UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

"ARAGON"

PROTOTIPO DE UN MECANISMO DE COMUNICACION PARA UN SISTEMA DE CONTROL DISTRIBUIDO DE UNA CENTRAL TERMO ELECTRICA

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

GONZALO MARTINEZ HURTADO

ENEP

ARAGON

SAN JUAN DE ARAGON, EDO. DE MEXICO 1995

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Dedicatoria

Este trabajo de tesis se lo dedico a mis Padres,

que me dieron la vida y de quienes siempre he

recibido el amor, apoyo y la orientación necesaria

para realizar todas mis metas.

Agradecimientos

A la Universidad Nacional Autónoma de México y a
la Escuela Nacional de Estudios Profesionales Aragón,
por mi formación profesional.

Al Instituto de Investigaciones Eléctricas y a la
Unidad de Resultados de Automatización de Procesos,
por la beca otorgada y los recursos brindados
para la realización de este trabajo.

Al M. en C. Enrique Ruiz, por su asesoría y dedicación
durante todo el desarrollo de la tesis.

Al M. en C. Luis García, por sus valiosos comentarios.

A la Ing. Silvia Vega, por su asesoría y orientación
durante la realización de esta tesis.

CONTENIDO

CAPÍTULO 2: MECANISMOS DE COMUNICACIÓN EN SISTEMAS DE CONTROL DISTRIBUIDO.

CAPÍTULO 3: ESPECIFICACIÓN DE UN MECANISMO DE COMUNICACIÓN CON TOLERANCIA A FALLAS PARA UN SCD DE UNA CTE.

LISTA DE FIGURAS

INTRODUCCIÓN.

a) Antecedentes.

La Comisión Federal de Electricidad (CFE) enfrenta el reto de satisfacer la creciente demanda de energía eléctrica en el país, con el compromiso de mantener niveles altos de rentabilidad, puntualidad y calidad en el sector eléctrico. Por esta razón, se ha requerido implantar nuevas estrategias orientadas a hacer más eficiente la generación, transmisión y distribución de la energía eléctrica.

Una de estas estrategias consistió en el desarrollo e implantación de sistemas automatizados que permitieron el mejoramiento de la operación de las centrales generadoras de electricidad.

Con el propósito de automatizar las centrales de generación eléctrica de la CFE, el Instituto de Investigaciones Eléctricas (IIE) desarrolló e implementó los Sistemas de Adquisición de Datos y Registro de Eventos (SADRES), que tienen como objetivo principal el facilitar y hacer más eficiente la puesta en servicio, operación y mantenimiento de las Centrales Termoeléctricas (CTE's) apoyando, así, a mejorar la confiabilidad y disponibilidad de estas centrales generadoras de energía.

Estos SADRES se encuentran instalados y en operación en la Central Termoeléctrica "Manzanillo II" en Manzanillo, Colima y en las Centrales Térmoelectricas de Tuxpan, Lerdo, Rosarito y Petacalco.

En la Unidad de Resultados de Automatización de Procesos (URAP), del IIE, se cuenta con proyectos cuyas metas son las de mejorar la operación de las CTE's a través de la modernización de los SADRES. Con este propósito, se ha identificado la conveniencia de desarrollar nuevas arquitecturas para los SADRES. Entre estas nuevas arquitecturas se encuentran las aplicadas para los **Sistemas de Control Distribuido (SCD's)**, que permiten explotar las ventajas de las nuevas tecnologías y cumplen con los requerimientos de confiabilidad y disponibilidad.

Un Sistema de Control Distribuido (SCD) es aquél en que las funciones de procesamiento de datos están repartidas entre varios procesos o elementos de cómputo separados físicamente. Cada proceso o elemento tiene capacidad de procesamiento, ya que cuenta con todos los recursos necesarios, que pueden ser propios o compartidos por medio de una red de comunicaciones.

Los elementos que conforman a un SCD son unidades de adquisición de información, de procesamiento, de presentación de información y de control.

Un elemento esencial de un SCD es su sistema de comunicaciones. El **sistema de comunicaciones** es el responsable de transportar mensajes del sistema y de los programas de aplicación entre estaciones o procesos, es decir, acepta mensajes de los procesos de una estación, para transmitirlos por un medio compartido y los distribuye entre los diferentes procesos que componen la red del sistema.

Una **red** puede verse como una colección de nodos interconectados, que realizan un intercambio de unidades de información (mensajes) entre ellos. Este intercambio de datos requiere que cada nodo o proceso comunicante se someta a ciertas reglas preestablecidas, para lograr un intercambio de información eficiente y confiable.

Un **mecanismo de comunicación** establece o define las reglas a seguir cuando se efectúa una comunicación entre elementos de la red, para que el intercambio de información entre los diversos elementos se haga de una manera ordenada y eficaz. Estos mecanismos de comunicación se conocen comúnmente como protocolos.

b) Objetivo de la Tesis.

La Unidad de Resultados de Automatización de Procesos, del **IIE**, ha identificado, como parte del desarrollo de Sistemas de Control Distribuido para Centrales Termoeléctricas, la necesidad de desarrollar un prototipo de mecanismo de comunicación que sea capaz de transferir información, de una forma transparente y eficiente, entre los nodos que conformen a un **SCD** y que permita cumplir con requerimientos de confiabilidad y disponibilidad de información.

En este trabajo de tesis se tiene el objetivo de especificar, diseñar, implantar e integrar un prototipo de mecanismo de comunicación al nivel de la capa de aplicación, de acuerdo con el modelo **OSI** para especificación de protocolos. El prototipo de mecanismo de comunicaciones a desarrollar se compondrá de dos elementos importantes:

1) Un protocolo de operación, que cumplirá con requisitos de flujo de información entre nodos de una red.

2) Un protocolo de respaldo, que servirá para implantar mecanismos tolerantes a fallas para satisfacer el requisito de disponibilidad en la transferencia de información.

c) Alcance de la Tesis.

El mecanismo de comunicación, por desarrollar en esta Tesis, tendrá la función de coordinar que la información obtenida en los procesos de adquisición de un **SCD** de una **CTE** y almacenada en estructuras en memoria, sea difundida, de una manera ordenada y segura, hacia todos aquéllos procesos o estaciones de trabajo donde se requiera de dicha información, ya sea para la presentación de la información o para el control de algún elemento del **SCD**.

La función de transferencia de información no se limitará a la transferencia de estructuras en memoria, sino que también se podrá transmitir otro tipo de información, como mensajes, comandos y archivos; todo esto con el propósito de integrar un servicio completo de comunicación entre nodos de un **SCD**.

El mecanismo de comunicación deberá contar, además, con un esquema tolerante a fallas a fin de que las funciones de transmisión y recepción de información, implantadas en cada nodo del SCD, contengan otra versión ubicada en otro nodo como respaldo y permitir que la transmisión y recepción de información no se vea afectada aún con la falla de algún nodo del SCD.

Dado que el desarrollo de prototipos se utiliza como fase inicial para el desarrollo de sistemas de calidad, una vez finalizado el prototipo se podrán detectar requerimientos adicionales del mecanismo de comunicación y se podrán identificar y redefinir los servicios que así lo requieran. Después de esto, el prototipo podrá evolucionar hasta el sistema final, que podrá ser implantado en cualquier arquitectura distribuida donde las aplicaciones requieran de servicios de transferencia de información entre nodos remotos, como es el caso de los proyectos de desarrollo e implantación de SCD's en las Centrales Termoeléctricas de la CFE.

d) Plataforma de desarrollo.

La plataforma software utilizada para el desarrollo del prototipo, se basará en el sistema operativo en Tiempo Real QNX de Quantum y en el lenguaje de programación "C", con el compilador de Watcom C, que es una implementación del lenguaje C para el S.O. QNX.

Este proyecto se desarrollará en una arquitectura distribuida compuesta de cinco PC's 386 enlazadas por la red ARCNET, caracterizada por tener una topología bus y un mecanismo de acceso al medio "Token Passing Bus", además de una velocidad de transmisión de 2.5 MBps.

e) Organización de la Tesis.

Esta tesis está organizada de la siguiente manera:

En el capítulo 1 se describen las características principales que conforman a un SCD, así como sus funciones, niveles y la arquitectura con la que se implantó en una Central Termoeléctrica.

En el capítulo 2 se definen conceptos generales relacionados con los mecanismos de comunicación. También se describe el modelo de referencia OSI (Open System Interconnection) de la ISO (International Standards Organization), para sistemas de comunicación y se describen las fases de desarrollo de mecanismos de comunicación (definición de requerimientos, especificación, diseño e implantación). En este mismo capítulo, se identifican y definen claramente todos los requerimientos de comunicación del SCD, que deberá satisfacer el mecanismo de comunicación por desarrollar en esta Tesis.

En el capítulo 3, una vez identificados los requerimientos, se realiza la especificación del mecanismo de comunicación. En esta especificación se incluye, entre otras cosas, el planteamiento del problema que se quiere resolver de una manera clara y breve, realizando un análisis detallado de los requerimientos del sistema a desarrollar.

En este capítulo, adicionalmente, se describen todos los servicios con que se cuenta para desarrollar el mecanismo y que son proporcionados por el S.O. QNX y la red ARCNET. Se definen los servicios que proporcionará el mecanismo de comunicación a los usuarios del sistema. Se hace una breve descripción del medio ambiente donde se encuentra el mecanismo, es decir, el S.O. QNX. Por último, se identifican y describen los distintos mensajes, y sus formatos, que el mecanismo utilizará para realizar sus servicios de transferencia de información.

En el capítulo 4 se describe la herramienta MEF (Máquina de Estado Finito) utilizada para el diseño de protocolos y se realiza la especificación de las reglas de procedimiento del mecanismo de comunicación con esta herramienta, es decir, se lleva a cabo el diseño del mecanismo.

En el capítulo 5 se realiza la implantación del mecanismo de comunicación, la cual se desarrolla utilizando diagramas de estructura, diagramas de flujo de datos y pseudocódigos, con el fin de dar una descripción detallada del comportamiento de todos los elementos que integran al mecanismo.

En el capítulo 6 se evalúa al mecanismo de comunicación, es decir, se especifican formalmente de las pruebas, que deben realizarse, con el propósito de observar el cumplimiento de todos los requerimientos definidos para el mecanismo.

En el capítulo 7 se exponen las conclusiones finales, obtenidas durante el desarrollo de este trabajo de Tesis.

Se incluye un apéndice, donde se encuentra el código fuente, en lenguaje C, de uno de los tres subsistemas que componen al mecanismo de comunicación.

CAPÍTULO 1: SISTEMA DE CONTROL DISTRIBUIDO (SCD) PARA UNA CENTRAL TERMOELÉCTRICA (CTE).

1.1 Introducción.

El desarrollo del sector eléctrico en México ha experimentado una gran evolución en nuestra época, caracterizándose por el incremento de centrales generadoras y la aplicación de sistemas computarizados para controlar los procesos de dichas centrales.

Debe tomarse en cuenta que cuando la computadora es aplicada en tareas de supervisión y control se debe trabajar en *tiempo real* (el tiempo necesario en la ejecución de cálculos debe ser mucho más pequeño que la constante de tiempo preponderante del proceso).

Básicamente, un *Sistema de Control Distribuido* (SCD) consiste de una colección de procesos distintos que están separados físicamente. Cada proceso realiza una actividad específica (tarea de aplicación, monitoreo, control, etc.) y está comunicado con los demás procesos mediante un sistema de comunicación común. La comunicación entre procesos se realiza por medio de un intercambio de mensajes.

Para implantar estos sistemas de control distribuido, se emplea un conjunto de computadoras enlazadas por una red de comunicaciones. Estas computadoras se encuentran distribuidas físicamente alrededor de la planta o dispositivos que estén controlando y por esto, deben comunicarse para coordinar y controlar sus actividades. Los SCD's tiene su área de aplicación en centrales de generación y distribución de electricidad, plantas químicas, industrias del acero, etc.

Un sistema que controle una central generadora de energía eléctrica (CTE), debe tener tres características fundamentales, que son: la adquisición de información de los procesos que se requieren controlar, el procesamiento de esta información de acuerdo con las funciones de software que constituyan al sistema y la presentación de dicha información a los operadores de la CTE con el fin de poder generar las acciones de control requeridas y la ejecución de las funciones de control.

Los sistemas de control distribuido para CTE's involucran del orden de 3000 a 3250 señales (2500 digitales y 750 analógicas) y por lo tanto existe un número importante de variables por controlar y monitorear, por lo que se ha requerido desarrollar sistemas que permitan manipular la información, que se va generando, con oportunidad, disponibilidad y confiabilidad.

1.2 Niveles generales de un SCD para una CTE.

Existen cuatro niveles principales que se pueden observar dentro de un sistema de control distribuido para una CTE:

1.2.1 Nivel de Equipo.

Las computadoras se incorporan junto con sensores, actuadores y controladores; todos estos interactuan con el propósito de controlar las actividades del sistema. El sistema puede estar dividido en subsistemas, cada uno controlando alguna actividad. Para tener un control global se requiere que dichos subsistemas se comuniquen por medio de un sistema de comunicación. Estos subsistemas se conocen comúnmente como nodos.

1.2.2 Nivel de Adquisición de Información.

Aquí se realiza la adquisición de las señales generadas por los procesos de la CTE por medio de unidades especiales llamadas Unidades de Adquisición (UA's). Aquí también se realiza el acondicionamiento de estas señales para poder ponerlas a disposición de otras funciones del SCD.

1.2.3 Nivel de Procesamiento.

En este nivel se realiza el procesamiento de las señales adquiridas. Esto significa que se realizan actividades tales como: cálculo de variables, elaboración de diagramas, reportes, etc.

1.2.4 Nivel de Presentación y Control.

Las computadoras transmiten la información adquirida y acondicionada hacia un cuarto de control o hacia una estación de operador donde es analizada, registrada o desplegada hacia un operador humano. Las computadoras son conectadas con desplegadores visuales (pantallas), impresoras, teclados, consolas especiales, etc. Todos éstos son utilizados para comunicarse con el operador; éste da como entrada puntos de referencia, parámetros o solicitudes de información, la cual es transmitida hacia algún dispositivo o controlador. Las salidas hacia el operador pueden ser las respuestas a la información solicitada o pueden ser mensajes de alarma de algún proceso en la central.

1.3 El sistema de comunicación de un SCD.

Todos los sistemas de control distribuido utilizan un sistema de comunicación, ya que requieren de una comunicación eficiente entre todos los procesos que componen dichos sistemas, para que estos procesos puedan coordinar sus actividades.

La función general de un *sistema de comunicación*, en un SCD, es la de transferir información entre los procesos que están en diferentes computadoras. Además, el sistema de comunicación debe contar con mecanismos para que no existan errores o pérdidas de información.

La información transferida a través del sistema de comunicación entre un proceso y otro consiste de bits, caracteres, estructuras de datos, etc. y la interpretación de la información depende de cada proceso de aplicación. Esta información es transferida en forma de *mensajes*, que pueden verse como unidades lógicas de información.

Es posible clasificar el tipo de información que se transfiere en un sistema de comunicación [6] de acuerdo a:

a) El tiempo de respuesta (qué tan rápido el mensaje debe ser transferido).

b) Longitud (la cantidad de información transferida).

c) Confiabilidad (la importancia de evitar errores).

d) Frecuencia (qué tan frecuentemente los procesos envían un tipo particular de mensaje).

Si un tipo particular de mensaje predomina en una aplicación, entonces el sistema de comunicación puede ser optimizado para ese tipo de transferencia.

Para comunicarse un proceso con otro, cada proceso requiere de un medio de identificación. Esta es una característica básica para la transferencia de mensajes, es decir, cada proceso necesita ser identificable; debe de tener un nombre y una dirección, indicando al sistema de comunicación donde se localiza dicho proceso.

Hay que notar que un nombre es un identificador lógico y una dirección es un identificador físico; por eso, los nombres deben ser traducidos a direcciones para ser utilizados por el sistema de comunicación.

Cuando un proceso transfiere información hacia otro proceso, el destino debe ser identificado para que el sistema de comunicación sepa donde entregar la información. Si la información requiere una respuesta y cuando pudo haber sido enviada por cualquiera de los procesos del sistema, la fuente debe también ser identificable para que el destino sepa donde enviar la respuesta. Generalmente tanto la fuente como el destino de un mensaje deben de ser identificables.

Cuando se desarrollan aplicaciones, es mejor utilizar nombres lógicos que direcciones físicas. La aplicación puede, entonces, ser desarrollada sin la necesidad de saber la configuración física de la red en la cual será implementada.

1.3.1 Características.

Una de las características más sobresalientes de un sistema de comunicación en un SCD puede verse mediante el concepto de su *confiabilidad*. Esta confiabilidad puede medirse en términos de su *integridad* y su *disponibilidad*.

1.3.1.1 Integridad.

La *integridad*, en un sistema de comunicación, es la medida de la exactitud y totalidad de la información que es recibida por un proceso de aplicación por medio del sistema de comunicación.

La integridad en la transferencia de información es de suma importancia en aplicaciones de control de procesos por razones obvias, ya que un parámetro de control erróneo o la pérdida de un mensaje puede llevar a una situación indeseable.

Por la razón anterior, cada sistema de comunicación debe contar con un mecanismo que sirva para la detección y corrección de errores y recuperación de esos errores que son detectados pero no pueden ser corregidos. Este mecanismo es usualmente incorporado dentro del sistema de comunicación en vez de en los procesos de aplicación, principalmente para evitar que estos procesos se vuelvan muy complejos y para separar las funciones de aplicación de las funciones de comunicación.

El control de error puede incrementar la longitud y el número de los mensajes utilizados por el sistema de comunicación y por lo tanto reducir el rendimiento. Así, existe un intercambio entre los requerimientos de rendimiento, por un lado, y la integridad, por el otro.

Los errores que puede ocurrir dentro de un sistema de comunicación en un SCD [6] incluyen los siguientes:

1) Pérdida de un mensaje (debido a una falla o porque el campo de dirección ha sido modificado).

2) Mensajes duplicados (resultado de la retransmisión dentro del sistema de comunicación).

3) Corrupción de mensajes (debido al ruido).

4) Mensajes fuera de secuencia (cuando hay más de una ruta hacia el proceso destino).

El mecanismo usado por el sistema de comunicación para control de error debe ser transparente a la aplicación. Hay varias técnicas de control de error, por ejemplo el chequeo cíclico redundante. Los errores de pérdida, duplicación y secuencia pueden ser corregidos por medio de números de secuencia incluidos en los mensajes que serán transmitidos.

1.3.1.2 Disponibilidad.

La *disponibilidad* del sistema de comunicación, es definida como el porcentaje del período operacional durante el cual el sistema es capaz de realizar su función específica, entre el período operacional requerido del sistema. Este término no sólo se aplica al sistema de comunicación, sino al sistema en general.

Ningún sistema puede garantizar el 100% de disponibilidad, ya que siempre hay una probabilidad finita de que un componente falle. La importancia de la disponibilidad dependerá de las consecuencias de una falla en el sistema. Un sistema de comunicación que no puede ser usado por algún proceso puede estar trabajando correctamente. Por ejemplo, un sistema puede ser incapaz de establecer conexión porque está saturado o por que no existe espacio buffer suficiente; estas condiciones lo harán parecer indispuesto para el proceso de aplicación, a pesar del hecho de que puede estar trabajando bien.

En la práctica, el sistema de comunicación es un componente clave en los SCD's y por esto su disponibilidad debe ser alta. Esto quiere decir, que el sistema debe continuar trabajando en un ambiente degradado, aún si algún otro componente falla.

Un criterio primario que se sigue en un SCD es que ninguna falla simple de cualquier dispositivo conectado al sistema de comunicación o ninguna falla de un simple componente dentro del sistema de comunicación debe causar la falla total del SCD.

La disponibilidad de un sistema de comunicación [6] depende de los siguientes factores:

a) La tasa de falla de los módulos de hardware y software dentro del sistema.

b) La habilidad de localizar los efectos de una falla; por ejemplo, la falla de una estación o línea de transmisión no debe impedir la comunicación entre las otras estaciones.

c) La habilidad de sobrepasar las fallas; por ejemplo, re-ruteando los mensajes alrededor de la falla.

d) El tiempo requerido para diagnosticar y reparar la falla.

1.4 Modelo Conceptual de un SCD para un CTE.

En la figura 1.1 se presenta a nivel abstracto, el modelo conceptual de un SCD para una Central Termoeléctrica [2]. En este modelo conceptual se considera a una planta generadora de energía eléctrica como un conjunto de procesos de los que se adquiere información que será procesada y distribuida para su adecuada utilización.

Las distintas funciones que forman el SCD procesan la información obtenida y los resultados de dicho procesamiento son repartidos en tres distintos puestos de mando, que son:

1) El Puesto del Operador de la planta, que tiene la responsabilidad de supervisar y controlar ciertas variables particulares de los procesos realizados en la CTE. Está compuesto por varios TRC (Tubos de Rayos Catódicos) y por teclados funcionales.

2) El puesto del Laboratorio Químico (LQ), que también tiene como función el supervisar y controlar variables de los procesos de la CTE. Contiene elementos tales como: Un TRC y un teclado.

3) El puesto del Ingeniero del Sistema, que tiene como objetivo fundamental supervisar y controlar la operación del SCD de la CTE. En este puesto pueden observarse todas las variables y solicitar todas las funciones disponibles en el SCD.

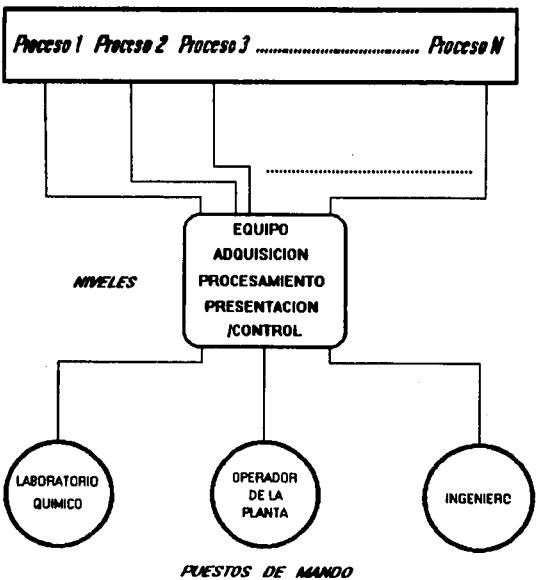

Figura 1.1. Modelo conceptual de un SCD.

1.4.1 Funciones del SCD.

En la figura 1.2 se puede observar un diagrama funcional del SCD y las relaciones que existen entre sus funciones [2]. Este diagrama está constituido de cuatro niveles, conteniendo diferentes funciones cada uno de ellos. Estos niveles son: Equipo, Adquisición, Procesamiento y Presentación.

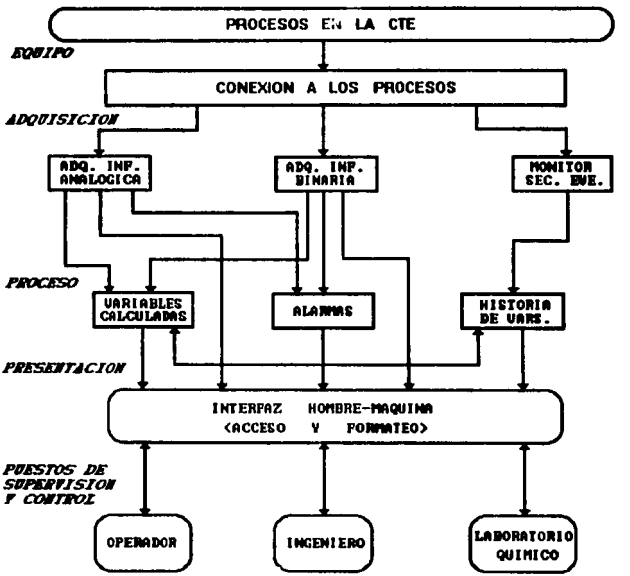

Figura 1.2. Diagrama funcional de un SCD.

El nivel de equipo está constituido por sensores, multiplexores, actuadores, controladores y acondicionadores de señales.

En el nivel de adquisición se llevan a cabo las siguientes funciones:

- Adquisición Binaria.
- Adquisición Analógica.
- Adquisición de Señales del MSE.

En el nivel de procesamiento se encuentran los equipos y mecanismos que se dedican a la interpretación y ejecución de algoritmos para el cálculo de variables, supervisión de procesos, elaboración de informes, etc. Todo esto con el fin de realizar:

1.- Una presentación detallada de las condiciones de los procesos.

2.- Una representación gráfica de los sistemas de la planta, incluyendo la información del estado de los procesos.

3.- Una reproducción de los estados operacionales de la planta.

4.- La elaboración de cálculos de parámetros de procesos.

5.- La documentación de la operación de la planta.

6.- El almacenamiento y presentación de los datos necesarios para analizar algún disturbio (disparo) en la planta.

En el nivel de presentación se lleva a cabo el control del sistema mediante el despliegue e impresión de la información adquirida y procesada por las funciones del SCD. Este nivel se distribuye en los diferentes puestos de mando, vistos anteriormente.

En cada nivel del SCD se realizan una variada gama de funciones [1]. Entre las cuales tenemos:

Adquisición Analógica.

Aquí se realiza un monitoreo continuo de las señales analógicas de campo, obteniendo los valores por medio de Unidades de Adquisición (UA's), para que sean digitalizadas, acondicionadas y transformadas en unidades de ingeniería, para que puedan ser procesadas por otras funciones del SCD que así lo requieran.

Cada Unidad de Adquisición está constituida en base a microprocesadores, controladores, convertidores programables y tarjetas de entrada/salida. La adquisición analógica puede, además, identificar el estado de las UA's (inicialización, configuración o adquisición continua).

Adquisición Binaria.

Esta función realiza la adquisición de las señales binarias de los procesos de la planta, por medio también de las UA's; y como la función anterior, acondiciona las señales para que puedan ser utilizadas por otras funciones del SCD.

Adquisición del MSE.

Por medio de esta función se realiza la adquisición de las señales del Monitor de Secuencia de Eventos (MSE) desde el campo, a través de UA's dedicadas. El MSE es el responsable de la detección y el registro a alta velocidad de los eventos que ocurran en los sistemas de protección del proceso, tomando como punto de partida el detectar una condición tal como una falla; teniéndose así un registro cronológico de aquellas variables que permitan hacer un análisis de los eventos relacionados con la falla y la secuencia de ocurrencia, con el propósito de efectuar un diagnóstico de falla.

Alarmas.

El sistema de alarmas ayuda al operador de la planta en el control del proceso, ya que puede detectar e informar algún cambio de estado normal a un estado de alarma (disparo), y viceversa, en las variables del proceso definidas para esta función. El informe y manejo de las alarmas se puede realizar de tres maneras distintas: Con un tablero convencional de alarmas, con impresoras y con despliegue en pantalla.

Historia de variables.

Esta función realiza un registro histórico y un almacenamiento del comportamiento (valor y estado) de ciertas variables adquiridas y calculadas a través del tiempo, para proporcionar esta información a otras funciones del SCD que requieran de esta información.

Variables calculadas.

Esta función realiza el cálculo de expresiones predefinidas en función de variables adquiridas o de otras previamente calculadas. Contiene un módulo para el cálculo de parámetros característicos.

Cálculo de parámetros característicos.

Aquí se calculan parámetros que caracterizan el comportamiento de los distintos procesos de la central (eficiencias, comportamientos, etc.).

Interfaz Hombre-Máquina.

La Interfaz Hombre-Máquina es la que permite que exista un "diálogo" entre el usuario y las funciones que conforman al SCD, presentándole al usuario información relativa a los procesos, los subsistemas y el sistema en general, para que pueda realizar la supervisión y control del sistema. Esto lo realiza por medio de reportes o desplegados visuales.

Reportes.

Esta función realiza la generación de información impresa de las condiciones de la CTE. Los reportes son generados ya sea automáticamente o por petición. Algunos reportes son: alarmas, balance diario, mensual, horas de operación, etc.

Monitor de Secuencia de Eventos.

El MSE tiene como objetivo generar los registros de información para:

a) Reportar secuencias de eventos.
b) Reportar secuencias de disparos.

Esto lo realiza registrando la información del comportamiento de todas las señales de MSE de los n conjuntos definidos en el sistema y de algunas otras señales analógicas y binarias.

El objetivo del MSE es realizar un registro cronológico y ordenado de estas señales, el cual debe mantenerse hasta que sea tomado por la Interfaz Hombre-Máquina.

Base de Datos en Tiempo Real.

Su función es la de almacenar la información actualizada de todas las variables adquiridas por el sistema. Esto lo realiza utilizando estructuras en memoria en cada nodo donde se realiza la adquisición de variables.

Base de Datos de Historia.

Su función es mantener el registro histórico de los valores y estados operativos de todas las variables adquiridas por el SCD y está replicada en cada puesto de supervisión.

Base de Datos de Reportes.

Contiene la información asociada a los reportes que son generados en el SCD.

1.5 Requerimientos de un SCD para una CTE.

Con las características antes mencionadas, el SCD para una CTE debe cumplir con una serie de requerimientos que están basados en el documento de especificaciones CFE-J100 para la instrumentación, control y automatización de Centrales Termoeléctricas normalizadas [4]. Estos requerimientos son:

a) El sistema debe realizar las siguientes funciones:

1.- Adquisición y acondicionamiento de datos de los procesos.
2.- Funciones matemáticas elementales (Aritméticas y Lógicas).
3.- Entrega de información.
4.- Entrega de datos hacia las salidas del sistema.
5.- Entrada/salida de datos hacia dispositivos periféricos.
6.- Cálculo de variables en función de las señales adquiridas.
7.- Cálculo de eficiencia y otro parámetros de la planta.
8.- Presentación de gráficas.

b) El sistema debe tener capacidad para manejar 2500 señales binarias, 750 analógicas y 125 señales binarias por interrupción.

c) El periodo de adquisición debe ser de 1 segundo.

d) El SCD debe tener una disponibilidad del 99%: es decir, el sistema puede estar, como máximo, fuera de operación 3 días con 16 horas al año.

e) Los valores de las señales adquiridas por el SCD deben poderse presentar tanto en pantallas como en impresoras.

f) El tiempo de respuesta debe ser de 1 segundo, es decir, el tiempo que transcurre entre que un valor es solicitado por el usuario y que le es presentado debe ser de 1 segundo.

g) El sistema debe contar con tres puestos de supervisión y control; el del Ingeniero de la planta, el del Operador de la planta y el del Laboratorio Químico.

h) Se deberá contar con una pantalla en color para mostrar los grupos de valores de medición en forma de barras, curvas características y diagramas de flujo de la planta. Esta pantalla debe instalarse en el tablero de control de la planta.

i) Se debe contar con una pantalla para fines de operación, que deberá ser instalada en una consola especial en el cuarto de control, otra para funciones de ingeniería en el cuarto de computadoras y otra para el operador del laboratorio químico.

j) El puesto correspondiente al Ingeniero debe tener las siguientes funciones:

- Manejo del sistema de cómputo.
- Manejo de la Base de Datos del sistema.
- Manejo de los dispositivos del sistema.
- Petición y envío de diagnósticos.

k) El puesto correspondiente al Operador debe tener las siguientes funciones:

- Manejo de la Interfaz Hombre-Máquina.
- Manejo de los dispositivos especiales.
- Demanda de reportes en pantalla.
- Reconocimiento de alarmas.

l) El puesto correspondiente al Laboratorio Químico debe tener las siguientes funciones:

- Manejo de la Interfaz Hombre-Máquina.
- Manejo de dispositivos especiales (registradores).
- Demanda de reportes en pantalla.

m) Para la documentación de la operación de la planta y el análisis posterior del proceso, se deben generar los siguientes reportes:

- Alarmas.
- Variables en alarmas.
- Puntos fuera de muestreo.
- Tendencia analógica.
- Horario.
- Balance diario y mensual.
- Estado de valores de medición y calculados
- Estado de señales binarias
- Horas de operación.
- Cuenta de eventos.
- Alarmas del sistema de computadoras.
- Fallas en los dispositivos periféricos.

n) El sistema debe contar con un reporte postdisparo mediante el cual deberá registrarse el historial previo y posterior al disturbio en secuencia cronológica, es decir, La historia previa deberá registrar valores previos y valores posteriores.

En resumen, podemos decir que un SCD para Centrales Termoeléctricas realiza las siguientes funciones [3]:

1.- Instrumentación y acondicionamiento de señales.
2.- Detección y registro de secuencias de eventos.
3.- Muestreo y validación de señales analógicas y digitales.
4.- Adquisición periódica de valores analógicos y digitales.

5.- Procesamiento de señales.
6.- Cálculo de variables elaboradas.
7.- Supervisión y control de procesos.
8.- Monitoreo de secuencias de eventos.
9.- Registro de la evolución de variables.
10.- Autovigilancia del SCD.
11.- Interfaz Hombre-Máquina.

