SIGECO)
- Sistema de Contabilidad (SISCONT)
- Sistema de Recursos Humanos (SIGRH)
- Sistema de Mantenimiento y Explotación de Centrales

1.3.2 Programas externos y auxiliares

En esta categoría se encuentran los programas desarrollados por terceras entidades y dos módulos auxiliares desarrollados por la Unidad Empresarial de Base Automática y Tecnologías de la Información (UEB ATI) Sancti Spíritus.

- Paquete Radial (Desarrollado en el Centro de Estudios Electroenergéticos de la Universidad Central de las Villas, UCLV)
- PSX (Desarrollado por la Facultad de Eléctrica de la UCLV)
- Paquete ESEC (Desarrollado por la empresa CIPEL)
- Programa GENERAL (Desarrollado por la UEB ATI Sancti Spíritus)
- Componente Callejero (Desarrollado por la UEB ATI Sancti Spíritus)
- Asistente de Subestaciones (Desarrollado por la UEB ATI Sancti Spíritus)

1.3.3 Subsistema de instalaciones

Este subsistema constituye el núcleo de todo el SIGERE, en el se encuentran todos los equipos importantes del SIGERE, que se identifican de acuerdo a sus especificaciones (fabricante, país, modelo, número de serie, etc.). Se recogen todos los datos nominales de cada uno de ellos de acuerdo a las normas internacionales existentes.

Este subsistema además se encarga de definir los nomencladores básicos de todo el Sistema y la ubicación de las instalaciones conjuntamente con su posición eléctrica y geográfica; el mismo se compone de los siguientes módulos:

- Nomencladores Básicos
- Ubicación de Instalaciones
- Esquemas Monolineales
- SIG-OBE (Sistema de Información Geográfica para la OBE)

1.3.4 Subsistema de operación

En este Subsistema se incluye todo el tratamiento Informático de las actividades de los Despachos Provinciales, Despachos de Distribución, Centros de Quejas y Operadores de

Subestaciones en la operación de las redes de distribución. Internacionalmente este subsistema está altamente automatizado debido al uso de Sistemas SCADA (*Supervisory Control And Data Adquisition*), y de Sistemas de Aviso Telefónico Integrados. El subsistema de operación está compuesto por los siguientes módulos:

- Gestión de Incidencias
- Atención al Cliente
- Control de Defectos
- Lecturas
- Control de la Red (Switching)
- Control de Vías Libres
- Gestión de la Calidad
- Informes y Reportes Operativos
- Servidor de Topología de la Red Eléctrica.

1.3.5 Subsistema de explotación

Este subsistema constituye la interfaz entre el Subsistema de Instalaciones y el personal técnico que lo explota en las OBE Territoriales y en los diferentes departamentos de la OBE Provincial, y proporciona el acceso regulado, de acuerdo al puesto de trabajo, a la actualización de las diferentes instalaciones a través de las acciones (levantamientos, nuevas instalaciones, mantenimientos, celajes, etc.) que se realizan según los procedimientos y registros establecidos por el Manual de Distribución. Este subsistema adquiere especial importancia al establecerse la Política de Mantenimiento que establece la necesidad de hacer el mantenimiento por diagnóstico; el mismo se compone de los siguientes módulos:

- Subestaciones de Distribución
- Líneas

- Subestaciones de Transmisión
- Líneas de Transmisión
- Desconectivos
- Protecciones
- Capacitares
- Transformadores de Distribución
- Generadores en Distribución
- Alumbrado Público
- Circuitos
- Apoyos
- Servicios
- Mediciones Tecnológicas

1.3.6 Subsistema de planificación

El Subsistema de Planificación comprende las funciones centralizadas relacionadas con los estudios, proyectos y control de las inversiones que se realizan sobre las instalaciones en las OBE Provinciales y Territoriales. El mismo abarca los siguientes módulos:

- Estudios
- Estudios de Desarrollo
- Proyectos
- Proyectos de la Transmisión
- Control de Inversiones
- Control de Inversiones Mayores

1.3.7 Subsistema de gestión y control

Agrupa todos los módulos que se relacionan con el intercambio y diseminación de la información, el control de los planes, acciones e indicadores básicos que permiten a los técnicos y gerentes tomar decisiones sobre la explotación, operación y planificación de las redes. Se compone de:

- Intercambios
- Información Gerencial
- Control de Pérdidas
- Programación y Control
- Auditorías
- Órdenes de Trabajo
- Accesos y Seguridad
- Información del SIGERE

1.4 La necesidad del módulo Servidor de Topología de la Red Eléctrica

El módulo Servidor de Topología de la Red Eléctrica (STRE), contenido en el subsistema de operación del SIGERE, tiene como funciones principales la operación, supervisión y adquisición de datos en tiempo real del estado de la red eléctrica, para modificar el estado de la misma actuando mediante telecomandos o directamente sobre sus interruptores; monitorear la evolución de sus parámetros; y recoger, procesar y almacenar la información recibida.

Actualmente en el proceso de operación de la red eléctrica el despachador es el encargado de gestionar y regir las operaciones que se realizan en su territorio. Para ello, cada vez que se requiere realizar alguna operación en la red, el despachador, de forma manual, siguiendo los procedimientos establecidos (Pérez, 2003), (Días, 2003), (Casalvilla, 2003), (Rodríguez, 2003), (Espinosa, 2003), tiene que dejar plasmadas todas las acciones acometidas en una planilla que recibe el nombre de Switching, Anexo I, y que constituye la única constancia de las tareas realizadas, y por tanto del estado de la red en cada instante.

Cuando varios despachadores operan la red eléctrica de forma concurrente, se corre el riesgo de que las operaciones se solapen en el tiempo o redunden, lo que atenta contra la respuesta en tiempo para su procesamiento, y la gestión adecuada de los recursos materiales y humanos. Por ejemplo, considérese dos despachadores A y B, que operan de forma concurrente sobre un circuito C de la red eléctrica. Si el despachador A ordena que se abra el interruptor X contenido en C, por concepto de sus operaciones a acometer y el despachador B, ordena que se abra inmediatamente X para acometer las suyas, como el despachador B no cuenta con el estado actual de la red eléctrica la operación que ha ordenado realizar ya ha sido acometida por el despachador A, lo que implica una pobre gestión de los recursos disponibles con gastos de horas hombre, de combustible, demora en otras operaciones, inutilización de recursos en el tiempo usado en el recorrido hasta llegar al interruptor, entre otras.

Por otra parte, la dirección del país ha introducido en el SEN, automáticas conectadas a los diferentes componentes de la red, así como interruptores inteligentes los que tienen la característica de poder almacenar y transmitir valores eléctricos, eventos y fallas, por los canales de comunicación y protocolos que se están utilizando actualmente en el SEN, además de que estos interruptores pueden ser telecomandados por los despachadores. Estas bondades actualmente no se utilizan para la gestión de la red en los despachos, solo en las subestaciones eléctricas donde un operador informa al despacho de la ocurrencia de algún evento o falla. Sin embargo aún no se cuenta en la mayoría de los territorios con estas tecnologías porque su costo en el mercado internacional asociado a los programas SCADAS que controlan, supervisan y monitorean estos sensores oscilan por los millares de dólares.

El funcionamiento del STRE debe permitir que los despachos cuenten con un servicio que garantice el estado en tiempo real de la red eléctrica de su territorio y la constante actualización de la misma basada en la operación de los despachadores sobre un módulo informatizado concebido para este propósito, denominado Módulo de Operación de la Red Switching, donde se plasman sus operaciones, que se ejecutan sobre la configuración de la red que está en el servidor, así como obtener información de los componentes eléctricos que poseen automáticas en el campo, lo que reduciría sustancialmente el tiempo en la operación de la red, así como las posibles colisiones que se pudieran originar en esta,

además de poder telecomandar los interruptores principales, dejando atrás la operación manual de los mismos por intermedio de brigadas de trabajo.

1.4.1 Propuesta de estructura del STRE

Como se ha explicado anteriormente con la implementación del STRE se pretende interactuar con varios elementos de diferentes características. En la siguiente figura se muestra la estructura general del STRE.

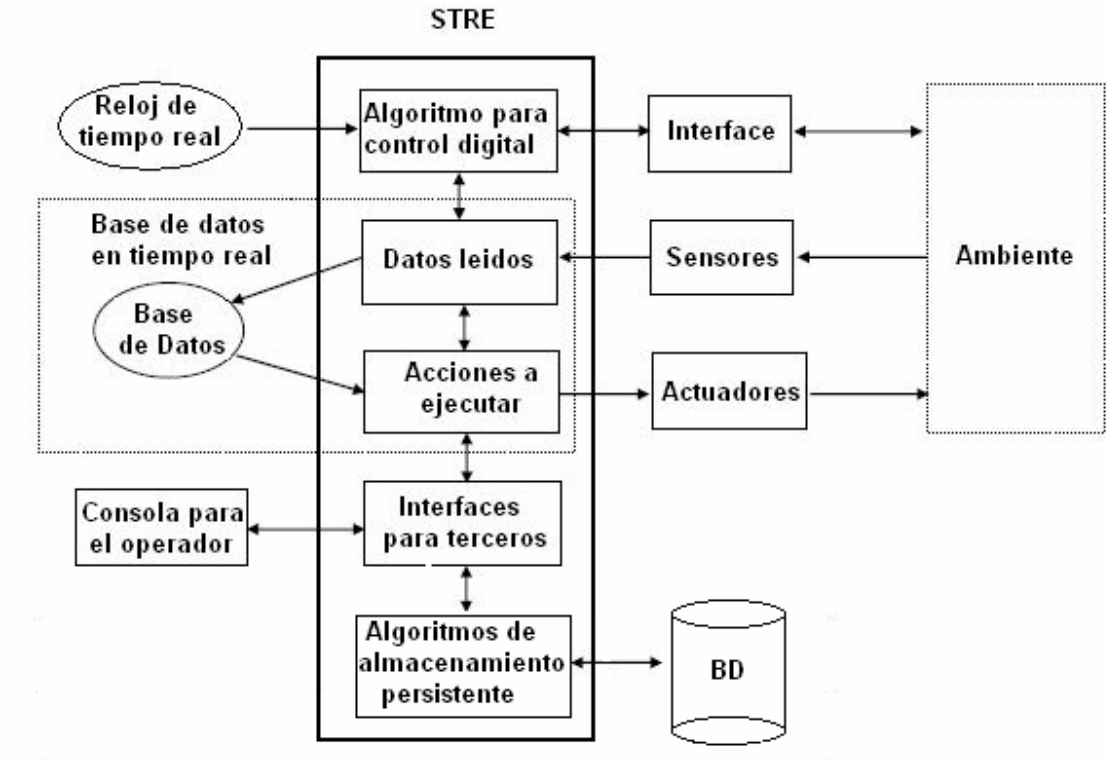


Figura 1.4.1 Propuesta de Estructura del STRE

Ambiente: se refiere al sistema controlado. Por ejemplo la red eléctrica de un territorio. El estado del ambiente (entorno físico) es supervisado por los sensores y puede cambiar por los actuadores.

Reloj de tiempo real: es el encargado de mantener concurrentes en el tiempo a todos los dispositivos electrónicos del esquema.

Base de Datos en tiempo real: es donde están distribuidos en tiempo real los valores de las variables que interesan monitorear y/o operar de la red eléctrica.

Algoritmo para control digital: usando el reloj, planifica la comunicación con el ambiente y las acciones que se realizan sobre este por intermedio de los actuadores los cuales se consideran transacciones.

Datos leídos: son aquellos que se obtienen a partir del cambio del ambiente.

Acciones a ejecutar: son las alertas o acciones a ejecutar según el análisis de los datos en tiempo real.

Interfaces para terceros: son las interfaces preparadas para que otras aplicaciones puedan establecer comunicación con el STRE, así como algunas vistas gráficas de la información que se tiene en la BDTR.

Consola para el operador: son vistas de la información que se tiene en la BDTR por intermedio de un monitor.

Algoritmos de almacenamiento persistente: cada un tiempo determinado se almacenan en la Base de datos física (BD), una copia del estado actual de la BDTR, además de todas las operaciones o cambios que se han producido en la misma.

BD: Base de datos física, donde se encuentran descritos los atributos de los componentes de la red eléctrica y las relaciones que hay entre ellos.

Actuadores: constituyen las operaciones de telemando sobre los dispositivos en el ambiente.

Sensores: captan la información acerca de los dispositivos que están en el ambiente para ser trasmitida en función de actualizar la BDTR.

Interface: son los canales de comunicación así como los protocolos usados en los mismos, para la interacción entre el sistema controlador, STRE, y el sistema controlado, el ambiente.

1.4.2 Tareas fundamentales del STRE

El servidor en su concepción se ha dividido en las siguientes tareas para su implementación modular.

1) Mecanismo de carga y construcción de la red eléctrica territorial para ser distribuida como una BDTR.

2) Mecanismo de actualización de la BDTR con componentes eléctricos no disponibles en tiempo real, pero disponibles en la BD física, mediante el Módulo de Supervisión del SIGERE.

3) Mecanismo para el cálculo de variables derivadas y la concurrencia en tiempo real de la red en consecuencia con las operaciones que se realizan sobre la misma desde el Módulo Switching del SIGERE.

4) Mecanismo de simulación de operaciones sobre la Red.

5) Mecanismo para salvas en la BD física de operaciones por despachador creando histórico de las mismas.

6) Planificador cíclico para el censado de las automáticas e interruptores inteligentes de campo y actualización de la BDTR.

7) Ejecutor de operaciones de telemando desde el despacho sobre interruptores físicos de la red.

1.4.3 La necesidad de desarrollar una BDTR en el STRE

Un factor importante a tener en cuenta es la característica de consistencia temporal de los elementos que componen la red eléctrica, también el tiempo de respuesta que brinda el STRE en relación a las operaciones sobre la red eléctrica, ya sea por medio de inserciones, eliminaciones o actualizaciones de la misma, pues el STRE debe realizar los cambios en la red eléctrica y dejar la misma en un estado consistente en el menor tiempo posible garantizando la disponibilidad de esta. Por lo general el STRE a partir de los cambios sobre la red calcula nuevos valores y los hace disponibles por medio de su base de datos, esto propicia que los algoritmos particulares usados para derivar nuevos valores deben ejecutarse en un tiempo crítico sin compilaciones de código intermedio y accesos a información en disco con el objetivo de contar con la información derivada de forma inmediata, además de crear mecanismos de notificación a las aplicaciones clientes implicadas sobre los cálculos realizados.

Los gestores de base de datos tradicionales implementan algoritmos eficientes que permiten una rapidez aceptable en las operaciones, no obstante los accesos a disco y el análisis en la mayoría de los casos del lenguaje usado, *Standard Query Lenguaje* (SQL), para realizar

operaciones de lectura, eliminación, actualización o inserción de datos en las mismas resulta una dificultad para cumplir los tiempos de respuesta que se desean. También tienen la limitante de no tratar consistencia temporal de los datos, factor fundamental en la red eléctrica.

Los SBDTR se utilizan eficazmente cuando *a)* se manejan volúmenes de datos grandes, b) las respuestas dependen de múltiples valores, *c)* se requiere respuesta a eventos aperiódicos, *d)* existen restricciones temporales. Los SBDTR se alojan y distribuyen en la memoria RAM (Randon Access Memory), a diferencia de las bases de datos tradicionales que se alojan en memoria física, garantizando un mejor rendimiento en cuanto al tiempo de respuesta sobre las operaciones que se realizan en la BDTR, (Ramamrithan, 2008), esta característica y la posibilidad de tratar la consistencia temporal de los datos, constituye un pilar prioritario en correspondencia al nivel de rendimiento que se persigue con la implementación del STRE.

CAPÍTULO 2. LOS SISTEMAS DE BASES DE DATOS EN TIEMPO REAL

En este capítulo se hace una introducción a las BDTR y se presentan sus características fundamentales, en particular los requisitos de consistencia temporal de los datos, así como una visión ampliada de las transacciones y sus particularidades en cuanto al tiempo de su completamiento y la asignación de prioridades. Se muestran una forma muy común en este tipo de base de datos de establecer el control de la concurrencia en las transacciones y de cómo resolver los conflictos entre ellas. También se evidencia como se implementan los desencadenadores basándonos en las técnicas de base de datos activas y agregando el factor tiempo en la ejecución a estos, además de evidenciar las prácticas usadas para las consultas o recuperación de los datos.