1.6 Arquitectura de un SCD para una CTE.

Una de las funciones primordiales de un SCD es el poder concentrar la información de los procesos de la CTE bajo supervisión y control y desplegarla en forma oportuna y ágil a los puestos de mando de la central, ya sea por petición de éstos, o de manera automática cuando existan eventos que modifiquen a los procesos bajo vigilancia. Todo esto implica tener una Base de Datos en Tiempo Real (BDTR) donde la información recolectada de la planta sea almacenada y pueda ser consultada por los usuarios del sistema.

De acuerdo al modelo conceptual visto anteriormente, un SCD se divide en cinco componentes que son: Equipo, Adquisición, Procesamiento, Presentación y Control.

La información adquirida periódicamente, de la CTE, se procesa para definir su valor y estado funcional y se organiza de acuerdo con las maniobras que realizan los operadores: control, vigilancia, atención de alarmas, etc.

La presentación de la información se reparte a través de tres puestos de mando que son, como ya se indicó: el Operador de la planta, el Laboratorio Químico y el Ingeniero del Sistema.

Para poder integrar todos los elementos mencionados anteriormente, se diseño una Red de Área Local, con una topología bus [1]. La red tiene un mecanismo de acceso al medio "Token Passing Bus" y está diseñada bajo el ambiente del Sistema Operativo de Tiempo Real Q.N.X. Esta arquitectura se muestra en la figura 1.3.

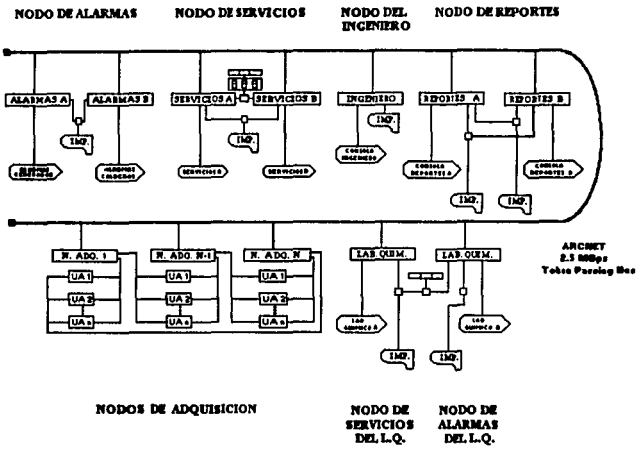

Figura 1.3. Arquitectura de un SCD para una CTE.

La arquitectura del SCD cuenta básicamente con 2 tipos de nodos: los Nodos de Adquisición y los Nodos de Presentación. Cada tipo de nodo tiene funciones y características específicas.

1.6.1 Nodos de Adquisición.

Estos nodos están compuestos de mecanismos que se dedican a la interpretación y ejecución de algoritmos de las funciones dedicadas a la adquisición periódica de valores obtenidos en las UA's.

Cada Unidad de Adquisición tiene mecanismos de comunicación con los Nodos de Adquisición y cuenta con funciones de autodiagnóstico que permiten evaluar el estado operacional de los elementos que las constituyen, además de realizar dos funciones: detección y registro de secuencias de eventos y adquisición de señales analógicas y digitales.

En los Nodos de Adquisición se ubican fisicamente las estructuras en memoria de las Bases de Datos de Tiempo Real y las Bases de Datos correspondientes a la historia de las variables, que contendrán la información adquirida. En estos nodos se localizan las siguientes funciones:

* Adquisición analógica.
* Adquisición binaria.
* Historia de variables.
* Variables calculadas.
* Monitor de secuencia de eventos.

Estos nodos tendrán, además, la función de difundir la información adquirida de señales analógicas, binarias y eventos almacenada en la BDTR, mediante el mecanismo de comunicaciones a desarrollar en esta tesis, a través del bus de enlace de la red, para que sea recibida por todos los Nodos de Presentación.

1.6.2 Nodos de Presentación.

Estos nodos tienen como propósito principal realizar el procesamiento de la información, el cálculo de variables, la Interfaz Hombre-Máquina con el operador y el control de procesos. Estos nodos se pueden clasificar en los siguientes tipos: Nodos de Alarmas, Nodos de Reportes, Nodo del Ingeniero, Nodos de Servicios, Nodo de Alarmas del LQ y Nodo de Servicios del LQ.

1.6.2.1 Nodos de Alarmas.

Los Nodos de Alarmas son los encargados del manejo de las variables en alarma del SCD, esto es, tienen como propósito la detección, registro y presentación de variables en alarma del proceso que son detectadas por el SCD.

1.6.2.2 Nodos de Servicios.

Los Nodos de Servicio tienen como propósito el llevar a cabo todas las funciones relacionadas con la Interfaz Hombre-Máquina orientada al operador de la planta, a través de despliegues de: diagramas de flujo, diagramas de barras, etc. y despliegues especiales: graficación en papel, indicadores digitales y cuadro convencional de alarmas.

1.6.2.3 Nodos de Reportes.

Los Nodos de Reportes están destinados a formatear la información y se encargan de realizar los reportes automáticos y por solicitud que ofrece el SCD de la información adquirida y procesada por las diferentes funciones.

1.6.2.4 Nodo del Ingeniero.

El Nodo del Ingeniero permite al Ingeniero de sistemas a accesar todas las funciones de servicio y propias del Ingeniero. Desde este nodo será posible darle mantenimiento a la Base de Datos de Tiempo Real del SCD, a través de un diálogo de servicios. El Ingeniero es el que tiene el control del SCD, es decir, es el que puede arrancar el sistema, sacar o poner en servicio dispositivos o Unidades de Adquisición, etc.

1.6.2.5 Nodos del Laboratorio Químico.

Los Nodos del LQ contienen todas las funciones de los nodos de Alarmas y Servicios, pero dedicados únicamente a las variables del LQ, con la única diferencia entre ellos de que solamente uno tendrá activada las funciones de alarmas, mientras que otro tendrá activadas las funciones de servicios.

1.7 Características principales de un SCD para una CTE.

Finalmente, podemos decir que las principales características del SCD para una CTE son:

a) Utiliza una red ARCNET, con una capacidad de transferencia de información de 25 Mbps.

b) La topología de la red de comunicación está basada en un bus simple; las trayectorias de conexión entre fuentes y destinos son directas (no existen nodos intermedios).

c) Cada nodo se conecta al bus mediante una interfaz manejada por el S.O. en Tiempo Real QNX.

d) Todos los nodos contienen un CPU, un coprocesador y manejan sus propias entradas y salidas.

e) Las computadoras de los nodos son compatibles.

f) Los nodos se comunican entre sí por medio de mensajes.

g) Cada nodo de la red tiene una réplica de la Base de Datos de Tiempo Real y de Historia.

h) La presentación de la información debe estar distribuida en toda la red.

CAPÍTULO 2: MECANISMOS DE COMUNICACIÓN EN SISTEMAS DE CONTROL DISTRIBUIDO.

2.1 Introducción.

Muchas actividades dentro de un SCD envuelven la coordinación simultánea de varios procesos. Debido a esto, la comunicación entre procesos juega un papel importante dentro de los SCD's, por lo que el sistema de comunicación representa un componente esencial en el SCD.

Un SCD puede verse como una colección de nodos interconectados (llamada *red*) que permite el intercambio de unidades de información entre los nodos o los procesos en dichos nodos. Este intercambio de información requiere, para que sea realizado ordenadamente, que cada nodo siga ciertas reglas preestablecidas.

Estas reglas permitirán que los procesos puedan intercambiar información de una manera ordenada y eficiente. Un *mecanismo de comunicación* es el que establece estas reglas que contienen, entre otras cosas, acuerdos sobre los mecanismos usados para:

- Inicialización y terminación de intercambio de datos.
- Sincronización de transmisores y receptores.
- Detección y corrección de errores de transmisión.
- Formateo y codificación de los datos.

El propósito primario de un mecanismo de comunicación, es establecer un orden en el intercambio de información entre los procesos de un SCD. Por esto, un mecanismo de comunicación puede verse como un conjunto de acuerdos entre dos procesos comunicantes.

Por ejemplo, considere un sencillo intercambio de mensajes entre dos procesos, A y B. La secuencia que se sigue para la transmisión de datos es mostrada en la figura 2.1.

Para comenzar, el proceso A envía un mensaje de iniciación de comunicación (Listo para Enviar) al proceso B. El proceso B recibe este mensaje y contesta transmitiendo un mensaje de respuesta (Listo para Recibir) al proceso A. Este intercambio puede verse como la inicialización de la transmisión de datos.

Después, los mensajes de datos son enviados por el proceso A y son recibidos por el proceso B, éste último envía un mensaje de reconocimiento positivo (recibido) por cada mensaje recibido de A.

Para terminar el intercambio, el proceso A envía un mensaje de terminación de comunicación (Fin de Datos) al proceso B, mismo que reconoce positivamente dicho mensaje.

Este mecanismo de comunicación, para un simple intercambio de datos, debe también especificar el formato de cada mensaje de control (iniciación y terminación), el encabezado para los mensajes, etc.

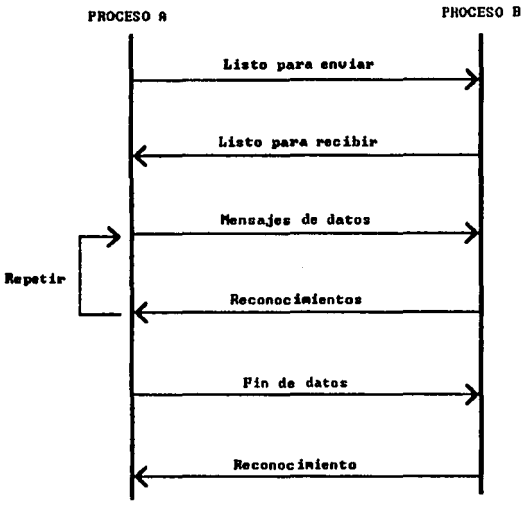

Figura 2.1. Transmisión de datos entre procesos.

Si dividimos en capas a los dos nodos interconectados por medio de una red, podemos ver dichas divisiones, en forma simple, como se muestran en la figura 2.2.

Un usuario accesa a la red a través de alguna entidad lógica (programa de aplicación) que llamamos proceso. Una capa de sesión recibe un mensaje desde la capa de proceso y realiza la acción necesaria para el establecimiento de la comunicación entre los usuarios de la red.

La capa de sesión envía mensajes a la siguiente capa baja, o el sistema de comunicación, el cual entrega los mensajes a la capa de sesión del nodo destino.

El principio fundamental es que una capa dada en un nodo, lógicamente intercambia mensajes con su correspondiente capa en otro nodo y el procesamiento en otras capas más bajas es transparente para dicha capa.

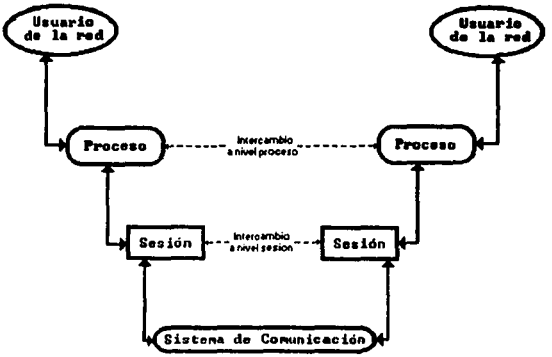

Figura 2.2. Capas generales en una red.

Los mecanismos de comunicación que permiten la comunicación entre capas correspondientes en diferentes nodos son llamados protocolos. Una capa dada también puede comunicarse con las capas adyacentes mediante una interface. Estos conceptos se ilustran en la figura 2.3. En resumen, podemos definir lo siguiente:

Una *capa* puede considerarse como un conjunto de entidades (procesos análogos de software y hardware) dentro de un nodo; cada capa adiciona servicios a aquellos proporcionados por la capa inferior y los ofrece a la capa superior. Las entidades en la misma capa y en nodos diferentes son llamadas entidades pares y se comunican mediante protocolos.

Un *mecanismo de comunicación* o *protocolo* define las reglas que gobiernan el intercambio de información entre entidades localizadas en una capa particular, en un nodo, con sus correspondientes entidades pares en otros nodos.

Un mecanismo de comunicación también define el formato y el orden de intercambio de la información, así como las acciones que se tengan que llevar a cabo en la recepción de la información.

Una *interface* entre dos capas locales de un nodo, define los mecanismos mediante los cuales una capa hace uso de los servicios provistos por una capa inferior, es decir, una interface define las reglas y formatos para el intercambio de información a través de dos capas adyacentes.

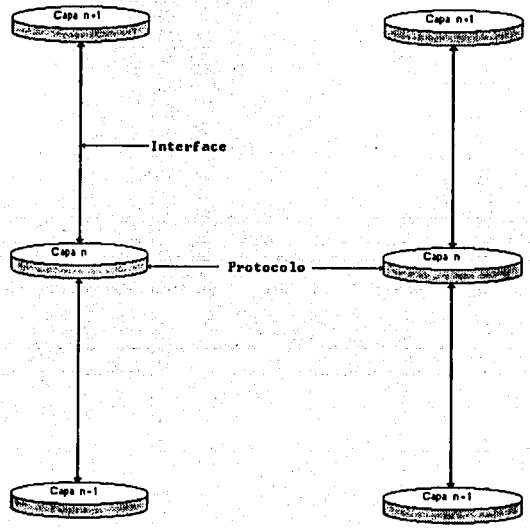

Figura 2.3. Capa, protocolo e interface.

Ahora bien, el tipo de flujo de información es determinado por las relaciones lógicas entre los procesos de aplicación que requieran comunicarse, como lo muestra la figura 2.4.

Hay que hacer notar que las relaciones físicas (líneas de transmisión) no necesitan corresponder a las relaciones lógicas; por ejemplo, el que dos nodos estén conectados a un bus simple común (relación física) no significa que deban intercambiar información (relación lógica).

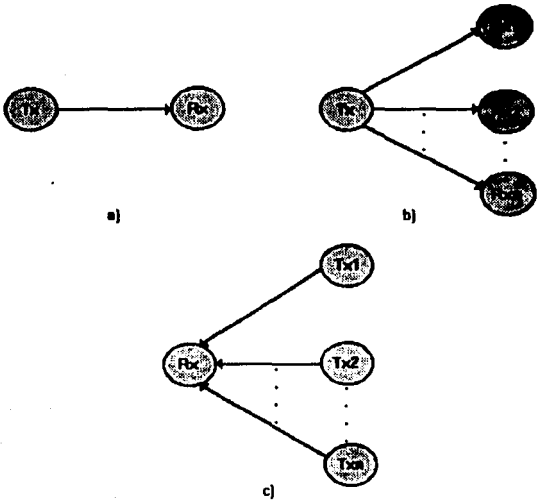

Figura 2.4. Flujos de información: a) Uno a uno, b) Uno a varios y c) Varios a uno.

Así, el flujo de información puede ser de alguno de los tres tipos siguientes:

a) Uno a uno: Los mensajes son enviados por un proceso a otro proceso utilizando un enlace punto a punto, es decir, cada proceso tendrá un enlace directo y dedicado con el otro.

b) Uno a varios (destino múltiple): Los mensajes son enviados por un proceso hacia dos o más procesos. Este tipo de flujo puede ser implementado por el llamado enlace "broadcast" (difusión) que está caracterizado por la propiedad de que una simple transmisión de un nodo fuente puede ser recibida simultáneamente por varios nodos destino. También puede ser implementado por un enlace punto-multipunto, donde hay que indicar explícitamente los destinos que tendrá la información.

c) Varios a uno: Un proceso puede recibir mensajes desde dos o más procesos.

En cada uno de los anteriores, la comunicación puede ser bidireccional, con respuestas retomadas por parte del receptor. Estas respuestas pueden ser reconocimientos positivos (mensaje recibido) o reconocimientos negativos (mensaje no recibido).

2.2 El modelo OSI de la ISO.

Existen muchas organizaciones que se han dedicado a la generación de modelos de referencia para mecanismos de comunicación. Como ejemplos, podemos mencionar al NIST (National Institute of Science and Technology), el FTSC (Federal Telecommunications Standards Committe) y el IEEE (Institute of Electrical and Electronics Engineers).

Las dos organizaciones más importantes en el área de generación de modelos de referencia son: el CCITT (Comité Consultatif International Télégraphique et Téléphonique) y la ISO (International Standard Organization).

Las organizaciones anteriores, realizan la tarea de estandarización de protocolos, esto es, definen un formato común para la especificación de protocolos (mecanismos de comunicación) y ayudan, así, a resolver un poco el problema del diseño de protocolos.

La ISO ha trabajado en arquitecturas para la interconexión de sistemas, tales como computadoras y terminales. Con esto, ha desarrollado una arquitectura que permite esta interconexión de sistemas y que es conocida como el modelo OSI (Open System Interconnection).

El modelo OSI está basado en tres elementos básicos: los procesos de aplicación, las conexiones que unen a estos procesos para proveer un intercambio de información y los sistemas que soportan a los procesos de aplicación.

El modelo OSI consiste de siete capas, como lo muestra la figura 2.5. Cada capa, con la excepción de la capa física, utiliza los servicios que le proporciona la capa inmediata inferior.

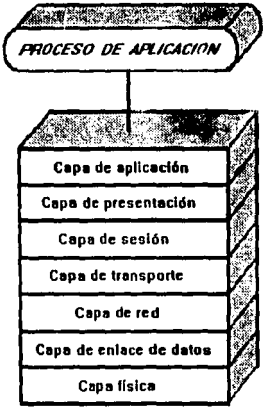

Figura 2.5. El modelo OSI.

2.2.1 Capa de aplicación.

La capa más alta, la capa de aplicación, provee la comunicación entre dos procesos de aplicación (tales como programas de aplicación) con un enlace fin-a-fin (usuario a usuario). Además, esta capa sirve de interfaz al usuario para tener acceso a un conjunto de servicios que son específicos a cada aplicación y que son incluidos en esta capa.

2.2.2 Capa de presentación.

Esta capa realiza servicios para la interpretación del significado de los datos intercambiados, esto es, realiza transformaciones en los datos transmitidos, con la finalidad de presentarlos al usuario en una forma entendible. Esto incluye la administración de la entrada, intercambio, despliegado y control de datos estructurados. Otra función de esta capa es la conversión entre códigos de caracteres. Lo más común entre computadoras es que no manejen los mismos formatos de almacenamiento de archivos, así que la capa de presentación debe brindar también facilidades de conversión entre formatos de archivos.

2.2.3 Capa de sesión.

La capa de sesión provee dos clases de funciones:

1) El ligado y desligado de dos entidades de presentación (que existen en la capa de presentación) para una relación lógica.

2) El control en el intercambio de datos, delimitando y sincronizando las operaciones entre las dos entidades de presentación. Así, esta capa provee la iniciación y terminación de sesiones, recuperación de sesiones y control de diálogo. El control de diálogo se refiere al orden en el cuál los mensajes debe fluir entre los usuarios.

En resumen, esta capa establece la conexión entre procesos localizados en otras computadoras. Una vez que la conexión a sido establecida, esta capa debe manejar el diálogo en forma adecuada.

2.2.4 Capa de transporte.

Esta capa es el límite entre los que son las capas orientadas a la aplicación y las capas orientadas a la comunicación. Es la capa más baja que utiliza un protocolo con significado fin-a-fin y provee una transferencia de datos transparente.

Esta capa asigna una dirección única de transporte a cada usuario y se maneja el concepto de *conexión de transporte*. Una o más conexiones de transporte pueden existir entre un par de direcciones de transporte. Una asociación entre dos entidades a nivel de la capa de aplicación sería mapeada por las capas de sesión y presentación dentro de una conexión ofrecida por la capa de transporte.

La capa de transporte usualmente ofrece funciones de multiplexaje, control de flujo y error de extremo a extremo, fragmentado y ensamblado de mensajes grandes dentro de paquetes.

2.2.5 Capa de red.

La capa de red aísla a las capas más altas de las consideraciones de comunicación y ruteo. Esta capa hace transparente a la capa de transporte de las particularidades de transferencia del medio tales como: enlace punto a punto, conmutación de paquetes, etc. Esta capa esta dividida en tres subcapas:

Subcapa Inter Red. Ofrece el ruteo, conmutación de paquetes de datos y funciones de retransmisión a través de subredes conectadas.

Subcapa de Soporte de Subred. Refuerza el nivel de servicios ofrecidos por una subred a los esperados por la subcapa inter red. Esto involucra funciones de control de secuencia, flujo y errores.

Subcapa de Acceso a la Red. Ofrece las funciones necesarias para transferir datos a través de la subred.

Hay que mencionar que un mensaje entre dos usuarios de red, atraviesa las tres capas más bajas en cada nodo transitado en la red.

2.2.6 Capa de enlace de datos.

La capa de enlace establece y mantiene uno o más enlaces entre entidades de red, y provee secuenciamiento y control de flujo de información. Esta capa también controla la detección de errores, control de acceso a la red local y sincronización de bloques de datos (mensajes). La capa de enlace está dividida en dos subcapas que son:

Subcapa de Control de Enlace Lógico. Relativa a la detección de errores y algunas veces corrección de errores y control de flujo. Sus funciones específicas son:

- Iniciar y controlar el intercambio de información.
- Organizar el flujo de datos.
- Realizar funciones de control y recuperación de errores.

Subcapa de Control de Enlace al Medio. Es la responsable del control de acceso al medio de transmisión compartido. Esta capa usa los servicios de la capa física para proporcionar servicio a la subcapa de enlace lógico.

2.2.7 Capa física.

Esta capa soporta los procedimientos mecánicos, eléctricos y funcionales para la creación, mantenimiento y liberación de circuitos físicos. Es la responsable de la transmisión de bits sobre el sistema físico. Lleva a cabo todas las funciones asociadas con señalización, modulación y sincronización de bits. Esta puede realizar la detección de errores por el monitoreo de la calidad de la señal, la especificación eléctrica y mecánica de la conexión física de la interfaz a la red.

2.3 Fases de desarrollo de mecanismos de comunicación.

La metodología empleada en el desarrollo de mecanismos de comunicación incluye las fases tradicionales del desarrollo del software en general, las cuales son:

a) La definición de los requerimientos de comunicaciones del sistema.
b) La especificación del mecanismo de comunicación.
c) El diseño de las reglas de procedimiento del mecanismo.
d) La implantación del mecanismo de comunicación.

El objetivo de la fase de definición de requerimientos es el de proveer una definición explícita de los requerimientos del mecanismo de comunicación para un sistema dado, es decir, se definen las necesidades de comunicación del sistema que deben ser satisfechas por el mecanismo de comunicación, tomando en cuenta los requisitos de cada elemento que conforma este sistema, siendo estos, componentes físicos, como equipo o lógicos, como procesos y funciones. En esta etapa, se lleva a cabo un análisis del flujo de información entre las funciones que conforman la arquitectura del sistema, para el cual se desarrollará el mecanismo de comunicación.

En la especificación del mecanismo de comunicación, se define claramente el problema por resolver, utilizando un lenguaje más técnico y detallado. Esta etapa se divide en sub-etapas, que incluyen la especificación de los requerimientos del mecanismo, el medio ambiente de desarrollo, los servicios que prestará el mecanismo, los mensajes y formatos utilizados por el mecanismo para realizar sus funciones.

En la fase del diseño del mecanismo de comunicación, se debe tratar con la estructura interna del mecanismo. La interacción entre procesos separados físicamente, constituye las reglas de procedimiento, es decir, el mecanismo de comunicación real. Las reglas de procedimiento describen la operación que realiza cada función interna del mecanismo, en respuesta a eventos generados en el sistema, mensajes de procesos remotos, etc.

El propósito de la fase de diseño, es el de emplear una herramienta formal para establecer las reglas de procedimiento, las cuales representan las interacciones del mecanismo de comunicación.

La última fase del desarrollo es la implantación del mecanismo de comunicación. Una vez que se tienen definidas y representadas todas las interacciones que debe realizar el mecanismo, se procede a su codificación en un lenguaje previamente definido. En esta fase se describen claramente todas las funciones que componen al mecanismo; para ello, se utilizan algunas de las herramientas de la Ingeniería de Software, tales como: diagramas de estructura, diagramas de flujo, diagramas de flujo de datos, pseudocódigos, etcétera.

En este capítulo se describe la fase de definición de los requerimientos del mecanismo de comunicación. La especificación del mecanismo se realiza en el capítulo 3, el diseño del mecanismo se encuentra en el capítulo 4 y su implantación en el capítulo 5.

2.4 Definición de los requerimientos del mecanismo de comunicación

Con base en la arquitectura del SCD, vista en el capítulo 1, podemos definir el flujo de información que se lleva a cabo entre los componentes del SCD. Este flujo de información se muestra en la figura 2.6.

La entrada de la información proveniente de las Unidades de Adquisición (UA's), es atendida por el módulo de adquisición y es acondicionada y almacenada en la zona de datos de tiempo real.

El módulo de detección de alarmas detecta las variables con cambio de estado funcional. Los cambios de estado funcional de las variables son almacenados en archivos (buffer de alarmas). El buffer de alarmas se divide en dos partes, la primera donde propiamente se depositan los atributos de las alarmas y la segunda que son listas que indican el orden en que se presentan las alarmas en las pantallas.

La función de la Historia de variables, realiza el registro histórico de los valores y estados operativos de todas las variables existentes en el sistema.

Cuando un módulo de adquisición, envía un comando de sincronización a las UA's, en respuesta se reciben las interrupciones de las señales del MSE (estampa de tiempo y valor de la señal), esta información se transfiere a un depósito temporal de eventos.

El depósito de eventos es actualizado en cada ciclo de adquisición; después, el módulo de MSE toma los eventos del buffer temporal, a la estampa de tiempo les agrega la hora real (tiempo en el que se envío el comando de sincronización a la UA), los ordena en forma cronológica de acuerdo con la hora real de cada evento, posteriormente deposita esta información en un buffer del MSE y verifica si se cumple con la terminación de secuencia de eventos del MSE.

En caso de cumplir con alguna terminación de secuencia de eventos, se toman los eventos del buffer del MSE y se almacenan en disco, en el archivo de historia de variables del MSE y se solicita la activación del reporte de MSE.

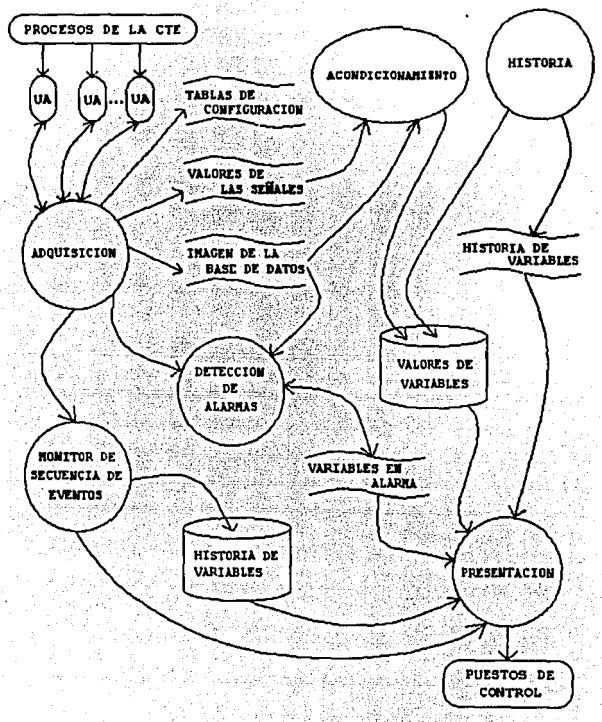

Figura 2.6. Flujo de información en el SCD.

Para cumplir con los objetivos anteriores de adquisición, selección y presentación de información, el SCD requiere de una Base de Datos. Esta Base de Datos debe cumplir con el requerimiento de funcionamiento en tiempo real; para satisfacer este requerimiento se debe contar con una imagen de la base de datos en memoria, la que se utiliza para realizar la definición de cada una de las variables y organizarlas de tal manera que su acceso sea rápido y eficiente. Esta Base de Datos se denomina Base de Datos en Tiempo Real (BDTR).

Los Nodos de Adquisición, son los que cuentan con el conjunto de estructuras en memoria (imagen de la base de datos). Estas estructuras son utilizadas para almacenar las variables que cada NA adquiere. En estas estructuras de memoria se encuentra información tal como:

- Características de las variables, por ejemplo: dirección en la BD, identificador, descripción, tipo, etc.
- Valores y estados operativos.
- Definición de las Unidades de Adquisición.
- Definición de tarjetas de las Unidades de Adquisición.
- Atributos de variables (analógicas y digitales).
- Códigos auxiliares (unidades de ingeniería, locuciones, límites de señales, etc.).

Para mantener la integridad de la información y simplificar el flujo de información entre funciones, cada nodo del SCD deberá contar con una réplica de la Base de Datos en Tiempo Real (BDTR).

Considerando que la información adquirida de los procesos en la CTE se almacena dentro de una BDTR, la cual se estará actualizando en forma continua en cada Nodo de Adquisición, entonces debe plantearse el esquema que permita que los Nodos de Presentación accesen esta información. Con lo anterior podemos definir el primer requerimiento del SCD: ***Transferencia de estructuras en memoria de los Nodos de Adquisición hacia los Nodos de Presentación.***

El SCD requiere de un mecanismo que le permita transferir la información adquirida por los Nodos de Adquisición, almacenada en estructuras de memoria (imagen de la BD), hacia todos los Nodos de Presentación: cada vez que dicha información haya cambiado en relación a un envío previo (estado operacional, valor, etc.). Esto permitirá contar siempre con información actualizada de los procesos de la CTE, en todos los Nodos de Presentación, para ejecutar las funciones de supervisión y control de la central.

Como un SCD se encuentra en un ambiente distribuido, es decir, los nodos que lo integran se encuentran dispersos físicamente, entonces existe un intercambio importante de mensajes de diferentes tipos entre estos nodos. Este intercambio es necesario para el correcto funcionamiento del sistema.

Como se describió en el capítulo uno, los nodos que conforman al SCD son los Nodos de Adquisición y los Nodos de Presentación. Estos últimos incluyen los siguientes: Reportes, Alarmas, Servicios, Ingeniero y Laboratorio Químico.

Los Nodos de Presentación se agrupan en tres puestos de mando, siendo estos: El puesto del Operador compuesto por el Nodo de Alarmas, el Nodo de Servicios y el Nodo de Reportes. El puesto del Ingeniero compuesto por el Nodo del Ingeniero y el puesto del Laboratorio Químico, compuesto por el Nodo de Alarmas del LQ y el Nodo de Servicios del LQ.

Considerando la arquitectura del SCD, se han identificado y clasificado las principales transferencias de información que se llevan a cabo entre los nodos del SCD [1]. Estas transferencias se resumen en las siguientes tablas, donde se identifican los diferentes flujos de información con su origen y destino.

NODOS DE ADQUISICIÓN	
RECEPCIÓN:	**NODO ORIGEN:**
- Comando de inicio, reinicio y sincronía	Ingeniero
- Comando de configuración de las UA's	Ingeniero
- Comando de cambio de estado operativo (tarjetas)	Ingeniero
- Mensaje de aviso de estado	Ingeniero
- Mensajes de reconocimiento	Cualquier nodo
TRANSMISIÓN:	**NODO DESTINO:**
- Mensaje periódico con información de la Base de Datos	Todos los nodos
- Resultado de diagnósticos	Ingeniero
- Mensajes de estados operativos	Ingeniero
- Mensajes de eventos	Reportes
- Mensajes de reconocimiento	Cualquier nodo

Figura 2.7. Tipos de transferencias realizadas por los Nodos de Adquisición.

NODOS DE REPORTES	
RECEPCIÓN:	NODO ORIGEN:
- Mensajes de información de la Base de Datos	Adquisición
- Mensajes de eventos	Adquisición
- Comandos de impresión	Servicios e Ingeniero
- Comandos de inicio, reinicio y sincronía	Ingeniero
- Mensajes de reconocimiento	Cualquier nodo
TRANSMISIÓN:	NODO DESTINO:
- Resultados de diagnósticos	Ingeniero
- Mensajes de reconocimiento	Cualquier nodo

Figura 2.8. Tipos de transferencias realizadas por los Nodos de Reportes.

NODOS DE ALARMAS	
RECEPCION:	NODO ORIGEN:
- Mensajes con información de la Base de Datos	Adquisición
- Comandos de presentación	Servicios
- Comandos de inicio, reinicio y sincronía	Ingeniero
- Mensajes de vigilancia	Otro nodo de Alarma
- Mensajes de reconocimiento	Cualquier nodo
TRANSMISIÓN:	NODO DESTINO:
- Mensajes de reconocimiento	Cualquier nodo
- Mensajes de vigilancia	Otro nodo de Alarma
- Resultados de diagnósticos	Ingeniero

Figura 2.9. Tipos de transferencias realizadas por los Nodos de Alarmas.