2.1 Introducción a las BDTR

Las investigaciones en el área de los SBDTR han recibido especial interés en los últimos años. Los SBDTR son sistemas de procesamiento de transacciones que están diseñados para manipular altas cargas de trabajo bajo restricciones temporales. Las transacciones en estos sistemas se caracterizan por tener plazos asociados a sus tiempos de completamiento. (Lam, 2005), (Haritsa, 2007).

Para soportar las aplicaciones en tiempo real, los SBDTR flexibilizan los requerimientos de Atomicidad, Consistencia, aIslamiento, y Durabilidad (ACID) de las transacciones para permitir un mejor soporte para consistencia temporal mientras mantienen el soporte para la consistencia de los datos. (Kang, 2006)

En muchas situaciones la puntualidad es tan importante como la durabilidad y la atomicidad. En una aplicación de base de datos diseñada para gestionar información acerca de una actividad crítica, como ocurre en un número importante de aplicaciones en tiempo real, entre las que pueden mencionarse el control de tráfico, el seguimiento de blancos, la manufactura ágil, entre otros, una transacción que lea y almacene la lectura actual de un sensor debe ser ejecutada en un intervalo de tiempo corto, cuanto más se demore su ejecución menos útil será la lectura del parámetro asociado al sensor. Los SBDTR a menudo se utilizan en estas aplicaciones para procesar transacciones dentro de sus plazos de completamiento, usando sensores que reflejan el estado actual de los datos. Los Sistemas de Bases de Datos Tradicionales (SBDT), son inapropiados para estas aplicaciones debido a que no consideran las restricciones temporales y de actualidad de los datos. (Kang, 2006)

En otras palabras, en muchas aplicaciones, una transacción que no se complete antes de un plazo específico se vuelve inútil y esta situación debe reportarse a las capas de aplicación y negocio para manejarse de la manera adecuada. Pero peor que volverse inútil, la ejecución retrasada de una transacción puede comprometer las propiedades de seguridad de un sistema. En tales casos, la detección de una violación de un plazo podría permitir la ejecución de acciones de recuperación, aislando y evitando la propagación de la falla a otros componentes y consecuentemente la ocurrencia de fallas más severas. (Ramamrithan, 2008)

2.1.1. Base de Datos Tradicionales y SBDTR. Consideraciones

Las bases de datos tradicionales tratan con datos persistentes. Las transacciones accesan a estos datos mientras se mantiene su consistencia. La seriabilidad es el criterio de correctitud usualmente asociado con las transacciones y el que permite que se garantice la consistencia de la base de datos.

Los sistemas de tiempo real tratan con datos temporales. Debido a la naturaleza temporal de los datos y los requerimientos de tiempo de respuesta impuestos por el ambiente, las tareas de tiempo real poseen restricciones de tiempo, períodos o plazos, que deben cumplirse.

Las bases de datos (BD) combinan diferentes características que facilitan:

1. La descripción de los datos.

2. El mantenimiento de la correctitud e integridad de los datos.
3. El acceso eficiente a los datos.
4. Las ejecuciones correctas de las consultas.
5. Las ejecuciones de las transacciones concurrentes.
6. El tratamiento de fallas.

Específicamente los esquemas de las BD ayudan a:

- La descripción de los datos y reducir la redundancia.
- Dar soporte al manejo de datos, como el indexado que ayuda al acceso eficiente a los datos.
- Dar soporte de transacción. Donde las transacciones tienen las propiedades ACID, estas aseguran el completamiento de las ejecuciones de las transacciones concurrentes y asegura el mantenimiento de la integridad de datos incluso en la presencia de fallas. (Vieira, 2000)

Sin embargo, según (Mendoza, 2006) y (Thomas, 2007), el soporte para sistemas de base de datos de tiempo real debe considerar lo siguiente:

- No todos los datos en la base de datos son permanentes; algunos son temporales.
- Los planes serializables temporalmente correctos son un subconjunto del conjunto de planes serializables.
- La temporalidad es más importante que la correctitud, en muchas situaciones, la correctitud puede ser tratada temporalmente.
- La atomicidad puede ser tolerada.
- Muchas de las extensiones para la seriabilidad que han sido propuestas en BD son también aplicables en BDTR, (Thomas, 2007). Algunas de estas asumen que el aislamiento de las transacciones no siempre es necesario.

Muchas de las ventajas de la tecnología de las BD, serían beneficiosas si podemos hacer uso de ellas para el manejo de datos obtenidos en STR.

En un sentido similar, los adelantos realizados en STR para procesar tareas con restricciones de tiempo, podrían ser aprovechados para tratar transacciones con restricciones de tiempo en SBDTR. (Mendoza, 2006)

2.2 Representación de los datos en BDTR.

Típicamente un STR consiste de un sistema de control y un sistema controlado, denominado ambiente. El sistema de control interactúa con el ambiente, de donde a través de sensores se obtienen los datos. Es indispensable que el estado del ambiente percibido por el sistema de control sea consistente, de otra forma, los efectos de las tareas de los sistemas de control pueden ser desastrosos. De aquí, que el monitoreo oportuno del ambiente y el procesamiento oportuno de la información censada sea necesario.

Las BDTR almacenan los datos en la forma exacta en que son usados por la aplicación, eliminando la necesidad de traducir los datos de una representación a otra. Tal traducción es necesaria, por ejemplo, si se usa una representación relacional. Todos los procesos locales de usuario tienen acceso directo a los datos, lo opuesto a usar un servidor, lo que introduce retardos no deseados. (Nogiec, 2003)

La necesidad de mantener la consistencia entre el estado actual del ambiente y el estado reflejado por los objetos contenidos en la BD conduce a la noción de la consistencia temporal. Siguiendo este criterio los elementos de la BDTR se categorizan en: *a)* elementos base y *b)* elementos derivados. La consistencia temporal tiene dos componentes: (Xiong, 2008), (Towsley, 2006),

- Consistencia absoluta entre el estado del ambiente y su reflejo en la base de datos.
- Consistencia relativa entre los datos usados para derivar a otros.

Definamos formalmente estos dos componentes. Denotemos un elemento de datos en la BDTR por, (Xiong, 2008), (Towsley, 2006)

$$d : (value, avi, timestamp)$$

Donde el *d(value)* denota el estado actual de *d*, y el *d(timestamp)* denota el tiempo en el cual la observación que relaciona a *d* fue hecha. *d(avi)* denota el intervalo de validez

absoluto o la longitud del intervalo de tiempo siguiente *d(timestamp)* durante el cual se considera que *d* tiene validez absoluta.

Un conjunto de elementos de datos usados para derivar un nuevo elemento de dato forma un conjunto de consistencia relativa. Cada conjunto *R* está asociado con un intervalo de validez relativo denotado por *Rrvi*. Se asume que *d Є R*.

d tiene un estado correcto, sí y solo sí: (Xiong, 2008), (Towsley, 2006)

1) *d(value)* es lógicamente consistente y por lo tanto satisface los restricciones de integridad.

2) *d* es temporalmente consistente, sí:

- Consistencia absoluta: (*current-time – d(timestamp)*) <= *d(avi)*

- Consistencia relativa: para toda *d' Є a R*, |*d(timestamp) – d'(timestamp)*| <= *Rrvi*

Los requisitos de consistencia temporal se traducen entonces en restricciones de tiempo sobre las transacciones. Así, para satisfacer la consistencia lógica usamos el control tradicional de concurrencia y para satisfacer los requisitos de consistencia temporal se consideran los tiempos conocidos para la administración de transacciones con restricciones de tiempo.

2.3 Transacciones en las BDTR.

Las transacciones en las BDTR pueden ser caracterizadas basándose en: *a)* la naturaleza de las transacciones en las BDTR, *b)* por la forma en la cual los datos son usados por las transacciones, y *c)* por las restricciones del tiempo, es decir, la importancia de la ejecución de la transacción en su plazo.

Las transacciones en BDTR van más allá de partir de un estado consistente de la BD, ejecutar un conjunto de operaciones y dejar la BD en otro estado consistente o en el mismo del que se partió. Las transacciones en BDTR tienen la característica de generar acciones para monitorear el ambiente cada un tiempo impuesto y reflejar su estado en la BDTR, así como actuar sobre el ambiente en un tiempo oportuno considerando las restricciones de

tiempo que garanticen la consistencia temporal de los elementos. (Ramamrithan, 2008), (Towsley, 2006), (Xiong, 2008)

Los STR realizan tres tipos de transacciones que son comunes en las bases de datos tradicionales:

- *Transacciones de solo escritura*, obtienen el estado del entorno y escriben en la base de datos.
- *Transacciones para actualización*, derivan nuevos datos y los almacena en la base datos.
- *Transacciones de solo lectura*, leen datos de la base de datos y se envían a los actuadores.

Esta clasificación puede ser usada para adaptarse a los esquemas de control de concurrencia sobre las BDTR por parte de los SBDTR de la siguiente manera. (Mendoza, 2006), (Towsley, 2006)

- *Transacciones sobre sensores,* obtienen el estado del entorno y reflejan el mismo escribiendo sobre la BDTR.
- *Transacciones para actualización*, derivan nuevos datos que se almacenan y publican en la BDTR, así como generan acciones sobre el ambiente si se cumplen un grupo de condiciones.
- *Transacciones de usuarios*, transacciones a nivel de usuario que pueden ser: *a)* de solo lectura, *b)* de solo escritura o *c)* o de modificación que involucra la lectura y la escritura.

Algunas restricciones de tiempo se derivan de los requerimientos de consistencia temporal y algunas de los requerimientos impuestos en el tiempo de reacción del sistema. Los requerimientos de periodicidad comúnmente tienen la siguiente forma:

Cada 10 segundos muestra el estado operativo del interruptor X.

Los requerimientos de reacción del sistema o actualización de valores comúnmente toman la forma de la restricción del plazo impuesto en las transacciones aperiódicas: Por ejemplo,

Si disparó interruptor X entonces en 10 mili segundos aislar transformador T1.

En este caso, la acción del sistema en respuesta a un disparo del interruptor *X* debe ser completada en 10 milisegundos.

Las transacciones también pueden ser distinguidas basadas en los efectos de la pérdida del plazo de una transacción. Considerando los diversos tipos de funciones que pueden asociarse a las transacciones, estas se categorizan de la forma siguiente: (Xiong, 2009)

- Transacciones de Plazo Crítico, son aquellas que pueden resultar en una catástrofe si el plazo falla.
- Transacciones de Plazo Suave, son aquellas que tienen algún valor aún después de un cierto tiempo de su plazo. Si el valor desciende hasta cero y alcanza su plazo, entonces obtenemos una transacción de plazo estable, pero si su plazo expira entonces no se toma en cuenta.

2.3.1 La necesidad de la predecibilidad

Las transacciones y procesamiento de consultas tienen restricciones de tiempo y hay diferentes consecuencias de no satisfacer esas restricciones. Una característica principal al problema del procesamiento de una transacción es la predecibilidad. Si *una transacción de plazo crítico* en tiempo real falla en su plazo, esta tiene consecuencias catastróficas. Considerando esto, es necesario predecir de antemano que las transacciones se completarán antes de que finalicen su plazo. Esta predicción será posible sólo si conocemos el peor tiempo de ejecución de una transacción, los datos y las necesidades de recursos de la transacción. Además, es deseable tener pequeñas variaciones entre los casos peores de predicción y las necesidades reales. La predecibilidad también es importante para *las transacciones de plazo suave*, aunque en menor grado. Conociendo de antemano al inicio de una transacción que esta no podrá completarse en su plazo permitirá al sistema descartar la transacción o acometer las acciones pertinentes para el completamiento de la misma en el plazo establecido según convenga, así se evita gastar tiempo en la transacción o darle recursos innecesarios para su completamiento. (Haritsa, 2007), (Mendoza, 2006)

En un Sistema de Base de Datos, existen un número de fuentes de impredecibilidad:

- Dependencia de la secuencia de ejecución de las transacciones sobre los valores de los datos.

- Datos y conflictos de recursos
- Paginación dinámica y I/O
- Aborto de transacciones y la restauración de resultados, y reinicialización.

Las bases de datos distribuidas tienen problemas adicionales retardos de comunicación y las fallas de sitios.

Es conveniente el análisis fuera de línea de una transacción porque este provee un estimado de requerimientos de tiempos de cómputo, datos y recursos. Para obtener la información necesaria de una transacción, se deben de tomar en cuenta cuatro fuentes de impredecibilidad. Las transacciones tienen dos fases. La primera fase es la pre-carga, es donde inicia la ejecución de la transacción para obtener los datos necesarios para su procesamiento posterior, no se consideran escrituras y los conflictos con otras transacciones. La demanda de los tiempos de cómputo de la transacción también son determinados en esta fase. Es posible que otras transacciones puedan cambiar los datos sin alterar el progreso de esta fase, (Franaszed, 1990). Una vez que se han completado los requerimientos se va a la segunda fase donde se puede garantizar la ejecución completa de la transacción en sus restricciones de tiempo. Para garantizar que la transacción se complete en su plazo, se construye un plan de ejecución que toma en cuenta los requerimientos de tiempo, recursos y datos. Si el plan no puede ser construido, la transacción es abortada, o descartada. (Ramamritham, 1990)

Si al final de la fase de pre-carga se detecta que el estado de los datos ha cambiando, entonces la fase de pre-carga debe ser reejecutada. En cualquier caso, si los datos no cambian se puede garantizar que una transacción se completará en su plazo y no se requiere de ninguna acción de recuperación, si una transacción se inhabilita para ejecutarse. El precio de esta última situación es la sobrecarga de la fase de pre-carga. Es posible aplicar diferentes optimizaciones, para mas detalles revisar. (O'Neil, 1992)

2.3.2 Tratando con Plazos Críticos.

Todas las transacciones con plazos críticos deben satisfacer sus restricciones de tiempo. Por consiguiente el manejo dinámico de transacciones no puede garantizarlo, los datos y el

procesamiento de recursos así como el tiempo necesitado por las transacciones tiene que ser garantizado y estar disponible cuando sea necesario. Esto tiene diferentes implicaciones.

Primero, se tiene conocimiento de cuando las transacciones sean probablemente invocadas. Esta información está disponible para transacciones periódicas, pero no para las transacciones aperiódicas. La separación más pequeña de tiempo entre dos ocurrencias de una transacción aperiódica puede ser vista como periódica. De aquí que podemos tratar a todas las transacciones con plazo crítico de tiempo real como transacciones periódicas.

Segundo, para asegurar a priori que sus plazos se cumplan, tenemos que determinar sus requerimientos de recursos y los peores casos de tiempos de ejecución de una transacción. Una vez que hemos logrado lo anterior, podemos tratar a las transacciones de manera similar como se tratan a las tareas periódicas en los sistemas de tiempo real, por ejemplo, usando planificadores con manejo de tablas estáticas o manejo de prioridad preferente. Además podemos utilizar herramientas para el análisis de la planificación, y por consiguiente verificar si el conjunto de transacciones son planificables considerando sus períodos y requerimientos de datos. (Rajkumar, 1988)

2.3.3 Tratando con Plazos Suaves

Las transacciones con plazos suaves de tiempo real, tienen más tiempo para procesarlas por lo tanto no requerimos encontrar los plazos todo el tiempo. La meta es encontrar el plazo de la transacción adoptando políticas de asignación de prioridad y mecanismos para la resolución de conflictos que explícitamente toman en cuenta el tiempo.

2.3.4 Asignación de prioridades

Los SBDTR asignan prioridades a las transacciones basadas en plazos y por el valor que representan para el sistema. Las posibles políticas de asignación son:

- Primero el plazo más temprano (*Earliest-deadline-first*).
- Primero los valores más altos (*Highest-values-first*).
- Primero los valores más altos por unidad de tiempo de cómputo (*Highest-value-per-unit-computation-time-first*).

- Primero la transacción de mayor duración de ejecución (*Longest-executed-transaction-first*).

En (Huang, 1989) se muestra que la política de asignación de prioridad tiene impacto significativo en el desempeño y que cuando las diferentes transacciones tienen valores diferentes, el plazo y valor deben ser considerados.

2.3.5 Control de concurrencia

Las transacciones en BDTR pueden controlar la concurrencia bloqueando la base de datos completamente, lo que no es significativo para la ejecución de las transacciones, pues aunque se reduce drásticamente el costo del control de la concurrencia también se imposibilita la ejecución de transacciones en paralelo que agilizan el funcionamiento de la BDTR en cuanto al tiempo de respuesta y ejecución de tareas de esta.