NODOS DEL LABORATORIO QUÍMICO	
RECEPCIÓN:	NODO ORIGEN:
- Mensajes de información de la Base de Datos	Adquisición
- Comandos de inicio, reinicio y sincronía	Ingeniero
- Comando de reconfiguración de funciones	Ingeniero
- Mensajes de vigilancia	Otro nodo LQ.
- Mensajes de reconocimientos	Cualquier nodo
TRANSMISIÓN:	NODO DESTINO:
- Resultado de diagnósticos	Ingeniero
- Mensajes de vigilancia	Otro nodo LQ.
- Mensajes de reconocimiento	Cualquier nodo

Figura 2.10. Tipos de transferencias realizadas por los Nodos del Laboratorio Químico.

NODOS DE SERVICIOS	
RECEPCIÓN:	NODO ORIGEN:
- Mensajes de información de la Base de Datos	Adquisición
- Comandos de inicio, reinicio y sincronía	Ingeniero
- Mensajes de reconocimiento	Cualquier nodo
TRANSMISIÓN:	NODO DESTINO:
- Resultado de diagnósticos	Ingeniero
- Comandos de presentación e impresión	Reportes
- Mensajes de reconocimiento	Cualquier nodo

Figura 2.11. Tipos de transferencias realizadas por los Nodos de Servicios.

NODO DEL INGENIERO	
RECEPCIÓN:	NODO ORIGEN:
- Mensajes de información de la Base de Datos	Adquisición
- Resultados de diagnósticos	Todos los nodos
- Mensajes de estados operativos	Adquisición
- Mensajes de reconocimiento	Cualquier nodo
TRANSMISIÓN:	NODO DESTINO:
- Comandos de inicio, reinicio y sincronía	Todos los nodos
- Configuración de las UA's	Adquisición
- Comando de cambio de estado operativo (tarjetas)	Adquisición
- Mensaje de aviso de estado	Adquisición
- Comando de reconfiguración de funciones	Laboratorio Químico
- Comando de impresión	Reportes
- Mensajes de reconocimiento	Cualquier nodo

Figura 2.12. Tipos de transferencias realizadas por el Nodo del Ingeniero.

Analizando el flujo de información, podemos definir un nuevo requerimiento del SCD: ***Transferencia de mensajes y comandos entre los nodos del SCD.***

Debe existir un mecanismo que le permita a los nodos del SCD transmitir mensajes o comandos, para lograr una comunicación completa entre todas las funciones integrantes del SCD.

Además de los requerimientos anteriores, se debe satisfacer la ***transferencia de archivos,*** entre los nodos que conforman la red. Esto se requiere para transmitir información estática, almacenada en disco en forma de archivo texto, y que resida en los nodos del SCD.

Por requerimientos de disponibilidad de la CTE, el SCD debe contar con una alta confiabilidad y con una disponibilidad mayor al 98%. Esta disponibilidad se logra mediante la tolerancia a fallas.

La tolerancia a fallas, es la propiedad de un sistema que le permite continuar realizando su función asignada, aún en la presencia de una o más fallas de sus componentes de hardware o software.

Para cumplir con el nivel de disponibilidad del SCD, se tiene que satisfacer el requerimiento de ***tolerancia a fallas*** en la transferencia de información. De esta manera, los requerimientos de transferencia de información no se verán afectados aún con la presencia de falla en algún nodo del SCD.

En resumen, los requerimientos de comunicación del SCD se pueden agrupar en dos clases:

1) Requerimiento de flujo de información, esto es, se requiere de un mecanismo que permita la transmisión de información entre los diferentes nodos componentes del SCD. Esta información puede ser de cuatro tipos:

   a) Estructuras en memoria, que conforman la BDTR.

   Este flujo de información se llevará a cabo desde los Nodos de Adquisición hacia los Nodos de Presentación. Dentro de este requerimiento se incluyen también:

   b) Mensajes.
   c) Comandos y
   d) Archivos.

   Estos tres tipos de transferencia de información podrán realizarse de cualquier nodo hacia cualquier otro nodo del SCD.

2) Requerimiento de tolerancia a fallas, lo que le permitirá al SCD cumplir con una transferencia de información, aún en caso de falla de algún nodo de la red. Para tener una tolerancia a fallas completa, se tendrían que considerar un sinnúmero de circunstancias, en donde se involucrarían tanto componentes de software como componentes de hardware. En este caso, se requiere que la tolerancia a fallas abarque los siguientes dos puntos:

   a) En caso de que algún nodo receptor falle, al momento de estar realizándose una transferencia de mensaje, comando o archivo, debe existir un mecanismo que le permita al nodo transmisor dirigir su información hacia un nodo respaldo. Este nodo respaldo asumirá las funciones de recepción del nodo caído.

   b) Debido a que la transferencia de estructuras es uno de los principales requerimientos del SCD, debe existir un mecanismo que permita que un Nodo de Adquisición asuma la responsabilidad de la transmisión de estructuras, en caso de falla de algún otro Nodo de Adquisición transmisor. Es decir, si un Nodo de Adquisición falla en el momento en que se encuentra transmitiendo estructuras de la BDTR, el mecanismo debe aportar las rutinas necesarias para que otro Nodo de Adquisición asuma las funciones de transmisión del nodo caído, y así cumplir con la disponibilidad en la transmisión de las estructuras.

CAPÍTULO 3: ESPECIFICACIÓN DE UN MECANISMO DE COMUNICACIÓN CON TOLERANCIA A FALLAS PARA UN SCD DE UNA CTE.

3.1 Introducción.

Una vez que se han identificado y definido los requerimientos de comunicación del SCD, se procede a la siguiente fase en el desarrollo de nuestro mecanismo: la especificación del mecanismo de comunicación. Una especificación debe consistir de 5 partes distintas para considerarse completa [9], ésta debe incluir explícitamente:

1) La **especificación de requerimientos**. El objetivo de este punto es el de proveer una definición explícita del problema que se quiere resolver, tomando en cuenta los requisitos del sistema.

2) El **medio ambiente** en el cual el mecanismo de comunicación se diseña, desarrolla y desenvuelve.

3) La **especificación de los servicios** que serán provistos por el mecanismo de comunicación. La especificación de estos servicios generalmente puede ser vista como una serie de primitivas de interacción. Estas primitivas describen de forma clara y precisa los servicios que prestará el mecanismo de comunicación [8]. La descripción de los servicios del mecanismo de comunicación no debe restringirse a la especificación de los servicios proporcionados al usuario, sino que también deben especificarse los servicios que le proporcionan a dicho mecanismo, las capas más bajas a él [7]. Tomando en cuenta una estructura de capas, los servicios proporcionados y requeridos por el mecanismo de comunicación pueden verse en la figura 3.1.

4) El **vocabulario** de mensajes usados para implantar el mecanismo de comunicación, y que está constituido por todos los mensajes que requerirá el mecanismo de comunicación para el desempeño de sus funciones. Aquí se incluyen los mensajes de control, de inicio y terminación de sesiones y mensajes con estructuras de datos.

5) El **formato** de cada mensaje en el vocabulario, es decir, la descripción de la estructura de los campos que forman los mensajes (longitud, tipo y contenido de los campos, etc.).

En cada parte de la especificación se puede definir una jerarquía de elementos. Por ejemplo, el vocabulario del mecanismo de comunicación puede consistir de una jerarquía de clases de mensaje. La definición del formato puede especificarse como mensajes de nivel alto, que pueden estar estructurados por elementos de nivel más bajo, etcétera..

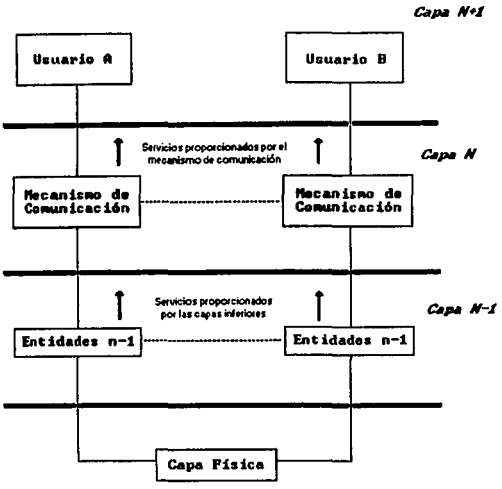

Figura 3.1. Especificación de servicios.

3.2 Especificación de los requerimientos.

En esta etapa de la especificación del mecanismo de comunicación, describiremos el problema que debe resolver el mecanismo, de acuerdo con los requerimientos de comunicación del SCD:

La información adquirida de los procesos en la CTE, es almacenada en una BDTR (Base de Datos en Tiempo Real), la cual está constituida por estructuras en memoria, las cuales se estarán actualizando en forma continua. Estas estructuras se encuentran en cada Nodo de Adquisición y se requiere que la información contenida en las estructuras sea transmitida hacia los Nodos de Presentación, para su consulta y ejecución de las funciones de presentación y control requeridas.

Para satisfacer el requerimiento descrito anteriormente, el mecanismo de comunicación a desarrollar, consistirá de una serie de reglas de procedimiento que permitirán que la información adquirida por los Nodos de Adquisición del SCD sea transmitida a todos los Nodos de Presentación, cada vez que dicha información haya cambiado con relación a un envío previo. Los Nodos de Presentación tomarán la información actualizada, para su procesamiento o presentación. Esta información será almacenada en réplicas de la Base de Datos, con la información requerida por cada nodo (figura 3.2).

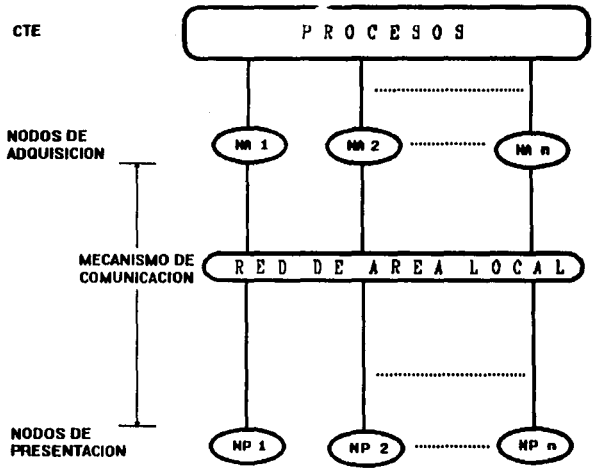

Figura 3.2. Localización del mecanismo de comunicación en el SCD.

Este mecanismo de comunicación deberá proporcionar un medio confiable y eficaz para la transmisión de estructuras de memoria (BDTR), transferencia de mensajes, comandos y archivos entre todos los nodos integrantes de la red.

Para satisfacer los requerimientos de disponibilidad del SCD, el mecanismo de comunicación deberá incluir funciones de tolerancia a fallas, esto es, si algún Nodo de Presentación o Adquisición llegara a fallar, el mecanismo tolerante a fallas deberá ejecutar las rutinas necesarias para evitar la pérdida de información y que dicha falla no sea la causante de la caída de la aplicación.

La tolerancia a fallas se implantará a través de la réplica de funciones en los nodos. Si un nodo llegará a fallar, existirá un nodo de respaldo que asumirá sus funciones. La duplicidad de funciones en los nodos se limitará a las funciones relacionadas con la transmisión y recepción de información, independientemente del tipo de información de que se trate.

Deberá existir un mecanismo tolerante a fallas para los Nodos de Adquisición y uno para los Nodos de Presentación. Esto se debe a la arquitectura del SCD.

Las funciones de tolerancia a fallas para los Nodos de Adquisición consideran un anillo lógico, donde el nodo N será el respaldo de las funciones realizadas por el nodo N+1, de tal manera que el último Nodo de Adquisición será el respaldo del primero. En caso de falla del nodo N, el nodo N-1 asumirá sus funciones, como se muestra la figura 3.3.

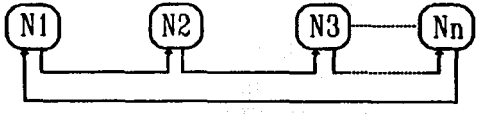

Figura 3.3. Anillo lógico para los Nodos de Adquisición.

El anillo lógico que se muestra en la figura 3.3 no se considera reconfigurable dinámicamente, debido al número máximo de canales de comunicación soportados por cada Unidad de Adquisición, lo cual limita a dos el número de canales interconectados desde las Unidades de Adquisición hacia cada Nodo de Adquisición [1], como se muestra en la figura 3.4.

En el caso que uno de los nodos fallara, el respaldo de este nodo tomará sus funciones de adquisición; sin embargo, si éste llegara a fallar inmediatamente después, antes de que el otro nodo se restableciera en operación, implicaría que el sistema operaría en modo degradado, y se perdería la adquisición de estas señales.

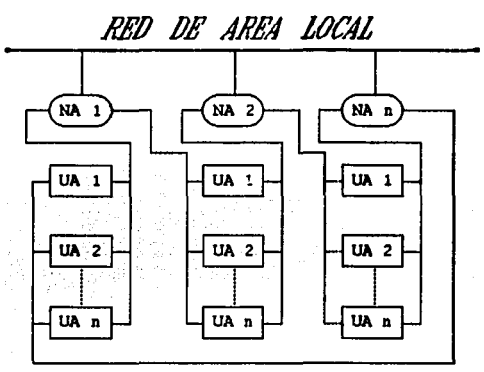

Figura 3.4. Conexión de los Nodos de Adquisición con las Unidades de Adquisición.

Para los Nodos de Presentación, y tomando en cuenta la configuración del SCD (véase figura 1.3), el mecanismo tolerante a fallas estará instalado por parejas, de tal manera que el nodo A será el respaldo del nodo B y viceversa, el nodo B será respaldo del nodo A en caso de falla (figura 3.5). Cabe señalar que el único Nodo de Presentación que no tendría nodo respaldo sería el Nodo del Ingeniero.

Figura 3.5. Tolerancia a fallas en los Nodos de Presentación.

Con lo anterior, podemos decir que el mecanismo de comunicación debe estar compuesto de dos elementos, para satisfacer los requerimientos establecidos:

1) Un protocolo de transferencia, que cumplirá con la función de transferencia de información (estructuras en memoria) desde los Nodos de Adquisición hacia los Nodos de Presentación. Esta transferencia de información se realizará cuando existan cambios en las variables adquiridas y almacenadas en la Base de Datos en Tiempo Real (BDTR). También será capaz de transmitir mensajes, comandos y archivos, entre cualquiera de los nodos.

2) Un protocolo de respaldo, que servirá para implantar los mecanismos de tolerancia a fallas y poder satisfacer la disponibilidad de transferencia de información del SCD. Las funciones que se implanten en un nodo, estarán replicadas en otro nodo, para respaldo en caso de falla. El protocolo de respaldo permitirá que un nodo adquiera las funciones del nodo al que está respaldando, en caso de falla de este último. De esta manera, cada nodo podrá estar en uno de los dos siguientes modos operativos:

   a) Primario-Primario (PP): En este modo operativo el nodo toma sus funciones más las funciones del nodo que está respaldando.

   b) Primario-Respaldo (PR): En este modo operativo el nodo N toma sus funciones y está listo para respaldar al nodo N+1.

La localización de los mecanismos tolerantes a fallas en el SCD puede verse en la figura 3.6.

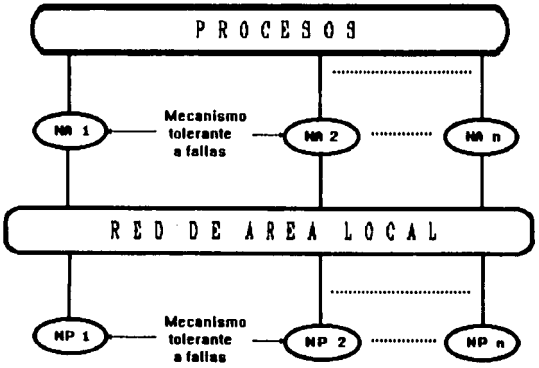

Figura 3.6. Localización de los mecanismos tolerantes a fallas en el SCD.

3.3 Medio Ambiente de desarrollo.

El mecanismo de comunicación a desarrollar en esta tesis se desenvuelve en el SCD para una CTE que se describió en el capítulo 1, y que está basado en el Sistema Operativo en Tiempo Real QNX.

El Sistema Operativo QNX se encuentra en aplicaciones tales como: Robótica, Control de Procesos, Adquisición de Datos en Tiempo Real y Estaciones de Reactores Nucleares.

Este Sistema Operativo fue seleccionado debido a que ofrece una solución completa en términos de Hardware, medio ambiente de desarrollo de Software y costo.

3.3.1 El Sistema Operativo QNX.

La principal responsabilidad del S.O. es administrar los recursos de la computadora. Todas las actividades en el sistema (programación de procesos, ejecución de programas de aplicación, escritura de archivos en disco, envío de datos a través de la red, etc.) deben funcionar, simultáneamente, tan transparentemente como sea posible. Las aplicaciones en tiempo real, por ejemplo, dependen del S.O. para manejar múltiples eventos dentro de un tiempo restringido.

El S.O. QNX es ideal para aplicaciones de tiempo real. Provee la multidemanda y el contexto de enlace rápido, que son los ingredientes esenciales para los sistemas en tiempo real. También es remarcablemente flexible; los diseñadores pueden fácilmente ambientar el S.O. para resolver las necesidades de su aplicación.

3.3.1.1 Ambiente de Hardware.

El S.O. QNX fue diseñado específicamente para microprocesadores de INTEL y puede usarse, por lo tanto, en IBM PC's, XT's, AT's y compatibles; además pueden ser mezcladas y combinadas, o bien adicionar o quitar cualquier tipo de estas computadoras en una red QNX, sin tener que reescribir los programas de aplicación.

El S.O. QNX es multiusuario y multitareas, así, una PC puede soportar hasta 10 dispositivos serie (terminales, modems, impresoras, etc.). Debido a que QNX es multitareas, cualquier máquina corriendo en QNX puede tener hasta 40 programas ejecutándose simultáneamente. Esto significa, por ejemplo, que si se tienen 4 usuarios del sistema, cada uno de ellos podría: Crear un documento, imprimir un documento y compilar un programa.

Para el diseño de redes, QNX ofrece:

a.- Un máximo de 10 dispositivos serie por PC y 16 por AT.

b.- Un máximo de 255 computadoras personales por red.

c.- Dado que QNX tiene capacidad de tiempo real, es adecuado para interactuar con controladores lógicos programables (PLC's), controladores de motores, equipo de adquisición de datos, sistemas de robótica, control de procesos, etc.

3.3.1.2 Ambiente de Software.

El S.O. QNX fue escrito en el lenguaje "C" estándar, un lenguaje conocido por incrementar la productividad de los programadores y con capacidad total para manejar multitareas y tiempo real.

QNX tiene disponibles un gran número de valiosas herramientas de desarrollo de software, entre las que destacan:

1) Un compilador de "C", para producir código optimizado.

2) Alrededor de 300 rutinas de librería (Watcom C).

3) Independencia de terminales entrada/salida y compatibilidad con UNIX.

4) Alrededor de 90 utilerías entre las que se encuentran MAKE, SPOOL y TALK (para comunicaciones con mainframes o micros a través de modems).

5) Un editor "full-screen" (vedit).

6) Un administrador que permite accesar archivos y directorios sobre DOS, como si fueran archivos y directorios de QNX.

7) Un filtro que traduce programas de QNX a formato DOS.

3.3.1.3 Características principales.

El sistema operativo QNX coordina y administra el uso de los recursos y suministra servicios tales como: ejecución remota de programas, administración de archivos, sincronización, intercomunicación entre procesos, etcétera. El S.O. QNX basa su eficiencia, modularidad y simplicidad en dos características fundamentales:

a) Arquitectura de microkernel (micronúcleo).

b) Comunicación entre procesos basada en mensajes (IPC).

El S.O. QNX consiste de un pequeño kernel (núcleo) a cargo de un grupo de procesos cooperativos (véase figura 3.7). El kernel es el corazón del S.O. y está dedicado sólo a 2 funciones esenciales:

1.- Paso de mensajes: El kernel supervisa el ruteo de todos los mensajes enviados entre todos los procesos a través de todo el sistema, también administra la IPC.

2.- Programación de ejecución de procesos: El *calendarizador* es una parte del kernel, y se invoca cuando un proceso cambia de estado como resultado de un mensaje o una interrupción. El *calendarizador* del kernel, decide a cuál proceso se le dará el control del CPU.

Los servicios de QNX, excepto los provistos por el kernel, son manejados por medio de los procesos estándares QNX. Una configuración típica de QNX, tiene los siguientes procesos de sistema:

a) Administrador de Procesos (Proc).

b) Administrador del Sistema de Archivos (Fsys).

c) Administrador del Dispositivos (Dev).

d) Administrador de Red (Net).

Los procesos del sistema son prácticamente iguales a cualquier programa escrito por un usuario; no tienen interfaces privadas o escondidas que no estén disponibles para los procesos del usuario. Esto es lo que le da a QNX una gran extensibilidad. Ya que la mayoría de los servicios del S.O. son provistos por los procesos estándar de QNX, es muy ampliar las capacidades del mismo S.O., con sólo crear nuevos programas, que provean nuevos servicios.

Administrador de Procesos (PROCESS MANAGER).

El *administrador de procesos* (Proc) trabaja en conjunto con el kernel para proveer servicios esenciales del S.O. El Proc trabaja como un proceso propio, por lo cual es programado para ejecutarse como cualquier otro proceso y usa las primitivas de paso de mensaje del kernel para comunicarse con otros procesos en el sistema.

El Proc es responsable de crear nuevos procesos en el sistema y administrar los recursos fundamentales asociados con un proceso. Estos servicios son todos provistos por medio de mensajes; por ejemplo, si un proceso ejecutándose quiere crear un nuevo proceso, lo hace enviando un mensaje conteniendo los detalles del nuevo proceso que será creado.

Administrador del Sistema de Archivos (FILESYSTEM MANAGER).

El *administrador del sistema de archivos* (Fsys) provee medios estandarizados para guardar y accesar datos en subsistemas de disco. El Fsys es responsable del manejo de todas las solicitudes de escritura, lectura, apertura y cierre de archivos.

En QNX un archivo es un objeto que puede ser leído o escrito. Por esto implementa cuatro tipos de archivo, que son:

1.- Un *archivo regular*, que es una secuencia de bytes accesibles aleatoriamente y que no tienen una estructura interna predefinida. Los programas de aplicación son los responsables de entender la estructura y contenido de cualquier archivo regular específico.

2.- Un *directorio* es un archivo que contiene entradas de archivos. Cada entrada de directorio asocia un nombre con un archivo. Un nombre de archivo es un nombre simbólico que permite identificar y accesar un archivo.

3.- Una *pipa* es un archivo sin nombre que sirve como un canal de E/S entre dos procesos cooperativos, de tal manera que un proceso escribe en la pipa y el otro lee de la pipa. El Fsys se ocupa de guardar los datos y sincronizar a los dos procesos.

4.- Los archivos *FIFO* son esencialmente lo mismo que las pipas, excepto que las pipas son archivos sin nombre, mientras que las FIFO son archivos permanentes con nombre y están guardados en directorios del sistema de archivos.

Administrador de Dispositivos (DEVICE MANAGER).

El *administrador de dispositivos* (Dev) es la interface entre los procesos y los dispositivos de terminal. Un dispositivo de terminal es presentado a un proceso QNX, como un flujo bidireccional de bytes que pueden ser leídos o escritos por el proceso.

El Dev regula el flujo de datos entre una aplicación y un dispositivo. El procesamiento de estos datos es realizado por Dev, de acuerdo con ciertos parámetros definidos en una *estructura de control terminal* (llamada **termios**), la cual existe para cada dispositivo. Los parámetros **termios** incluyen la disciplina de control de línea (razón de transferencia en baudios, paridad, etc.).

El Dev también provee un grupo de servicios auxiliares, disponibles para los procesos, para la administración de dispositivos, tales como:

- Realizar operaciones de lectura.
- Notificar a un proceso de la disponibilidad de datos en dispositivos.
- Desconectar un canal de comunicación.
- Insertar datos de entrada.

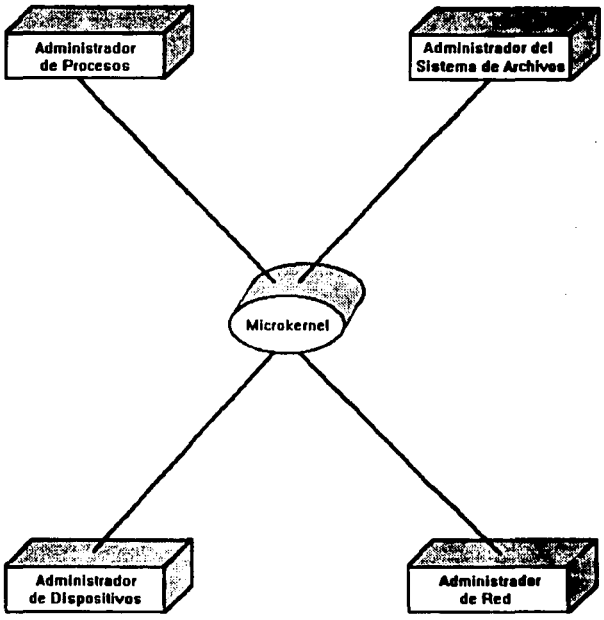

Figura 3.7. Arquitectura del Sistema Operativo Q.N.X.

Administrador de Red (NETWORK MANAGER).

El *administrador de red* (Net) les da a los usuarios de QNX una extensión de las capacidades del IPC del sistema. Comunicándose directamente con el kernel, el Net mejora al IPC, mediante la propagación de mensajes hacia procesos remotos, es decir, el Net es responsable de la propagación de las primitivas de mensajeo de QNX a través de la LAN.

En su forma más simple, una red de área local (LAN) provee mecanismos para compartir archivos y dispositivos periféricos entre varias computadoras interconectadas. El S.O. QNX va mucho más allá de este simple concepto e integra a toda la red en un grupo simple y homogéneo de recursos. Cualquier proceso, en cualquier máquina, en la red puede hacer uso directamente de cualquier recurso, de cualquier otra máquina, localizada en cualquier otro punto de la red.

Desde la perspectiva de las aplicaciones, no hay diferencia entre un recurso local y uno remoto, y no se necesitan elementos especiales en las aplicaciones para hacer uso de los recursos remotos. Los usuarios pueden tomar el servicio de cualquier dispositivo periférico y correr las aplicaciones en cualquier máquina en la red (suponiendo que se tienen los privilegios apropiados). Los procesos pueden comunicarse a cualquier lugar a través de toda la red. El IPC de QNX hace fluida y transparente a la red.

Como las aplicaciones pueden accesar todos los servicios por medio de mensajes y dado que el Net le permite a los mensajes fluir transparentemente sobre la red, los nodos de QNX actúan juntos como una única computadora lógica.

3.4 Especificación de los servicios.

El mecanismo de comunicación que se desarrolla en esta tesis, está basado en la transferencia de información desde nodos fuente hacia nodos destino. El propósito principal del mecanismo de comunicación consiste en implantar servicios confiables de transferencia fin-a-fin (usuario a usuario) de estructuras en memoria (BDTR), archivos, mensajes y comandos. Estos servicios incluyen el establecimiento y terminación de enlaces con los nodos remotos, sesiones de transferencia de información y tolerancia a fallas, en caso de falla en algún nodo de la red.

Por lo anterior, se puede establecer que el mecanismo de comunicación puede clasificarse dentro de los dos siguientes tipos, descritos a continuación:

a) Como un protocolo *punto-a-punto*, es decir, cuenta con un transmisor y un receptor, en los casos de transferencia de información (mensajes, archivos, etc.) entre dos nodos.

b) Así también como un protocolo *multipunto*, ya que tiene más de un receptor (en el caso de transferencia de estructuras en memoria hacia los Nodos de Presentación).

De cualquier manera, el mecanismo de comunicación provee un servicio usuario-a-usuario entre diferentes nodos.

Para poder implantar todos estos servicios, el mecanismo de comunicación debe beneficiarse y hacer uso de los servicios que le proporcionan las capas más bajas del modelo OSI.

Como se muestra en la figura 3.8 y de acuerdo con el modelo OSI de la ISO, visto en el capítulo 2, el mecanismo de comunicación se basa en los servicios ofrecidos por ARCNET, que corresponden a la capa dos del modelo OSI, y en los servicios ofrecidos por el S.O. QNX, los cuales corresponden a las capas cuatro y cinco de dicho modelo.

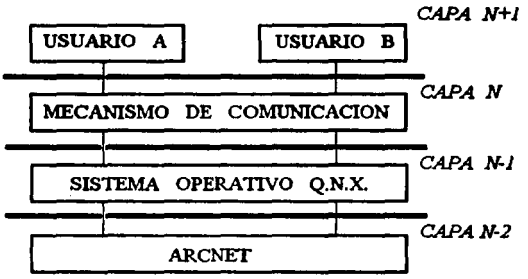

Figura 3.8. Capas de servicio proporcionadas a la aplicación.

3.4.1 Servicios proporcionados por ARCNET.

Como se dijo anteriormente, la topología de la red de comunicaciones del SCD para la CTE está basada en un bus simple. La topología se refiere a la manera en que los nodos de la red están interconectados.

En la topología bus todos los nodos se ligan, a través de las interfaces hardware apropiadas, directamente a un medio de transmisión lineal o bus. Una transmisión de cualquier nodo en la red se propaga por el medio y puede ser recibida por todos los otros nodos de la red.

3.4.1.1 Características de ARCNET.

La red sobre la que se basa nuestro SCD es la ARCNET, que tiene la capacidad de soportar hasta 255 nodos y con una velocidad de transmisión de 2.5 Mbps. El controlador de ARCNET (AC) y su circuitería de soporte se localizan en un módulo de interface de red (NIM), el cual interfaza una estructura de bus de microcomputadora a la ARCNET. El NIM se ensambla en una ranura disponible de una PC y se consideran (el NIM y la PC) como un nodo en ARCNET con una dirección única. Las direcciones válidas abarcan de 0 a 255, estando la dirección 0, reservada para mensajes que requieran dirigirse a todos los nodos de la red.

ARCNET cuenta con un mecanismo de acceso al medio denominado "token passing bus". En este sistema, los nodos en el bus forman un anillo lógico, esto es, los nodos son posiciones lógicas asignadas en una secuencia ordenada, con el último miembro de la secuencia seguido por el primero. Cada nodo sabe la identidad de su nodo anterior y posterior. El orden físico de las estaciones en el bus es independiente del orden lógico, como lo muestra la figura 3.9.

Como todos los nodos comparten un medio de transmisión común, sólo uno de ellos puede transmitir a la vez. Para esto, se requiere de un control de acceso que determine cuál nodo puede transmitir. Para evitar contención en la comunicación, existe un "frame" de control (grupo de bits) conocido como "token" que regula el derecho de acceso al medio.

El "token" pasa de un nodo a otro en orden ascendente, según el valor de sus direcciones, formando el anillo lógico. El nodo que recibe el "token" tiene el privilegio de controlar el medio; éste puede entonces transmitir datos y puede elegir a su receptor. Cuando el nodo termina o no tiene datos que transmitir, éste pasa el "token" al siguiente nodo de la secuencia lógica; por lo que, la operación consiste de la alternación del "token" entre los nodos de la red y de transferencias de datos mientras se tiene el "token".

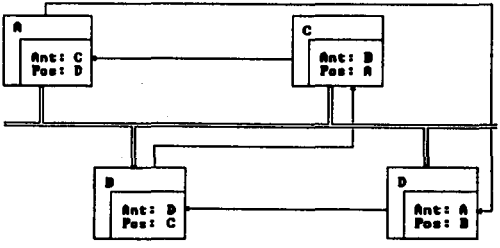

Figura 3.9. Sistema "Token Passing Bus".

3.4.1.2 Tipos de transmisión.

El protocolo que sigue la red ARCNET está conformado por cinco tipos de transmisión básicos [1], que son:

ITT: Invitación para transmitir.
FBE: Consulta de buffer vacío.
PAC: Paquetes.
ACK: Reconocimiento.
NAK: No reconocimiento.

Cada uno de estos tipo de transmisión son precedidos por una ráfaga de alerta, que está constituida por seis intervalos de caracteres nulos consecutivos.

Como el protocolo de ARCNET esta orientado a caracteres, cada tipo de transmisión se descompone en series de caracteres de ocho bits. Antes de que se transmita un caracter, ARCNET envía dos intervalos de marcas y un intervalo de espacio.

Invitación a transmitir (ITT).

El ITT proporciona el control de la red al nodo que lo recibe y es referido como el "token". El "token" se transfiere cuando el nodo que está enviado datos termina su transmisión, o cuando un nodo recibe el "token" y no tiene datos que transmitir. El formato de este tipo de transmisión es el siguiente:

ITT =	ALERT	EOT	DID	DID

Donde los campos son: ALERT (ráfaga), EOT (End of Transmission) y DID (Destination Identification number).