Una alternativa viable es que los procesos en la BDTR pueden adquirir bloqueos de objetos simples o múltiples, examinando los estados de bloqueo y liberando bloqueos por la fuerza. Adquiriendo múltiples bloqueos en un momento puede prevenir los bloqueos de plazos. La capacidad de desbloquear objetos bloqueados por otros procesos puede ser útil cuando se implementa un mecanismo de recuperación a partir del procesamiento de excepciones y procesos terminales sin liberar todos los bloqueos previamente adquiridos. Los SBDTR implementan mecanismos de control de concurrencia básicos pero las políticas deben designarse e implementarse por la aplicación. (Nogiec, 2003)

El criterio de corrección más ampliamente usado para el control de concurrencia es el de seriabilidad. Se debe insistir en que cuando varias transacciones se ejecutan de manera concurrente, la consistencia de la BDTR se puede perder aún cuando cada transacción sea correcta, el concepto de seriabilidad ayuda a identificar las ejecuciones que aseguran la consistencia. (Lam, 2005)

Una forma de asegurar la seriabilidad de las transacciones es exigir que el acceso a los datos se haga de manera mutuamente excluyente, es decir, mientras una transacción accede a un dato, ninguna otra puede modificarlo; el método más común que se utiliza para instrumentar lo anterior es permitir que una transacción aceda a un solo dato sólo si tiene un bloqueo sobre el dato. (González, 2005)

Existen varios modos de bloqueo, los básicos son dos:

1) Compartido: si una transacción *T* ha obtenido un bloqueo compartido *S* sobre el elemento *Q*, *T* puede leerlo pero no modificarlo.
2) Exclusivo: si una transacción *T* ha obtenido un bloqueo exclusivo *X* sobre el elemento *Q*, *T* puede leerlo y modificarlo.

Matriz de Compatibilidad entre bloqueos:

	Compartido(*S*)	Exclusivo(*X*)
Compartido(*S*)	Verdadero	Falso
Exclusivo(*S*)	Falso	Falso

Para acceder a un dato, la transacción primero debe bloquearlo, si el dato ya está bloqueado por otra transacción con un modo incompatible debe esperar hasta que se desbloquee.

Una transacción debe bloquear un dato mientras accede a él y no es siempre deseable que se desbloquee de inmediato después de su último acceso pues puede no garantizarse la seriabilidad, o sea se puede caer en situaciones de abrazo mortal (DeadLock) con otras transacciones que acceden a los mismos datos de forma concurrente.

Cuando ocurre un bloqueo mutuo el sistema debe reiniciar una de las dos transacciones, los datos bloqueados se liberan y la otra puede continuar.

Lo anterior nos indica que el bloqueo debe usarse con cuidado, si queremos aumentar la concurrencia desbloqueando tan pronto como se puede se puede incurrir en estados inconsistentes y si no, podemos originar un bloqueo mutuo.

2.3.5.1 Protocolo de bloqueo en dos fases

Este protocolo garantiza la seriabilidad y se basa en las siguientes fases: (Carlson, 1999)

- Fase de crecimiento: una transacción obtiene bloqueos.

- Fase de mejoramiento: Una vez que una transacción libere un bloqueo solo puede hacer liberaciones, no puede adquirir ningún otro.

Un protocolo de bloqueo en 2 fases (2 PL) garantiza la seriabilidad en conflictos pero no evita el bloqueo mutuo. El protocolo de bloqueo en dos fases se puede refinar para aumentar la concurrencia, a continuación ejemplificamos esta técnica. (González, 2005)

Si se bloquea de modo exclusivo *X* a un elemento *A* desde el inicio en la transacción *T1*, se elimina la posibilidad de concurrencia con otra transacción *T2*. Se puede primero solicitar un bloqueo compartido *S* para *A* y después elevarlo a *X*, para ello se cuenta con las sentencias:

upgrade: convierte de modo *S* a *X*.

downgrade: convierte de modo *X* a *S*.

Estas sentencias no se pueden usar de manera arbitraria, *upgrade* solo se puede usar en fase de crecimiento y *downgrade* en fase de encogimiento. (González, 2005)

Cuando una transacción ejecute una lectura sobre el elemento *A*, el sistema automáticamente ejecuta un bloqueo compartido de *A*, si una transacción ejecuta una escritura de *A*, se comprueba si dicha transacción tiene un bloqueo compartido sobre *A* en cuyo caso ejecuta *upgrade*(*A*), si no se ejecuta bloqueo exclusivo de *A*.

La implementación del protocolo de bloqueo en dos fases es muy usado en BDTR por su simplicidad de implementación y eficiencia en cuanto a costo computacional, el gran problema consiste en la pérdida de tiempo en los chequeos del estado de DeadLock entre las transacciones y la posible inconsistencia que se puede originar si se liberan los bloqueos de forma incorrecta, para minimizar dicho inconveniente y mantener una eficiente consistencia de los datos la mayoría de los manejadores 2PL en BDTR son 2PL estricto, liberan todos sus bloqueos cuando la transacción termina, a diferencia del protocolo 2PL que libera sus bloqueos en escala. (Carlson, 1999)

2.3.5.2 Manejo de DeadLock

Como ya se ha dicho, los algoritmos de control de concurrencia pueden conducir a un estado de DeadLock, o sea un conjunto de transacciones esperan unas por otras.

Un DeadLock es un fenómeno permanente, si se presenta en un sistema, no se resuelve si no es con una acción externa de parte del usuario, del SBDTR o del sistema operativo.

Una técnica útil para analizar los DeadLock son los grafos de espera (WFG), (González, 2005), este es un grafo dirigido que representa la interrelación "espera – por" entre transacciones. Los nodos de este grafo representan las transacciones concurrentes del sistema, un arco entre *Ti* y *Tj* (*Ti*→*Tj*) indica que la transacción *Ti* está esperando porque *Tj* libere un dato. En la figura 2.3.5.2.1 se muestra la representación en WFG del estado de DeadLock entre las transacciones *T1* y *T2*.

Figura 2.3.5.2.1. Representación en WFG del estado de DeadLock entre 2 transacciones.

2.3.5.3 Métodos que evitan el DeadLock

Los esquemas que evitan el DeadLock emplean técnicas de control de concurrencia que nunca incurren en DeadLock o requieren que los planificadores detecten situaciones potenciales de deadlock por adelantado y aseguran que no ocurra.

La forma más simple de evitar DeadLock es ordenar los recursos o unidades de bloqueos y que las transacciones siempre adquieran los bloqueos en tal orden, este ordenamiento de unidades de bloqueo puede hacerse de forma global o local. (González, 2005)

Otra alternativa es hacer uso de las horas de entrada de las transacciones y resolver el DeadLock abortando ya sea la de mayor prioridad o la de menor. Para instrumentar este tipo de alternativa, el manejador de bloqueo debe ser modificado de la siguiente manera:

Si una solicitud de bloqueo de una transacción *Ti* es denegada, el manejador de bloqueo no debe forzar a que *Ti* espere sino que debe aplicarse una prueba. Si *Ti* pasa con éxito la prueba, puede esperar, de lo contrario una de las dos transacciones involucradas en el conflicto debe abortar o reiniciarse. Los métodos más conocidos para estas pruebas son: *a)* Esperar - Morir y *b)* Herir - Esperar.

Ambos métodos se basan en la asignación de horas de entrada sobre las transacciones y se describen a continuación: (González, 2005)

Regla Esperar - Morir: si *Ti* solicita un bloqueo sobre un elemento de dato que ya está bloqueado por *Tj*, *Ti* espera si y solo si *Ti* es más vieja que *Tj*. Si *Ti* es más nueva que *Tj* entonces *Ti* aborta y se reinicia con la misma hora de entrada.

Regla Herir - Esperar: si *Ti* solicita un bloqueo sobre un elemento de dato que ya está bloqueado por *Tj*, entonces *Ti* espera si y solo si *Ti* es más nueva que *Tj* sino *Tj* aborta y el bloqueo se concede a *Ti*.

Observar que en ambas reglas la transacción más joven es la que aborta.

La detección del DeadLock se hace sobre el análisis del WFG para determinar si hay ciclos y su solución una vez detectado usualmente se hace seleccionando una o más víctimas que deben abortar para romper el ciclo. La selección de una víctima se hace entre las transacciones involucradas en el DeadLock y no es necesariamente la transacción que originó el ciclo, puede ser elegida en base a determinados criterios como son:

- La más joven.
- La que ha adquirido menos bloqueos.
- La que ha hecho la menor cantidad de cambios.
- La que aún le falta más para terminar.
- La que forma parte de más de un ciclo en el grafo de espera.
- La de menor prioridad.

2.4 Relación entre las BDTR y las Bases de Datos Activas. Los desencadenadores en las BDTR

La construcción básica de un bloque en una base de datos activa es la siguiente:

ON *evento*
  IF *condición*
    DO *acción.*

En la ocurrencia del *evento* especificado, si la *condición* se mantiene, entonces la *acción* especificada puede realizarse. Este constructor provee un buen mecanismo por el cual las restricciones de integridad de los datos pueden mantenerse. El evento puede ser arbitrario, incluyendo eventos externos, como en el caso de eventos en tiempo real generados por el ambiente, eventos temporales, o eventos relacionados con transacciones, como es el inicio y confirmación de transacciones. La condición puede corresponder a condiciones del estado de los datos o del entorno. Se dice que la acción es activada (Korth, 1990), (Dayal, 1988) y esta puede ser una transacción arbitraria.

Las bases de datos activas proveen un buen modelo para el arribo de tareas periódicas y aperiódicas basadas en eventos y condiciones. Aunque la estructura anterior implica que una base de datos activa pueda reaccionar a las interrupciones, las restricciones de tiempo no son explícitamente consideradas por la transacción subyacente en el mecanismo de procesamiento.

La deficiencia principal en bases de datos activas es la ausencia de restricciones de tiempo en el completamiento de las transacciones, por lo que se hace necesario que las bases de datos activas consideren las restricciones de tiempo. (Korth, 1990), (Dayal, 1988)

Considere que se desea activar una protección que aísla el funcionamiento de un transformador de alto voltaje en una subestación de distribución dentro de la red eléctrica de un territorio, si al medir la potencia activa que llega a un interruptor esta se comporta fuera de los límites establecidos, la protección se ejecuta abriendo el interruptor y como consecuencia el transformador queda aislado de la red.

Esta situación se podría describir de la siguiente manera:

ON (Operación de cierre del interruptor X)

IF (potencia activa fuera de límites)

DO (en 3 milisegundos abrir interruptor X)

En resumen, mientras las bases de datos activas poseen las características necesarias para tratar con muchos aspectos de los STR, el ingrediente crucial faltante es la persecución del procesamiento oportuno de las acciones. (Korth, 1990), (Dayal, 1988), (Carlson, 2003)

Las BDTR ofrecen un mecanismo similar al mecanismo de desencadenadores (triggers) típicamente encontrado en los sistemas de gestión de bases de datos relacionales. Un trigger puede definirse para activarse al insertar, eliminar, modificar o acceder un objeto.

Los triggers en una BDTR no son más que eventos asociados a los distintos cambios que sufren los objetos, se ejecutan como una transacción más, y en la mayoría de los casos están orientados a generar nuevos elementos en la BDTR o para ordenar ejecución de operaciones sobre el ambiente que se controla dentro de los plazos establecidos. (Korth, 1990), (Dayal, 1988), (Carlson, 2003), (Nogiec, 2003).También hay triggers que se ejecutan en modo difundido (broadcast) permitiendo la notificación a todas las aplicaciones conectadas a la BDTR de alguna alarma, dato o evento importante que se ha generado. Una característica importante de los triggers en las BDTR es que son usados para asegurar la consistencia temporal de los datos. (Nogiec, 2003), (Carlson, 2003)

2.5 Consultas basadas en atributos en las BDTR

Las BDTR incorporan un mecanismo de recuperación de objetos que se basa en los atributos de estos para ser seleccionados. Cada objeto en la DBTR tiene un conjunto de atributos públicos que pueden ser manipulados libremente por la aplicación. Entonces los valores de los atributos para un objeto particular pueden cambiar durante el tiempo de vida de la aplicación. El significado de los atributos es suministrado por la aplicación. La BDTR proporciona operaciones para seleccionar objetos, basado en subconjuntos de atributos. Por ejemplo, se pueden usar atributos para definir clases o estados de objetos. En tal caso, según (Nogiec, 2003) y (Ramamrithan, 2008), las siguientes consultas son posibles:

select class = * **and** state = suspended

select class = dataSource **and** locked != true

select class = pressure **and** alarmRedZone = true

Las implementaciones actuales en BDTR no contemplan el uso de un lenguaje de consulta para impedir retardos en tiempo de ejecución introducidos por el traslado de las sentencias de consulta, así como los análisis léxico y sintáctico que se imponen antes de ejecutar las mismas, en su lugar se usa una librería de funciones que permiten definiciones de búsqueda especificando parámetros sobre los atributos de los objetos en sí.

Este mecanismo de consulta es muy sugerente y rápido, los accesos a objetos se logran usando estructuras como: B-Tree, B+, B- y tablas Hash, los objetos dentro de la BDTR son accedido por sus nombres que son únicos, aunque representa un compromiso entre lo lento y lo complejo, pues una BDTR que presente un mecanismo poderoso y diversificado de consultas como las bases de datos tradicionales, reduciría el rendimiento en tiempo de la misma, que en su caso es una característica fundamental y deseada.

Podemos resumir que las consultas en BDTR se realizan a través de librerías de funciones que se apoyan en estructuras de datos propicias para el almacenamiento y acceso rápido de los elementos.

CAPÍTULO 3. IMPLEMENTACIÓN DE LA BDTR EN EL SERVIDOR DE TOPOLOGÍA DE LA RED ELÉCTRICA

En este capítulo se describe el estado actual del STRE y se informa acerca de las particularidades de los componentes en su entorno, se caracteriza la tecnología .Net Remoting usada en la implementación del STRE, se muestran los ficheros de configuración usados para lograr las comunicaciones entre el STRE y las aplicaciones clientes que se conectan con este, se detalla el modelo usado para la representación de los datos en la BDTR, se define y ejemplifica como se implementaron las transacciones y el control de la concurrencia entre estas, se muestra como se implementaron los desencadenadores y el mecanismo de consultas en la BDTR.

3.1 Descripción actual del STRE

El STRE se ha implementado usando el Borland Developer Studio 2006, sobre un proyecto del tipo WinForm .Net, aprovechando muchas de las bondades que brinda la tecnología .NET de la compañía Microsoft; explotándose fundamentalmente la heterogénea librería de clases que brinda el FrameWork 1.1 y la tecnología para la distribución de objetos en tiempo real, .Net Remoting. El uso de la mencionada tecnología agiliza el proceso de implementación al usar o heredar en muchos casos, las clases disponibles en el FrameWork, así como se mantiene la compatibilidad en cuanto al lenguaje de programación con el SIGERE, que lleva alrededor de 10 años implementándose de forma absoluta con el lenguaje Object Pascal.

El STRE cuenta con un entorno dinámico a su alrededor, donde intervienen usuarios, aplicaciones clientes, comunicaciones y un servidor de datos, en la figura 3.1.1 se muestra la estructura actual del STRE y el medio en que realiza sus operaciones.