Consulta de buffer vacío (FBE).

Antes de que un nodo receptor reciba datos, éste debe ser consultado, para ver si tiene un buffer libre para aceptar los datos. Para verificar la existencia de este buffer vacío, el nodo transmisor envía un FBE. Si el nodo receptor envía una respuesta positiva (ACK), el nodo transmisor puede, entonces, enviar los datos. El formato del FBE es el siguiente:

FEB =	ALERT	ENQ	DID	DID

Donde el primer caracter es un caracter ENQ en ASCII (05 hex) el cual significa consulta.

Paquetes (PAC).

Mediante este tipo de transmisión, los datos se transfieren entre los nodos como paquetes. ARCNET soporta paquetes de longitud variable y envía solamente los caracteres mínimos necesarios que se requieren transmitir. ARCNET tiene un tamaño de paquete mínimo de un caracter y un tamaño de paquete máximo de 508 caracteres.

Solamente un paquete se puede enviar antes que el nodo transmisor deba pasar el "token", es decir, los mensajes que excedan la longitud máxima permitida de paquete requerirá de múltiples transmisiones para que se transmita el mensaje completo, por lo que tendrán que ocurrir varios viajes a través del anillo lógico antes de que se transmita el mensaje completo.

El formato utilizado para este tipo de comunicación se muestra a continuación:

PAC =	ALERT	SOH	SID	DID	DID	CP	DATOS	DATOS	CRC	CRC

La transmisión de paquetes comienza con una ráfaga, el primer caracter es el SOH en ASCII (01 hex) que significa el comienzo del encabezado. El siguiente caracter es el SID, el cual contiene la dirección del nodo origen (una de las 255 direcciones posibles). Después sigue el DID que requiere de dos caracteres, después ARCNET adiciona el CP (apuntador de continuación).

El CP le informa al nodo de destino en que localidad de su memoria se localizará el inicio de los datos transmitidos. Después se envían los datos (de 1 a 508 caracteres). Los caracteres de datos son seguidos por dos caracteres CRC (Cyclic Redundancy Check) que forman una palabra de 16 bits que verifican la integridad de los datos en el paquete. El valor del CRC es determinado por el patrón de datos y es adicionado por el nodo transmisor.

El nodo receptor lleva a cabo los mismos cálculos y compara sus resultados con el CRC de los datos transmitidos. El CRC debe coincidir, sino los datos se consideran erróneos y el paquete debe ser retransmitido en el siguiente turno del nodo transmisor.

Reconocimiento (ACK).

La transmisión de un ACK tiene como propósito reconocer positivamente un PAC o un FBE (el paquete fue recibido o el buffer de recepción está vacío, respectivamente).

Se considera que solamente el nodo destino generará un ACK, y el nodo de origen es el único que espera por la respuesta, por lo que no se requiere direccionamiento del nodo. La transmisión del ACK comienza con una ráfaga seguida por el código ASCII ACK (06 hex). Su formato es:

ACK =	ALERT	ACK

No reconocimiento (NAK).

Este reconocimiento negativo es generado por el nodo receptor, cuando su buffer no está disponible o cuando un paquete de datos se recibió alterado. La transmisión del NAK comienza con una ráfaga de bits y es seguida por un NAK en ASCII (15 hex).

NAK = | ALERT | NAK |

3.4.1.3 Servicios de ARCNET.

Los servicios que ofrece ARCNET se pueden discutir de acuerdo como se comporta la red. Podemos englobar dichos servicios en: paso del "token", configuración y reconfiguración de la red, baja de un nodo en la red, alta de un nodo en la red y transmisión de datos.

Paso del "token".

Cuando un nodo recibe un ITT, éste checa si tiene datos que transmitir. Si no tiene datos, pasa el "token" (ITT) al siguiente nodo activo con la dirección inmediatamente mayor que la actual (NID, Next Destination Identification Address).

El nodo que recibe el ITT tiene, momentáneamente, el derecho de transmitir datos a través del medio. Así, el "token" continúa circulando en el anillo lógico.

Configuración y reconfiguración.

Antes de que la red inicie el paso del "token", o cuando ocurre una falla que altere el paso del "token", la red debe ser configurada o reconfigurada, respectivamente. En ambos casos ARCNET requiere la identificación de todos los nodos activos, por lo que se consideran como un mismo proceso.

Cuando un nodo activo falla en la recepción de un ITT después de 840 µs o si el nodo se energiza, éste generará una ráfaga de bits de reconfiguración (RECON). Esta ráfaga está constituida por una serie de ocho intervalos de marca seguidos por un intervalo de espacio repetidos 765 veces.

Con lo anterior, el nodo estará activo por 2754 µs lo cual alterará el paso exitoso del "token"; la ráfaga RECON excede el máximo tiempo de transmisión y subsecuente recepción del siguiente ITT. El nodo que debería recibir el "token" no obtiene el control del medio y el nodo que transmite el "token" libera el control de la red, provocando que el "token" se pierda.

Después de 78 µs de inactividad, todo los nodos se percatan de que está ocurriendo una reconfiguración. Cada nodo, entonces, inicializa un temporizador, calculado de la siguiente manera:

temporizador = 146 µs * (255 - ID)

donde ID es el identificador de dirección de cada nodo.

El nodo que experimenta un temporizador primero (el nodo con la dirección más alta), envía un ITT a su NID y luego escucha la actividad en la línea. Si no ocurre *ninguna* actividad, después de 44 µs, el nodo transmisor asume que el nodo con el NID no existe o está inactivo. Entonces incrementa el NID y envía otro ITT.

Si el nodo direccionado por NID existe, éste asume el control de la red y prosigue con el método anterior. Este proceso continúa hasta que todos los nodos activos son determinados. Después de esto, el "token" comienza a viajar a través de la red.

Baja de un nodo en la red.

Cuando un nodo se da de baja, generará actividad en la línea por un período de 74 µs, cuando se le quiera transmitir el "token". El nodo predecesor al nodo dado de baja detectará esto y supondrá que su nodo sucesor está inactivo, por lo que incrementará su NID y transmitirá el "token" a dicha dirección. El nodo inactivo será excluido de las sucesivas rondas en el anillo lógico.

Alta de un nodo en la red.

Cuando un nodo se da de alta debe esperar 840 ms, si en ese lapso no recibe el "token" entonces dicho nodo generará una ráfaga de reconfiguración que alterará el paso del "token". Después se sigue el procedimiento de reconfiguración que se explico anteriormente, con la finalidad de integrar al nuevo nodo al anillo lógico.

Transmisión de datos.

Cuando un nodo recibe el "token" y tiene datos que transmitir, envía un FBE a su nodo receptor. Si el nodo receptor tiene disponible su buffer entonces retorna un ACK. En caso contrario enviará de regreso un NAK.

Si el nodo transmisor recibe un NAK, simplemente pasará el "token" al siguiente nodo y esperará su turno para volver a intentar la transmisión de datos. Si, por el contrario, el nodo transmisor recibe un ACK entonces enviará un PAC.

El nodo receptor del PAC verificará la integridad de los datos. Si no existe ningún error enviará de retorno un ACK. El nodo transmisor al recibir el ACK cederá el "token" al siguiente nodo. Si el nodo transmisor no recibe respuesta dentro de un lapso de 74 µs entonces supone que el mensaje se perdió e intentará la transmisión en su siguiente turno.

3.4.2 Servicios proporcionados por el S.O. QNX.

Cuando varios procesos corren simultáneamente, como es el ambiente de un SCD en tiempo real, el S.O. debe proveer servicios que le permitan a los procesos comunicarse entre ellos. El S.O. QNX provee un grupo simple y poderoso de servicios de IPC (Inter-Process Communication), que simplifican el trabajo para los diseñadores de aplicaciones compuestas por procesos cooperativos.

El S.O. QNX se basa en el principio de que: "*la comunicación efectiva es la llave de una operación efectiva*". La IPC, entonces, forma la piedra angular de la arquitectura de QNX y mejora la eficiencia de todas las transacciones entre todos los procesos a través de todo el sistema.

El sistema operativo QNX soporta varios tipos de IPC, que son: la IPC por medio de mensajes, por medio de "proxies" y por medio de señales. Estos tipos de IPC se describen a continuación:

a) **Mensajes:** Es la forma fundamental de IPC en QNX. Provee comunicación síncrona entre procesos cooperativos, donde el proceso que envía los mensajes requiere de una comprobación de recepción por parte del receptor y, además, potencialmente de una respuesta al mensaje.

b) **Proxies:** Son una forma especial de mensaje. Están hechos especialmente para la notificación de eventos, donde el proceso transmisor no necesita interactuar con el proceso receptor.

c) **Señales:** Son la forma tradicional de IPC. Son usadas para soportar comunicación entre procesos asíncrona.

3.4.2.1 IPC con mensajes.

En QNX, un mensaje es un paquete de bytes que son transmitidos sincronamente de un proceso a otro. El S.O. QNX no le toma especial importancia al contenido de un mensaje, de hecho los datos de los mensajes sólo tienen significado para el transmisor y el receptor y no para otros procesos.

Para comunicarse directamente con otro, los procesos cooperativos pueden utilizar una serie de funciones disponibles en la Librería C de QNX (Watcom C), y que pueden ser usadas localmente o a través de la red:

1) Para envío de mensajes: **Send(pid,smsg,rmsg,smsg_len,rmsg_len)**.
Donde:

pid es el ID (identificador) del proceso receptor del mensaje.
smsg es el buffer del mensaje que será enviado.
rmsg es el buffer que contendrá la respuesta del proceso receptor.
smsg_len es la longitud del mensaje que será enviado.
rmsg_len es la longitud del mensaje de respuesta.

2) Para recepción de mensajes: **Receive(pid,msg,msg_len)**.
Donde:

pid es el ID del proceso transmisor.
msg es el buffer donde se guardará el mensaje recibido.
msg_len es la longitud del mensaje que será aceptada en el buffer de recepción.

Si el **smsg_len** en **Send()** y el **msg_len** en **Receive()** no son del mismo tamaño, el más pequeño de los dos determinará la cantidad de datos que serán transferidos.

3) Para réplica de mensajes: **Reply(pid,msg_reply, msg_reply_len)**.
Donde:

pid es el ID del proceso al cual va dirigida la respuesta (proceso transmisor).
msg_reply es el buffer de respuesta.
msg_reply_len es la longitud del mensaje de respuesta.

Si **msg_reply_len** en **Reply()** y **rmsg_len** en **Send()** no son del mismo tamaño, el más pequeño de los dos determinará cuantos datos serán transferidos.

En la figura 3.10 se ilustra una secuencia simple de eventos en donde dos procesos, A y B, utilizan las funciones anteriores para comunicarse entre ellos. Los pasos que se siguen en la comunicación entre procesos son los siguientes:

1) El proceso A manda un mensaje al proceso B emitiendo una solicitud **Send()** al kernel. En este punto, el proceso A se bloquea (*Send BLOCKED*) hasta que el proceso B emita un **Receive()** para recibir el mensaje.

2) El proceso B emite un **Receive()** y recibe el mensaje que estaba en espera (enviado por el proceso A). El proceso A cambia a un estado de bloqueo de espera de respuesta (*Reply BLOCKED*). Ya que un mensaje estaba en espera, el proceso B no se bloquea.

Si el proceso B hubiera emitido un **Receive()** antes de que el mensaje se enviara, el proceso B entraría a un estado de bloqueo de recepción (*Receive BLOCKED*) hasta que el mensaje arribara. En este caso, el transmisor se cambiaría inmediatamente al estado de bloqueo de espera (*Reply BLOCKED*) cuando enviara su mensaje.

3) El proceso B completa el proceso asociado con el mensaje que recibió del proceso A y envía una respuesta por medio de **Reply()**. El mensaje de respuesta es copiado al proceso A, que puede continuar ejecutándose. Un **Reply()** no bloquea al proceso B, que también está listo para continuar su procesamiento.

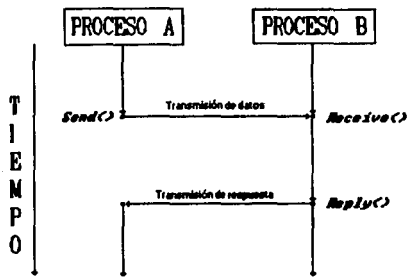

Figura 3.10. Comunicación entre procesos (IPC).

El paso de mensajes no sólo permite a los procesos transmitir datos a otros, sino que también provee medios de sincronización de ejecución de varios procesos. Desde que envían, reciben y responden a mensajes, los procesos soportan varios "cambios de estado" cuando se encuentran ejecutándose. Esto se puede ver claramente en el ejemplo de la figura 3.10.

Una vez que el proceso A emite la solicitud **Send()**, es incapaz de reanudar su ejecución hasta que haya recibido la respuesta al mensaje enviado anteriormente. Esto asegura que las acciones ejecutadas por el proceso B para el proceso A están completas antes que el proceso A pueda reanudar su ejecución. Más aún, una vez que el proceso B emite su solicitud de **Receive()**, no puede continuar su proceso hasta que reciba un mensaje.

Se deben mencionar algunos otros aspectos del IPC que deben tomarse en cuenta:

a) Los datos del mensaje son mantenidos en el proceso de envío, hasta que el receptor este listo para procesar el mensaje. No se hace ninguna copia del mensaje dentro del kernel. Esto sucede, dado que el proceso transmisor está en un estado *Send BLOCKED* y es incapaz de modificar inadvertidamente los datos del mensaje.

b) Los datos del mensaje de respuesta se copian del proceso que respondió hacia el proceso *Reply BLOCKED* como una operación atómica, cuando la solicitud **Reply()** es emitida. El proceso *Reply BLOCKED* se desbloquea después que los datos han sido copiados dentro de su espacio.

c) Si es necesario, el mensaje y/o la respuesta pueden tener una longitud de cero.

d) El proceso transmisor no necesita saber nada acerca del estado del proceso receptor antes de enviar un mensaje. Si el proceso receptor no está preparado para recibir un mensaje cuando el proceso transmisor lo envíe, este último simplemente se bloqueará (*Send BLOCKED*).

e) Puede haber mensajes de varios procesos con destino a un sólo proceso. Normalmente el proceso receptor recibe los mensajes en el orden en que fueron enviados por los otros procesos; sin embargo, el proceso receptor puede especificar que los mensajes serán recibidos en un orden basado en la prioridad de los procesos transmisores.

3.4.2.2 IPC con proxies.

Un "proxy" es una forma de mensaje sin bloqueo, que está especialmente hecha para la notificación de eventos, donde el proceso transmisor no necesita interactuar con el receptor, esto es, el proceso transmisor no se bloquea o tiene que esperar por una respuesta. La única función de un "proxy" es enviar un mensaje a un proceso específico que reconozca el "proxy". Como los mensajes, los "proxies" trabajan a través de toda la red.

Algunos casos en que es necesario utilizar "proxies" entre procesos son los siguientes:

a) Un proceso quiere notificar a otro proceso que un evento ha ocurrido, pero no puede bloquearse hasta que el receptor envíe un **Receive()** y un **Reply()**.

b) Un proceso quiere enviar datos a otro proceso, pero no necesita ni respuesta ni otro tipo de reconocimiento de que el proceso receptor ha recibido el mensaje.

Los "proxies" se pueden crear con la función de la Librería C de QNX: **proxy = qnx_proxy_attach(pid,msg,msg_len,priority)**.
Donde:

proxy será el identificador del "proxy".
pid es el ID del proceso receptor del "proxy".
msg es el buffer del mensaje.
msg_len es la longitud del mensaje.
priority es la prioridad del "proxy".

Ahora bien, para enviar el "proxy" hacia el proceso elegido se utiliza la función **Trigger(proxy)**. Donde **proxy** es el identificador del "proxy" que se creó con la función **qnx_proxy_attach()**.

3.4.2.3 IPC con señales.

Las señales son eventos asincronos que interrumpen un proceso. Permiten una forma tradicional y muy primitiva de comunicación entre procesos. Una señal se libera (envia) hacia un proceso, cuando una acción definida por el proceso para esa señal se realiza. Un proceso puede enviar una señal hacia si mismo.

Un proceso puede recibir una señal en una de las siguientes tres formas, dependiendo de cómo fue definido el ambiente de manejo de la señal:

1) Si el proceso no ha tomado alguna acción especial para manejar señales, se lleva a cabo la acción por "default". Usualmente esta acción es para terminar el proceso.

2) El proceso puede ignorar la señal. Si un proceso ignora una señal, no hay efecto en el proceso cuando la señal es liberada hacia éste.

3) El proceso puede proveer un manejador de señales para la señal. Un manejador es una función en el proceso que se invoca cuando la señal es liberada. Cuando un proceso contiene un manejador de señales, se dice que es capaz de 'atrapar' a la señal. Cualquier proceso que 'atrapa' una señal, está recibiendo una forma de interrupción de software.

Alguna señales que se manejan en QNX se mencionan a continuación. En la mayoría de estas señales, la acción por "default" es la terminación anormal del proceso:

SIGINT: Señal de atención interactiva (<CTRL><C> en el teclado).
SIGQUIT: Señal de terminación interactiva.
SIGILL: Instrucción ilegal encontrada.
SIGFPE: Operación aritmética errónea.
SIGKILL: Señal de terminación. No puede ser 'atrapada' ni ignorada.
SIGBUS: Error de bus.
SIGSEGV: Detección de una referencia ilegal de memoria.
SIGPWR: Señal de caída de energía.

Para enviar una señal especifica hacia un proceso especifico, se utiliza la función: **kill(pid, sig)**. Donde:

pid es el identificador del proceso al que se quiere enviar una señal determinada.
sig es la señal que se requiere enviar.

Ahora bien, para determinar una acción (que no sea la acción por "default") que será realizada cuando se reciba una señal especifica se utiliza la función: **signal(sig, *func)**.

Donde:

sig es la señal que será 'atrapada' por el manejador de señales.
***func** es un puntero a una función. Esta función será la que manejará la señal especificada.

3.4.2.4 Circuitos Virtuales.

En el S.O. QNX, un proceso puede comunicarse con otro proceso, en otra computadora de la red, como si estuviera hablando con otro proceso en la misma máquina. De hecho, desde el punto de vista de aplicaciones, no hay diferencia entre un recurso local y uno remoto. Este remarcable grado de transparencia es posible gracias a los 'circuitos virtuales', que son caminos que el administrador de red (Net) provee para transmitir mensajes, "proxies" y señales a través de la red.

Un proceso transmisor en un nodo local es responsable de inicializar un circuito virtual entre él y el proceso en un nodo remoto con el que se quiere comunicar; para esto, el proceso transmisor usualmente emite una llamada a la función de la Librería C de QNX: **qnx_vc_attach(nid,pid,buffer)**.
Donde:

nid es el nodo donde se encuentra el proceso remoto.
pid es el proceso remoto con quien se quiere ligar..
buffer es el buffer que se creará en ambos extremos.

Una vez que el circuito virtual ha sido creado, los dos procesos, en distintos nodos, pueden comunicarse utilizando las funciones antes mencionadas (**Send(), Receive()** y **Reply()**).

Además de crear un circuito virtual, esta función también crea un proceso virtual ID o VID, en cada extremo del circuito. El VID (identificador virtual) parece tener el ID del proceso remoto al que se quiere comunicar.

Para comprender lo anterior más claramente, vea el ejemplo de la figura 3.11. Un circuito virtual conecta el PID 1 al PID 2. En el nodo 1, donde reside el PID 1, un VID 2 representa el PID 2. En el nodo 2, donde reside el PID 2, un VID 1 representa al PID 1. Ambos, PID 1 y PID 2 pueden referirse al VID en sus nodos como si fueran cualquier otro proceso local. Así, el PID 1 puede mandar un mensaje al VID 2 en su extremo y este VID 2 enviará el mensaje a través de la red al VID 1, representando al PID 1 en el otro extremo. Este VID 1 entonces ruteará al mensaje al PID 2.

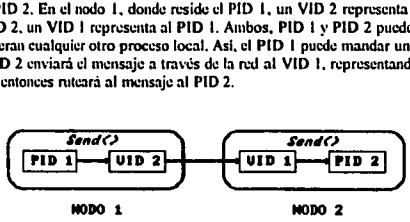

Figura 3.11. Circuitos Virtuales.

Si existe la necesidad de comunicarse con un proceso remoto, pero no se cuenta con su identificador (PID), la amplia gama de funciones de la Librería C de QNX nos permite ligar a dicho proceso con un circuito virtual mediante las funciones: **qnx_name_locate(nid, "p_name", buffer)** o **qnx_vc_name_attach(nid, buffer, "p_name")**.
Donde:

nid es el nodo donde se encuentra el proceso remoto.
p_name es una cadena de caracteres que indican el nombre bajo el cual está registrado el proceso remoto y
buffer es la longitud del buffer de comunicación.

Ambas funciones realizan la misma tarea: enlazar a un proceso remoto por medio de un circuito virtual. Los circuitos virtuales contribuyen al uso eficiente de los recursos de una red QNX por varias razones:

a) Cuando un circuito virtual es creado, se le da la habilidad de manejar mensajes de una longitud específica. Pero, si se necesita enviar mensajes más grandes que dicha longitud, el circuito virtual es automáticamente reajustado para acomodar el mensaje más grande que se genere en la comunicación.

b) Si un proceso se liga a un circuito virtual existente y solicita un buffer más grande que el que se está usando, el tamaño del buffer es automáticamente incrementado.

c) Cuando un proceso termina, su circuito virtual asociado es automáticamente desligado.

Gracias a la gran variedad de funciones que nos ofrece la Librería C de QNX se pueden implementar infinidad de aplicaciones donde la comunicación interprocesos esté involucrada facilitando, en mucho, la tarea de implementación de "Sistemas IPC".

3.4.3 Servicios proporcionados por el mecanismo de comunicación.

Los servicios ofrecidos por el mecanismo de comunicación proporcionarán, de manera transparente al usuario (que pueden ser programas de aplicación), los servicios requeridos para una transferencia eficiente y confiable de información a través de la red.

Este mecanismo de comunicación será visto por los usuarios como una caja negra, donde sus entradas serán la información que se requiere transmitir y el destino de la información, y la salida será esa misma información pero en el nodo destino (véase figura 3.12).

La especificación de los servicios se realiza a través de las interacciones entre los usuarios del mecanismo y el mecanismo en sí mismo. Por lo tanto, la especificación de los servicios está basada en un conjunto de primitivas de servicio que, de manera abstracta, describen las operaciones de la interface a través de la cual son accesados los servicios.

Figura 3.12. El mecanismo de comunicación como *caja negra*.

Las primitivas de servicio son abstractas en el sentido que describen las operaciones realizadas y sus efectos, pero deja abiertos los formatos exactos y los mecanismos que se utilizaran para llevarlas a cabo, es decir, para su implementación (por ejemplo: llamadas a procedimientos, funciones, parámetros de entrada, etcétera). En seguida se realiza una definición formal de todos los servicios proporcionados a los programas de aplicación y usuarios del mecanismo de comunicación.

3.4.3.1 Establecimiento de enlaces.

Mediante este servicio un nodo transmisor (fuente) podrá requerir el establecimiento de un enlace con cualquier nodo remoto (receptor) del SCD. Su sintaxis la podemos describir como:

rpid = conexión(nodo);

Donde *rpid* no indicará si hubo éxito en el enlace. Un valor de -1 nos indicará un enlace fallido, de otra manera *rpid* contendrá el identificador del proceso receptor. El parámetro *nodo* nos indicará el nodo con el que se requiere el establecimiento del enlace.

3.4.3.2 Transferencia de estructuras.

Este servicio será ofrecido a los procesos de adquisición residentes en los Nodos de Adquisición. La información procedente de los procesos de la CTE se adquiere y almacena en estructuras en memoria principal, que son la imagen de la BD del SCD, por medio de procesos de adquisición. Este tipo de almacenamiento se realiza en estructuras en memoria, dado que el SCD trabaja en tiempo real. La alternativa de almacenamiento en disco (acceso, almacenamiento, etc.) consume demasiado tiempo, por lo que es poco factible para trabajar en tiempo real.

Una vez que los "procesos de adquisición" han adquirido las variables de los procesos de la CTE, llevarán a cabo una comparación con los valores anteriores de dichas variables. Si se detecta un cambio en algún parámetro de las variables, estos procesos solicitarán el servicio de transferencia de estructuras al mecanismo de comunicación, el que a su vez, que se encargará de transferir la estructura hacia todos los Nodos de Presentación.

Cada Nodo de Presentación recibirá la estructura y la almacenará en su Base de Datos en Tiempo Real para ponerlas a disposición de los procesos de procesamiento y presentación que la requieran. La primitiva para la transferencia de estructuras se define como sigue:

éxito_env_est = estructura(estructura,resp);

Donde *éxito_env_est* es la bandera que indicará si la estructura fue recibida por el Nodo de Presentación, *estructura* será la estructura que se requiere transferir, misma que será pasada por medio de un mensaje del proceso de adquisición hacia el mecanismo de comunicación. El parámetro *resp* es el buffer que contendrá el reconocimiento de estructura recibida por parte de los Nodos de Presentación.

3.4.3.3 Transferencia de archivos.

Cuando exista información estática, almacenada en disco, cualquier nodo de la red podrá transferirla hacia otro nodo de la red por medio de este servicio.

La transferencia de archivos será administrada mediante la división del archivo en paquetes (mensajes), donde cada paquete tendrá una longitud máxima de 64 kbytes (por restricción del S.O. QNX). La primitiva que describe este servicio es la siguiente:

éxito_env_arch = archivo(nodo,"archivo",resp);

Donde *éxito_env_arch* indicará el éxito o falla de la transferencia, *nodo* es el nodo destino al cual se dirige el archivo, *"archivo"* es el nombre del archivo que se requiere transferir y *resp* es el buffer del reconocimiento.

3.4.3.4 Transferencia de mensajes.

El mecanismo de comunicación prestará el servicio de envío de mensajes hacia cualquier nodo de la red, siendo el nodo fuente también cualquier nodo en la red. Esta transferencia será del tipo mensaje/respuesta, es decir, el nodo transmisor del mensaje esperará una respuesta por parte del nodo receptor de dicho mensaje. Su primitiva es:

éxito_env_msg = mensaje(nodo,"mensaje",resp);

El parámetro *nodo* indica el nodo destino, *"mensaje"* es el mensaje por transferir y *resp* es el buffer del reconocimiento.

3.4.3.5 Transferencia de comandos.

Este servicio permitirá que un nodo fuente pueda enviar un comando hacia un nodo receptor, en este nodo ejecutarse el comando recibido y retornar como respuesta una bandera de comando ejecutado o la información generada por la ejecución de dicho comando. La primitiva que define a este servicio es la siguiente:

éxito_env_cmd = comando(nodo,"comando",resp,ban);

La primitiva anterior está compuesta de los siguientes parámetros: *nodo* que representa al receptor del comando, *"comando"* que es el comando que se requiere ejecutar, *ban* es la bandera que indicará si se requiere como respuesta sólo el reconocimiento de comando ejecutado o si se requiere la información generada en la ejecución del comando. Para almacenar la respuesta se utiliza el buffer *resp*.

3.4.3.6 Estado del nodo.

Este servicio indicará el modo operativo en que se encuentra el nodo donde se está corriendo la aplicación. Este servicio no requiere de una llamada explícita, ya que se estará ejecutando continuamente a lo largo de la operación del mecanismo de comunicación (cada vez que haya alguna operación de transmisión o recepción de información en un nodo). La primitiva que describe este servicio es:

mo = modo_operativo();

Donde *mo* nos indicará si el nodo se encuentra en modo operativo PP (Primario-Primario) o se encuentra en modo operativo PR (Primario-Respaldo).

3.4.3.7 Tolerancia a fallas.

El mecanismo de comunicación contará con un mecanismo software, que le permitirá contar con una disponibilidad alta, en la transferencia de información en el sistema y así evitar la pérdida de información, en caso de que algún nodo (de adquisición o presentación) llegara a fallar.

Dado a que la tolerancia a fallas no es un servicio que se deba solicitar explícitamente, por parte del usuario, ya que éste se encuentra integrado al mecanismo de comunicación, es difícil generar una primitiva abstracta que englobe a todo este servicio, ya que esto implica llamadas a varias funciones, la generación de procesos recursivos, etc.

Para el caso de servicios de transferencias de mensajes, comandos y archivos, las funciones que integren dichos servicios utilizarán banderas *éxito*, que servirán para la llamada a una rutina tolerante a fallas. Si el valor de retorno de *éxito* fuera -1 (existió un error en la transferencia de información o algún nodo participante en la transferencia falló) se utilizará nuevamente la primitiva:

rpid = conexión(nodo);

Esta primitiva, ahora nos devolverá el identificador del nodo **respaldo** del nodo identificado por *nodo*. Este identificador servirá para el enlace con el nodo respaldo y el cambio del flujo de información hacia dicho nodo.

La anterior primitiva sólo se utiliza en caso de que el nodo caído fuera el nodo receptor. El flujo de información cambiará hacia el nodo respaldo del nodo receptor inicial, para que la información pueda ser consultada en dicho nodo y no perder la información transferida. Para el caso de falla en el nodo transmisor, la tolerancia a fallas se puede representar por medio de dos primitivas básicas:

ok = aviso(nodo_resp);

Esta primitiva permite que un nodo transmisor avise a su nodo respaldo, el parámetro *nodo_resp*, que va a realizar una transferencia de información. El parámetro *ok* nos indicará si se pudo avisar al nodo respaldo de la transferencia. En caso negativo (el nodo respaldo está inactivo), la transferencia se realizará sin tolerancia a fallas. En caso de que el aviso fuera exitoso, el nodo respaldo utilizará la primitiva:

falla = vigila(nodo_respd);

Esta primitiva checará el estado operativo del nodo respaldado, durante toda su transferencia de información. Si se detectara la caída del nodo respaldado (la variable *falla* tendrá el valor de -1), el nodo respaldo tomará el mando de la transferencia.

Debe hacerse notar que estas dos últimas primitivas, sólo son válidas en la transferencia de estructuras de memoria, debido a que se requiere en el SCD, contar con la disponibilidad de la información, generada de los procesos de la CTE, en cualquier Nodo de Presentación.

3.5 Vocabulario y formatos de mensaje.

El siguiente paso en la especificación del mecanismo de comunicación, es la especificación del vocabulario y los formatos de los mensajes, es decir, en este punto se especificarán todos los mensajes que requerirá el mecanismo de comunicación para el desempeño correcto de sus funciones. También se especificarán las estructuras internas de cada mensaje.

3.5.1 Tipos de mensajes.

Como se ha podido observar, existen varios tipos de intercambio de información entre los nodos del SCD (datos, mensajes, comandos, etc). Aparte de estos tipos de información, el mecanismo de comunicación requiere de otros adicionales, que le permitan llevar a cabo un control en el intercambio de información. Así, podemos definir los siguientes mensajes que se requieren en el mecanismo de comunicación:

3.5.1.1 Inicio de transmisión.

Cuando se va a comenzar una transferencia de información, el proceso transmisor debe enviar un mensaje de inicio de transmisión, al proceso receptor. Esto se hace con dos propósitos:

a) El proceso transmisor podrá saber si el proceso receptor se encuentra disponible para realizar una transferencia de información. Si el proceso transmisor no recibe respuesta por parte del proceso receptor dentro de un tiempo determinado, el primero asumirá que el segundo se encuentra ocupado por lo que intentará establecer comunicación con algún otro proceso receptor.

b) En el mensaje de inicio de transmisión, se incluyen las características más importantes del intercambio de información que se llevará a cabo, como: el ID del proceso transmisor y del proceso receptor, el tipo de información que se intercambiará (mensaje, comando, estructura, archivo), etc.

Estos mensajes pueden ser vistos como los inicializadores de alguna transferencia de mayor importancia.

3.5.1.2 Estructuras.

Estos mensajes contendrán las estructuras de la BD que representan a las variables adquiridas en los procesos de la CTE y que deben transferirse, ya que cambiaron en algún parámetro que las define. Este mensaje estará compuesto de todos los campos necesarios para la identificación exacta de la variable (número de Nodo de Adquisición, número de Unidad de Adquisición, identificador, etc.).

3.5.1.3 Comando.

Un comando es un mensaje que se envía de un proceso a otro, solicitando que se lleve a cabo una acción, o que se retorne un parámetro. La respuesta será la réplica al comando y puede representar una confirmación, de que la acción ha sido realizada o se puede solicitar la información. El comando y su respuesta generalmente forman una transacción lógica simple y ambos mensajes son usualmente muy pequeños.

3.5.1.4 Mensaje.

Los mensajes pueden ser utilizados simplemente para el intercambio de información, no muy importante, entre los procesos. Esta información puede ser la notificación de algún evento, poco relevante, ocurrido en el sistema.

3.5.1.5 Archivo.

Nuestra aplicación requiere la transferencia de archivos almacenados en disco. Este tipo de mensaje tendrá una longitud muy grande, por lo que se administrará mediante la división del archivo en mensajes más pequeños (datagramas), por lo tanto su tiempo de transmisión será mayor. Cada mensaje podrá tener hasta una longitud máxima de 64 kbytes.

3.5.1.6 Reconocimiento.

Este tipo de mensaje se utiliza para informar cuando se requiere, a un proceso transmisor, que la información transmitida fue o no recibida. Para esto, existen los reconocimientos positivos (recibido) y los reconocimientos negativos (no-recibido).

3.5.1.7 Final de transmisión.

Este mensaje tiene como finalidad el de comunicar, al nodo receptor, que la transferencia de información ha sido terminada. Este mensaje permitirá al nodo receptor, volver a esperar un nuevo mensaje de inicio de transmisión.

3.5.1.8 Aviso.

Un mensaje de aviso se requiere para informar a un Nodo de Adquisición respaldo, que su nodo respaldado va a iniciar una transferencia de estructuras. Con la recepción de dicho mensaje, el nodo respaldo puede iniciar las rutinas de vigilancia de su nodo respaldado.

3.5.2 Formatos de mensajes.

Dada las grandes facilidades que nos da el S.O. QNX para transmitir mensajes, gracias a su librería Watcom C, podemos implementar cualquier tipo de mensaje mediante estructuras simples en lenguaje C. Este tipo de estructuras nos permiten implementar desde mensajes simples hasta archivos. Los diferentes tipos de mensajes, que se describieron anteriormente, se pueden implementar de la siguiente manera:

a) Un mensaje de inicio de transmisión se puede construir con una serie de campos que indique el origen, destino y el tipo de transacción que se realizará, como se muestra a continuación:

```
struct msj0 {
        char inicio[3];
        char nm[7];
        int nd_tx;
        int nd_rx;
        int tipo;
        int ban;
        } msg0;
```

El primer campo nos permitirá indicar, si el mensaje se trata del inicio de una transferencia de información o el inicio de operaciones del nodo, que ha generado dicho mensaje. Este campo podrá tener los siguientes valores:

"ok" = Indicará un inicio de operaciones o inicio de una transferencia de información.
"up" = Indicará un mensaje de alta, por parte del nodo respaldado.
"ur" = Indicará un mensaje de alta, por parte del nodo respaldo.
"db" = Indicará que un programa de aplicación requiere el servicio de transferencia de archivo.
"mg" = Indicará que un programa de aplicación requiere el servicio de transferencia de mensaje.
"cd" = Indicará que un programa de aplicación requiere el servicio de transferencia de comando.

El segundo campo sirve para almacenar el nombre bajo el cual se encuentra registrado el proceso transmisor. Este campo será utilizado por el proceso receptor cuando se requiera localizar al proceso transmisor y así, enviarle la respuesta, si ésta es requerida.

El tercer y cuarto campo servirán para almacenar los números de nodo del proceso transmisor y proceso receptor, respectivamente. El quinto campo nos indicará el tipo de transferencia que será realizada, de acuerdo con los siguiente valores:

1 = Transferencia de mensaje.
2 = Transferencia de comando.
4 = Transferencia de archivo.

El último campo nos indicará, si el servicio de transferencia de información fue solicitado por un usuario (ban = 0) o por un programa de aplicación (ban = 1).

b) Para el manejo de las estructuras en memoria, o BDTR, se contempla la definición de todas las variables y sus diferentes campos, de tal forma, que se tenga toda la información necesaria para los procesos de presentación y control. La estructura de la BDTR [5] se define de la siguiente manera:

```
struct canal{
        char id[6];
        char de[35];
        unsigned char
                señal_ocupada: 1,
                sc: 1,
                cf: 1,
                cd: 1,
                po: 1;
        int posición;
        float aa;
        float ab;
        float of;
        float fe;
        float vv;
        int au;
        int gs;
        int nd;
        int ra;
        int se;
        };

struct bit{
        char id[6];
        char de[35];
        unsigned char
                señal_ocupada: 1,
                sc: 1,
                bn: 1,
                vl: 1;
        int posición;
        int nd;
        int ra;
        int se;
        int no;
        };

struct tarjeta{
        char tipo_tarj[7];
        int dirección;
        int núm_señal_analog;
        float tiempo_adq_analog;
        int rango;
        int núm_señal_digital;
        float tiempo_adq_digital;
        unsigned char
```

```
                        edo_op: 1;
                struct canal canal[16];
                struct bit bit[8];
                };

        struct nodo{
                char nombre[6];
                unsigned char
                        edo_op_prim: 1,
                        edo_op_resp: 1;
                int núm_tarj_usadas;
                int núm_tarj_ocupadas;
                struct tarjeta tarjeta[8];
                };
struct nodo msg2;
```

En la estructura *nodo* se definen los siguientes campos:

nombre: El nombre del nodo donde se encuentra la estructura ("nodo1", "nodo2", etc.).
edo_prim: Indica si el nodo se encuentra en estado operativo primario (1 = en servicio).
edo_resp: Indica si el nodo se encuentra en estado operativo respaldo (1 = en servicio).
núm_tarj_usadas: El número de tarjetas de adquisición que contiene el nodo.
núm_tarj_ocupadas: El número de tarjetas de adquisición que actualmente están en funcionamiento.
tarjetas[8]: El número de tarjetas que maneja el nodo.

En la estructura *tarjeta* se definen los siguientes campos:

tipo_tarj[7]: Una cadena donde se define el nombre de la tarjeta de adquisición (SAC-410, etc.).
dirección: La dirección del puerto de la tarjeta.
núm_señal_analog: Número de señales analógicas que adquiere la tarjeta.
tiempo_adq_analog: Tiempo de adquisición para las señales analógicas.
rango: El rango de adquisición.
núm_señal_digital: Número de señales digitales que adquiere la tarjeta.
tiempo_adq_digital: Tiempo de adquisición de las variables digitales.
edo_op: Estado operativo de la tarjeta (1 = En servicio).
struct canal canal[16]: Define 16 estructuras del tipo *canal*.
struct bit bit[8]: Define 8 estructuras del tipo *bit*.

En la estructura *canal* se define lo siguiente:

id[6]: El identificador de la variable analógica.
de[35]: Descripción de la variable.
señal_ocupada: Si se está utilizando la variable (1 = en adquisición).

sc: Scan Inhibido, indica si la variable se está actualizando o no, es decir, si se encuentra en servicio o fuera de servicio (1 = en servicio).
ef: Estado funcional, define si la variable está fuera de servicio por falla o por mantenimiento (1 = por falla).
cd: Calidad del Dato, indica si la calidad del valor de la variable es buena o mala, esto es, que no haya rebasado sus límites de operación (1 = calidad buena).
po: Polaridad, indica si la tarjeta de conversión es unipolar o bipolar (1 = unipolar).
posicion: Indica la posición de la variable en la BD.
aa: Límite crítico alto, define el límite superior de la variable.
ab: Límite crítico bajo, define el límite inferior de la variable.
of: Offset, el valor de "b" en la ecuación y = mx + b, utilizada para la linealización de las variables.
fe: Factor de Escala, factor de conversión del valor de la variable en unidades de ingeniería.
vv: Valor de la variable.
au: Apuntador a Unidades, apuntador a las unidades de la variable.
gs: Ganancia de la variable.
nd: Nodo donde se adquiere la variable.
ra: La tarjeta de adquisición a la que pertenece la variable.
se: Número de la señal (del 1 al 16).

La estructura *bit* define los siguientes campos:

bn: Estado anormal, indica que el valor de la variable corresponde a un estado anormal o normal (1 = estado anormal).
vl: Valor lógico (0 = falso, 1 = cierto).
no: Estado Normal, apuntador a donde se encuentran las descripciones de las locuciones de la variable.

Los demás campos definen las mismas características de la señal descritas en la estructura *canal*.

c) Un comando puede verse como una cadena de caracteres, que será transmitida hacia un proceso servidor, entonces tenemos:

```
struct msj6 {
        char comando[20];
        int bandera;
        } msg6;
```

Una vez que la variable *comando* ha sido recibida en el proceso receptor, dicho proceso puede tomar la cadena y hacer los procedimientos que se requieran para ejecutar el comando.

Ya que ha sido ejecutado el comando, el proceso receptor del comando puede enviar como respuesta al proceso transmisor la indicación de que ya se ejecutó el comando, o bien puede enviarle la información generada de la ejecución de dicho comando. El campo *bandera*, será el indicador del tipo de respuesta que se requiere, en este caso podrá tomar dos valores:

0 = información generada.
1 = indicación de ejecución de comando.

d) Un mensaje simple también se puede implementar mediante una cadena de caracteres de una longitud definida, como sigue:

```
struct msj5 {
            char mensaje[50];
            } msg5;
```

Donde *mensaje* será la variable que contendrá el mensaje que se requiere enviar, utilizando las funciones de la Librería C de QNX (*Send(), Receive()*, etc.).

e) Para la transmisión de un archivo, podemos dividir el archivo en mensajes pequeños, como se dijo anteriormente. Una vez hecho esto, un archivo puede ser implementado por *n* mensajes. La estructura de este tipo de mensaje será la siguiente:

```
struct msg9{
            int indice;
            int bandera;
            int falla;
            char nombre[15];
            char cadena[RENGLÓN][COLUMNA];
            } msg9;
```

Donde el campo *índice* nos indicará el número de renglones del archivo que se encuentran en el mensaje. El campo *bandera* nos indicará si se trata del último mensaje del archivo (fin de archivo). El campo *falla* nos servirá para indicar una apertura errónea del archivo (el archivo no existe o no se pudo abrir). El campo *nombre* contendrá el nombre del archivo transferido y el campo *cadena* será el espacio buffer donde se almacenará cada bloque del archivo.

f) Un reconocimiento será implementado por medio de una estructura similar a la de un mensaje simple, donde una cadena predefinida nos indique un "reconocimiento positivo" y otra que signifique un "reconocimiento negativo". Más aún, esta última estructura puede ser omitida gracias al valor de retorno de la función *Send()*.

La función *Send()* retorna el PID del proceso al que entregaron el mensaje contenido en su buffer. En caso de cualquier falla (algún problema en la entrega de mensaje) la función retorna el valor de -1, que nos indicará que el mensaje o respuesta no fueron entregados al nodo destino. Para fines de desbloqueo de la función antes mencionada, se utilizará la cadena "ok" para reconocer positivamente a los mensajes recibidos, como se muestra a continuación:

```
struct msj1 {
        char resp[3];
        } msg1;
```

Para reconocimientos que requieran una respuesta más grande, como por ejemplo las respuestas para mensajes o comandos, se puede utilizar la siguiente estructura:

```
struct msg7 {
        char ack[50];
        } msg7;
```

Donde *ack* contendrá la respuesta a un mensaje o la respuesta de la ejecución de un comando.

g) Un mensaje de fin de transmisión, puede ser simplemente implementado por medio de una cadena, que indique el final de la comunicación, por ejemplo "fn".

```
struct msg4{
        char final[3];
        } msg4;
```

h) Finalmente, un aviso puede ser implementado por un mesaje que sólo contenga el número de nodo que requiere ser respaldado, en su transferencia de estructuras. Así, el nodo respaldo que reciba este mensaje, sabrá de antemano que debe vigilar la transferencia de estructuras de su nodo respaldado. Entonces, este mensaje quedaría implementado de la siguiente forma:

```
struct msj10 {
        int nid_tx;
        } msg10;
```

Donde *nid_tx* indicará el nodo que requiere ser respaldado durante su transferencia de estructuras.

Hay que señalar que todas las estructuras anteriores, estarán contenidas en el archivo "mensajes.h". Este archivo será utilizado, posteriormente, en la implantación del mecanismo de comunicación.

CAPÍTULO 4: DISEÑO DEL MECANISMO DE COMUNICACIÓN CON TOLERANCIA A FALLAS.

4.1 Introducción.

Una vez que se han determinado los requerimientos, especificado los servicios y se han identificado todos los elementos necesarios para el desarrollo del mecanismo de comunicación, se procede a su diseño. El propósito de esta etapa, la cual se presenta en este capítulo, es el de utilizar una herramienta formal para especificar **las reglas de procedimiento** del mecanismo de comunicación.

Las reglas de procedimiento expresan la secuencia lógica de transferencias de comandos, datos, etc. y el intercambio ordenado de mensajes entre diferentes procesos. Estas reglas conforman el mecanismo de comunicación real y la especificación de estas reglas, constituye el diseño del mecanismo de comunicación.

Una de las técnicas para resolver problemas de software o hardware establece que un problema muy grande se divida en subproblemas, que sean más fáciles de resolver. Por esto, el diseño del software de protocolos o mecanismos de comunicación, se estructura en módulos. Este diseño modular nos permite tener algunas ventajas, como son:

a) Un diseño modular expresa la estructura lógica del mecanismo de comunicación, mediante la separación de tareas de alto nivel, de las tareas de bajo nivel; para tener una visión clara de la interrelación entre los componentes del mecanismo y por lo tanto contar con una visión más clara de su funcionamiento.

b) Facilita el llevar a cabo modificaciones o extensiones posteriores del mecanismo de comunicación.

En el proceso de diseño se asumen 2 puntos cruciales, que son:

1) El diseño del mecanismo de comunicación es un proceso iterativo. Es probable que el diseño no sea correcto la primera vez que sea escrito y es muy probable que no será correcto la segunda o tercera vez.

2) Cada vez que una fase de diseño se completa, se estará convencido de que está libre de errores. Una revisión *paso a paso* del código puede revelar los errores más grandes, pero no puede esperarse que también revele errores sutiles.

Existen algunas reglas básicas, que han sido establecidas para diseñar mecanismos de comunicación y que pueden ayudarnos en nuestro diseño. Estas reglas son:

1) Asegurarse que el problema está bien definido. Todos los criterios de requerimientos y necesidades deben ser perfectamente definidos antes de comenzar el diseño.

2) Definir los servicios que serán realizados por el mecanismo, antes de decidir cuáles estructuras deben ser usadas para realizar estos servicios (el *qué* viene antes del *cómo*).

3) Diseñar la *funcionalidad externa* antes que la *funcionalidad interna*, es decir, hay que considerar la solución al problema como una "caja negra" y decidir cómo debe interactuar con su ambiente, para después decidir cómo la caja negra estará organizada internamente.

4) Realizar un diseño simple. Los mecanismos de comunicación excesivamente "adornados" son más difíciles de implementar, de verificar y a menudo menos eficientes. Hay pocos problemas complejos en el diseño de mecanismos de comunicación. Los problemas que parecen complejos pueden ser problemas simples, que se encuentran mezclados. El trabajo del diseñador es identificar los problemas simples, separarlos y después resolverlos uno a uno.

5) No introducir lo que es inmaterial. Esta regla se relaciona en mucho con la regla anterior.

6) Realizar un diseño fácilmente extendible. Un buen diseño resuelve una clase de problemas en lugar de un caso simple. El mecanismo de comunicación, debe ser lo suficientemente flexible para soportar todas las condiciones posibles que puedan presentarse.

7) Antes de implementar un diseño, se recomienda construir un prototipo de alto nivel y verificar que el criterio de diseño es el idóneo.

8) Implementar el diseño, midiendo su rendimiento y si es necesario, optimizar el diseño.

9) Verificar que la implantación optimizada final, sea equivalente al diseño de alto nivel que fue verificado.

10) No saltarse de las reglas 1 a la 7.

Finalmente, se puede establecer, que entre las características que se deben tener presentes en el diseño de un mecanismo de comunicación se encuentran las siguientes:

a) Modularidad.

Un mecanismo de comunicación puede construirse con un número pequeño de piezas bien definidas, donde cada pieza realiza una función específica.

Para comprender la funcionalidad del mecanismo de comunicación, debe ser suficiente entender la función de cada una de las partes y la manera en que éstas interactuan. Los mecanismo de comunicación que se diseñan tomando en consideración estos criterios, son más fáciles de entender y de implantar eficientemente.

b) Definición.

Un mecanismo de comunicación, no debe sobreespecificarse, es decir, no debe contener ningún código inalcanzable o inejecutable; tampoco debe estar incompleto. Un mecanismo de comunicación con definición, es aquél que está perfectamente delimitado.

c) Robustez.

El diseñar mecanismos de comunicación que trabajen bajo circunstancias normales representa una menor dificultad. Es lo inesperado, lo que incrementa su complejidad. Esto significa que, el mecanismo de comunicación debe diseñarse considerando cualquier acción posible y secuencia de acciones bajo todas las condiciones.

d) Consistencia.

Existen trayectorias de ejecución no deseadas, en las cuales pueden caer los mecanismos de comunicación. Un mecanismo de comunicación debe evitar caer en cualquiera de ellas. Aquí mencionamos tres de las más importantes:

* **Deadlocks:** Son estados en los cuales no existe un estado de ejecución posible del mecanismo de comunicación, por ejemplo, todos los procesos del mecanismo de comunicación están esperando condiciones que pueden nunca cumplirse.

* **Livelocks:** Son secuencias de ejecución que pueden ser repetidas indefinidamente, sin hacer un progreso efectivo en la ejecución.

* **Terminaciones impropias:** Es la finalización de una ejecución del mecanismo de comunicación, sin satisfacer alguna de las condiciones de terminación adecuadas.

4.2 Máquinas de Estado Finito (MEF).

La herramienta más usual utilizada para la especificación de las reglas de procedimiento de un mecanismo de comunicación, es el diagrama de secuencia de tiempo, pero es inadecuado para el análisis del mecanismo de comunicación.

Las reglas de procedimiento deben describirse de manera clara y concisa. Sin embargo, una descripción detallada provee claridad, pero carece de la brevedad necesaria para el estudio del mecanismo de comunicación. Por otro lado, una descripción breve y general, carece de los detalles necesarios para implantar el mecanismo.

Por lo anterior, se han desarrollado varias técnicas para la descripción de las reglas de procedimiento. Entre estas técnicas, se encuentran las redes de Petri, los diagramas de secuencia y los diagramas de estado, estos últimos conocidos formalmente como Máquinas de Estado Finito.

Debido a las facilidades ofrecidas por las MEF, para expresar las interacciones de los mecanismos de comunicación, se decidió emplear esta herramienta.

Las MEF proveen una representación concisa del mecanismo de comunicación, evitando la simplicidad o complejidad de la descripción, en caso de usar diagramas de secuencia o redes de Petri, respectivamente. Las MEF son el punto intermedio entre complejidad y simplicidad, por lo que es la herramienta más recomendable para el diseño de mecanismos de comunicación.

4.2.1 Definición y características principales.

Los mecanismos de comunicación se comprenden fácilmente, en forma de diagrama de estados. Una MEF define qué acciones le son permitidas a un proceso, qué eventos se espera que ocurran y cómo se responderá a estos eventos, que transiciones ocurren en el proceso y que mensajes son intercambiados por el proceso.

Una MEF se especifica en forma de una tabla de transición, donde para cada estado de control de la MEF, la tabla especifica un conjunto de reglas de transición. Hay una regla por renglón en la tabla y usualmente más de una regla por estado. Una tabla de transición de estados típica puede verse en la figura 4.1. La 1a. y 2a. columnas son condiciones que deben ser satisfechas para que la regla sea ejecutable. Las otras dos columnas de la tabla definen el efecto que se produce al aplicar una regla de transición.

Condición		Efecto	
Estado Actual	Entrada	Salida	Siguiente Estado
q0	-	1	q2
q1	-	0	q0
q2	0	0	q0
q2	1	0	q1

Figura 4.1. Tabla de transición de estados.

En el modelo MEF tradicional, el ambiente de la máquina consiste de dos conjuntos finitos de señales: de entrada y de salida. Cada señal tiene un arbitrario, pero finito, rango de valores posibles. Una regla de transición se ejecuta cuando una señal de entrada se aplica a un estado, generándose una señal de salida y un cambio de estado asociados, ambos, a la señal de entrada y al estado inicial de la máquina.

El guión en una de las dos primeras columnas significa una condición *no importa*. Un guión en la 1a. columna indica que la regla de transición se aplica a todos los estados de la máquina y un guión en la 2a. columna indica que la regla de transición se aplica con cualquier señal de entrada. Los guiones en la 3a. columna significan que la señal de salida no cambia y un guión en la última columna significa que el estado no se modifica. La MEF de la figura 4.1 se entiende fácilmente cuando se representa en la forma de un diagrama de transición de estados, como el mostrado en la figura 4.2.

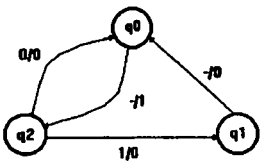

Figura 4.2. Diagrama de transición de estados.

Los estados son representados por círculos y las reglas de transición por arcos dirigidos. Las etiquetas de los arcos son del tipo E/S, donde E especifica la condición de transición (la señal de entrada) y S correspondiendo al efecto (señal de salida).

Cuando se trata de especificar MEF's de sistemas un poco más complejos, las reglas de transición usualmente toman la forma siguiente:

MSG?, PRED

ACCION, MSG!

Donde: **MSG?** representa la recepción de un mensaje, **PRED** es el predicado que debe cumplirse para que ocurra la transición, **ACCION** es el efecto que se tendrá con la transición y **MSG!** es el mensaje de salida que puede ser enviado al realizarse la transición.

También es posible construir sistemas elaborados con máquinas interactivas, es decir, donde las señales de salida de una máquina se conecten a las señales de entrada de otra máquina.

En el diseño de mecanismos de comunicación, las MEF's son más útiles, si pueden modelar fenómenos de un sistema de computadoras distribuido. Existen dos maneras diferentes de cumplir con este objetivo: con los modelos de comunicación síncrona y asíncrona.

4.2.2 MEF en comunicación síncrona.

En este tipo de modelo, las condiciones de transición son las selecciones que la máquina puede hacer para comunicarse. Se permite un sólo evento por regla de transición. La máquina puede seleccionar una señal de entrada o una señal de salida, para la cual una regla de transición se especifica.

Para llevar a cabo una transición, una señal se selecciona precisamente por dos máquinas simultáneamente; en una máquina como salida y en la otra máquina como entrada. Si se cumple esta condición, ambas máquinas realizan la transición correspondiente, de manera simultánea y cambian sus selecciones de acuerdo con el nuevo estado que alcanzaron.

Las tablas A y B, de la figura 4.3, muestran un ejemplo de enlace síncrono de MEF's. La máquina de la tabla A puede hacer sólo una selección de entrada X en el estado q0 y una selección de salida Y en el estado q1. La segunda máquina es casi la misma que la primera, pero tiene sus selecciones invertidas.

Estado	Entrada	Salida	Sig. Edo.
q0	X	-	q1
q1	-	Y	q0

Tabla A

Estado	Entrada	Salida	Sig. Edo.
q0	-	X	q1
q1	Y	-	q0

Tabla B

Figura 4.3. Tablas de transición de estados en comunicación síncrona.

Nótese que la comunicación síncrona se ha establecido como binaria, por lo que, exactamente 2 máquinas deben participar, una con una selección de entrada dada y la otra con su correspondiente selección de salida.

4.2.3 MEF en comunicación asincrona.

Con el modelo asincrono, las máquinas son enlazadas a través de colas de mensaje FIFO (first input - first output). Las señales de una máquina, se consideran como objetos abstractos denominados *mensajes*.

Las señales de entrada se accesan o recuperan desde las colas de entrada y las señales de salida se añaden a las colas de salida. La sincronización se realiza definiendo las señales de entrada y salida, para que estas sean condicionales, respecto a las colas de mensaje. Si una cola de entrada está vacía, no se tendrá ninguna señal de entrada disponible en esa cola y las reglas de transición que requieran una señal serán incjecutables. Si una cola de salida está llena, ninguna señal de salida puede generarse para esa cola, y las reglas de transición que la producen, también son incjecutables.

Para ilustrar la comunicación asincrona, en las figuras 4.4 y 4.5 se definen dos tablas de transición, para un mecanismo sencillo de comunicación. Los diagramas de estados de dichas tablas se muestran en la figura 4.6.

Estado	Entrada	Salida	Sig. Edo.
q0	-	msg0	q1
q1	ack1	-	q0
q1	ack0	-	q2
q2	-	msg1	q3
q3	ack0	-	q2
q3	ack1	-	q0

Figura 4.4. Comunicación asincrona: Transmisor.

Estado	Entrada	Salida	Sig. Edo.
q0	msg1	-	q1
q0	msg0	-	q2
q1	-	ack1	q3
q2	-	ack0	q0
q3	msg0	-	q4
q3	msg1	-	q5
q4	-	ack0	q0
q5	-	ack1	q3

Figura 4.5. Comunicación asincrona: Receptor.

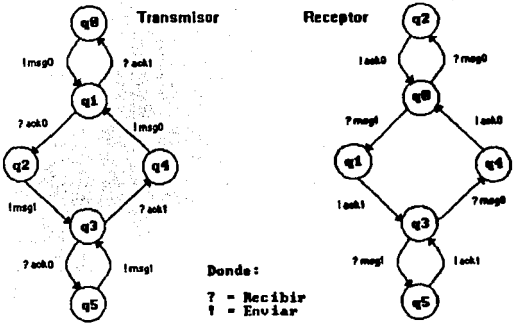

Figura 4.6. Diagramas de transición de estados en comunicación asíncrona.

4.3 Diseño del mecanismo de comunicación con la herramienta MEF.

Teniendo en cuenta que los servicios principales del mecanismo de comunicación, tanto en los Nodos de Adquisición como en los Nodos de Presentación, son los relacionados a la transferencia de información, entre la que se encuentran mensajes, estructuras, etc., el mecanismo de comunicación debeb de contar con un protocolo de operaciones, que se encargará de esta transferencia de información.

Además de transmitir y recibir información, el mecanismo de comunicación debe incluir un protocolo de respaldo, que le permitirá al SCD mantener la disponibilidad en la transferencia de información, en caso de falla en algún nodo del sistema.

Con lo anterior, podemos dividir el comportamiento global del mecanismo de comunicación en tres fases principales: a) *Arranque*, b) Establecimiento del modo operativo y c) Operación. Estas fases se pueden observar en la figura 4.7.

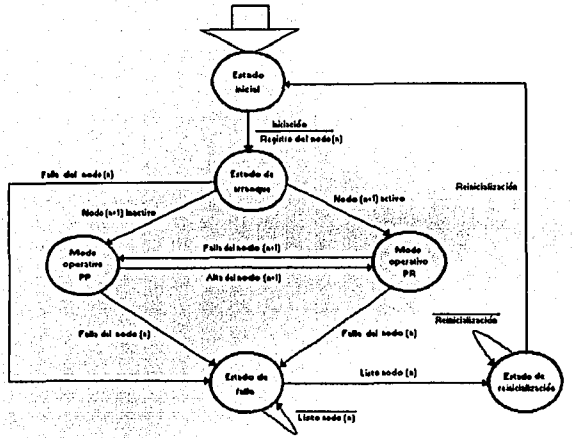

Figura 4.7. Diagrama general del mecanismo de comunicación.

En la fase de arranque, el mecanismo de comunicación se dará de alta en cada nodo del sistema y en cada nodo se verificará la presencia del nodo respaldado, y de su nodo respaldo (en caso de tratarse de un Nodo de Adquisición). Esto se realiza con el propósito de verificar cuáles nodos se encuentran en operación y cuáles están fuera de servicio.

En la fase de establecimiento del modo operativo, el mecanismo de comunicación utilizará la información obtenida en la fase anterior para establecer el estado operacional del nodo, es decir, se determinará si el nodo se encuentran en modo operativo *Primario-Primario* (el nodo toma sus funciones y las funciones del nodo al que está respaldando) o el nodo se encuentra en modo operativo *Primario-Respaldo* (donde el nodo toma sus funciones y está listo para tomar las funciones de su nodo respaldado en caso de falla de este último).

En la última fase, la fase de operación, el mecanismo de comunicación ofrecerá sus servicios al usuario o a programas de aplicación. En los Nodos de Adquisición y en los de Presentación se podrán utilizar los servicios de transferencia de mensajes, comandos y archivos. En los NA's se tendrá la responsabilidad de transferir las estructuras de la BDTR, cada vez que los procesos de adquisición requieran de dicho servicio. Los NP's almacenarán la información recibida en sus respectivas BD's.

4.3.1 Diseño del protocolo de respaldo para los Nodos de Adquisición.

El objetivo del protocolo de respaldo, de los Nodos de Adquisición, es el de establecer un modo operativo (PP o PR) para cada nodo donde se ejecute el mecanismo de comunicación (*mc*). La figura 4.8 nos muestra el protocolo de respaldo para los Nodos de Adquisición, expresado en una Máquina de Estados Finitos.

Hay que señalar que los mensajes definidos en el capítulo anterior, y que son intercambiados por los protocolos diseñados, son utilizados con otro nombre para un mayor entendimiento del funcionamiento de los protocolos. En cada mensaje se señala a que mensaje representa verdaderamente. La MEF de la figura 4.8 está constituida por los siguientes elementos:

ESTADOS QUE CONSTITUYEN AL AUTÓMATA:

Q0 = Estado Inicial (Nodo *NAi(n)* listo para operar).
Q1 = Estado de arranque (Registro del *mc* en el nodo *NAi(n)*).
Q2 = Estado de verificación de anillo lógico.
Q3 = Estado con nodo *NAi(n)* sin nodo respaldo *NAi(n-1)*.
Q4 = Estado con nodo *NAi(n)* con nodo respaldo *NAi(n-1)*.
Q5 = Estado con modo operativo *PP*: Primario de si mismo y primario del nodo *NAi(n+1)*.
Q6 = Estado con modo operativo *PR*: Primario de si mismo y respaldo del nodo *NAi(n+1)*.
Q7 = Estado de falla del nodo *NAi(n)* (Fuera de servicio).
Q8 = Estado de reinicialización del nodo *NAi(n)*.

MENSAJES INTERCAMBIADOS POR EL PROTOCOLO DE RESPALDO:

Alta_NAi(n) = Mensaje de inicio de operación del *mc* en el nodo *NAi(n)* (msg0.inicio = "up" o "ur", según sea el caso).
Alta_NAi(n-1) = Mensaje de inicio de operación del *mc* en el nodo *NAi(n-1)* (msg0.inicio = "ur").
Alta_NAi(n+1) = Mensaje de inicio de operación del *mc* en el nodo *NAi(n+1)* (msg0.inicio = "up").
Alta_PAi(n) = Mensaje de inicio de operación del *mc* para el proceso de adquisición en el nodo *NAi(n)* (msg0.inicio = "ok").

MENSAJES GENERADOS PARA EL USUARIO:

Falla_Reg = Falla en el registro del *mc* en el nodo *NAi(n)*.
NAi(n)_SR = Nodo *NAi(n)* trabajando sin nodo respaldo.
NAi(n)_CR = Nodo *NAi(n)* trabajando con nodo respaldo.
NAi(n)_MO_PP = Nodo *NAi(n)* trabajando en modo operativo PP.
NAi(n)_MO_PR = Nodo *NAi(n)* trabajando en modo operativo PR.
Fail_NAi(n+1) = Falla en el nodo respaldado *NAi(n+1)*.

EVENTOS QUE OCURREN EN EL SISTEMA:

Ini_Fun = Inicialización de operación del *mc* en el nodo *NAi(n)*.
Reboot = Reinicialización del nodo *NAi(n)*.
Falla_NAi(n) = Falla en el nodo *NAi(n)*.
Falla_NAi(n+1) = Falla en el nodo *NAi(n+1)*.
Falla_NAi(n-1) = Falla en el nodo *NAi(n-1)*.
Ok_NAi(n) = Nodo *NAi(n)* listo.

PREDICADOS UTILIZADOS POR LA MEF:

Ok_Reg_NAi(n) = Registro del *mc* en el nodo *NAi(n)* realizado.
Ok_NAi(n-1) = Nodo *NAi(n-1)* del anillo lógico se encuentra operando.
Ok_NAi(n+1) = Nodo *NAi(n+1)* del anillo lógico se encuentra operando.

ACCIONES REALIZADAS POR EL AUTÓMATA:

Reg_NAi(n) = Registrar al *mc* en el nodo *NAi(n)*.
Loc_NAi(n-1) = Localizar al nodo respaldo *NAi(n-1)*.
Loc_NAi(n+1) = Localizar al nodo respaldado *NAi(n+1)*.

El funcionamiento de la MEF de la figura 4.8 se puede describir de la siguiente manera: En el estado Q0, el NAi(n) está listo para entrar en operación. Al recibir el *mc*, una inicialización de operaciones (el mecanismo de comunicación es ejecutado), lo primero que hará el *mc* será registrarse, como proceso, bajo un nombre con el S.O. (estado Q1). Este registro se hará de acuerdo con el nodo donde se encuentre corriendo el *mc* (por ejemplo, si el *mc* se encuentra en el nodo 1 se registrará con el nombre "*nodo1*").