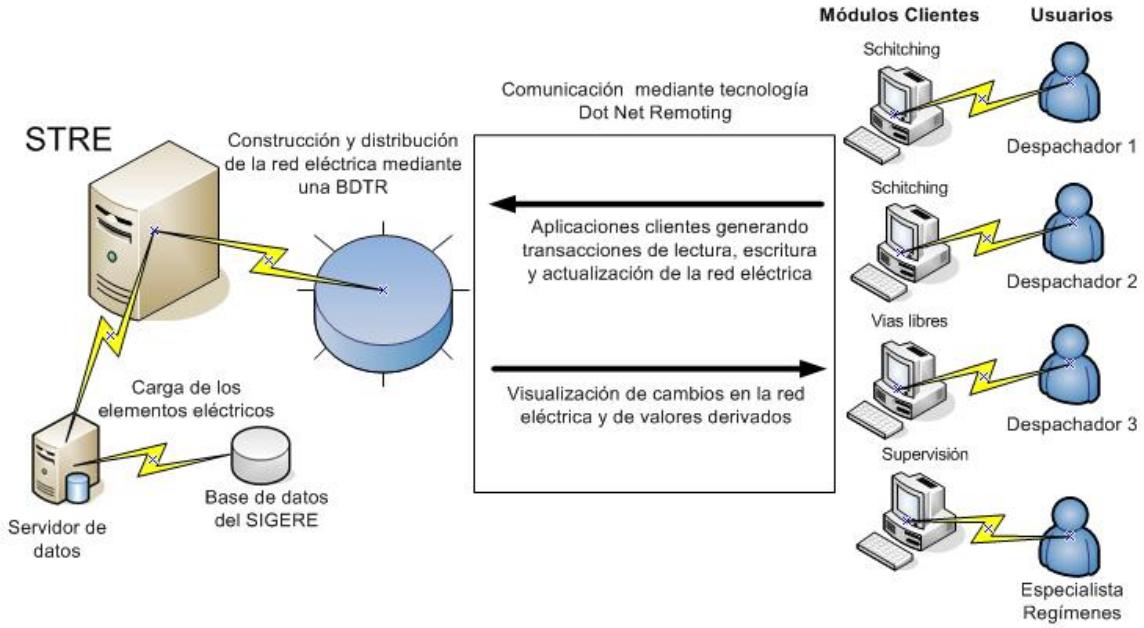


Figura 3.1.1 Estructura actual del STRE

Base de datos del SIGERE: es donde se almacenan físicamente los componentes eléctricos que componen la red eléctrica del territorio y las relaciones existentes entre estos. Además se almacenan todas las operaciones realizadas sobre la red eléctrica por medio de los despachadores.

Esta base de datos está alojada en un servidor SQL SERVER 2000, tecnología usada por el SIGERE desde hace más de 8 años, suficientemente potente que garantiza los requerimientos deseados en cuanto al almacenamiento, operación y recuperación de los datos.

La figura 3.1.2 muestra el diagrama lógico de la porción de la BD que involucra las tablas que contienen los componentes eléctricos, así como las relaciones entre los mismos.

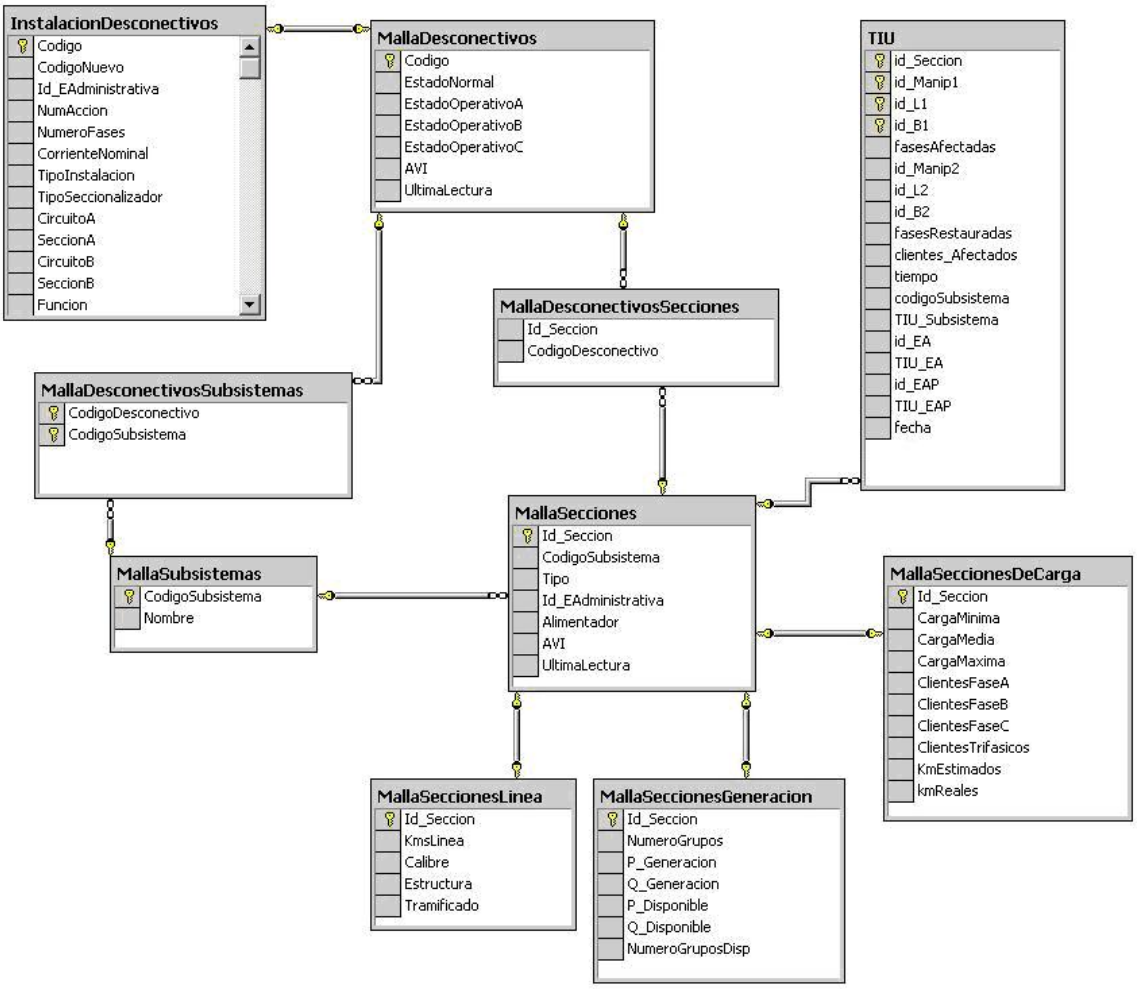


Figura 3.1.2 Tablas y sus relaciones donde se describen físicamente los componentes eléctricos de la red del territorio.

Las principales características del diagrama radican en:

- Un desconectivo, interruptor, limita con dos secciones, una por cada extremo, y una sección es limitada por varios desconectivos.
- Las secciones pueden ser de varios tipos: de carga, donde se encuentran conectados los clientes a garantizar el servicio; de línea, por donde se transmite la corriente eléctrica y; de generación, donde se origina la corriente eléctrica, además de otros tipos de funciones las cuales no se modelan en tablas separadas.
- Los desconectivos pueden o no pertenecer a varios subsistemas a la vez, circuitos eléctricos, los que están en más de uno se denominan desconectivos comunes.

- Las secciones pertenecen a un solo subsistema, circuito eléctrico.
- Cuando a una sección pasa del estado energizada a no energizada y viceversa, se generan valores auxiliares los cuales se registran en la tabla TIU, es por eso que una sección puede contar con varios valores de tiempo de interrupción por usuario (tiu).
- En las tablas MallaDesconectivos y MallaSecciones se cuenta con los atributos distintivos AVI y UltimaLectura, los mismos se refieren al tiempo establecido para garantizar consistencia absoluta y temporal de los objetos en la BDTR.

En la figura 3.1.3 se muestra el diagrama lógico de la porción de la base de datos que involucra las tablas que contienen la información acerca del almacenamiento de todas las operaciones que se realizan sobre la red eléctrica.

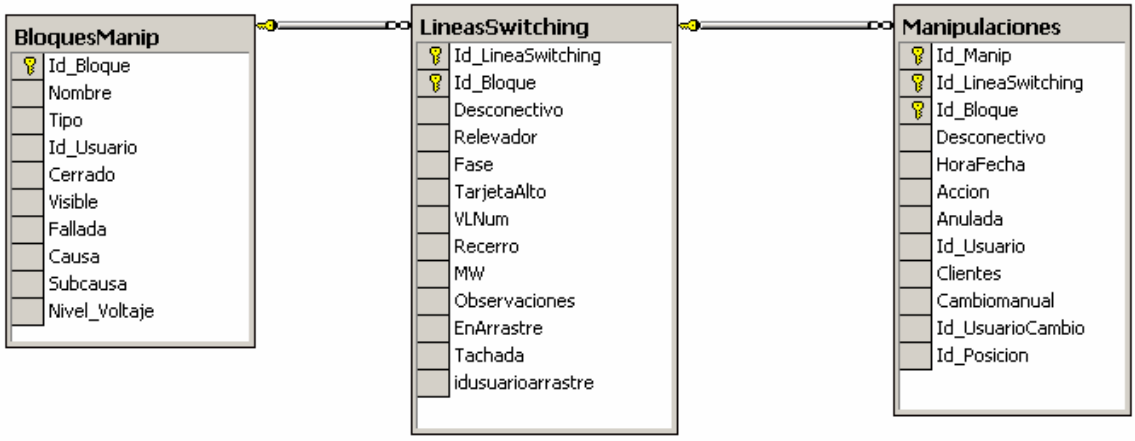


Figura 3.1.3 Tablas y sus relaciones donde se describen las operaciones que se realizan sobre la red eléctrica.

Las principales características del diagrama radican en:

- Un bloque de manipulaciones, operaciones sobre la red, contiene varias líneas de manipulación y una línea está en un solo bloque de manipulaciones.
- Una línea de manipulaciones contiene dos manipulaciones y una manipulación se encuentra en una sola línea de manipulaciones.
- La tabla Manipulaciones entre sus atributos cuenta con la identificación del usuario, *id_Usuario*, que realizó la manipulación para futuras auditorías de sus operaciones lo que da al traste a su evaluación en la operación de la red.

Servidor de datos: constituye un servidor profesional, de gama alta, donde está instalado el Microsoft SQL SERVER 2000 que alberga la base de datos del SIGERE.

STRE: Servidor de Topología de la Red Eléctrica, carga los componentes de la red y las relaciones entre ellos, construye la BDTR y distribuye la misma mediante una interface de comunicación por medio de tecnología .Net Remoting, además gestiona las transacciones de lectura, escritura y actualización que generan las aplicaciones clientes que se conectan al servidor para interactuar con la BDTR. También notifica a las aplicaciones conectadas de los cambios que se originan en la red distribuida, además almacena los mismos mediante un mecanismo de persistencia de la red y de las operaciones realizadas sobre la misma.

Módulos Clientes: Supervisión, Switching y Vias Libres, se conectan a la interface que exporta el servidor por intermedio de tecnología .Net Remoting y generan transacciones sobre la BDTR. En los Anexos II y III se muestra una vista gráfica de los módulos Switching y Supervisión.

Usuarios: Generalmente despachadores y especialistas del departamento Regímenes que usan los módulos clientes para visualizar, operar y actualizar la red eléctrica.

El principio de funcionamiento del STRE consiste en, primeramente al iniciar, el STRE crea una conexión con la BD física del SIGERE con el propósito de cargar los componentes eléctricos y sus relaciones, los que describen la red eléctrica del territorio. Posteriormente construye la BDTR y la distribuye usando .Net Remoting, inmediatamente se coloca a la escucha de conexiones de los módulos clientes, los cuales generan transacciones de lectura, escritura, actualización, inserción y eliminación sobre la red eléctrica. Varias transacciones pueden ejecutarse de forma concurrente por intermedio de hilos de ejecución, el STRE mantiene el control de la concurrencia mediante 2PL estricto usando métodos ya descritos en el epígrafe 2.3.5 para evitar el DeadLock.

Debe tenerse en cuenta que en los momentos actuales, el STRE no cuenta en su entorno con toda la tecnología necesaria para comunicarse con los elementos eléctricos que están en el campo, apoyándose en el reloj de tiempo que garantiza el sincronismo en la concurrencia de todos los dispositivos; es por ello que todas las implementaciones realizadas en la BDTR del STRE están orientadas a satisfacer las aplicaciones clientes que se conectan y a su notificación.

3.3.1 Características de la tecnología .Net Remoting

En el STRE se ha implementado un mecanismo para distribuir en memoria RAM los objetos que se encuentran almacenados en la BDTR con el fin de que estos puedan ser accedidos de forma rápida desde las distintas aplicaciones que implementan las interfaces requeridas para lograrlo.

Con el fin de garantizar este propósito se escogió la tecnología .Net Remoting de Microsoft, la cual provee facilidades para crear entornos distribuidos por los distintos canales de comunicación que soporta.

Algunas de las características fundamentales que se tuvieron en cuenta para seleccionar la tecnología son las siguientes: (Rammer, 2002)

- Tiene como objetivo crear herramientas que faciliten la distribución de la aplicación en red de forma transparente. *Es lo que se desea con el STRE.*
- Tecnología de objetos distribuidos sucesora de DCOM con nuevas potencialidades y mejoramiento de algunas ya existentes. *Hace que se discrimine el uso de DCOM.*
- Presenta un marco variado y extensible para que los objetos de distintos dominios de aplicaciones, procesos y equipos se puedan comunicar sin problemas. *Garantiza las comunicaciones entre distintos artefactos, aplicaciones, servicios, equipos físicos, etcétera.*
- Está implementada sobre las ideas fundamentales encontradas en CORBA y Java RMI, aunque la combinación final es algo diferente que se distingue por corregir las limitaciones fundamentales de estas últimas tecnologías. *Hace que se discrimine las tecnologías CORBA y Java RMI.*
- Posee gran flexibilidad, y es una tecnología independiente del lenguaje de programación y con las comunicaciones en un nivel de abstracción elevado. *Permite compatibilidad con el SIGERE en cuanto al lenguaje de programación y garantiza la fácil implementación de las comunicaciones.*
- Modelo de programación sencillo y eficaz, así como compatibilidad en tiempo de ejecución que permite interacciones transparentes. *Un modelo de programación*

sencillo agiliza el proceso de construcción de las aplicaciones, en este caso el STRE.

3.3.1 Ficheros de configuración para establecer comunicación entre el STRE y las aplicaciones clientes.

Una forma sencilla y práctica para establecer comunicación entre los diferentes dominios de aplicación usando tecnología .Net Remoting es a través de ficheros de configuración.

Existen dos tipos de ficheros de configuración, 1) se aloja junto con la aplicación que va a distribuir los objetos, denominado fichero de configuración del servidor y 2) está junto a cada aplicación cliente que desea conectar con la aplicación que distribuye los objetos, denominado fichero de configuración cliente.

Al inicial el STRE, se carga y procesa el archivo Server.config, fichero de configuración del servidor. Las propiedades fundamentales que se describen en él son: el nombre del ensamblado de donde se va distribuir, el espacio de nombre y la clase que se desea distribuir dentro del ensamblado, el nombre lógico que se atribuye a la distribución, el modo en que se van a instanciar los objetos distribuidos, el protocolo de comunicación y el puerto, entre otros. En el Anexo IV se muestra una de las posibles configuraciones del contenido del fichero Server.config.

Al iniciar cualquiera de las aplicaciones clientes, se carga y procesa el archivo Client.config, fichero de configuración cliente. La propiedad fundamental que se describe en él es la dirección donde se encuentran los objetos distribuidos, donde se especifica el protocolo para establecer la comunicación, el nombre del servidor, el puerto y el nombre lógico de la distribución. En el Anexo V se muestra una de las posibles configuraciones del contenido del fichero Client.config.

3.2 Modelo de representación de los datos en la DBTR del STRE.

La representación de los datos de la BDTR del STRE en lugar de utilizar el modelo relacional, se basa en el modelo orientado a objetos. Los objetos contienen la información de los elementos de la red eléctrica, sus características fundamentales y las relaciones entre los mismos.

Todos los atributos de las clases que se presentarán en los diagramas a continuación son públicos, pues aunque se pierda la propiedad de encapsulamiento de los atributos en los objetos, el objetivo es lograr un acceso directo a los atributos, y con ello una mayor transparencia y rapidez al no tener que contar con métodos para el acceso, esta característica es esencial para el manejo de BDTR modeladas a través de objetos.

La figura 3.2.1 se muestran las clases del modelo que describen los componentes eléctricos y las relaciones que existen entre estas.