Si el registro fue fallido (no se pudo registran el *mc* en el NAi(n)) se regresará al estado Q0, en espera de una nueva inicialización de operaciones.

En caso de un registro exitoso, se enviará un mensaje de alta del *mc* desde el nodo NAi(n), dirigido al proceso de adquisición asociado al nodo donde está corriendo el *mc*. Esto se realiza con el propósito de que el proceso de adquisición tenga conocimiento de la inicialización del *mc* y pueda solicitar los servicios de transferencia de estructuras.

Después se realiza la transición al estado Q2, donde el *mc* buscará al nodo respaldo del nodo NAi(n). Si en el NAi(n-1) se encuentra ya operando el *mc*, se hará una transición hacia el estado Q4, se enviará un mensaje hacia el nodo NAi(n-1) indicando que el *mc* del nodo NAi(n), ha sido dado de alta y se generará un mensaje indicando que se está trabajando con nodo respaldo.

En caso contrario (el *mc* del nodo NAi(n-1) o el nodo respaldo, no se encuentran operando), la transición se hará hacia el estado Q3 y se generará un mensaje que indicará que el nodo NA(n) se encuentra operando sin su nodo respaldo.

El siguiente paso del protocolo de respaldo es el verificar si el NAi(n+1) está operando y el *mc* ya se encuentra ejecutándose en ese nodo. Esto se realiza en los dos estados Q3 y Q4. Si el nodo respaldado NAi(n+1) se encuentra en funcionamiento junto con el *mc*, se entrará al estado Q6, esto es, el NAi(n) adquirirá el modo operativo *Primario-Respaldo* y se enviará hacia dicho nodo el mensaje de alta del *mc*. En caso contrario, el NAi(n) adquirirá el modo operativo *Primario-Primario* con lo que el estado alcanzado será el Q5.

Existen dos eventos importantes que pueden ocurrir cuando ya se tiene un modo operativo establecido:

1) La falla en el nodo respaldado NAi(n+1) y

2) La inicialización o reinicialización de operaciones por parte del nodo NAi(n+1).

En caso de una falla en el nodo respaldado, se hará una transición del estado Q6 al estado Q5 indicándose la caída del nodo y la transición del modo operativo. La transición será al contrario si se detecta un mensaje de alta por parte del nodo respaldado NAi(n+1).

En cualquiera de los estados anteriores, el nodo NAi(n) puede sufrir una falla. En tal caso se realizará una transición al estado de falla Q7. En este estado, permanecerá el NAi(n) hasta que éste sea reparado o la causa de la falla haya sido solucionada. Cuando tales eventos ocurran, se realizará la transición hacia el estado Q8 en donde el nodo NAi(n) estará listo para recibir una reinicialización de las funciones del *mc*.

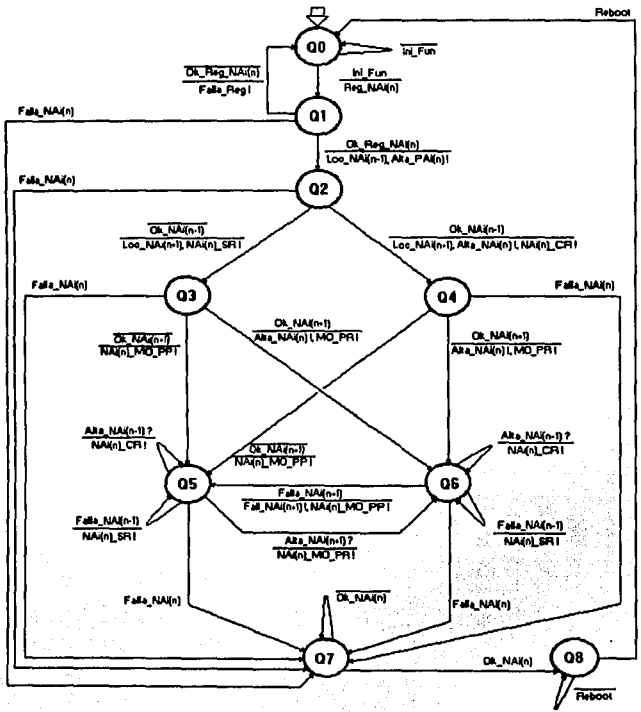

Figura 4.8. MEF del protocolo de respaldo de los Nodos de Adquisición.

4.3.2 Diseño del protocolo de operación para los Nodos de Adquisición.

Una vez que el Nodo de Adquisición ha determinado su modo operativo (gracias al protocolo de respaldo del *mc*), el *mc* de este Nodo de Adquisición, ya puede ofrecer sus servicios de transferencia de información: Envío de mensajes, estructuras, etc.

El protocolo de operación para los Nodos de Adquisición está representado por la Máquina de Estado Finitos de la figura 4.9. Los elementos de dicha máquina son los siguientes:

ESTADOS QUE CONSTITUYEN AL AUTÓMATA:

Q0 = Estado de verificación del estado operativo del nodo respaldado *NAi(n+1)*.
Q1 = Estado en modo operativo **PP** (*Primario-Primario*).
Q2 = Estado en modo operativo **PR** (*Primario-Respaldo*).
Q3 = Estado de conexión con nodos receptores de información.
Q4 = Estado de transferencia de información y recepción de reconocimientos.
Q5 = Estado de recepción de información.
Q6 = Estado vigía de condición operativa del nodo *NAi(n+1)*.
Q7 = Estado de creación de procesos para transferencia de estructuras.
Q8 = Estado de espera de solicitud de envío de estructura.
Q9 = Estado de falla del nodo *NAi(n)*.
Q10 = Estado de reinicialización del nodo *NAi(n)*.
Q11 = Estado de inicialización de funciones del *mc*.

MENSAJES INTERCAMBIADOS POR EL PROTOCOLO DE OPERACIÓN:

Cmd =	Comando (msg6).
Res_cmd =	Resultado de la ejecución de un comando (msg7).
Msg =	Mensaje (msg5).
Res_msg =	Respuesta a un mensaje (msg7).
Arch =	Archivo (msg9).
Ack_arch =	Reconocimiento de archivo recibido (msg1).
Est =	Estructura actualizada de la BDTR (msg2).
Ack_est =	Reconocimiento de la estructura recibida (msg1).
Tx_NA =	Inicio de transmisión de estructuras por parte del nodo *NAi(n+1)* (msg10 + msg2).
Ack_tx =	Reconocimiento del mensaje *Tx_NA* (msg1).
Tx_ok =	Fin de transmisión de estructuras por parte del nodo *NAi(n+1)* (msg4).
Ack_tx_ok =	Reconocimiento del mensaje *Tx_ok* (msg1).
Msg_ini =	Mensaje de inicio de transferencia de información (msg0.inicio = "ok").
Ack_ini =	Reconocimiento del mensaje *msg_ini* (msg1).
Info(x) =	Mensaje con información: mensaje, comando o archivo (msg5, msg6 o msg9, según sea el caso).
Ack_info(x) =	Reconocimiento del mensaje *info(x)* (msg1 o msg7).

Msg_fin = Mensaje de fin de transmisión de información (msg4).
Ack_fin = Reconocimiento del mensaje *msg_fin* (msg1).
Sol_env_bd = Solicitud de transferencia de estructura de la BDTR (msg2).
Ack_bd = Reconocimiento del mensaje *Sol_env_bd* (msg1).
Msg_proc = Mensaje de caída de nodo *NAi(n+1)* hacia proceso de adquisición en nodo *NAi(n)*.

MENSAJES GENERADOS POR EL PROTOCOLO PARA EL USUARIO:

NAi(n)_PP = Nodo *NAi(n)* trabajando en modo operativo PP.
NAi(n)_PR = Nodo *NAi(n)* trabajando en modo operativo PR.
NDi(p)_FS = Nodo receptor primario *NDi(p)* fuera de servicio.
NDi(r)_FS = Nodo receptor respaldo *NDi(r)* fuera de servicio.
Ok_tx = Transferencia de información realizada.
Falla = Transferencia de información fallida.
Falla_tx = Falla en el nodo transmisor.
Falla_rx = Falla en el nodo receptor.
ND_tx_ocup = Nodo receptor ocupado.

EVENTOS EN EL SISTEMA:

Reboot = Reinicialización del nodo *NAi(n)*.
Ini_fun = Inicializar funciones del *mc* en el nodo *NAi(n)*.
Falla_NAi(n) = Falla en el nodo *NAi(n)*.
Falla_Nd_tx = Falla en el nodo transmisor.
Falla_Nd_rx = Falla en el nodo receptor.
Ok_NAi(n) = Nodo *NAi(n)* listo.
Sol_env_inf = Solicitud de transferencia de información (mensaje, comando o archivo).

PREDICADOS UTILIZADOS:

Ok_NAi(n+1) = Nodo *NAi(n+1)* en operación.
Ok_NDi(p) = Conexión con nodo primario *NDi(p)* realizada.
Ok_NDi(r) = Conexión con nodo respaldo *NDi(r)* realizada.
Ok_NDi = Conexión establecida con nodo *NDi(p)* o nodo *NDi(r)*.
Ok_procs = Procesos transmisores creados.
Env1 = Transferencia de mensaje, comando o archivo.
Env2 = Transferencia de estructura.
TO_ini = Tiempo de espera por reconocimiento del mensaje *msg_ini* terminado.

ACCIONES REALIZADAS:

Ejec_cmd = Ejecutar comando.
Ejec_PR = Ejecutar protocolo de respaldo.
Con_NDi(p) = Realizar conexión con nodo primario *NDi(p)*.
Con_NDi(r) = Realizar conexión con nodo respaldo *NDi(r)*.
Vg_NAi(n+1) = Vigilar transferencia de información del nodo*NAi(n+1)*.
Alm_arch = Almacenar archivo.
Crea_procs = Crear procesos concurrentes de transmisión.

Como se puede observar en la figura 4.9, el estado Q0 representa al protocolo de respaldo respecto a la verificación del estado operativo del nodo respaldado por el nodo NAi(n), es decir del nodo NAi(n+1).

Como el Nodo de Adquisición NAi(n) ha asumido un modo operativo (estado Q1 modo operativo *Primario-Primario* o estado Q2 modo operativo *Primario-Respaldo*), el *mc* puede ofrecer sus servicios de transferencia de información. En cualquiera de los dos estados anteriores, el nodo NAi(n) puede recibir un mensaje de inicio de transferencia de información. Con la recepción de este tipo de mensaje se hace una transferencia hacia el estado Q5 (recepción de información). En este estado el nodo NAi(n) está preparado para recibir mensajes, comandos o archivos.

Con la recepción de un comando, el NAi(n) ejecutará el comando y enviará la réplica de dicho comando. Con la recepción de un mensaje, el nodo procesará el mensaje y enviará de retorno la respuesta del mensaje. Si recibe un archivo, lo almacenará en su disco y retomará su reconocimiento correspondiente.

A partir del estado Q5 se puede salir de dos formas:

1) Por la recepción de un mensaje de fin de transferencia, indicando la terminación del intercambio de información, o

2) Por la falla en el nodo transmisor, que se indicará por un mensaje de caída de nodo transmisor.

Ambas opciones harán una transición al estado Q0.

Debe notarse que el protocolo de respaldo estará funcionando implícitamente durante toda la operación del nodo NAi(n). Si se llegara a detectar la caída del nodo respaldado NAi(n+1), el NAi(n) pasaría del modo operativo *PR* al modo operativo *PP* (habría una transición del estado Q2 al estado Q1 o viceversa), como ya se ha explicado anteriormente.

Cuando algún usuario solicite algún servicio de transferencia (mensaje, comando o archivo), el protocolo de operación hará una transición al estado Q3, donde se realiza el enlace o *circuito virtual* con el nodo receptor requerido.

En el estado Q3 existen dos fases, la primer fase consiste del enlace con el nodo inicialmente solicitado. Si el nodo con el que se requiere enlazarse se encuentra fuera de servicio o el *mc* en dicho nodo no se ha ejecutado, se indicará el impedimento de enlazarse con el nodo requerido. De ser así, se realizará la segunda fase: el enlace con el nodo respaldo del nodo receptor original. Si la segunda fase también resulta fallida, es decir, no se logró el enlace con el nodo receptor respaldo, se generará un mensaje de transferencia fallida y se retomará al estado Q0.

Si se logra el enlace con cualquiera de los nodos receptores, ya sea el primario NDi(p) o el respaldo NDi(r), el protocolo de operación realizará la transición al siguiente estado, el de transferencia de información (Q4). Aquí se enviará la información que el usuario requiere transmitir.

En el estado Q4 existe una característica particular. El primer mensaje que se enviará hacia el nodo receptor, será el mensaje de inicio de transferencia; acto seguido, se esperará por un lapso de tiempo, por el reconocimiento de dicho mensaje (5 seg.). Si el nodo receptor no contesta en ese tiempo, el protocolo de operación asumirá que el nodo receptor se encuentra ocupado, para recibir la información (porque esta realizando otra transferencia con otro nodo) y regresará al estado inicial Q0.

Una vez que se recibe el reconocimiento del mensaje de inicio, el protocolo de operación transmite la información del usuario (mensaje, comando o archivo) y espera por el reconocimiento de recepción exitosa de dicha información por parte del nodo receptor. Ya que se transmitió la información requerida, se envía un mensaje de fin de transferencia, para indicar que la transmisión terminó. A continuación se realiza la transferencia hacia el estado inicial Q0.

Si durante la transferencia ocurriera una falla en el nodo receptor, se generaría un mensaje informando tal evento y se realizaría una transición al estado Q3, para volver a realizar los pasos antes descritos.

Cuando el proceso de adquisición del nodo NAi(n) solicite una transferencia de estructuras hacia los Nodos de Presentación, se realizará una transición al estado Q7. Esta solicitud debe contener la estructura que se requiere transferir a los NP's.

En el estado Q7 se generarán *n* procesos concurrentes transmisores de estructuras, donde *n* será igual al número de Nodos de Presentación existentes en el sistema. Esto se realiza con el propósito de aumentar la velocidad de transmisión de las estructuras. Esto se debe a los requerimientos del *mc*, para trabajar en un ambiente de tiempo real.

Una vez que todos los procesos concurrentes han sido creados, el protocolo de operación del nodo NAi(n), enviará un mensaje hacia su nodo respaldo NAi(n-1), indicándole que realizará una transmisión de estructuras y requiere de la vigilancia de dicha transmisión.

Después de enviar el mensaje de aviso, hacia el nodo respaldo, se realiza la transición hacia el estado Q3, donde cada proceso transmisor se enlazará con un proceso receptor de estructuras de un Nodo de Presentación en particular. Ya que cada proceso transmisor se enlazó con su Nodo de Presentación correspondiente, se realiza la transición al estado Q4.

En el estado Q4 se envía la estructura, hacia todos los Nodos de Presentación. Aquí se omiten los mensajes de inicio y terminación, para darle mayor disponibilidad a este tipo de transferencia. Una vez que se transfirió la estructura, se envía un mensaje de fin de transferencia de estructura hacia el nodo respaldo NAi(n-1), con el propósito de que este nodo termine la vigilancia de la transmisión. El siguiente paso es el retorno al estado inicial Q0.

El estado Q2 tiene la particularidad siguiente: En este estado se puede recibir un mensaje, indicando que el nodo respaldado NAi(n+1) está por realizar una transferencia de información (envío de estructuras en memoria), hacia los Nodos de Presentación. Si se recibe tal mensaje (que incluye a la estructura que se requiere transmitir), se realiza una transición al estado Q6 donde el protocolo de operación del NAi(n) realizará una "vigilancia" de su nodo respaldado, con la finalidad de que la transferencia de información se realice correctamente.

Si se llegara a detectar la caída del nodo respaldado, antes de haber completado la transferencia de información, el nodo NAi(n) asumiría la transferencia de la estructura del nodo NAi(n+1). Debido a que la caída del nodo NAi(n+1) implica que el proceso de adquisición de variables, en ese nodo, dejará de funcionar, es necesario enviar un mensaje al proceso de adquisición asociado al nodo NAi(n), para que se dedique a la adquisición de las variables del nodo caído y que este proceso sea, ahora, el encargado de solicitar el envío de estructuras. La transición que seguiría sería al estado Q7, donde el nodo NAi(n) ejecutará las mismas acciones descritas anteriormente.

Los estados Q8 y Q9 son análogos a los estados Q7 y Q8 del protocolo de respaldo: Si hay una falla del nodo NAi(n) en cualquier estado del protocolo de operación (Q0, Q1, ..., Q8) se pasa al estado Q9, donde se permanecerá hasta que el nodo esté listo. En caso de que el nodo se encuentre listo para operar se pasará al estado Q9 donde se esperará una reinicialización. Si se recibe esta reinicialización se realiza la transición al estado Q10, donde se esperará una inicialización del *mc* para poder comenzar de nuevo sus funciones asignadas.

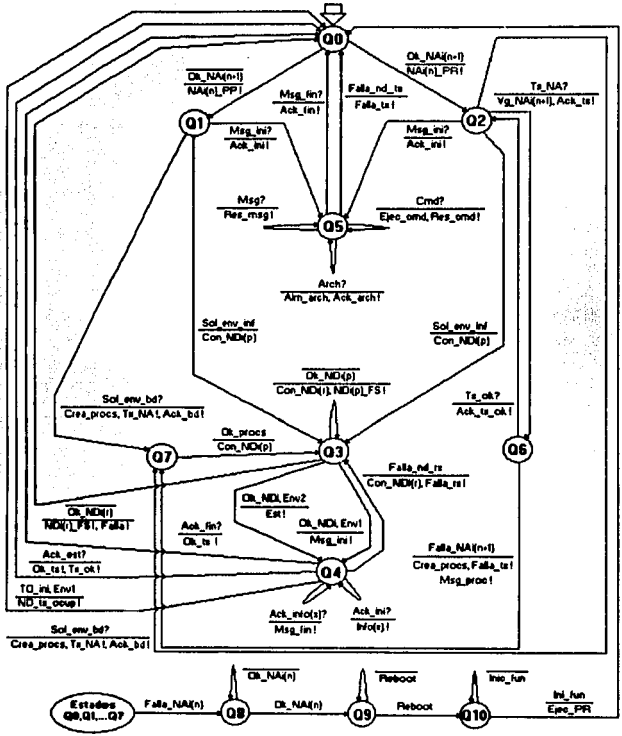

Figura 4.9. MEF del protocolo de operación de los Nodos de Adquisición.

4.3.3 Diseño del protocolo de respaldo para los Nodos de Presentación.

Al igual que los Nodos de Adquisición, los Nodos de Presentación requieren de un mecanismo tolerante a fallas (protocolo de respaldo) para mantener el nivel de disponibilidad, con funciones específicas en la transferencia de información en el SCD.

A diferencia de los NA's, que forman un anillo lógico, los NP's están respaldados por parejas. Por ejemplo, el nodo de Alarmas A está respaldo por el nodo de Alarmas B y viceversa, el nodo B respalda al A; el nodo de Reportes A está respaldado por el de Reportes B, etcétera. Esto lo vemos claramente en la figura 1.3, donde se muestra la arquitectura del SCD para una CTE.

En cualquiera de las aplicaciones, el objetivo del protocolo de respaldo para los Nodos de Presentación es el mismo: Establecer el modo operativo (PP o PR) con el cuál los Nodos de Presentación ejecutarán sus funciones.

La figura 4.10 nos muestra el protocolo de respaldo para los Nodos de Presentación. El protocolo está conformado por los siguientes elementos:

ESTADOS QUE CONSTITUYEN A LA MAQUINA DE ESTADOS FINITOS:

Q0 = Estado inicial (Nodo de presentación *NPi(A)* listo para operar).
Q1 = Estado de arranque (Registro del *mc* en el nodo *NPi(A)*).
Q2 = Estado de búsqueda del nodo respaldado *NPi(B)*.
Q3 = Estado de modo operativo PP: Primario de sí mismo y primario del nodo *NPi(B)*.
Q4 = Estado de modo operativo PR: Primario de sí mismo y respaldo del nodo *NPi(B)*.
Q5 = Estado de falla del nodo *NPi(A)*.
Q6 = Estado de reinicialización del nodo *NPi(A)*.

MENSAJES INTERCAMBIADOS POR EL PROTOCOLO:

Alta_NPi(A) = Mensaje de inicio de operaciones del *mc* del nodo *NPi(A)* (msg0.inicio = "up").
Alta_NPi(B) = Mensaje de inicio de operaciones del *mc* del nodo *NPi(B)* (msg0.inicio = "up").

MENSAJES GENERADOS POR EL PROTOCOLO PARA EL USUARIO:

Falla_reg = Falla en el registro del *mc* en el nodo *NPi(A)*.
NPi(A)_MO_PP = Nodo *NPi(A)* trabajando en modo operativo PP.
NPi(A)_MO_PR = Nodo *NPi(A)* trabajando en modo operativo PR.
Fail_NPi(B) = Falla en el nodo respaldado *NPi(B)*.

EVENTOS QUE OCURREN EN EL SISTEMA:

Ini_Fun = Inicialización de operaciones del *mc* del nodo *NPi(A)*.
Reboot = Reinicialización del nodo *NPi(A)*.
Falla_NPi(A) = Falla en el nodo *NPi(A)*.
Falla_NPi(B) = Falla en el nodo *NPi(B)*.
Ok_NPi(A) = Nodo *NPi(A)* listo.

PREDICADOS UTILIZADOS POR LA MEF:

Ok_Reg_NPi(A) = Registro del *mc* en el nodo *NPi(A)* fallido.
Ok_NPi(B) = Nodo *NPi(B)* se encuentra operando.

ACCIONES REALIZADAS:

Reg_NPi(A) = Registrar al *mc* en el nodo *NPi(A)*.
Loc_NPi(B) = Localizar al nodo respaldado *NPi(B)*.

El funcionamiento de la MEF de la figura 4.10 es similar a la MEF del protocolo de respaldo de los Nodos de Adquisición. La diferencia principal que existe entre el funcionamiento de ambos protocolos de respaldo, es que los NP's solamente deben verificar si su nodo respaldado está en funcionamiento (Nodo A o Nodo B, según sea el caso), mientras que los NA's deben verificar la operatividad de su nodo respaldado y de su nodo respaldo (por formar un anillo lógico).

En el estado Q0 el NPi(A) está listo para operar. Si se recibe una inicialización de operaciones del mecanismo de comunicación (*mc*), el *mc* tratará de registrarse en el sistema (realizando la transición del estado Q0 al estado Q1).

En el estado Q1, si fue exitoso el registro del nodo, se realizará una transición al estado Q2, donde se verifica que el nodo pareja (nodo NPi(B)) se encuentre operando y su *mc* se encuentre ejecutándose. Si no es así, el NPi(A) asume el modo operativo *Primario-Primario* (estado Q3). En caso contrario, se asume el modo operativo *Primario-Respaldo* (estado Q4) y se envía un mensaje de inicio de operaciones hacia el nodo pareja.

Una vez más, se llevará a cabo una transición del estado Q4 al estado Q3 cuando se detecte una falla en el nodo NPi(B) y una transición del estado Q3 al estado Q4, cuando exista una inicilización o reinicialización de operaciones por parte de este nodo.

Cuando exista una falla, en el nodo actual, se realiza una transición al estado Q5. Se permanecerá en este estado, hasta que el nodo este listo para operar nuevamente. En este caso, se alcanzará el estado Q6, donde el nodo NPi(A), está listo para recibir una reinicialización y poder ejecutar nuevamente su *mc*.

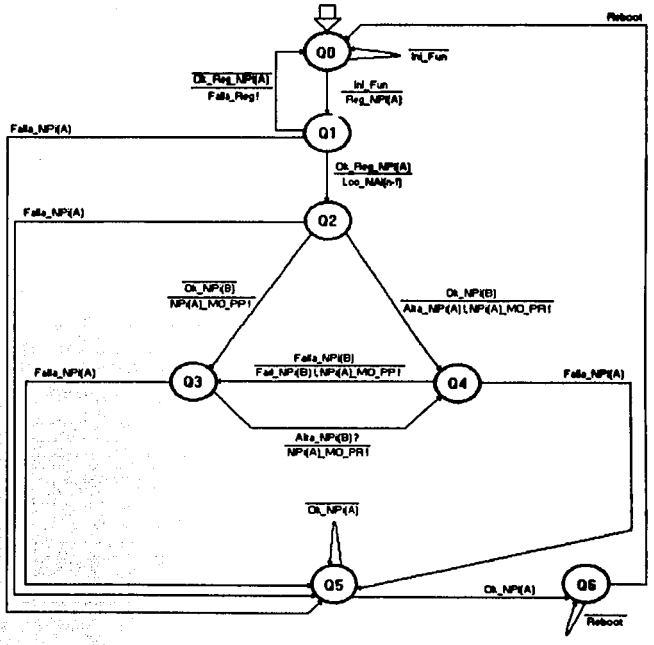

Figura 4.10. MEF del protocolo de respaldo de los Nodos de Presentación.

4.3.4 Diseño del protocolo de operación para los Nodos de Presentación.

Los Nodos de Presentación, contienen un número importante de funciones similares a los Nodos de Adquisición, en lo relacionado con la transferencia de comandos, mensajes y archivos.

La diferencia principal entre el protocolo de operación de los NA's y el de los NP's, es que estos últimos incluyen como función principal, la de recibir estructuras provenientes de los NA's, que constituyen a la BDTR. Es responsabilidad de los NP's, el recibir estas estructuras y almacenarlas en su propia Base de Datos.

El protocolo de operación de los Nodos de Presentación se muestra en la figura 4.11. Sus elementos son:

ESTADOS QUE CONSTITUYEN AL AUTÓMATA:

Q0 = Estado de arranque.
Q1 = Estado de creación de procesos receptores concurrentes y chequeo del estado operativo del nodo respaldado *NAi(n+1)*.
Q2 = Estado en modo operativo PP (*Primario-Primario*).
Q3 = Estado en modo operativo PR (*Primario-Respaldo*).
Q4 = Estado de recepción de información.
Q5 = Estado de conexión con nodos receptores de información.
Q6 = Estado de transferencia de información y recepción de reconocimientos.
Q7 = Estado de falla del nodo *NAi(n)*.
Q8 = Estado de reinicialización del nodo *NAi(n)*.
Q9 = Estado de inicialización de funciones del *mc*.

MENSAJES INTERCAMBIADOS POR EL PROTOCOLO DE OPERACIÓN:

Cmd =	Comando (msg6).
Res_cmd =	Resultado de la ejecución de un comando (msg7).
Msg =	Mensaje (msg5).
Res_msg =	Respuesta a un mensaje (msg7).
Arch =	Archivo (msg9).
Ack_arch =	Reconocimiento de archivo recibido (msg1).
Est =	Estructura actualizada de la BDTR (msg2).
Ack_est =	Reconocimiento de la estructura recibida (msg1).
Msg_ini =	Mensaje de inicio de transferencia de información (msg0.inicio = "ok").
Ack_ini =	Reconocimiento del mensaje *msg_ini* (msg1).
Info(x) =	Mensaje con información: mensaje, comando o archivo (msg5, msg6 o msg9).
Ack_info(x) =	Reconocimiento del mensajes *info(x)* (msg1 o msg7).
Msg_fin =	Mensaje de fin de transmisión de información (msg4).
Ack_fin =	Reconocimiento del mensaje *msg_fin* (msg1).

MENSAJES GENERADOS POR EL PROTOCOLO PARA EL USUARIO:

NPi(A)_PP =	Nodo *NPi(A)* trabajando en modo operativo **PP**.
NPi(A)_PR =	Nodo *NPi(A)* trabajando en modo operativo **PR**.
NDi(p)_FS =	Nodo receptor primario *NDi(p)* fuera de servicio.
NDi(r)_FS =	Nodo receptor respaldo *NDi(r)* fuera de servicio.
Ok_tx =	Transferencia de información realizada.
Falla =	Transferencia de información fallida.
Falla_tx =	Falla en el nodo transmisor.
Falla_rx =	Falla en el nodo receptor.
ND_tx_ocup =	Nodo receptor ocupado.

EVENTOS EN EL SISTEMA:

Reboot =	Reinicialización del nodo *NPi(A)*.
Ini_fun =	Inicializar funciones del *me* en el nodo *NPi(A)*.
Falla_NPi(A) =	Falla en el nodo *NPi(A)*.
Falla_Nd_tx =	Falla en el nodo transmisor.
Falla_Nd_rx =	Falla en el nodo receptor.
Ok_NPi(A) =	Nodo *NPi(A)* listo.
Sol_env_inf =	Solicitud de transferencia de información (mensaje, comando o archivo).

PREDICADOS UTILIZADOS:

Ok_NPi(B) =	Nodo *NPi(B)* en operación.
Ok_NDi(p) =	Conexión con nodo primario *NDi(p)* realizada.
Ok_NDi(r) =	Conexión con nodo respaldo *NDi(r)* realizada.
Ok_NDi =	Conexión establecida con nodo *NDi(p)* o nodo *NDi(r)*.
TO_ini =	Tiempo de espera por reconocimiento del mensaje *msg_ini* terminado.

ACCIONES REALIZADAS:

Ejec_cmd =	Ejecutar comando.
Ejec_PR =	Ejecutar protocolo de respaldo.
Con_NDi(p) =	Realizar conexión con nodo primario *NDi(p)*.
Con_NDi(r) =	Realizar conexión con nodo respaldo *NDi(r)*.
Crea_procs =	Crear procesos concurrentes de recepción.
Act_est =	Actualizar estructura.
Alm_arch =	Almacenar archivo.

De acuerdo a la figura 4.11, en el estado Q0, que representa el arranque de operaciones, se crean *n* procesos receptores de estructuras; donde *n* es igual al número de Nodos de Adquisición en el SCD. Estos procesos servirán como receptores secundarios, para que siempre exista algún proceso disponible para la recepción de estructuras.

Una vez de que se hayan creado los procesos receptores secundarios, se realiza una verificación del estado operativo del nodo respaldado por el nodo NPi(A), es decir del nodo NPi(B). Una vez que se establece el modo operativo (estado Q2 modo operativo *Primario-Primario* o estado Q3 modo operativo *Primario-Respaldo*), el *mc* puede ofrecer sus servicios de transferencia de información.

En cualquiera de los dos estados anteriores (Q2 y Q3), el nodo NPi(A) podrá recibir estructuras provenientes de los NA's. Estas estructuras serán recibidas por los *n* procesos receptores secundarios, ya que cada uno de estos procesos estará asignado a recibir estructuras de un Nodo de Adquisición en particular.

Una diferencia notable entre el protocolo de operación de los Nodos de Presentación y el protocolo de operación de los Nodos de Adquisición es la siguiente:

Como no existe transferencia de estructuras entre Nodos de Presentación, no existe la solicitud de envío de estructuras de Nodos de Presentación hacia otros Nodos de Presentación. Lo anterior implica, que los estados vigía y de creación de procesos transmisores, en el protocolo de operación de los NA's (estados Q6 y Q7), son suprimidos en este protocolo. A partir de este punto, el protocolo de operación de los Nodos de Presentación se comporta de la misma forma que el protocolo de operaciones de los Nodos de Adquisición.

Cuando se recibe un mensaje de inicio de transferencia, indicando que se requiere la recepción de información, el protocolo de operación llevará a cabo una transición al estado Q4, donde recibirá la información y, así mismo, la reconocerá. Una vez que haya terminado la transferencia o exista una falla en el nodo transmisor, se regresará al estado Q1.

Si se requiere enviar un mensaje, comando o archivo, se realiza una transición al estado Q5, donde se realizan los enlaces con los nodos receptores solicitados. Si no se logra el enlace, se regresa al estado Q1; en caso contrario se pasa al estado Q6.

En el estado Q6, se envía la información requerida. Si existe una falla en el nodo receptor, se retornará al estado Q5, donde se intentará restablecer el enlace o se iniciará el enlace con el nodo receptor respaldo, del nodo receptor original. Si se logró transmitir toda la información solicitada, a continuación se realiza la transición hacia el estado Q1.

Los estados Q7, Q8 y Q9 son equivalentes a los estados Q8, Q9 y Q10 del protocolo de operación, de los Nodos de Adquisición. Si ocurre una falla en el nodo NPi(A), se realiza la transición hacia el estado Q7, donde se permanecerá hasta que el nodo esté listo. Cuando el nodo se encuentre listo para operar nuevamente, se trasladará al estado Q8, donde se esperará la reinicialización. Si se recibe esta reinicialización, se realiza la transición al estado Q11, donde se esperará una inicialización de las operaciones del *mc*.