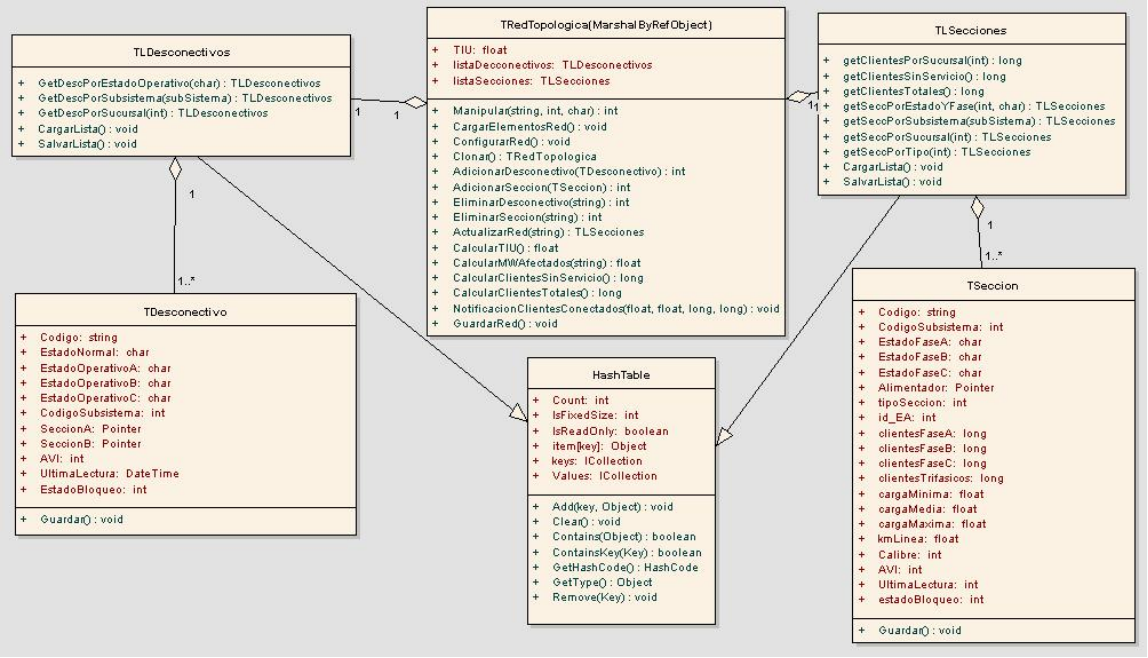


Figura 3.2.1 Clases que describen los componentes eléctricos de la red y la relación entre estas.

En la figura se observan dos clases fundamentales donde sus instancias contienen la información fundamental de la red eléctrica, estas se describen en detalle a continuación.

TDesconectivo: clasifica los objetos de tipo interruptor, a continuación se detallan sus atributos.

- Codigo: código único en la base de datos del SIGERE y en la BDTR del interruptor.
- EstadoNormal: estado en que se debe encontrar el interruptor en condiciones óptimas de la red eléctrica, puede tomar el valor '*A'*, que significa normalmente abierto o el valor '*O'*, que significa normalmente cerrado.

- EstadoOperativoA, EstadoOperativoB, EstadoOperativoC: se refiere al estado actual del interruptor en cada una de sus fases, estos atributos pueden tomar valores de *'A'*, abierto u *'O'*, cerrado, de manera independiente, pues los interruptores pueden ser operados por fase eléctrica.
- CodigoSubsistema: contiene un código de tipo entero que describe el subsistema, circuito, al cual pertenece el interruptor. En el caso de que el interruptor sea común a dos subsistemas se toma el subsistema de mayor nivel de voltaje, en el caso de que este último sea el mismo se toma cualquiera de ellos.
- SeccionA, SeccionB: se refiere a los códigos de las dos secciones que están conectadas al interruptor una por cada extremo de este. Estos atributos son fundamentales en el proceso de construcción de la red eléctrica.
- AVI: contiene un valor entero positivo que especifica el intervalo de validez absoluta del interruptor, este es uno de los atributos que se usa para garantizar la validez absoluta y relativa del dato, además que constituye la información fundamental para el muestreo a través de sensores del interruptor físico.
- UltimaLectura: contiene un valor de tipo *DateTime* que específica el instante de tiempo en que se realizó el último muestreo del interruptor físico, también es usado para garantizar la consistencia temporal del dato.
- EstadoBloqueo: se refiere a si el objeto está bloqueado por algún proceso o transacción, los valores que puede tomar son: 0, que significa no bloqueado; 1, bloqueo compartido y; 2, bloqueo exclusivo.

TSeccion: clasifica los objetos de tipo sección, a continuación se detallan sus atributos.

- Codigo: código único en la base de datos del SIGERE y en la BDTR de la sección.
- CodigoSubsistema: contiene el identificador del subsistema, circuito, al que pertenece la sección.
- EstadoFaseA, EstadoFaseB y EstadoFaseC: se refiere a los estados en que está la sección en cada una de sus fases de forma independiente, cada atributo posee el valor de *'A'*, que significa sin energía o valor *'O'*, que significa energizada.

- Alimentador: contiene la dirección en memoria del interruptor, desconectivo, que alimenta la sección en el estado óptimo de la red, este es un atributo fundamental para la construcción de la red eléctrica.
- tipoSeccion: es un atributo que toma valor entero y que clasifica la sección según su tipo: 1, bus finito; 2, bus infinito; 3, generación; 4, tierra; 5, línea; 6, carga; 7, enlace y; 8, línea carga. Vale destacar que en las secciones de tipo 6 y 8 es donde se encuentran los clientes, también es necesario aclarar que no se modeló una estructura de herencia según el tipo de sección para ganar en rapidez de operación sobre las secciones con el ahorro de las conversiones entre tipos de datos (typeCast) a implementar en el código del STRE.
- Id_EA: contiene un entero que especifica la estructura administrativa, sucursal, a la que pertenece la sección.
- clientesFaseA, clientesFaseB, clientesFaseC y clientesTrifasicos: estos atributos contienen valores enteros grandes positivos que describen la cantidad de clientes por fase de la sección, así como los clientes trifásicos de la misma.
- cargaMinima, cargaMedia y cargaMaxima: contienen valores de tipo float que especifican cual es la carga de la sección en MegaWatt(MW) según el horario del día, la mínima se enmarca desde las 12 AM hasta las 6 AM, la media desde las 6 AM hasta las 6 PM y la máxima desde las 6 PM hasta las 11:59 PM.
- kmLinea: es la cantidad de kilómetros que mide la sección.
- Calibre: es un entero par positivo que describe el grosor de la sección, puede tomar valores desde 0 hasta 10, donde el menor valor corresponde al mayor grosor.
- AVI: contiene un valor entero positivo que especifica el intervalo de validez absoluta de la sección, este es uno de los atributos que se usa para garantizar la validez absoluta y relativa del dato, además que constituye la información fundamental para el muestreo a través de sensores de la sección, principalmente en secciones de línea utilizadas como barras.

- UltimaLectura: contiene un valor de tipo DateTime que especifica el instante de tiempo en que se realizó el último muestreo de la sección física, también es usado para garantizar la consistencia temporal del dato.
- EstadoBloqueo: se refiere a si el objeto está bloqueado por algún proceso o transacción, los valores que puede tomar son: 0, que significa no bloqueado; 1, bloqueo compartido y; 2, bloqueo exclusivo.
- La clase TLDesconectivos está preparada para cargar y salvar en la BD del SIGERE la lista de todos los desconectivos operables de la red eléctrica, hereda de la clase HashTable del Framework .Net 1.1 para garantizar el rápido acceso a los objetos, además de las demás operaciones a tener en cuenta, como son la inserción, la eliminación, algunas funciones de búsqueda por objeto y clave, propiedades que devuelven la lista de los valores o la lista de las claves, etc. También contiene un grupo de métodos definidos los cuales se consideran consultas sobre la lista de interruptores y sobre los cuales se ampliará en el epígrafe 3.5.

La clase TLSecciones está preparada para cargar y salvar en la BD del SIGERE la lista de todas las secciones que componen la red eléctrica y que enlazan los interruptores, hereda de la clase HashTable del Framework .Net 1.1 para garantizar el rápido acceso a los objetos, además de las demás operaciones a tener en cuenta, como son la inserción, la eliminación, algunas funciones de búsqueda por objeto y clave, propiedades que devuelven la lista de los valores o la lista de las claves, etc. También contiene un grupo de métodos definidos los cuales se consideran consultas sobre la lista de secciones y de los cuales se ampliará en el epígrafe 3.5.

La clase TRedTopologica que hereda de la clase MarchalByRefObject para contar con el mecanismo de distribución de sus instancias a través de la tecnología .Net Remoting, es la encargada de funcionar como controladora de la red eléctrica en sí, contiene dos atributos esenciales listaDesconectivos y listaSecciones los cuales permiten el acceso a todos los componentes de la red, la instancia que crea el STRE de esta clase permite:

- Cargar los elementos de la red eléctrica.

- Enlazar los elementos en forma de seudo grafo usando listas por el carácter poroso de la matriz de incidencia.
- Operar los interruptores que componen la red.
- Ejecutar métodos automáticos que se consideran triggers para el cálculo de variables y notificaciones a clientes conectados, así como actualizar el estado de la red que es un aspecto de importancia mayor para lograr la concurrencia en tiempo real de la misma.
- Adicionar y eliminar nuevos componentes eléctricos a la red como interruptores y secciones.
- Hacer salvas automáticas cada un período de tiempo establecido de los objetos de la red eléctrica que han cambiado de estado durante el período mencionado.

También es necesario llevar el control de las operaciones que realiza cada despachador sobre la red eléctrica, para ello se creó la siguiente estructura de clases que se muestra en el siguiente diagrama. Es necesario aclarar que estos objetos no requieren de consistencia temporal aunque sean distribuidos en la BDTR.

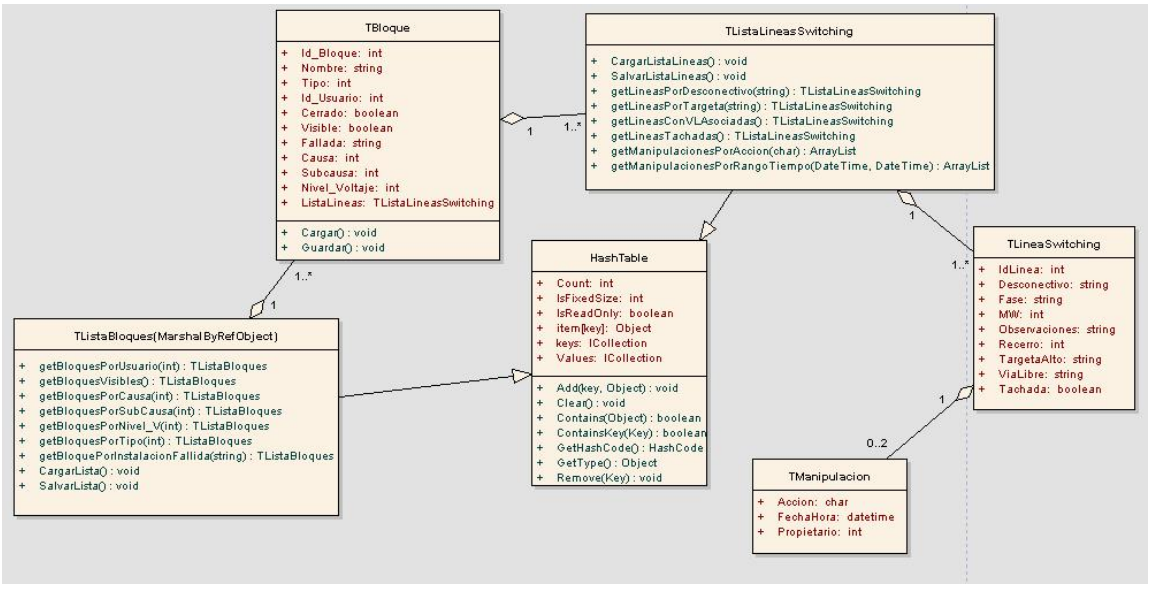


Figura 3.2.2 Clases que describen las operaciones sobre la red y la relación entre estas.

Un despachador divide sus trabajos en bloques de operación, cada bloque contiene varias líneas y las líneas cuentan con 0, 1 o 2 manipulaciones que en sí son las operaciones sobre un interruptor determinado.

La clase TManipulacion es una agregación de la clase TLineaSwitching y contiene el usuario que ejecutó la manipulación, la acción que realizó y en qué momento.

En la clase TLineaSwitching se encuentran un grupo de atributos distintivos, Desconectivo se refiere al interruptor que se le aplicaron las manipulaciones, en que fases, los MW que se operaron, cuantos recierres hubo, si hay vías libres asociadas y si la línea esta tachada o no.

La clase TListaLineasSwitching que hereda de HashTable para garantizar un buen rendimiento en las búsquedas sobre los elementos que la conforman, contiene un conjunto de métodos que garantizan cargar y salvar la lista de líneas del bloque a que pertenecen, además de contar con otros que se consideran consultas sobre las líneas y sus manipulaciones y que serán explicados más adelante.

La clase TBloque contiene atributos que describen al bloque, su nombre, de que tipo es, que usuario lo creo, si está cerrado y/o visible, la instalación fallada que dio origen a la creación del mismo, que causa, subcausa y nivel de voltaje se le asignan, además posee la lista de líneas que lo conforman.

La clase TListaBloques que hereda de MarchalByRefObject para contar con el mecanismo de distribución de sus instancias a través de la tecnología .Net Remoting y de HashTable para optimizar las búsquedas sobre los elementos de la lista, contiene el listado de todos los bloques en el STRE, posee métodos que permiten cargar los bloques que están físicos en la base de datos y que fueron guardados para reiniciar el STRE por alguna razón, además de contar con un grupo de métodos que permiten consultar los elementos de la lista por diferentes criterios de selección, en el diagrama se muestran los más importantes y usados que serán explicados en el epígrafe 3.5.

Cuando se realizan manipulaciones sobre los interruptores de la red eléctrica, en la mayoría de las ocasiones los clientes que reciben el servicio eléctrico son afectados, es de gran interés llevar el control de estas afectaciones y generar cálculos de otras variables derivadas, como es el caso particular del *tiu*, valor de importancia para la UNE.

El la figura 3.2.3 se muestran las clases usadas para llevar la información en relación a los cálculos que realiza el STRE cuando se manipulan interruptores de la red.

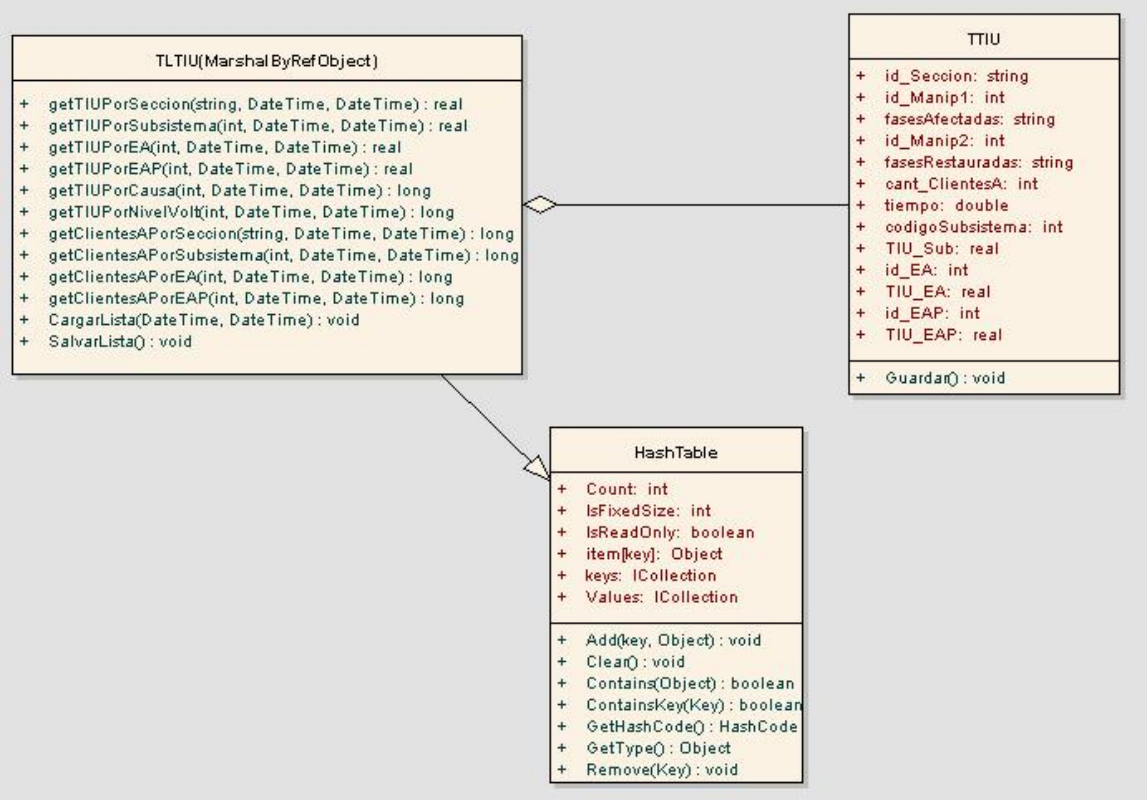


Figura 3.2.3 Clases utilizadas para almacenar los valores derivados en al BDTR.

Las instancias de la clase TIU contienen todos los valores calculados por el STRE que se requieren almacenar para posteriores análisis y consultas. A continuación se detallan sus atributos.

- id_Seccion: se refiere a la sección que quedó sin energía en algún momento del tiempo real y que posteriormente fue energizada.
- id_Manip1: representa el identificador de la manipulación que dejó a la sección sin energía.
- facesAfectadas: es un valor string de hasta tres caracteres que informa sobre las fases eléctricas de la sección que quedaron sin energía después de aplicarse la manipulación id_Manip1 sobre la red eléctrica.

- id_Manip2: representa el identificador de la manipulación que introdujo energía en la sección ya cuando esta estaba sin energía.
- fasesRestauradas: es un valor string de hasta tres caracteres que informa sobre las fases eléctricas de la sección que incorporaron energía después de aplicarse la manipulación id_Manip2 sobre la red eléctrica.
- cant_ClientesA: valor entero positivo que representa la cantidad de clientes afectados en el servicio eléctrico, los mismos se calculan después de aplicar la manipulación de cierre id_Manip2.
- tiempo: representa el tiempo de interrupción del servicio eléctrico medido en horas en que la sección estuvo sin energía.
- codigoSubsistema: es el identificador del subsistema, circuito, al que pertenece la sección afectada y por consiguiente los usuarios afectados.

El tiu, es el tiempo de interrupción por usuario, su cálculo depende de tres variables, *1)* los clientes a los que se les ha interrumpido el servicio eléctrico, *2)* la cantidad de clientes totales según el alcance del cálculo deseado, ya sea a nivel de sección, a nivel de circuito, a nivel de estructura administrativa o a nivel de estructura administrativa provincial y *3)* el tiempo que dichos usuarios estuvieron interrumpidos, la fórmula para obtener su valor es:

*tiu := cantidadClientesAfectados / cantidadClientesTotales * tiempoInterrupcion;*

Dada la anterior explicación se simplifica la comprensión de los atributos definidos a continuación.

- TIU_Sub: es un valor real que describe el tiu sobre el subsistema afectado, codigoSubsistema.
- id_EA: es el identificador de la estructura administrativa afectada, en este caso a nivel municipal.
- TIU_EA: es un valor real que describe el tiu sobre la estructura administrativa afectada, id_EA.
- id_EAP: es el identificador de la estructura administrativa provincial afectada.

- TIU_EAP: es un valor real que describe el tiu sobre la estructura administrativa provincial afectada, id_EAP, cada vez que se genera este valor es sumado al atributo *TIU* perteneciente a la instancia de la clase TRedTopologica, esta sumatoria se mantiene de forma anual.

La clase TLTIU que hereda de MarchalByRefObject para contar con el mecanismo de distribución de sus instancias a través de la tecnología .Net Remoting y de HashTable para optimizar las búsquedas sobre los elementos de la lista, contiene el listado de todos los objetos de tipo TTIU en el STRE, posee métodos que permiten salvar, así como cargar los objetos que están físicos en la base de datos en un intervalo de tiempo determinado y que fueron guardados en su momento, además de contar con un grupo de métodos que permiten consultar los elementos de la lista por diferentes criterios de selección, en el diagrama se muestran los más importantes y usados que serán explicados en el epígrafe 3.5.

3.3 Implementación de las transacciones y del control de la concurrencia

En este epígrafe se tratarán las transacciones para actualización y las transacciones del nivel de usuario, dejando pendiente las transacciones sobre sensores por su no implementación en esta etapa al no ser objetivo en la misma.

La BDTR del STRE no tiene en consideración transacciones de plazo crítico por la ausencia del reloj de tiempo, además, no se consideran de este tipo las transacciones de actualización o de nivel de usuario, a no ser aquellas que sean para telecomandar dispositivos de campo, lo cual en esta fase no se ha implementado por ausencia de comunicaciones en el entorno del STRE.

3.3.1 Descripción de métodos que se consideran transacciones en la BDTR

Muchos de los métodos que se ejecutan sobre la red eléctrica o sobre el esquema de la lista de bloques por parte de los objetos que componen la BDTR se consideran transacciones, por lo que es necesario se haga un tratamiento especial en cuanto al control de la concurrencia evitando los DeadLock. Estos pueden clasificarse en métodos: *a)* orientados al usuario o que colaboran con dichos métodos para la operación, *b)* triggers que se ejecutan dinámicamente, *c)* que se ejecutan de forma periódica y *d)* que son ejecutados en modo broadcast para las notificaciones.

A continuación describimos los métodos implementados en cada una de las clasificaciones anteriores.

a) Métodos orientados al usuario

- Desde la Clase TRedTopologica

    1) Manipular(CodigoDesconectivo: string; fases: int; oper: char): int, coloca en *oper*, (‘A’ o ‘O’), el interruptor con código CodigoDesconectivo en las fases especificadas como parámetro, devuelve 0 si la acción se ejecutó con éxito, se usa desde el módulo del Switching.
    2) Clonar: TRedTopologica, devuelve una copia exacta de la red eléctrica, se usa para hacer simulaciones desde el módulo Vias Libres.
    3) AdicionarDesconectivo(desc: TDesconectivo): int, adiciona el objeto desc a la red eléctrica, devuelve 0 si la acción culminó con éxito, se usa desde el módulo de Supervision.
    4) AdicionarSeccion(secc: TSeccion): int, adiciona el objeto secc a la red eléctrica, devuelve 0 si la acción culminó con éxito, se usa desde el módulo de supervisión.
    5) EliminarDesconectivo(desc: string): int, elimina de la red eléctrica el interruptor con código desc, se usa desde el módulo de Supervision.
    6) EliminarSeccion(secc: string): int, elimina de la red eléctrica la sección con código secc, se usa desde el módulo de Supervision.

- Desde la clase TLBloques, todos estos métodos se usan desde el módulo Switching

    1) Add(clave: Key; bloque: TBloque), adiciona un nuevo bloque a la lista.
    2) Remove(clave: Key), elimina un bloque de la lista.
    3) getBloquesPorUsuario(id_usuario: int): TListaBloques, devuelve los bloques que pertenecen al usuario id_usuario

También se consideran transacciones la mayoría de los métodos de las clases TLDesconectivos, TLSeciones, TLTIU, TListaBloques y TListaLineasSwitching que no

han sido explicados en este epígrafe y que se tratarán, por considerarse consultas, en el epígrafe 3.5

b) Métodos considerados triggers.

- Desde la Clase TRedTopologica

1) ActualizarRed(desc: string): TLSecciones, se le atribuye una importancia mayor, ya que se encarga de actualizar la red eléctrica a partir de la manipulación de los interruptores. Mediante un algoritmo de flujo de energía, refleja el ambiente real de la red, actualizando la energía en cada una de las fases de las secciones afectadas por la manipulación, devuelve el listado de secciones actualizadas, se usa como un disparador después de que algún objeto de la lista de interruptores cambie de estado operativo.

2) CalcularTIU: float, se dispara o ejecuta cuando al menos una sección de carga o línea carga de la lista de secciones sufre cambios en el estado energético de alguna de sus fases. Devuelve el valor del tiu provincial que se obtuvo en ese momento, además crea varios objetos del tipo TTIU en correspondencia con la cantidad de secciones afectadas y los adiciona a la lista del tipo TLTIU.

3) CalcularMWAfectados(desc: string): float, se ejecuta a partir de que el interruptor desc cambie de estado operativo, calculándose la carga dejada de servir o ingresada a la red eléctrica después de la operación del interruptor.

4) CalcularClientesSinServicio: long, se ejecuta cuando al menos una sección de carga o línea carga de la lista de secciones sufre cambios en el estado energético de alguna de sus fases. Devuelve la cantidad de clientes que han quedado sin servicio en toda la red en ese momento.

5) CalcularClientesTotales: long, se ejecuta bajo las mismas condiciones del trigger CalcularClientesSinServicio. Devuelve la cantidad de clientes totales de la red eléctrica.

c) Métodos se ejecutan de forma periódica

- Desde la Clase TRedTopologica

1) GuardarRed, guarda los elementos de la red eléctrica que han cambiado su estado después de la última ejecución del método. Se ejecuta sobre un hilo de ejecución en forma periódica cada 10 segundos, tomando en cuenta que en el caso que fallase el *hardware* donde corre el STRE, en este intervalo de tiempo no se pierden muchos cambios de estado de los elementos que componen la red según la observación realizada en el despacho provincial de la provincia Sancti Spíritus y que no es objetivo abordar en este informe.

- Desde la Clase TListaBloques

1) SalvarLista, guarda de la lista de bloques, los nuevos y los que han modificado alguno de sus atributos, líneas o manipulaciones después de la última ejecución del método. Se ejecuta sobre un hilo de ejecución en forma periódica cada 10 segundos, tomando en cuenta que en el caso que fallase el *hardware* donde corre el STRE, en este intervalo de tiempo no se pierden muchas operaciones realizadas por parte de los despachadores, según la observación realizada en el despacho provincial de la provincia Sancti Spíritus y que no es objetivo abordar en este informe.

d) Métodos ejecutados en modo *broadcast* para las notificaciones.

- Desde la Clase TRedTopologica

1) NotificacionClientesConectados(mw: float; tiu: float; cli: long; cliT: long), se ejecuta siempre que se creen o modifiquen objetos del tipo TTIU, funciona como un desencadenador que notifica sobre los MW afectados, el tiu calculado, el total de clientes que han quedado sin servicio y el total de clientes totales a todas las aplicaciones clientes, Switching, conectadas. Este método siempre es precedido por la ejecución del método Manipular de la clase TRedTopologica.

3.3.2 El manejador de transacciones en el STRE

El manejador de transacciones se ocupa de mantener la BDTR en un estado consistente aún cuando ocurran accesos concurrentes o fallas. Para ello se basa en una estructura de datos

en forma de lista que almacena las particularidades de cada transacción. En la figura 3.3.2.1 se muestran las clases que describen dicha estructura.

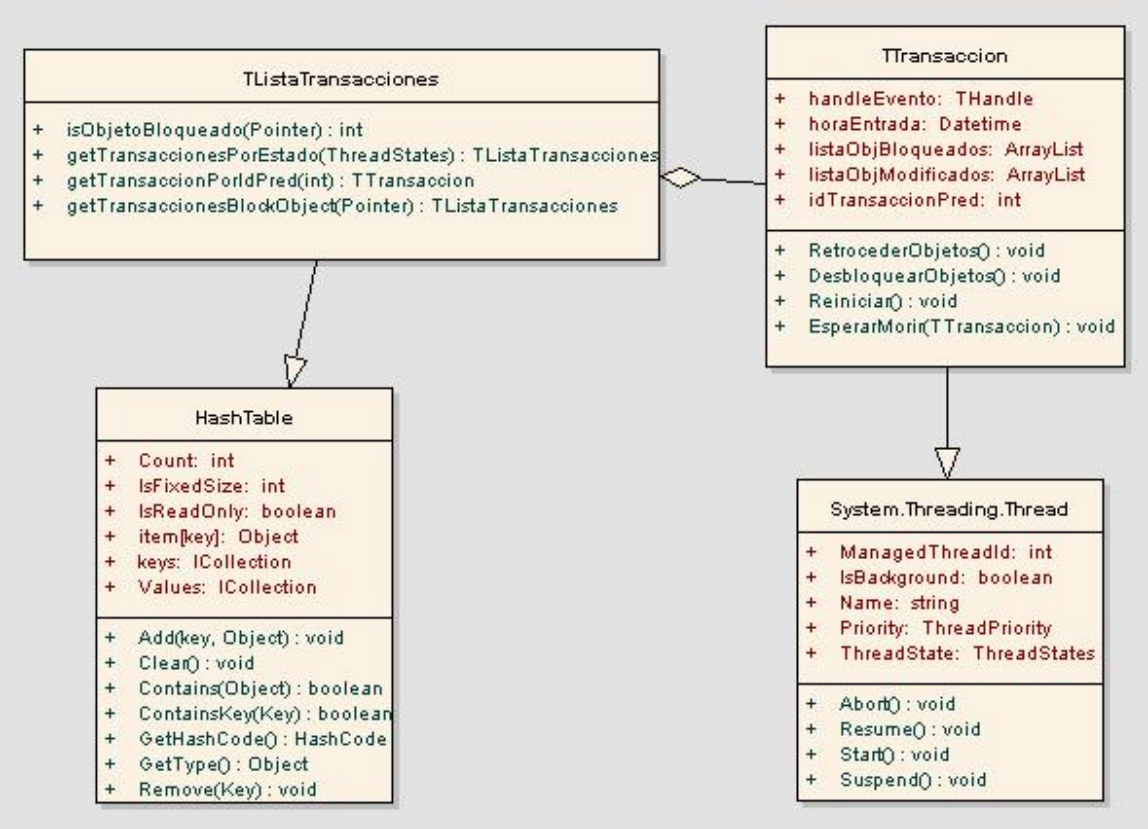


Figura 3.3.2.1 Estructura para el manejo de transacciones.

En el diagrama se observan las relaciones que existen entre las clases. Lo más significativo radica en que la clase TTransaccion hereda de la clase System.Threading.Thread la que ha sido descrita parcialmente por su amplitud, permitiendo la ejecución de las transacciones en hilos concurrentes.

La clase TListaTransacciones contiene todas las transacciones que se manejan en ese momento, hereda de la clase HashTable del Framework .Net 1.1 para garantizar el rápido acceso a los objetos. Sus métodos se describen a continuación.

- isObjetoBloqueado (obj: Pointer): int, esta función devuelve el estado de bloqueo en que se encuentra el objeto *obj*.

- getTransaccionesPorEstado (estado: ThreadStates): TListaTransacciones, devuelve todas las transacciones de la lista que se encuentra en el *estado* especificado como parámetro.
- getTransaccionPorIdPred (pred: int): TTransaccion, devuelve la transacción que posee en el campo idTansaccionPred el valor *pred.*
- getTransaccionesBlockObject(objeto: Pointer): TListaTransacciones, devuelve todas las transacciones que comparten un bloqueo compartido sobre el *objeto* especificado como parámetro.

La clase TTransaccion constituye el eje fundamental en el control de la concurrencia, implementando el mecanismo 2PL estricto y usando la técnica Esperar-Morir para evitar el DeadLock, es por ello que a continuación describimos los atributos y operaciones que la componen.

- handleEvento: almacena el manejador del evento que constituye una transacción y que se ejecuta en este hilo de ejecución.
- horaEntrada: especifica el instante de tiempo en que la transacción comenzó su ejecución, este tiempo se toma del reloj de la maquina donde este corriendo el STRE.
- listaObjBloqueados: contiene la lista de objetos que han sido bloqueados por esta transacción.
- listaObjModificados: es una lista que contiene una copia de los objetos antes de ser modificados por la transacción.
- idTransaccionPred: es el identificador de la transacción por la cual la transacción actual está esperando, en caso de no estar en ese estado este atributo toma el valor de -1;
- RetrocederObjetos, método que toma la lista listaObjModificados y restaura cada objeto en su estado original antes de ser modificado por la transacción. Se usa como un mecanismo de recobrado.

- DesbloquearObjetos, toma la lista listaObjBloqueados y desbloquea todos los objetos bloqueados por la transacción, se usa cuando se activa el mecanismo de recobrado, así como cuando la transacción termina su ejecución de forma exitosa.
- Reiniciar, reinicia la transacción, el siguiente código fuente muestra la implementación del método.

```
Begin
  Self.RetrocederObjetos;
  Self.DesbloquearObjetos;
  Self.Abort;
  Self.Start;
End;
```

- EsperarMorir(tranc : TTransaccion), este método implementa la regla seleccionada para evitar el *DeadLock*, se le pasa como parámetro la transacción que posee el objeto bloqueado sobre el cual la transacción actual está tratando de adquirir un bloqueo. El siguiente fragmento de código muestra el uso de la regla Esperar-Morir.

```
Begin
    If Self.horaEntrada < tranc.horaEntrada then
     Self.Suspend;
  Else
     Self.Reiniciar;
End;
```

Cada transacción puede pasar en su tiempo de vida por diferentes estados, la figura 3.3.2.2 muestra el diagrama de transición de estados de las transacciones.