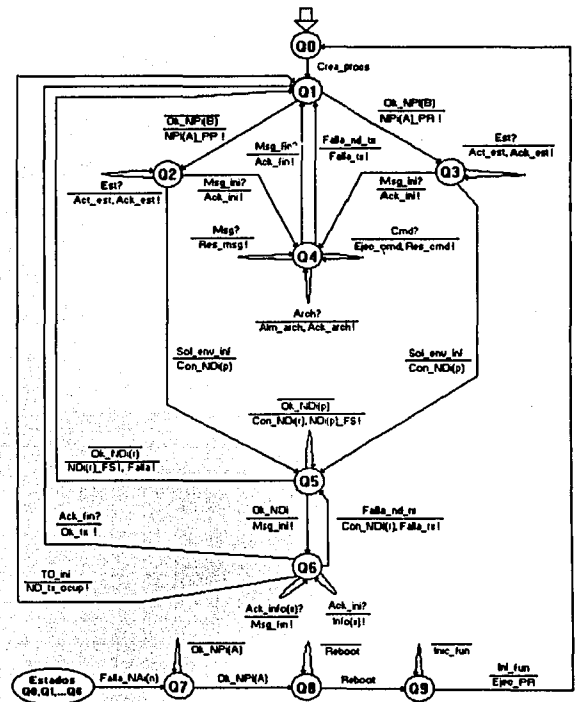

Figura 4.11. MEF del protocolo de operación de los Nodos de Presentación.

CAPÍTULO 5: IMPLANTACIÓN DEL MECANISMO DE COMUNICACIÓN CON TOLERANCIA A FALLAS.

5.1 Introducción.

En este capítulo se describe la cuarta fase del desarrollo del mecanismo de comunicación. Una vez que se ha terminado la fase de diseño, se procede a realizar la implantación del mecanismo. En esta fase, todos los intercambios de mensajes, acciones y cambios de estado, realizados por el mecanismo de comunicación, que fueron representados por medio de Máquinas de Estado Finito, son implantados en un lenguaje de programación.

En la implantación del mecanismo de comunicación, se desarrolla el conjunto de programas que realizarán todas las funciones que fueron especificadas en el diseño. Para el desarrollo de este conjunto de programas se utilizan diversas notaciones, como: diagramas de flujo de datos, diagramas de estructura, etc., que son herramientas desarrolladas para expresar los diseños de software.

El desarrollo de una aplicación de software se realiza en varias etapas [10], que son:

1) Se deben establecer los subsistemas que componen a la aplicación de software.

2) Cada subsistema debe dividirse en componentes individuales (programas) y ha de establecerse la especificación de los subsistemas, definiendo la operación de los mismos.

3) Cada componente del subsistema se implanta como un conjunto de funciones, que actúan de manera conjunta.

4) Se lleva a cabo una refinación de cada una de las funciones.

5) Se especifican los algoritmos utilizados en cada una de las funciones.

Las principales características que debe cumplir una aplicación de software son: mantenibilidad, coherencia y poco acoplamiento. La mantenibilidad consiste en minimizar el costo de los cambios de la aplicación, y esto significa que la aplicación tiene que ser comprensible y que las modificaciones deben tener un efecto local. La coherencia se logra, si los componentes de la aplicación muestran un alto grado de relación funcional, es decir, cada componente de la aplicación es esencial para que ese sistema alcance su propósito.

El acoplamiento está relacionado con las conexiones entre componentes. Los sistemas acoplados tienen conexiones fuertes, en las que sus componentes dependen uno de otro, mientras que los sistemas poco acoplados se conforman de componentes independientes o casi independientes. Las ventajas de los sistemas con coherencia y poco acoplamiento es que cualquier componente se puede reemplazar por uno equivalente con poco o nigún cambio en los otros componentes del sistema.

5.2 Consideraciones del mecanismo de comunicación.

Considerando que la aplicación desarrollada conforma parte de un prototipo, previo a la descripción de los componentes del mecanismo de comunicación diseñado, debemos establecer algunos puntos importantes: El mecanismo de comunicación diseñado es un prototipo. Este prototipo se desarrolló con una arquitectura distribuida integrada por cinco PC's 386, enlazadas por una red ARCNET y que trabaja bajo el medio ambiente del Sistema Operativo de Tiempo Real QNX.

De las cinco PC's, tres de ellas se asignaron como tres Nodos de Adquisición, formando entre ellas el anillo lógico descrito en la figura 5.1, donde el nodo #2 respalda al nodo #4, el nodo #4 respalda al nodo #6 y el nodo #6 respalda al nodo #2.

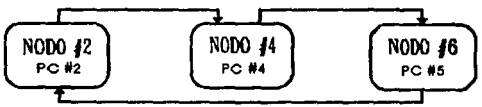

Figura 5.1. Nodos de Adquisición para el mecanismo de comunicación.

Las dos PC's restantes, se asignaron como dos Nodos de Presentación, respaldándose una con otra (véase figura 5.2). Así, el nodo #1 respalda al nodo #3 y viceversa, el nodo #3 respalda al nodo #1.

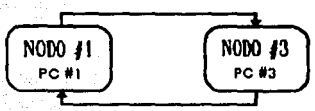

Figura 5.2. Nodos de Presentación para el mecanismo de comunicación.

Bajo este enfoque se realizó la implantación del mecanismo de comunicación, cumpliendo con los requerimientos de comunicación, disponibilidad y confiabilidad establecidos.

5.3 Subsistemas del mecanismo de comunicación.

Como primera etapa de la implantación del mecanismo de comunicación, debemos definir los subsistemas que conformarán al mecanismo. En los diseños en MEF del capítulo 4, se expresa la primer división del mecanismo de comunicación, en dos subsistemas: El subsistema para los Nodos de Adquisición y el subsistema para los Nodos de Presentación, como lo ilustra la figura 5.3.

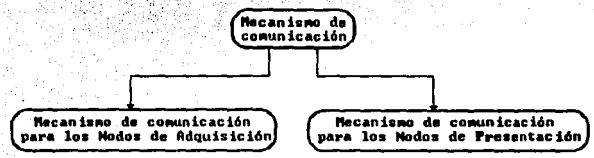

Figura 5.3. Subsistemas del mecanismo de comunicación.

Como segunda etapa, se determinan los componentes que conforman estos subsistemas, donde cada componente debe tener tareas específicas. De acuerdo a las MEF's para los Nodos de Adquisición, podemos dividir al subsistema de los NA's en los componentes de: Inicio y configuración, Recepción de información, Transmisión de información y Tolerancia a fallas. Estos componentes se muestran en la figura 5.4. De igual forma, los componentes del subsistema para los Nodos de Presentación, serían los mostrados en la figura 5.5.

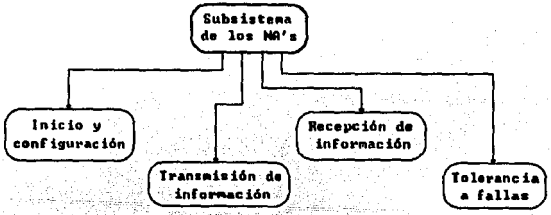

Figura 5.4. Componentes del subsistema de los Nodos de Adquisición.

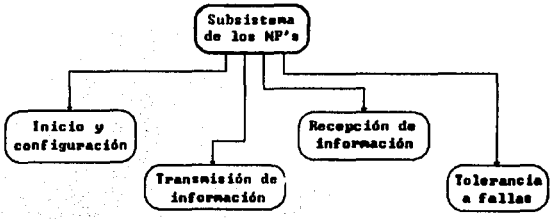

Figura 5.5. Componentes del subsistema de los Nodos de Presentación.

5.3.1 Subsistema para los Nodos de Adquisición.

En este subsistema, como se muestra en la figura 5.4, se pueden describir por separado cada uno de sus componentes. Cada componente puede dividirse en varios módulos, donde cada módulo realizará una función específica.

5.3.1.1 Inicio y configuración.

En este componente se realizan todas las funciones de inicialización y configuración, y se establece el modo operativo con el que trabajará el mecanismo de comunicación. Estas funciones son las encargadas de preparar al mecanismo de comunicación para que realice sus operaciones de transferencia (envío y recepción), es decir, realizan las acciones especificadas en el protocolo de respaldo de los NA's. Estas funciones se agrupan en cuatro módulos, que se muestran en el diagrama de la figura 5.6.

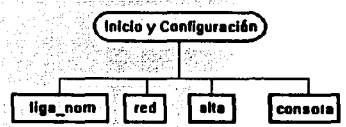

Figura 5.6. Módulos de Inicio y Configuración.

1) Módulo liga_nom().

Este módulo le permite al mecanismo de comunicación obtener el número del nodo, donde se está ejecutando (*nid*) y ligar el proceso (ejecución del mecanismo) a este nodo, por medio del registro del proceso bajo un nombre (por ejemplo, si se encuentra en el nodo 2 se registrará con el nombre "/nodo2"). Con este registro se le asigna un identificador, que se guarda en una variable (*id*). Una vez que el proceso se ha registrado, los demás procesos localizados en nodos remotos, pueden localizarlo mediante la búsqueda del nombre bajo el cual se registró. El pseudocódigo de este módulo se muestra a continuación:

```
MÓDULO liga_nom():
Inicio
 Obtener número de nodo actual nid;
 Registrar proceso en nid bajo el nombre "/nodonid";
 Si (Registro igual a ERROR) Entonces
  Retornar ERROR;
 Sino
  Retornar identificador del registro del proceso id;
 FinSi
Fin
```

2) Módulo red(nid).

Este módulo define la configuración del anillo lógico, que forman los Nodos de Adquisición. De acuerdo a la configuración mostrada en la figura 5.1, el argumento de entrada *nid*, obtendrá el nodo respaldo y el nodo respaldado del nodo donde se encuentra ejecutándose el mecanismo de comunicación. Las variables *nr* y *nrd* guardarán el valor del número del nodo de respaldo y respaldado del nodo *nid*, respectivamente. El pseudocódigo de este módulo es como sigue:

```
MÓDULO red(nid):
Inicio
  Opciones(nid)
    Caso 2: Asignar a nr = 6;
            Asignar a nrd = 4;
            FinCaso
    Caso 4: Asignar a nr = 2;
            Asignar a nrd = 6;
            FinCaso
    Caso 6: Asignar a nr = 4;
            Asignar a nrd = 2;
            FinCaso
  FinOpciones
Fin
```

3) Módulo **alta()**.

Este módulo realiza la notificación, al nodo respaldo y nodo respaldado, que el mecanismo de comunicación en el nodo actual ha sido dado de alta y comienza sus operaciones. Esta notificación la realiza con la verificación de que los nodos (respaldo y respaldado) están activos y el mecanismo de comunicación en esos nodos ya se encuentra ejecutándose. En este caso, se les envía un mensaje indicando el inicio de operaciones. La búsqueda del proceso respaldado permite conocer el modo operativo (*Primario-Primario* o *Primario-Respaldo*) con el cual estará trabajando el mecanismo. El pseudocódigo de este módulo es:

```
MÓDULO alta():
Inicio
  Buscar nodo respaldo "/nodonr";
  Si (nodo respaldo está activo) Entonces
    Notificar nodo respaldo activo;
    Asignar a msg0.inicio = "up";
    Enviar mensaje de alta (msg0) hacia nodo respaldo;
  Sino
    Notificar nodo respaldo inactivo;
  FinSi
  Buscar nodo respaldado "/nodonrd";
  Si (nodo respaldado está activo) Entonces
    Notificar nodo respaldado inactivo;
    Asignar a msg0.inicio = "ur";
    Enviar mensaje de alta (msg0) hacia nodo respaldado;
  Sino
    Notificar nodo respaldado inactivo;
  FinSi
Fin
```

4) Módulo **consola()**.

Este módulo tiene la función de inicializar la consola de trabajo actual, con la finalidad de que pueda estar pendiente, simultáneamente, de dos eventos importantes:

1) La recepción de algún mensaje enviado por el mecanismo de comunicación de un nodo remoto (mensaje de inicio de transferencia de información), un mensaje de alta por parte de algún Nodo de Adquisición o algún mensaje de solicitud de servicio de transferencia, por parte de algún programa de aplicación, y

2) La solicitud de envío de información por parte de un usuario del mecanismo de comunicación, mediante el desplegado de un menú de servicios de transmisión (envío de mensaje, comando o archivo).

Este módulo, inicialmente, obtiene el número de consola en la que se está ejecutando el mecanismo de comunicación. Después, limpia ("resetea") todos los canales de E/S de dicha consola y la prepara para la recepción de los eventos mencionados anteriormente. Una vez que se ejecuta este módulo, el mecanismo de comunicación quedará en espera de que ocurra alguno de los eventos anteriores. Si se recibe un mensaje de inicio de transmisión, automáticamente el mecanismo de comunicación comienza a ejecutar el componente de Recepción de información. Si se recibe una solicitud de transferencia de información, el mecanismo de comunicación pasa al componente de Transmisión de información.

MÓDULO consola():
Inicio
  Obtener el número de consola actual;
  Limpiar ("resetear") canales E/S de la consola actual;
  Desplegar menú de servicios de transferencia;
  Esperar evento;
  Si (se recibio mensaje de inicio de transmisión (msg0.inicio = "ok")) Entonces
    Llamar a **recibir(FALSO)**;
  FinSi
  Si (se recibio mensaje de alta (msg0.inicio = "up" o msg0.inicio = "ur")) Entonces
    Replicar mensaje (msg1);
    Notificar alta de nodo;
  FinSi
  Si (se recibio solicitud de transmisión, por parte de un usuario) Entonces
    Llamar a **envio(***tipo de transferencia***)**;
  FinSi
  Si (se recibio mensaje de solicitud de transmision de archivo (msg0.inicio = "db")) Entonces
    Replicar mensaje (msg1);
    Llamar a **obten_nombre()**;
  FinSi
  Si (se recibio mensaje de solicitud de transmisión de mensaje o comando (msg0.inicio = "mg" o msg0.inicio = "cd")) Entonces
    Replicar mensaje (msg1);
    Llamar a **pids(***nodo receptor, tipo de transferencia***)**;
  FinSi
Fin

5.3.1.2 Transmisión de información.

En este componente se realizan tanto las transferencias de estructuras en memoria, hacia los Nodos de Presentación, como las transferencias de mensajes, comandos y archivos hacia cualquier nodo del sistema. Debido a que uno de los requerimientos principales del mecanismo de comunicación consiste en transferir estructuras de la BDTR hacia todos los Nodos de Presentación, es necesario que el mecanismo este siempre disponible para ejecutar este tipo de transferencia. Por esta razón, se requiere que el mecanismo contenga más de un proceso transmisor de información.

El S.O. QNX nos permite la creación de procesos "hijos" dentro de una aplicación. Estos procesos se estarán ejecutando concurrentemente con el proceso principal (aplicación). Con esta característica, podríamos crear un proceso "hijo" que se encargue exclusivamente del envío de estructuras. Así, el proceso principal se encargaría solamente del envío de mensajes, comandos y archivos (véase figura 5.7).

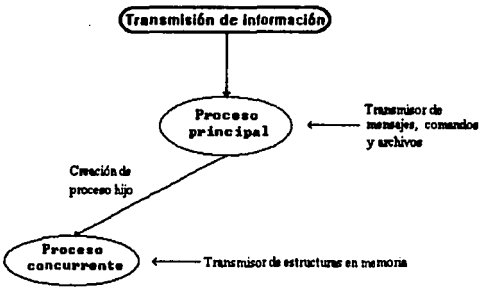

Figura 5.7. Creación de proceso transmisor concurrente.

Como el mecanismo de comunicación tiene que trabajar en un ambiente de Tiempo Real, donde se debe responder en un tiempo finito y específico, un proceso transmisor de estructuras no es lo suficientemente eficiente para poder transmitir las estructuras hacia los *n* Nodos de Presentación que existan en el sistema (en nuestro caso existen dos Nodos de Presentación). Para resolver este problema, se pueden crear *n* procesos "hijos", que se dediquen, cada uno, a transmitir las estructuras hacia los *n* Nodos de Presentación; así, cada "hijo" se dedicará a enviar estructuras hacia un Nodo de Presentación específico.

En conjunto con los procesos transmisores anteriores, debe existir un proceso adicional, que se dedique a la recepción de la solicitud de envío de estructuras (esta solicitud incluye la estructura que se requiere enviar) y que coordine el envío de dicha estructura en los demás procesos "hijos". Este proceso se comunicará, directamente, con el proceso de adquisición asociado al nodo donde se está ejecutando el mecanismo de comunicación. El proceso de adquisición será el que solicite el servicio de transferencia de estructuras.

Una vez que el proceso "coordinador" recibió la solicitud, tendrá la responsabilidad de comunicarse con los *n* procesos transmisores, para que éstos envíen la estructura hacia los Nodos de Presentación. Todo lo anterior puede ilustrarse en el diagrama de la figura 5.8.

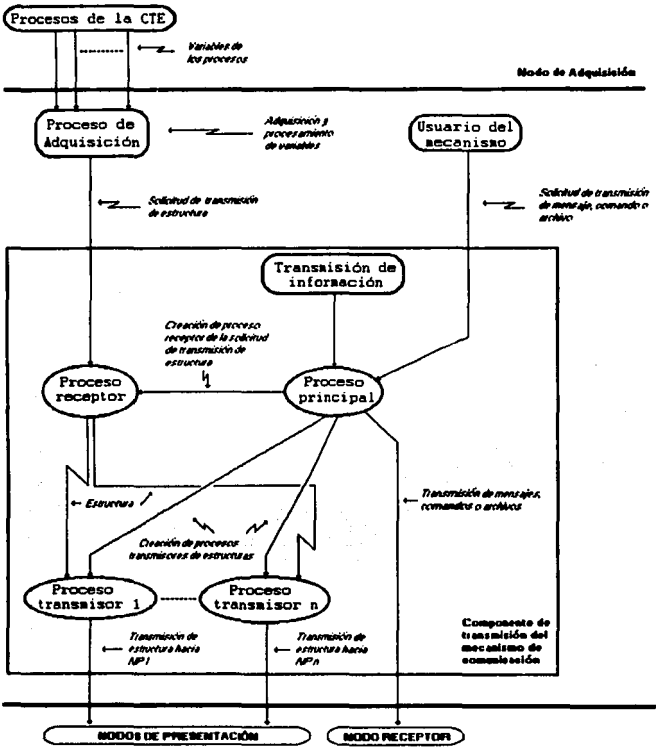

Figura 5.8. Diagrama de flujo de datos para la transmisión de información.

Las solicitudes de servicio de transferencia de información, se obtienen de dos formas distintas:

1) Por solicitud mediante un menú de servicios. El mecanismo de comunicación desplegará un menú (interface visual usuario-aplicación), que indicará los tipos de transferencias que ofrece como servicios. El usuario del mecanismo seleccionará el servicio requerido (envío de mensaje, comando o archivo). El mecanismo le solicitará al usuario, los parámetros necesarios para realizar la transferencia (el mensaje, comando o archivo y el nodo receptor), para después realizar todas las funciones requeridas para establecer y realizar la comunicación (véase figura 5.9).

2) Por solicitud directa de una aplicación al mecanismo de comunicación. En este caso, podemos ver que la solicitud de envío de estructuras es de este tipo: por ejemplo, un proceso de aplicación (proceso de adquisición que se encuentra ejecutándose en el mismo nodo que el mecanismo de comunicación), se comunica directamente con el mecanismo, para solicitar el servicio de transferencia de estructuras.

Para los servicios de transferencia de información, solicitados por programas de aplicación, existe un módulo independiente del mecanismo de comunicación, que permite a los programas de aplicación solicitar los servicios de transferencia de información directamente al mecanismo de comunicación.

Este módulo, llamado interface aplicación-aplicación (IAA), permitirá a los programas de aplicación, tener acceso a todos los servicios de transferencia de información de una manera transparente y sencilla. De esta manera, existirán dos tipos de transferencia: Interactiva, cuando el solicitante es un usuario del mecanismo de comunicación y una transferencia transparente, cuando los solicitantes de los servicios son programas de aplicación (véase figura 5.10).

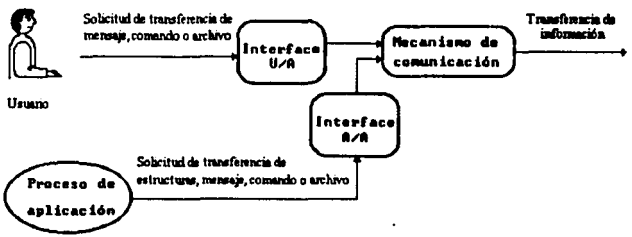

Figura 5.9. Tipos de solicitud para transferencia de información.

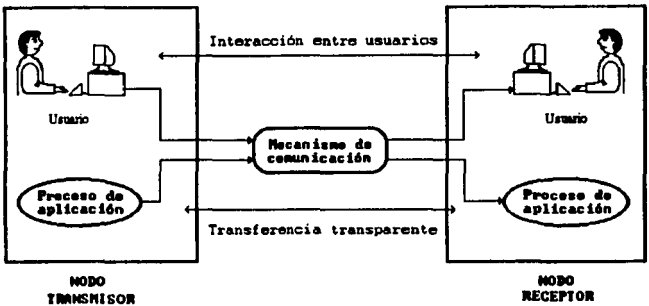

Figura 5.10. Clases de transferencia: interactiva y transparente.

Los módulos que constituyen al componente de Transmisión de información se muestran en el diagrama de la figura 5.11.

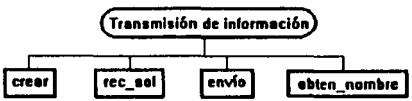

Figura 5.11. Módulos de Transmisión de información para los Nodos de Adquisición.

1) Módulo crear(n,i,*zid).

El módulo **crear()** tiene la función de crear *n* procesos "hijos" transmisores de estructuras. El parámetro *i* sirve como identificador numérico para el proceso creado. El parámetro *zid* servirá para apuntar a un arreglo de direcciones, donde se almacenarán los identificadores del sistema para los procesos creados (PID's).

Cada proceso creado llamará a la función **env_est()**, que tiene como tarea el envío de la estructura hacia un Nodo de Presentación específico. Esta función, a su vez, es la responsable de la creación de los enlaces con los NP's, como lo muestra el diagrama de la figura 5.12.

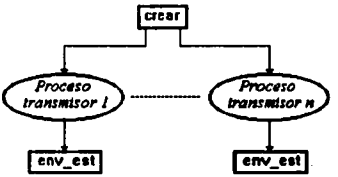

Figura 5.12. Diagrama del módulo crear( ).

El pseudocódigo de este módulo se muestra a continuación:

```
MÓDULO crear(n,i,*zid):
Inicio
  Si (n es mayor que 0) Entonces
    Generar proceso env_est(i);
    Asignar identificador de proceso a *zid;
    Incrementar i;
    Incrementar zid;
    Decrementar n;
    Llamar a crear(n,i,zid);
  FinSi
Fin


Módulo env_est(i).
```

El módulo **env_est()** estará a la espera de una recepción de estructura, para después realizar la conexión con el NP correspondiente y enviar la estructura hacia dicho nodo. El parámetro *i* le indicará con cuál NP deberá conectarse.

Cabe señalar que en cada NP habrá *n* procesos receptores de estructuras, donde *n* representa al número de NA's que existen en el sistema (en nuestro caso existen tres NA's). Esto se explica, en el componente de Recepción de información del subsistema del mecanismo de comunicación para los NP's.

De acuerdo con lo anterior, el parámetro *i* y el número de NA (*nid*), desde donde se están enviando las estructuras, determinarán el proceso receptor con el cual se deberá enlazar este módulo. La variable *proceso* contendrá el nombre del proceso con el cual se realizará el enlace.

```
MÓDULO env_est(i):
Inicio
  Recibir estructura (msg2);
  Asignar a proceso = "/nodo";
  Opciones (i)
    Caso 1: Concatenar(proceso,"1");
            FinCaso
    Caso 2: Concatenar(proceso,"3");
            FinCaso
  FinOpciones
  Opciones (nid)
    Caso 2: Concatenar(proceso,"a");
            FinCaso
    Caso 4: Concatenar(proceso,"b");
            FinCaso
    Caso 6: Concatenar(proceso,"c");
            FinCaso
  FinOpciones
  Realizar conexión con proceso proceso;
  Enviar estructura (msg2) hacia proceso;
  Replicar mensaje recibido;
Fin
```

2) Módulo rec_sol(*zid).

Este módulo es el responsable de recibir la solicitud de transferencia de estructura, emitida por el proceso de adquisición del sistema. Hay que recordar que dicha solicitud contiene la estructura que se desea difundir a los NP's. El diagrama de este módulo se muestra en la figura 5.13.

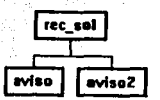

Figura 5.13. Diagrama de estructura del módulo rec_sol( ).

Se debe considerar que, el proceso de adquisición utilizará una función especial para la solicitud de transferencia de estructura. Esta función viene contenida en la interface aplicación-aplicación del mecanismo de comunicación y en esta función se incluye una rutina que le permite el enlace con el mecanismo y el paso de la estructura, por medio de un mensaje. El proceso de recepción de solicitud de envío de estructuras es independiente al proceso principal del mecanismo de comunicación, por lo que requiere registrarse, independientemente, bajo un nombre que sea diferente al del proceso principal.

Una vez que el módulo **rec_sol()** recibe una solicitud, mandará llamar a la función **aviso()**, que tiene como objetivo el avisar al nodo respaldo, del nodo actual, que se va a realizar una transferencia de estructura. Este aviso permitirá que el nodo respaldo realice rutinas de tolerancia a fallas, concernientes a la vigilancia de la transferencia de la estructura.

Después del aviso de transferencia de estructura, el módulo **rec_sol()** enviará la estructura, solicitada en transferencia, hacia todos los procesos "hijos" transmisores. El parámetro ***zid*** contiene el apuntador al arreglo de direcciones de los identificadores de cada proceso "hijo". Estos se encargarán de la difusión de la estructura hacia los NP's. Finalmente, se llama a la función **aviso2()** que le indicará, al nodo respaldo, que la transferencia se ha realizado. Este aviso causará que el nodo de respaldo termine las rutinas de tolerancia a fallas.

MÓDULO rec_sol(*zid):
Inicio
  Obtener número de nodo actual *nid*;
  Registrar proceso en *nid* bajo el nombre "/nodo*nid*/b";
  Para ( )
    Recibir solicitud de envío de estructura (msg2);
    Llamar a **aviso()**;
    Para (***zid***)
      Enviar estructura (msg2) hacia proceso *zid;
      Incrementar zid;
    FinPara
    Llamar a **aviso2()**;
    Replicar solicitud recibida;
  FinPara
Fin

Módulo **aviso()**.

El módulo **aviso()** tiene la función de enviar un mensaje, hacia el NA respaldo del NA actual, indicando que se va a realizar una transferencia de estructura. Este aviso sirve para que el NA respaldo realice las funciones de tolerancia a fallas.

Inicialmente, el módulo **aviso()** debe localizar, en el NA respaldo, al proceso que se encargará de la vigilancia de la transferencia. Este proceso será registrado, independientemente, del proceso principal del mecanismo de comunicación. Esto último se describe con detalle en el componente de Tolerancia a fallas.

La variable *nr*, definida por el módulo **red()**, ayuda a conocer el número de NA respaldo del NA actual. Una vez que se localizó al NA respaldo, el módulo le enviará un mensaje, que contendrá el número del NA (*nid*) que requiere ser vigilado durante la transferencia. Después de este mensaje, se envía la estructura que se transmitirá hacia los NP's. Lo anterior se realiza con el objeto de que, si existe una caída del NA (*nid*) durante la transferencia, el NA respaldo tenga en su poder la estructura que debe ser enviada hacia los NP's y se encargue de su envío. El pseudocódigo de este módulo es:

MÓDULO aviso():
Inicio
  Realizar conexión con proceso respaldo "/nodo*nr*a";
  Asignar a msg10.nid_tx = *nid*;
  Enviar mensaje (msg10) hacia proceso respaldo;
  Enviar mensaje (msg2) hacia proceso respaldo;
Fin

3) Módulo envío(tipo).

Este módulo tiene la función de obtener el nodo de destino (la variable *nodo*), hacia el cual se dirigirá la información, para después llamar al módulo **pids()**, que administra la transmisión de información. El parámetro *tipo* nos indica el tipo de transferencia solicitada, de acuerdo con los siguientes valores: (1) para transferencia de mensaje, (2) para transferencia de comando y (4) para transferencia de archivo. El diagrama de estructura del módulo **envío()**, se presenta en la figura 5.14.

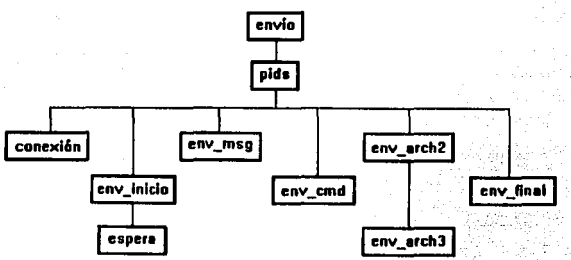

Figura 5.14. Diagrama de estructura del módulo envío( ).

Módulo **pids(nodo,tipo)**.

El módulo **pids()** requiere de dos argumentos de entrada, que son: *nodo*, que es el nodo hacia el cual se enviará la información (nodo receptor) y *tipo*, que indica el tipo de información por transmitir (mensaje, comando o archivo).

La primer acción que realiza este módulo, consiste en establecer un enlace (circuito virtual), con el nodo receptor, mediante el módulo **conexión()**. Una vez establecido el enlace con el nodo receptor, se envía el mensaje de inicio de transmisión que contiene la información de inicialización de la transferencia (nodo transmisor, nodo receptor, tipo de información, etc.). Este envío se realiza por medio del módulo **env_inicio()**.

La segunda acción consiste en verificar el tipo de transferencia solicitada. Si se requiere transmitir un mensaje, se llama al módulo **env_msg()**. Si se va a transmitir un comando, se llama al módulo **env_cmd()**. Para la transmisión de un archivo se requiere del módulo **env_arch2()**. El módulo **env_final()**, tiene la función de transmitir el mensaje de fin de transmisión, que le indicará al nodo receptor la terminación de la transferencia de información.

Si la solicitud de transferencia fue hecha por un programa de aplicación, el mensaje, comando o archivo que se requieren transferir serán recibidos, por el mecanismo de comunicación, por medio de mensajes enviados por la interface aplicación-aplicación (IAA). El pseudocódigo del módulo **pids()** es el siguiente:

```
MÓDULO pids(nodo,tipo):
Inicio
  Asignar a rpid valor retornado por conexión(nodo);
  Si (rpid es diferente de ERROR) Entonces
    Llamar a env_inicio(rpid,nodo,tipo);
    Si (env_inicio() retorna un ERROR) Entonces
      Notificar nodo receptor ocupado;
      Si (solicitud hecha por programa de aplicación (msg0.ban = 1)) Entonces
        Asignar a msg7.ack = "Nodo ocupado";
        Si (se solicito transferencia de mensaje (msg0.tipo = 1)) Entonces
          Recibir mensaje (msg5);
        FinSi
        Si (se solicito transferencia de comando (msg0.tipo = 1)) Entonces
          Recibir mensaje (msg5);
        FinSi
      FinSi
      Replicar con mensaje (msg7);
    Sino
      Opciones (tipo)
        Caso 1: Llamar a env_msg(rpid);
                Si (env_msg() retorna un ERROR) Entonces
                  Notificar falla en nodo receptor;
```

```
            Llamar a pids(nodo,tipo);
          Sino
            Llamar a env_final(rpid);
          Finsi
          FinCaso
      Caso 2: Llamar a env_cmd(rpid);
          Si (env_cmd() retorna un ERROR) Entonces
            Notificar falla en nodo receptor;
            Llamar a pids(nodo,tipo);
          Sino
            Llamar a env_final(rpid);
          Finsi
          FinCaso
      Caso 4: Llamar a env_arch2(rpid);
          Si (env_arch2() retorna un ERROR) Entonces
            Notificar falla en nodo receptor;
            Llamar a pids(nodo,tipo);
          Sino
            Llamar a env_final(rpid);
          Finsi
          FinCaso
    FinOpciones
  FinSi
 Sino
  Notificar enlace fallido;
  Si (solicitud hecha por programa de aplicación (msg0.ban = 1)) Entonces
    Asignar a msg7.ack = "Enlace fallido";
    Si (se solicito transferencia de mensaje (msg0.tipo = 1)) Entonces
      Recibir mensaje (msg5);
    FinSi
    Si (se solicito transferencia de comando (msg0.tipo = 1)) Entonces
      Recibir mensaje (msg5);
    FinSi
  FinSI
  Replicar mensaje (msg7);
 FinSi
Fin
```

En caso de una falla, por parte del nodo receptor, en cualquiera de los módulos de transmisión, se vuelve a llamar a **pids()**, para tratar de reestablecer la comunicación ya sea con el nodo receptor original o con su nodo de respaldo.