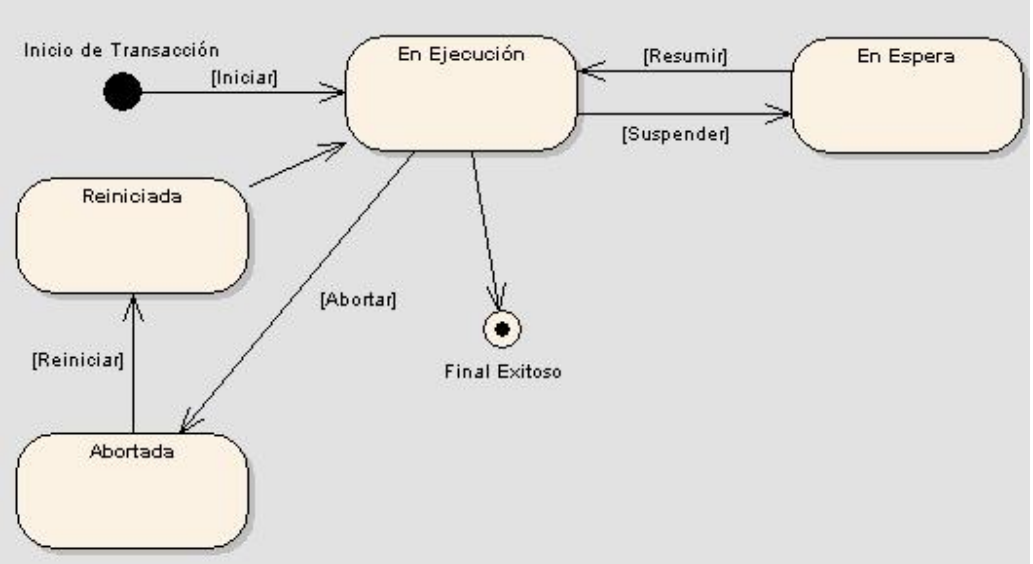


Figura 3.3.2.2 Diagrama de transición de estados de las transacciones en el manejador del STRE.

En los siguientes códigos fuentes se muestra como se alcanzan cada uno de los estados mostrados en la figura anterior.

Inicio de transacción: se crea la transacción y se enlaza con el método a ejecutar.

```
Var
  hilo : Thread;
  hiloDelegado : ThreadStart;
……..
Begin
  …….
  hiloDelegado := ThreadStart.create(self.handle);
  hilo := Thread.create(hiloDelegado);
  …….
End;
```

En ejecución: a) se toma la hora de inicio de ejecución de la transacción del sistema y se ordena a la transacción comenzar su ejecución, b) se restaura la transacción desde el punto en que se quedo.

a) Self.horaEntrada := DateTime.Now;
Self.Start;

b) Self.Resume

En espera: cuando a una transacción no se le concede un bloqueo y pasa la prueba Esperar-Morir queda en espera hasta que sea liberado este, para llevarla a este estado se le aplica la siguiente sentencia.

```
Self.Suspend;
```

Abortada: cuando a una transacción no se le concede un bloqueo y no pasa la prueba Esperar-Morir debe ser abortada y posteriormente reiniciada.

```
Self.Abort;
```

Reiniciada: después de que una transacción pasa al estado de abortada debe ser reiniciada manteniendo la misma hora de entrada que se le asignó al ser iniciada por primera vez.

```
Self.Reiniciar;
```

Final Exitoso: la transacción se ha completado de forma exitosa alcanzando el estado *Stopped.*

```
Self.DesbloquearObjetos;
Self.destroy;
listaTrans.Remove(Self.ManageThreadId);
```

Las transacciones periódicas son implementadas usando las clases System.Threading.Timer y System.Threading.TimerCallback, a continuación se muestra a través de un pequeño segmento de código como se implementaron las transacciones periódicas GuardarRed de la clase TRedTopologica y SalvarLista de la clase TListaBloques.

```
var
  salvaAutomRed : System.Threading.Timer;
  salvaAutomRedCallBack : System.Threading.TimerCallback;
```

```
  salvaAutomBloques : System.Threading.Timer;
  salvaAutomBloquesCallBack : System.Threading.TimerCallback;
Begin
   salvaAutomRedCallBack := Red.guardarRed;
   salvaAutomRed:= System.Threading.Timer.Create(salvaAutomRedCallBack,nil,0,36000000);
   salvaAutomBloquesCallBack := listaBloques.SalvarLista;
   salvaAutomBloques:= System.Threading.Timer.Create(salvaAutomBloquesCallBack,nil,0,36000000);
End;
```

3.3.2 Implementación de bloqueos basándose en 2PL estricto.

Como ya se ha explicado, para garantizar la consistencia de los datos en la BDTR se hace necesario bloquear de forma compartida o exclusiva los objetos por parte de las transacciones. Para lograr este propósito se implementó la clase TBloqueo, los métodos fundamentales de la misma se describen a continuación mediante una porción de su código fuente.

Métodos:

- EstadoBloqueo(objeto : pointer) : int, devuelve el estado de bloqueo del *objeto* pasado como parámetro.

```
If  objeto is TSeccion then
    Result := TSeccion(objeto).EstadoBloqueo
Else
If objeto is TDesconectivo then
    Result := TDesconectivo(objeto).EstadoBloqueo;
```

- Bloquear(objeto: pointer; estado : integer) : boolean, este método devuelve *true* si pudo ejecutar un bloqueo del tipo *estado* sobre el *objeto* sino devuelve *false*. También es usado para desbloquear usando *estado = 0*.

```
If (EstadoBloqueo(objeto) = 0) or (estado = 0)
   or ((EstadoBloqueo(objeto) = estado) and estado = 1)  then
 begin
    If objeto is TSeccion then
      begin
        TSeccion(objeto).EstadoBloqueo := estado
        Result := true;
```

```
        end
      else
      If objeto is TDesconectivo then
        begin
          TDesconectivo(objeto).EstadoBloqueo := estado
          Result := true;
        end
    end
  else
  result := false;
```

- UpgradeBloqueo(objeto: pointer), convierte el estado de bloqueo del *objeto* que se pasa como parámetro de compartido a exclusivo.

```
  If listaTransaciones .getTransaccionesBlockObject(objeto).count = 1 then
   If objeto is TSeccion then
      TSeccion(objeto).EstadoBloqueo := 2
   else
     TDesconectivo(objeto).EstadoBloqueo := 2;
```

- DowngradeBloqueo(objeto: pointer), convierte el estado de bloqueo del *objeto* que se pasa como parámetro de exclusivo a compartido.

```
  If objeto is TSeccion then
     TSeccion(objeto).EstadoBloqueo := 1
  else
    TDesconectivo(objeto).EstadoBloqueo := 1;
```

3.4 Implementación de desencadenadores (Triggers)

La implementación de triggers se considera un factor muy importante a tener en cuenta en las BDTR, las facilidades que brinda el uso de los mismos para generar nuevos elementos en la BDTR, notificar a las aplicaciones conectadas, así como actuar sobre los elementos de campo, son de gran importancia para obtener las funcionalidades deseadas.

Por la ausencia de comunicaciones, el STRE se ve imposibilitado a poder telecomandar los elementos que están en el campo, es por eso que en este epígrafe solo se hace referencia a

los triggers implementados para calcular nuevos valores y publicarlos en la BDTR y para las notificaciones automáticas de cambios en algunas variables a las aplicaciones clientes.

A continuación se ejemplifica a través de código fuente como se han implementado los triggers en la BDTR.

Cuando se ejecuta el método Manipular por intermedio de alguna transacción es necesario actualizar el estado en que han quedado algunos componentes de la red eléctrica, con este propósito se implementó el trigger ActualizarRed de la clase TRedTopologica, el cual se ejecuta bajo las condiciones que se muestra en el siguiente fragmento de código fuente.

```
auxManip := Red.Manipular(auxDesc,fases,operacion);
If auxManip = 0 then
  listaSeccionesAfectadas := Red.ActualizarRed(auxDesc) ;
```

Al ejecutar el trigger ActuializarRed de la clase TRedTopologica puede que se modifique el estado de las fases eléctricas de un grupo de secciones, por lo que se hace necesario derivar nuevos valores y publicarlos en la BDTR como son, los MW introducidos o retirados por la actualización, los clientes que han quedado sin servicio, los clientes totales de la red y los valores de tiu, en todos los casos se notifica a las aplicaciones clientes Switching de los valores derivados. Para lograr estas funcionalidades se implementaron los triggers, CalcularMWAfectados, CalcularClientesSinServicio, CalcularClientesTotales, CalcularTIU y NotificacionClientesConectados, todos de la clase TRedTopologica y que en el siguiente segmento de código se muestra las condiciones en que se ejecutan los mismos.

```
mw := -1;
clientesSS := -1;
clientesT := -1;
tiu := -1;

if listaSeccionesAfectadas.Count > 0 then
  mw := calcularMWAfectados(auxDesc);

if (listaSeccionesAfectadas.getSeccPorTipo(6).Count > 0)
 or (listaSeccionesAfectadas.getSeccPorTipo(8).Count > 0)  then
  begin
    clientesSS :=  CalcularClientesSinServicio;
```

```
    clientesT := CalcularClientesTotales;
    tiu := CalcularTIU;
end;

If mw + clientesSS + clientesT + tiu <> -4 then
 NotificacionClientesConectados(mw, tiu, clientesSS, clientesT );
```

3.5 Implementación de consultas a la BDTR.

Las consultas en una BDTR no son más que librerías de funciones que filtran las colecciones de objetos por medio de condiciones sobre los atributos contenidos en los mismos. A continuación se describen los principales métodos implementados desde las diferentes clases que componen el modelo de datos de la BDTR.

- Desde TLDesconectivos

    1) GetDescPorEstadoOperativo (estOp: char): TLDesconectivos, filtra los desconectivos de la lista que se encuentran en el estado operativo *estOp* en alguna de sus fases de operación (*EstadoOperativoA, EstadoOperativoB, EstadoOperativoC*).

    2) GetDescPorSubsistema(subSistema:int): TLDesconectivos, filtra los desconectivos de la lista que pertenecen al *subSistema* especificado como parámetro.

    3) GetDescPorSucursal (ea: int): TLDesconectivos, filtra los desconectivos de la lista que pertenecen a la estructura administrativa *ea* especificada como parámetro.

- Desde TLSecciones

    1) getClientesPorSucursal (sur: int): long, devuelve los clientes que pertenecen a la estructura adminitrativa *sur*.

    2) getSeccPorEstadoYFace (fase: int, estado: char): TLSecciones, filtra las secciones por estado y fase, devolviendo las secciones que se encuentran en este *estado* en las fases *fase*.

3) getSeccPorSubsistema (subsistema: int): TLSecciones, filtra las secciones que pertenecen al *subsistema* especificado como parámetro.

4) getSeccPorTipo (tipo: int): TLSecciones, filtra las secciones por su tipo, esta consulta es muy usada por los triggers que calculan tiu y clientes sin servicio ya que estos solo se disparan si son afectadas secciones del tipo 6 o 8.

- Desde TListaLineasSwitching

1) getLineasPorDesconectivo (desc: string): TListaLineasSwitching, filtra las líneas donde se manipuló el desconectivo *desc.*

2) getLineasPorTargeta (tarjeta: string): TListaLineasSwitching, filtra las líneas donde se ha ubicado una *tarjeta* de alta, 'TA' o de baja, 'TB'.

3) getLineasConVLAsociadas: TListaLineasSwitching, devuelve las líneas en las cuales se han asociado permisos de trabajo, este aspecto es muy importante en la revisión de las operaciones realizadas por el despachador.

4) getLineasTachadas: TListaLineasSwitching, cuando los despachadores cometen errores en el completamiento de las líneas, tienen la opción de tachar estas, esta consulta brinda dichas líneas filtradas lo que se usa en la revisión de las operaciones del despachador.

5) getManipulacionesPorAccion (accion: char): ArrayList, filtra todas las manipulaciones según la *accion* contenida en estas. Devuelve una lista donde cada objeto es del tipo TManipulacion.

- Desde la clase TListaBloques

1) getBloquesPorUsuario (usuario: int): TListaBloques, devuelve el listado de bloques que pertenecen al *usuario.*

2) getBloquesPorCausa (causa: int): TListaBloques, devuelve los bloques que han sido cerrados indicando una *causa* especifica, se usa en el análisis del funcionamiento de líneas eléctricas.

3) getBloquePorInstalacionFallida (instalacion: string): TListaBloques, filtra los bloques que han sido creados por una falla en la *instalacion* especificada como

parámetro, se usa en el análisis de fallas de instalaciones por el módulo de Supervicion.

- Desde la clase TLTIU

  1) getTIUPorSeccion (idsecc: string, fecha1: DateTime, fecha2: DateTime): real, devuelve el valor del tiu de la sección *idsecc* en el rango desde *fecha1* hasta *fecha2*.

  2) getTIUPorSubsistema (sub: int, fecha1: DateTime, fecha2: DateTime): real, análogo al anterior pero el valor de tiu se calcula por subsistema.

  3) getTIUPorEA (ea: int, fecha1: DateTime, fecha2: DateTime): real, análogo al anterior pero el valor de tiu se calcula por estructura administrativa, *ea.*

  4) getTIUPorCausa (causa: int, fecha1: DateTime, fecha2: DateTime): long, devuelve el valor del tiu según la *causa* especificada como parámetro en el rango desde *fecha1* hasta *fecha2*.

  5) getClientesAPorSubsistema (sub: int, fecha1: DateTime, fecha2: DateTime): long, devuelve la cantidad de clientes afectados en el subsistema *sub* en el rango desde *fecha1* hasta *fecha2*.

- Desde la clase TListaTransacciones

  1) getTransaccionesPorEstado (estado: ThreadStates): TListaTransacciones, filtra las transaciones que se encuentran en el estado especificado como parámetro.

  2) getTransaccionesBlockObject (objeto: Pointer): TListaTransacciones, devuelve un listado de las transacciones que poseen un bloqueo compartido sobre el *objeto* que se especifica como parámetro.

El esquema general usado en la implementación de las funciones anteriormente descritas es el siguiente:

- se crea una variable auxiliar que será el resultado de la consulta, en la mayoría de los casos, una lista del tipo a devolver.
- se define un ciclo que recorre la lista completa a analizar

- si el elemento i-ésimo de la lista cumple con las condiciones de igualdad en relación con los parámetros de la consulta en el caso de que estos existan, se adiciona a la variable auxiliar que será devuelta como parámetro.
- se devuelve la variable auxiliar como resultado de la consulta.

Existen esquemas más atractivos para la implementación de consultas, estos consideran técnicas de manejo de índices similar a las base de datos tradicionales, por su complejidad, y por no ser necesario en esta etapa de implementación de la BDTR, no han sido consideradas.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Con el desarrollo del presente trabajo se ha dado cumplimiento a los objetivos propuestos en el mismo, lo cual se ve reflejado en las siguientes conclusiones:

1 Con la elaboración de varios diagramas de clases, para describir las particularidades de los atributos de las clases y las relaciones que existen entre las mismas, se modeló la Base de Datos en Tiempo Real del Servidor de Topología de la Red Eléctrrica

2 Se describió, a) el protocolo de bloqueo en dos fases (2PL) estricto y b) las técnicas para evitar deadlock, Esperar-Morir y Herir-Esperar. Implementándose a) y de b) la técnica Esperar-Morir en la Base de Datos en Tiempo Real para controlar la concurrencia y los conflictos entre las transacciones.

3 Se obtuvo, una versión del Servidor de Topología de la Red Eléctrica que cuenta con una BDTR donde están los componentes eléctricos de la red distribuidos usando la tecnología .Net Remoting, que permite la comunicación e interacción con las aplicaciones clientes en una mejor operación de la red eléctrica por parte de los despachadores.

Recomendaciones

1 Implantar el STRE en el despacho provincial de Holguín, que funciona como polígono de pruebas del SIGERE, para su posterior generalización a los restantes despachos del país.

2 Desarrollar una versión mejorada del STRE que incorpore transacciones de plazo crítico en la BDTR, así como el control, supervisión y monitoreo de los interruptores físicos que se encuentran en el terreno.

3 Realizar mejoras en la implementación del mecanismo de consultas teniendo en cuenta la utilización de índices sobre las listas que componen la BDTR.

BIBLIOGRAFÍA

Adalberg, B. (2002). *Appliying Update Streams in a Soft Real Time Database System.*

Anderson, J. H. (2004). *Lock-Free Transactions for Real Time Systems.*

Banerjee, A. (2003). *Building Web Services with .Net Remoting and ASP.Net.*

Bestavros, A. (2007). *Value-cognizant Admission Control Strategies for Real Time DBMS.*

Brohede, M. (2000). *Using Distributed Active Real Time Database Funtionality in Information-Fusion Infrastructures.*

Burns, A. (1998). *A Worst Case Behaviour Analysis for Hard Real Time Transactions.*

Carlson, J. (2003). *Event-Pattern Triggered Real Time Tasks.*

Carlson, J. (1999). *Structural Locking Mechanisms and Their Effect on Real Time Database System Performance.*

Casalvilla, J. B. (2003). *Instrucción para el cargo de los despachadores de distribución.*

Dayal, U. (1988). *Combining Active Database and Timing Constraints.*

Días, j. (2003). *Instrucción para la confección del modelo de orden de Vía Libre.*

Espinosa, L. R. (2003). *Instrucción para la confección del modelo diario de operación.*

Falkenroth, E. (1997). *Database in Control and Simulation.*

Fernández, R. (2007). *Visión del Sistema de Gestión de Redes (SIGERE).*

Franaszed, P. A. (1990). *Access Invariance ant its Use in High Contention Environments .*

González, L. G. (2005). *Consurrencia y Recobrado, UCLV.*

Hansson, J. (2002). *DeeDS Towards a Distributed and Active Real Time Database System.*

Haritsa, J. R. (2007). *Real Time Database System.*

Huang, H. (1989). *Esperimental Evaluation of Real Time Transaction Processing .*

Kang, K.-D. (2006). *An Approach for Real Time Database Modeling and Performance Management.*

Kao, B. (2007). *Maintaining Temporal Consistency of Discrete Objects in Soft Real Time Database Systems.*

Korth, H. F. (1990). *Triggered Real Time Databases with Consistency Contraints.*

Lam, K. Y. (2005). *Evaluation of Concurrencia Control Strategies for Mixed Soft Real Time Database Systems.*

Lam, K. Y. (2005). *Strategies for Resolving Inter-Class Data Conflicts in Mixed Real Time Database Systems.*

McCarthy, T. P. (2005). *Performance-Polymorphic Execution of Real Time Queries.*

Mellin, J. (2008). *Refining Design Constraints using a System Services Model of a Real Time DBMS.*

Mendoza, L. A. (2006). *Diseño de un Manejador de Bases de Datos para un Sistema Operativo de Tiempo Real.*

Nogiec, J. M. (2003). *RTDB: A Memory Resident Real-Time Object Database.*

Obemeyer, P. (2001). *Microsoft .NET Remoting: A Technical Overview.*

O'Neil, P. (1992). *Towards Predictable Transaction Executions in Real Time Database Systems.*

O'Neil, P. (2002). *Towards Predictable Transaction Executions in Real Time Database Systems.*

Peng, C. S. (2000). *Real Time Database Benchmark Design for Avionics Systems.*

Pérez, P. (2003). *Instrucción para el control del estado físico de los alimentadores de distribución.*

Puente, J. A. (2003). *Apuntes del Doctorado en Sistemas de Tiempo Real.*

Rajkumar, R. (1988). *Concurrency Control for Distrubuted Real Time Database.*

Ramamritham, K. (1990). *Efficient Scheduling Algorithms for Real-Time Multiprocesor Systems.*

Ramamrithan, K. (1996). *Real Time Database.*

Ramamrithan, K. (2008). *Real Time Database Systems in the New Millenium.*

Rammer, I. (2002). *Advanced .Net Remoting.*

Rodríguez, L. E. (2003). *Instrucción para la confección del modelo manipulación de interruptores.*

Squadrito, M. (1999). *Towards Priority Ceilings in Object-Based Semantic Real Time Concurrency Control.*

Srinivasan, P. (2001). *An Introduction to Microsoft .NET Remoting FrameWork .*

Thomas, S. (2007). *Integrating Standard Transactions in Firm Real Time Database System.*

Towsley, D. (2006). *Maintaining Temporal Consistency: Issues and Algorithms.*

Vieira, M. (2000). *Towards Timely ACID Transactions in DBMS.*

Vinay, V. (2000). *Scheduling Algorithms for the Broadcast Delivery of Multimedia Products.*

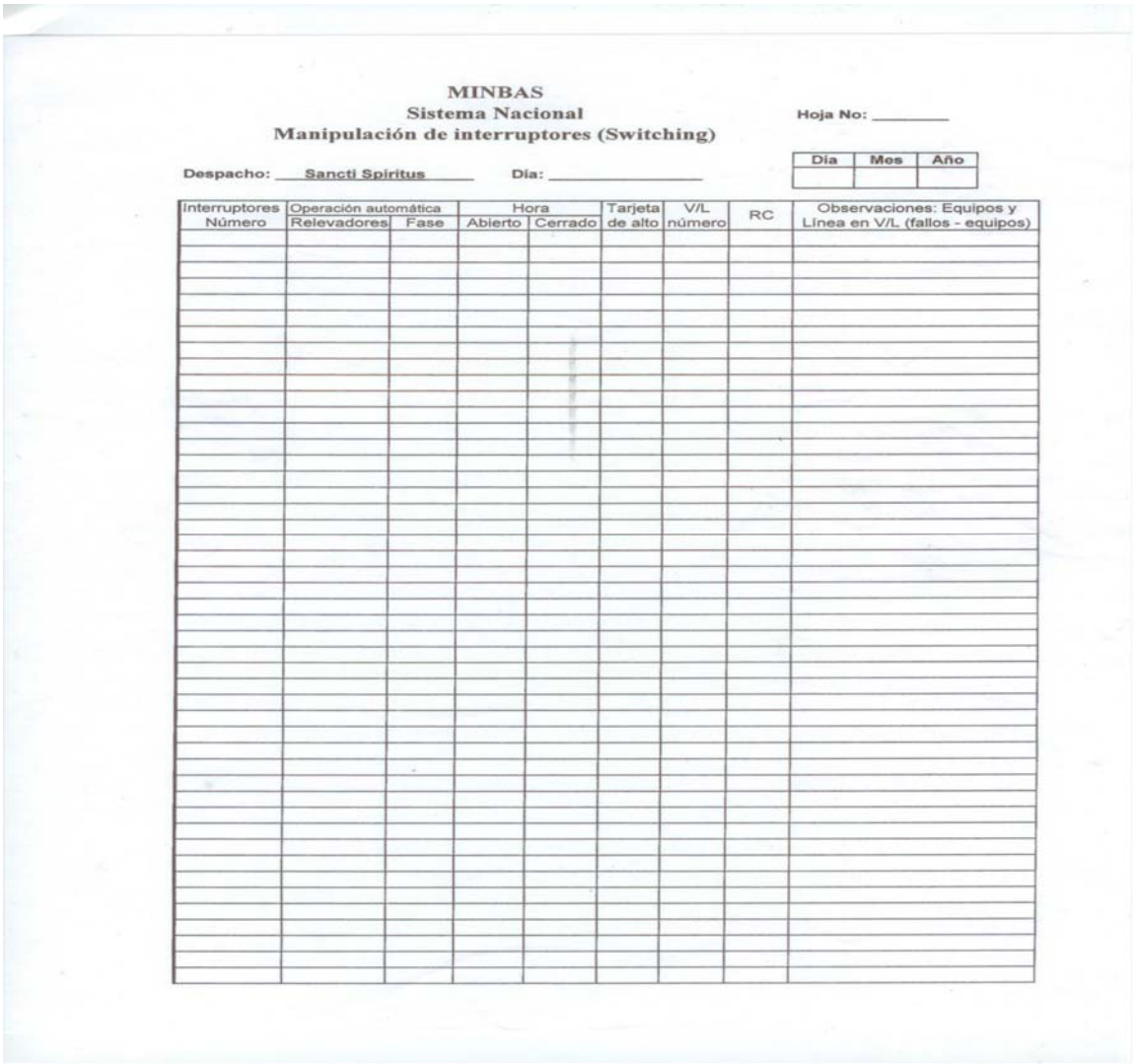
Xiong, M. (2009). *Mirror: A State-Conscious Concurrency Control Protocol for Replicated Real Time Databases.*

Xiong, M. (2008). *Scheduling Access to Temporal Data in Real Time Database.*

Xiong, M. (2009). *Towards the Specification and Analysis of Transactions in Real Time Active Database.*

ANEXOS

Anexo I Planilla Switching



MINBAS
Sistema Nacional
Manipulación de interruptores (Switching)

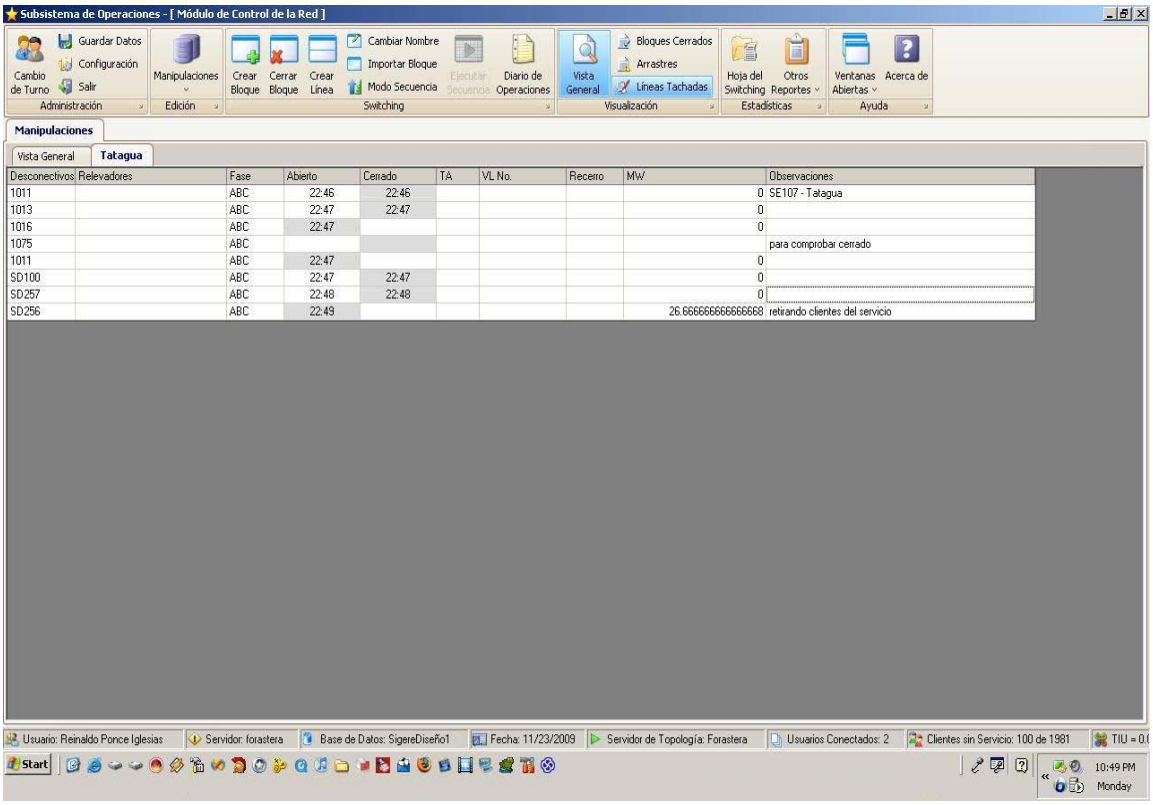
Hoja No: _____

Despacho: Sancti Spiritus Día: _____

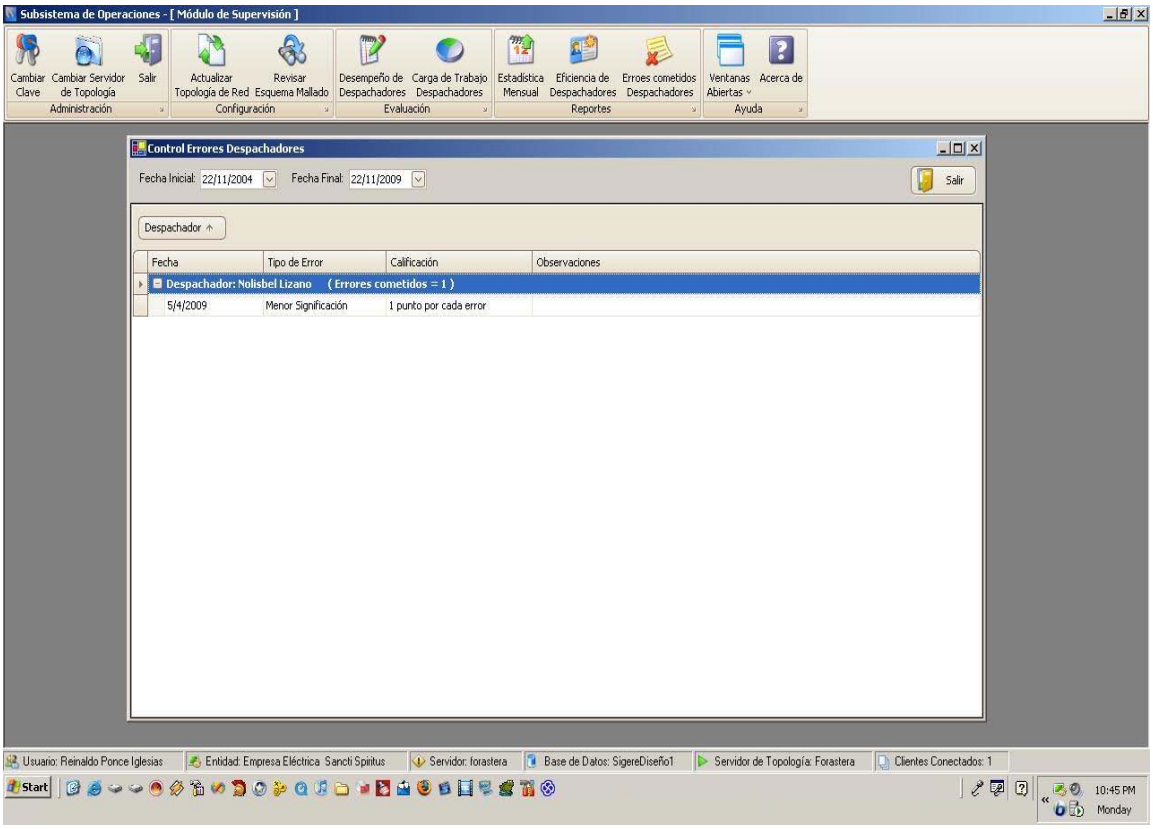
Día	Mes	Año

Interruptores Número	Operación automática Relevadores	Fase	Hora Abierto	Cerrado	Tarjeta de alto	V/L número	RC	Observaciones: Equipos y Línea en V/L (fallos - equipos)

Anexo II Vista gráfica del módulo de control de la red, Switching.



Anexo III Vista gráfica del módulo Supervision.



Anexo IV Contenido del fichero de configuración del servidor, Server.config

```
<configuration>
  <system.runtime.remoting>
    <application name = "DllSwitching">
        <lifetime leaseTime="365D" sponsorshipTimeOut="365D" renewOnCallTime = "365D" />
        <service>
            <wellknown type="UTSwitching.TRedTopologica, DllSwitching" objectUri="Topologia"
                                                    mode="Singleton" />
        </service>
        <channels>
          <channel ref="tcp" port="9999">
          <serverProviders>
              <formatter ref="binary" typeFilterLevel="Full" />
          </serverProviders>
          <clientProviders>
             <formatter ref="binary" />
          </clientProviders>
        </channel>
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

Anexo V Contenido del fichero de configuración del cliente, Client.config

```
<configuration>
  <system.runtime.remoting>
    <application name="Server">
      <lifetime leaseTime="365D" sponsorshipTimeOut="365D" renewOnCallTime="365D" />
      <client url="tcp://Forastera:9999/Topologia">
        <wellknown type="UTSwitching.TSwitching, DllSwitching" url="tcp://Forastera:9999/Topologia" />
      </client>
      <channels>
       <channel ref="tcp" port="0" clientConnectionLimit="20">
          <clientProviders>
             <formatter ref="binary" />
          </clientProviders>
          <serverProviders>
              <formatter ref="binary" typeFilterLevel="Full" />
          </serverProviders>
        </channel>
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```