Módulo conexión(nodo).

Este módulo tiene la función de establecer un enlace, con el mecanismo de comunicación, en el nodo receptor indicado por el parámetro *nodo*. Si el mecanismo se encuentra en ejecución en el nodo receptor, se retorna el identificador del proceso del mecanismo (*rpid*).

Si no se encuentra activo el mecanismo o el nodo receptor está fuera de funcionamiento, se retorna un error. En este caso se inicia una rutina tolerante a fallas, que tiene comopropósito establecer conexión con el nodo respaldo del nodo receptor.

Módulo env_inicio(rpid,nodo,tipo).

Una vez que se establece el enlace con el nodo receptor, se procede al envío del mensaje de inicio de transmisión hacia el proceso *rpid*. Este envío tiene una característica importante, por ser el primer mensaje que se intercambiará, también tiene la función de indicar si el nodo receptor está disponible para recibir información o no. Esto se realiza mediante la creación de un temporizador ("timer") de espera (módulo **espera()**).

Cada vez que se envía el mensaje de inicio, se inicializa este temporizador. Si el temporizador llega a un tiempo predeterminado (en el mecanismo de comunicación de 5 seg.) y aún no se recibe respuesta al mensaje de inicio, el módulo asumirá que el nodo receptor se encuentra ocupado en alguna otra transferencia y retornará un valor de ERROR. Si la respuesta es recibida dentro del tiempo definido, el módulo retornará un valor EXITO, y se continuará con la transferencia (véase figura 5.15).

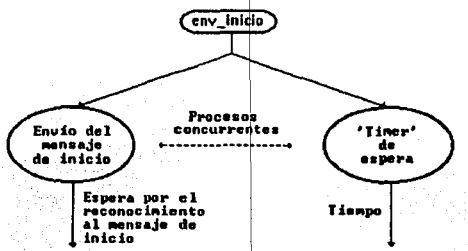

Figura 5.15. Creación de un temporizador ['timer'] de espera.

Los pseudocódigos de los módulos **env_inicio()** y **espera()** son mostrados a continuación:

```
MÓDULO env_inicio(rpid,nodo,tipo):
Inicio
  Asignar a msg0.inicio = "ok";
  Asignar a msg0.nid_tx = nid;
  Asignar a msg0.nid_rx = nodo;
  Asignar a msg0.inicio = tipo;
  Generar proceso espera();
  Enviar mensaje (msg0) hacia proceso rpid;
  Si (tiempo transcurrido y no se obtuvo respuesta) Entonces
    Notificar nodo receptor ocupado;
    Retornar ERROR;
  Sino
    Terminar proceso espera();
    Retornar EXITO;
  FinSi
Fin

MÓDULO espera():
Inicio
  Contar  5 segundos;
  Notificar tiempo transcurrido;
Fin
```

Módulo env_msg(rpid).

Este módulo lleva a cabo la transmisión de un mensaje. Para realizar lo anterior, lo primero que realiza el módulo es pedirle al usuario el mensaje que se requiere transmitir. Si la solicitud fue hecha por un programa de aplicación, el mensaje se recibe por medio de la IAA. A continuación, se envía el mensaje hacia *rpid* y se espera por la respuesta al mensaje. Cuando se recibe la respuesta, se despliega en pantalla o se retorna la respuesta a la IAA, según sea el caso. En caso de que se detecte la caída del nodo receptor, se retornará el valor de ERROR; en caso contrario, se retorna un EXITO. Su pseudocódigo es:

```
MÓDULO env_msg(rpid):
Inicio
  Si (solicitud fue hecha por programa de aplicación (msg0.ban = 1)) Entonces
    Recibir mensaje (msg5);
  Sino
    Obtener mensaje por transmitir;
    Asignar a msg5.mss = mensaje;
  FinSi
  Enviar mensaje (msg5) hacia proceso rpid;
  Si (Envío retorna ERROR) Entonces
    Asignar a msg0.ban = 0;
    Retornar ERROR;
```

```
  Sino
    Desplegar respuesta;
    Asignar a msg7.ack = "Mensaje recibido";
    Replicar mensaje recibido (msg5) con mensaje (msg7);
    Retornar EXITO;
  FinSi
Fin
```

Módulo **env_cmd(rpid)**.

Este módulo realiza la transmisión de comandos. Primero, se solicita el comando que se requiere ejecutar en el nodo receptor. Después, se pide el tipo de respuesta que se requiere. Se podrá solicitar la notificación de ejecución del comando o se podrá solicitar la información generada por la ejecución. Estos parámetros pueden recibirse por medio de un mensaje, enviado por la IAA, si la solicitud fue hecha por un programa de aplicación. En caso de falla en el nodo receptor, se retoma un ERROR. Su pseudocódigo es:

```
MÓDULO env_cmd(rpid);
Inicio
  Si (solicitud fue hecha por porgrama de aplicación (msg0.ban = 1)) Entonces
    Recibir mensaje (msg6);
  Sino
    Obtener comando por transmitir;
    Asignar a msg6.cmd = comando;
    Obtener tipo de respuesta requerida;
    Asignar a msg6.bandera = tipo;
  FinSi
  Enviar mensaje (msg6) hacia proceso rpid;
  Si (Envio retoma ERROR) Entonces
    Asignar a msg0.ban = 0;
    Retornar ERROR;
  Sino
    Desplegar respuesta;
    Asignar a msg7.ack = "Comando ejecutado";
    Replicar mensaje recibido (msg6) con mensaje (msg7);
    Retornar EXITO;
  FinSi
Fin
```

Módulo **env_arch2(rpid)**.

Este módulo, le ofrece al usuario, el servicio de envío de archivos. El módulo **env_arch2()** obtiene el nombre del archivo por enviar y posteriormente, llama al módulo **env_arch3()**. Este último módulo realiza la verificación de la existencia del archivo.

Si el archivo no existe, entonces se genera un mensaje, indicando la apertura fallida del archivo. En caso contrario, se comienza a leer el archivo y a llenar el mensaje donde se enviará dicho archivo. Un archivo puede utilizar más de un mensaje para lograr su transferencia, tomando en cuenta que un mensaje puede ser de hasta 64 kb.

Cada mensaje contendrá un bloque de almacenamiento (matriz de caracteres de [renglón][columna]) para guardar cada bloque en que sea dividido un archivo, en caso de requerirse la división para la transmisión. Por motivos demostrativos, el mecanismo de comunicación utilizará un bloque de 5 kb (matriz de caracteres de [50][100]). El pseudocódigo de los módulos **env_arch2()** y **env_arch3()** se muestran a continuación:

```
MÓDULO env_arch2(rpid):
Inicio
  Obtener archivo por transmitir;
  Asignar a msg9.nombre = archivo;
  Asignar a valor = env_arch3(rpid);
  Retornar valor;
Fin

MÓDULO env_arch3(rpid):
Inicio
  Abrir el archivo msg9.nombre;
  Si (Archivo no existe) Entonces
    Asignar a msg9.falla = VERDADERO;
    Enviar mensaje (msg9) hacia proceso rpid;
    Si (solicitud fue hecha por programa de aplicación (msg0.ban != 0) Entonces
      Asignar a msg7.ack = "No existe archivo";
      Replicar mensaje recibido (msg9) con mensaje (msg7);
    FinSi
    Retornar EXITO;
  Sino
    Asignar msg9.falla = FALSO;
    Asignar a renglón = 0;
    Mientras (No se alcance fin de archivo) Hacer
      Leer renglón del archivo;
      Asignar a msg9.cadena[renglón] = lectura del archivo;
      Incrementar renglón;
      Si (Renglón leido no es igual al fin de archivo) Entonces
        Si (renglón es igual a 50) Entonces
          Asignar a msg9.indice = renglón;
          Asignar a msg9.bandera = FALSO;
          Asignar a renglón = 0;
          Enviar mensaje (msg9) hacia proceso rpid;
          Si (Envio retorna ERROR) Entonces
            Cerrar archivo;
```

```
          Retornar ERROR;
        FinSi
      FinSi
    Sino
      Asignar a msg9.indice = renglón - 1;
      Asignar a msg9.bandera = VERDADERO;
      Enviar mensaje (msg9) hacia proceso rpid;
      Si (Envio retorna ERROR) Entonces
        Cerrar archivo;
        Retornar ERROR;
      Sino
        Cerrar archivo;
        Asignar a msg7.ack = "Archivo recibido";
        Replicar mensaje recibido (msg9) con mensaje (msg7);
        Retornar EXITO;
      FinSi
    FinSi
  FinMientras
 FinSi
Fin
```

Módulo **env_final(rpid)**.

Este módulo tiene la función de enviar el mensaje de final de transmisión (msg4), hacia el nodo receptor, para indicar la terminación de la transferencia de información.

4) Módulo obten_nombre().

Este módulo es el encargado de comunicarse con la IAA, cuando un proceso de aplicación requiere el servicio de transferencia de archivos. Este servicio puede ofrecer la transferencia de un archivo hacia todos los Nodos de Presentación (con el mensaje msg0.ban = 2). El diagrama de estructura del módulo **obten_nombre()** se muestra en la figura 5.16.

La IAA deberá enviar un mensaje de inicio de transmisión, donde especifique que un proceso de aplicación requiere del servicio de tranferencia de archivo (msg0.inicio = "db"). El mecanismo de comunicación, al recibir tal mensaje, mandará llamar al módulo **obten_nombre()**. Este módulo recibirá el nombre del archivo que se requiere transferir, esto es, el proceso de aplicación que solicitó el servicio, mediante la IAA, enviará el mensaje que contendrá el nombre del archivo (msg9).

Una vez que se tiene el archivo por transferir, el módulo se enlazará con el nodo receptor o con cada NP, si se requiere difundir el archivo, para después enviar un mensaje de inicio de transmisión a cada uno de ellos. A continuación se llama al módulo **env_arch3()**, que será el encargado de transmitir el archivo. Por último, se enviará el mensaje de final de transmisión.

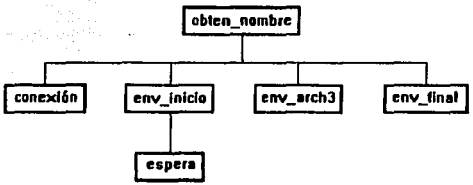

Figura 5.16. Diagrama de estructura del módulo obten_nombre( ).

El pseudocódigo del módulo **obten_nombre()** se muestra a continuación:

```
MÓDULO obten_nombre():
Inicio
  Recibir mensaje (msg9);
  Si (msg0.ban = 2) Entonces
    Para (cada NP) Hacer
      Asignar a rpid = conexión(NP);
      Si (rpid es diferente de ERROR) Entonces
        Llamar a env_inicio(rpid,NP,4);
        Si (env_inicio() retorna ERROR) Entonces
          Notificar nodo ocupado;
          Asignar a ms7.ack = "Nodo ocupado";
          Replicar mensaje recibido (msg9) con mensaje (msg7));
        Sino
          Llamar a env_arch3(rpid);
          Si (env_arch3() retorna ERROR) Entonces
            Notificar falla en nodo receptor;
            Asignar a ms7.ack = "Falla en nodo receptor";
            Replicar mensaje recibido (msg9) con mensaje (msg7));
          Sino
            Llamar a env_final(rpid);
          FinSi
        FinSi
      Sino
        Notificar enlace fallido;
        Asignar a msg7.ack = "Enlace fallido";
        Replicar mensaje recibido (msg9) con mensaje (msg7));
      FinSi
    FinPara
```

```
  Sino
    Asignar a rpid = conexión(msg0.nid_rx);
    Si (rpid es diferente de ERROR) Entonces
      Llamar a env_inicio(rpid,msg0.nid_rx,4);
      Si (env_inicio() retorna ERROR) Entonces
        Notificar nodo ocupado;
        Asignar a ms7.ack = "Nodo ocupado";
        Replicar mensaje recibido (msg9) con mensaje (msg7));
      Sino
        Llamar a env_arch3(rpid);
        Si (env_arch3() retorna ERROR) Entonces
          Notificar falla en nodo receptor;
          Asignar a msg7.ack = "Falla en nodo receptor";
          Replicar mensaje recibido (msg9) con mensaje (msg7));
        Sino
          Llamar a env_final(rpid);
        FinSi
      FinSi
    Sino
      Notificar enlace fallido;
      Asignar a msg7.ack = "Enlace fallido";
      Replicar mensaje recibido (msg9) con mensaje (msg7));
    FinSi
  Fin
Fin
```

5.3.1.3 Recepción de información.

Este componente está conformado por una serie de módulos de recepción, donde cada uno tiene la función de recibir un tipo de información específico. Todos estos módulos pueden verse en la figura 5.17.

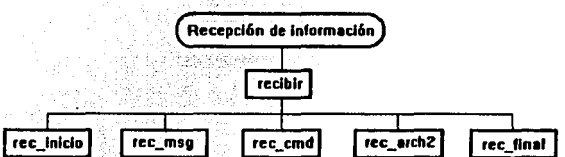

Figura 5.17. Módulos de Recepción de información para los NA's.

1) Módulo recibir(*ban*).

Si el mecanismo de comunicación recibe un mensaje de inicio de transmisión (msg0.inicio = "ok"), se llama a **recibir()**. Una vez que se recibe tal mensaje, este módulo llamará al submódulo **rec_inicio()**, que tiene por objetivo replicar dicho mensaje y poder determinar que tipo de información será recibida.

Ya que se determinó el tipo de información que será recibida (indicado por la variable *tipo* del mensaje msg0), el módulo **recibir()** llama al submódulo **rec_msg()**, si se va a recibir un mensaje. Se llama al submódulo **rec_cmd()**, si se trata de un comando o se llama al submódulo **rec_arch2()**, si se trata de recibir a un archivo. Una vez que se ha recibido la información, se llama al módulo **rec_final()** que nos permitirá recibir el mensaje de final de transmisión.

Si se detecta la caída del nodo transmisor, en cualquiera de los submódulos anteriores, se vuelve a llamar al módulo **recibir()**, para que el mecanismo este atento a una posible retransmisión de información. El parámetro *ban* nos indicará si el módulo **recibir()** fue llamado a causa de una caída del nodo transmisor, lo que permitirá recibir nuevamente el mensaje de inicio de transmisión.

Los tres tipos de información que puede recibir un Nodo de Adquisición son: mensajes, comandos y archivos. La recepción de estructuras en memoria entre Nodos de Adquisición, no es requerida en el mecanismo de comunicación, ya que este tipo de información sólo tendrá un flujo de Nodos de Adquisición hacia Nodos de Presentación. El pseudocódigo del módulo **recibir()** es el siguiente:

```
MÓDULO recibir(ban):
Inicio
  Si (ban es igual a VERDADERO) Entonces
    Recibir mensaje de inicio (msg0);
  FinSi
  Llamar a rec_inicio();
  Si (rec_inicio retorna ERROR) Entonces
    Notificar falla en nodo transmisor;
    Llamar a recibir(VERDADERO);
  Sino
    Opciones (tipo)
      Caso 1: Llamar a rec_msg();
              Si (rec_msg() retorna ERROR) Entonces
                Notificar falla en nodo transmisor;
                Llamar a recibir(VERDADERO);
              Sino
                Llamar a rec_final();
                Notificar transmisión concluida;
              FinSi
              FinCaso
      Caso 2: Llamar a rec_cmd();
              Si (rec_cmd() retorna ERROR) Entonces
```

```
            Notificar falla en nodo transmisor;
            Llamar a recibir(VERDADERO);
          Sino
            Llamar a rec_final();
            Notificar transmisión concluida;
          FinSi
          FinCaso
    Caso 4: Llamar a rec_arch2();
          Si (rec_arch2() retorna ERROR) Entonces
            Notificar falal en nodo transmisor;
            Llamar a recibir(VERDADERO);
          Sino
            Llamar a rec_final();
            Notificar transmisión concluida;
          FinSi
          FinCaso
  FinOpciones
 FinSi
Fin
```

2) Módulo rec_inicio().

Este módulo permite la réplica del mensaje de inicio de transmisión (msg0). Este mensaje contiene la información básica con respecto a la transmisión que está por iniciarse (nodo transmisor, nodo receptor, tipo de información, tipo de solicitud, etc.)

Si existiera alguna falla durante la réplica del mensaje recibido, se retorna el valor de ERROR; en caso contrario, se identifica el tipo de mensaje. Si se trata de un mensaje de alta, se notifica el evento indicando qué nodo fue dado de alta. Si se trata de una transferencia de información, entonces se devuelve el tipo de información que será recibida por el nodo actual, de acuerdo a lo siguiente: (1): para indicar que se trata de un mensaje, (2): para indicar un comando y (4): para indicar un archivo.

```
MÓDULO rec_inicio():
Inicio
 Replicar mensaje recibido;
 Si (Réplica retorna ERROR) Entonces
  Retornar ERROR;
 Sino
  Si (se recibió alta de nodo) Entonces
   Desplegar aviso de evento de alta de nodo;
  Sino
   Retornar tipo de transmisión (msg0.tipo);
  FinSi
 FinSi
Fin
```

3) Módulo rec_msg().

Aquí se recibe un mensaje, se despliega la información contenida en el mensaje y se responde al mensaje recibido. Además, se verifica que la respuesta se reciba en el nodo transmisor del mensaje. El pseudocódigo de este módulo es el siguiente:

```
MÓDULO rec_msg():
Inicio
  Recibir mensaje (msg5);
  Desplegar mensaje recibido;
  Obtener mensaje de respuesta;
  Guardar respuesta en msg7.ack;
  Replicar mensaje recibido (msg5) con mensaje (msg7);
  Si (Réplica retorna ERROR) Entonces
    Retornar ERROR;
  Sino
    Retornar EXITO;
  FinSi
Fin
```

4) Módulo rec_cmd().

Con este módulo se recibe un comando, se ejecuta el comando y la información generada se guarda en un mensaje, mismo que se envía hacia el nodo transmisor del comando. Si el comando no se pudo ejecutar, se responde con un mensaje de error; en caso contrario se envía un mensaje indicando que el comando se ejecutó satisfactoriamente.

```
MÓDULO rec_cmd():
Inicio
  Recibir comando (msg6);
  Ejecutar comando recibido;
  Si (Ejecución retorna ERROR)
    Replicar con mensaje de error;
  Sino
    Almacenar información generada en mensaje (msg7);
    Replicar mensaje recibido (msg6) con mensaje (msg7);
  FinSi
  Si (Réplica retorna ERROR) Entonces
    Retornar ERROR;
  Sino
    Retornar EXITO;
  FinSi
Fin
```

5) Módulo rec_arch2().

Con este módulo, se realiza la recepción de archivos. Dado que el tamaño de los archivos de texto, que se esperan recibir, es grande, este tipo de recepción se realiza por medio de paquetes, donde cada paquete contiene un bloque del archivo. El funcionamiento de este módulo puede ser descrito de la siguiente manera:

```
MÓDULO rec_arch2():
Inicio
  Recibir mensaje (msg9);
  Replicar mensaje recibido;
  Guardar información en disco;
  Si (msg9.bandera es igual a VERDADERO) Entonces
    Retornar EXITO;
  Sino
    Llamar a rec_arch2();
  FinSi
Fin
```

6) Módulo rec_final().

La función de este módulo se limita a recibir y replicar el mensaje de terminación de transferencia de información, que cada nodo transmisor envía una vez que finaliza la transmisión.

5.3.1.4 Tolerancia a fallas.

En el componente de Tolerancia a fallas, se incluyen las funciones requeridas para mantener la disponibilidad en la transferencia de información. Este componente está constituido por los módulos mostrados en la figura 5.18.

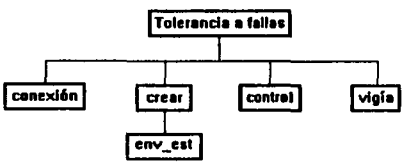

Figura 5.18. Módulos de Tolerancia a fallas.

1) Módulo conexión(nodo).

La función del módulo **conexión()** consiste en establecer un enlace (circuito virtual) con un nodo receptor específico, indicado por el parámetro *nodo*. Además de la función anterior, este módulo tiene otra función, que consiste en establecer un enlace con el nodo respaldo, del nodo receptor inicial, de acuerdo con la configuración del sistema.

Esta función se requiere, en el caso en que el nodo receptor inicial se encuentre inactivo o haya sufrido una falla durante una transferencia de información. Así, este módulo proporcionará una trayectoria opcional, para el flujo de información, hacia el nodo respaldo del nodo receptor caído, manteniendo la disponibilidad en la transferencia de información (véase figura 5.19).

El pseudocódigo de este módulo es el siguiente:

```
MÓDULO conexión(nodo):
Inicio
  Opciones (nodo)
    Caso 1: Asignar a nr = 3;
            FinCaso;
    Caso 2: Asignar a nr = 6;
            FinCaso;
    Caso 3: Asignar a nr = 1;
            FinCaso;
    Caso 4: Asignar a nr = 2;
            FinCaso;
    Caso 6: Asignar a nr = 4;
            FinCaso;
  FinOpciones
  Buscar el proceso "/nodonodo";
  Si (proceso está activo) Entonces
    Asignar a rpid el identificador del proceso;
    Retornar rpid;
  Sino
    Buscar el proceso "/nodonr";
    Si (proceso está activo)
      Asignar a rpid el identificador del proceso;
      Retornar rpid;
    Sino
      Retornar ERROR;
    FinSi
  Finsi
Fin
```

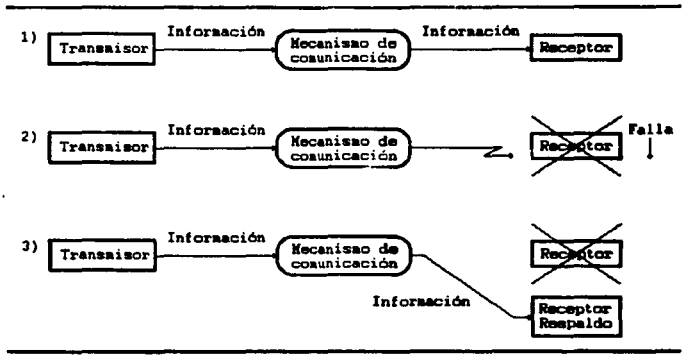

Figura 5.19. Tolerancia a fallas en la recepción de información.

2) Módulo crear().

Este módulo se describió en el componente de Transmisión de Información. El módulo es utilizado por el componente de Tolerancia a fallas, en caso de que un NA*n* esté respaldando a un NA*n+1*, y este NA*n+1* sufra una falla durante una transferencia de estructura hacia los NP's. El módulo **crear()** permitirá al NA*n* transmitir la estructura, que el NA*n+1* estaba transfiriendo, hacia los Nodos de Presentación.

3) Módulo control(*zid).

Este módulo es el encargado de recibir los avisos de inicio y terminación de transferencia de estructuras. Estos avisos le permiten iniciar y terminar la vigilancia del NA respaldado, durante sus transferencias de estructuras.

Este módulo es independiente al proceso principal del mecanismo de comunicación (al igual que el proceso receptor de solicitudes de transferencia de estructuras), por lo que es registrado con un nombre diferente al del proceso principal. Por lo anterior, este módulo trabajará como un proceso concurrente al proceso principal.

El módulo se registrará en el sistema y esperará un aviso de inicio de transferencias de estructuras (mensajes msg10 y msg2) por parte del NA al cuál se está respaldando. Cuando se recibe este inicio, el módulo creará un proceso **vigía()** que se encargará de verificar el estado operativo del NA respaldado (véase figura 5.20). El proceso vigía utilizará el mensaje "msg10" para saber el número del nodo que requiere dicha verificación.

Figura 5.20. Procesos concurrentes en el subsistema de los Nodos de Adquisición.

Después de la creación del proceso **vigía()**, el módulo de **control()** esperará por uno de los dos mensajes siguientes:

1) Un mensaje de terminación de transferencia (msg4.final = "ok"), enviado por el NA respaldado.

2) Un mensaje de falla en el NA respaldado (msg4.final = "fl"), enviado por el proceso **vigía()**.

Si se recibe un mensaje de terminación, el módulo **control()** terminará al proceso **vigía()** y esperará un nuevo mensaje de inicio de transferencia.

En caso de recibir un mensaje de falla, el módulo utilizará el parámetro *zid* (puntero al arreglo de direcciones que contienen los PID's de los procesos, creados por el módulo **crear()**), para enviar la estructura hacia los procesos respaldo transmisores de estructuras. Estos últimos se encargarán de transferir la estructura hacia los NP's, como se explicó anteriormente.

Debido a que una caída en el NA respaldado implica, entre otras cosas, que los procesos de adquisición y el mecanismo de comunicación dejarán de funcionar en ese nodo, el **módulo control()** debe encargarse de comunicar al proceso de adquisición, del nodo respaldo, dicha falla. Esto se realiza, ya que con el aviso de caída del NA respaldado, el proceso de adquisición del nodo respaldo, comenzará a adquirir las variables que le corresponden al nodo caído (hay que recordar la configuración de los NA's y las UA's en el SCD mostrada en la figura 3.4) y, así, dicho proceso de adquisición será el encargado de solicitar la transferencia de estructuras del nodo caído. El aviso al proceso de adquisición se logra mediante el envío del mensaje msg2; así, el proceso de adquisición sabrá desde que estructura deberá comenzar a solicitar la transferencia de estructuras. Con lo anterior, se asegura la disponibilidad de la información, generada en los procesos de la CTE, en los Nodos de Presentación.

Los pseudocódigos de los módulos **control()** y **vigía()** se muestran a continuación:

MÓDULO control(*zid):
Inicio
  Registrar proceso bajo el nombre "/nodo*nida*";
  Asignar a *rpid* el identificador del proceso;
  Para ( ) Hacer
    Recibir mensaje (msg10);
    Replicar mensaje recibido;
    Recibir mensaje (msg2);
    Replicar mensaje recibido;
    Generar proceso **vigía(rpid, msg10.nid_tx)**;
    Recibir mensaje (msg4);
    Replicar mensaje recibido;
    Si (msg4.final es igual "ok") Entonces
      Terminar proceso **vigía()**;
    Sino
      Para (*zid*) Hacer
        Enviar mensaje (msg2) hacia proceso **zid*;
        Incrementar *zid*;
      FinPara
    FinSi
    Enviar mensaje (msg2) hacia proceso de adquisición.
  FinPara
Fin

MÓDULO vigía(rpid, nid_tx):
Inicio
  Checar estado operativo del NA *nid_tx*;
  Si (se detecta falla en NA) Entonces
    Enviar mensaje de falla (msg4.final = "f") hacia *rpid*;
  FinSi
Fin

5.3.2 Subsistema para los Nodos de Presentación.

En la figura 5.5 se presentaron los componentes que conforman al subsistema del mecanismo de comunicación para los Nodos de Presentación. Si comparamos al subsistema de los Nodos de Adquisición con el subsistema de los Nodos de Presentación, se puede observar que ambos subsistemas contienen los mismos componentes, es decir, Inicio y configuración, Transmisión de información, Recepción de información y Tolerancia a fallas.

Los componentes del subsistema para los NP's están constituidos, en su mayoría, por los mismos módulos que los componentes del subsistema para los NA's; por lo que sólo se explicarán aquellos módulos que se implantaron específicamente para los Nodos de Presentación.

5.3.2.1 Inicio y configuración.

Los módulos que conforman al componente de Inicio y configuración se muestran en la figura 5.6. Estos módulos realizan las funciones de inicialización, configuración y establecimiento del modo operativo.

El módulo **liga_nom()** tiene el mismo funcionamiento que su análogo en los NA's. En el módulo **red()** se realiza la configuración de los NP's.

Recordando la configuración mostrada en el punto 5.2 y asumiendo que el parámetro *nid* representa al número de Nodo de Presentación actual, y que es obtenido por el módulo **liga_nom()**, y que la variable *nd* representará al nodo respaldado de *nid*, el pseudocódigo del módulo **red()** para los NP's, es el siguiente:

```
MÓDULO red(nid):
Inicio
  Opciones (nid)
    Caso 1: Asignar a nd = 3;
        FinCaso
    Caso 3: Asignar a nd = 1;
        FinCaso
  FinOpciones
Fin
```

De acuerdo con la MEF del protocolo de respaldo de los NP's, el NP actual debe avisar su inicio de operaciones a su nodo respaldado. Esto se realiza por medio del módulo **alta()**. Este módulo, entonces, nos indicará el modo operativo del NP (MO: *Primario-Primario* o *Primario-Respaldo*). El pseudocódigo de este módulo es:

```
MÓDULO alta():
Inicio
  Buscar proceso respaldado "/nodond";
  Si (proceso respaldado está activo) Entonces
    Notificar modo operativo Primario-Respaldo;
    Enviar mensaje de alta(msg0);
  Sino
    Notificar modo operativo Primario-Primario;
  FinSi
Fin
```

El módulo **consola()** permite recibir el mensaje de alta, por parte del NP respaldado, y el mensaje de inicio de transmisión, por parte de algún nodo del sistema. Adicionalmente, permite la recepción de las solicitudes de transferencia de información, ya sea por parte de un usuario o un programa de aplicación. Este módulo es el mismo que se describió en el subsistema de los Nodos de Adquisición.

5.3.2.2 Transmisión de información.

Este módulo realiza las transferencias de información solicitadas, es decir, realiza las transferencias de mensajes, comandos y archivos hacia cualquier nodo del sistema. Los elementos que constituyen al componente de Transmisión de información se muestran en la figura 5.21.

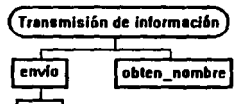

Figura 5.21. Módulos de Transmisión de información para los Nodos de Presentación.

Dado que el flujo de transmisión de estructuras es desde los Nodos de Adquisición hacia los Nodos de Presentación, los módulos **crear()** y **rec_sol()**, del componente de Transmisión de los NA's, se suprimen en el componente de los NP's.

5.3.2.3 Recepción de información.

Con este componente, los Nodos de Presentación realizan la recepción de todos los tipos de información, es decir, aquí se encuentran los módulos necesarios para la recepción de estructuras (BDTR), mensajes, comandos y archivos.

La función principal del mecanismo de comunicación, en los NP's, es la concerniente a la recepción de las estructuras de datos provenientes desde los Nodos de Adquisición. Dado que la Base de Datos del SCD, que contiene todos los parámetros para la definición de todas las variables que se adquieren en la CTE, deberá estar actualizándose continuamente, es necesario que la BD sea cargada a memoria por medio de estructuras que definan una Base de Datos en Tiempo Real (BDTR). Cada Nodo de Presentación tiene una réplica de la BD del SCD.

Por lo anterior, es necesario la creación de un proceso que se encargue de cargar la BD a memoria, para que ésta última pueda ser actualizada dinámicamente (en Tiempo Real). El acceso a disco incrementaría, en mucho, el tiempo de actualización requerido para la recepción y actualización de las estructuras, que definen a la BDTR (véase figura 5.22).

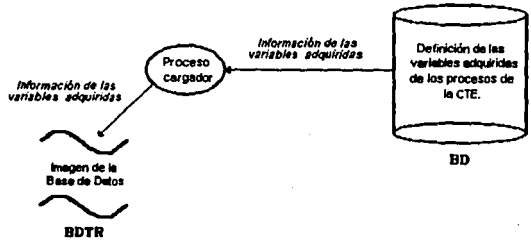

Figura 5.22. Cargado de la Base de datos a la memoria.

Una vez cargada la BD en memoria, el proceso cargador, de la figura 5.22, tendrá la responsabilidad de recibir las estructuras, provenientes de los Nodos de Adquisición, y actualizar a la BDTR con los datos recibidos, como lo muestra la figura 5.23.

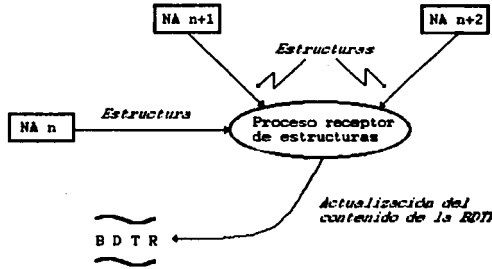

Figura 5.23. Recepción de estructuras.

Como puede verse en la figura anterior, el proceso receptor debe atender la recepción de estructuras, desde diferentes NA's. Para que esta recepción se realice de manera más eficiente, se pueden crear *n* procesos receptores de estructuras, donde *n* será el número de NA's en el sistema (en nuestro prototipo se consideran tres Nodos de Adquisición).

Cada proceso receptor estará dedicado, exclusivamente, a la recepción de estructuras por parte de un NA específico, dejándole al proceso principal, del mecanismo de comunicación, la función de recibir mensajes, comandos y archivos. El proceso receptor, de la figura 5.23, ahora sólo tendrá la responsabilidad de actualizar la BDTR (figura 5.24). Cada vez que se requiera, los datos en memoria deberán ser vaciados en la Base de Datos en disco.

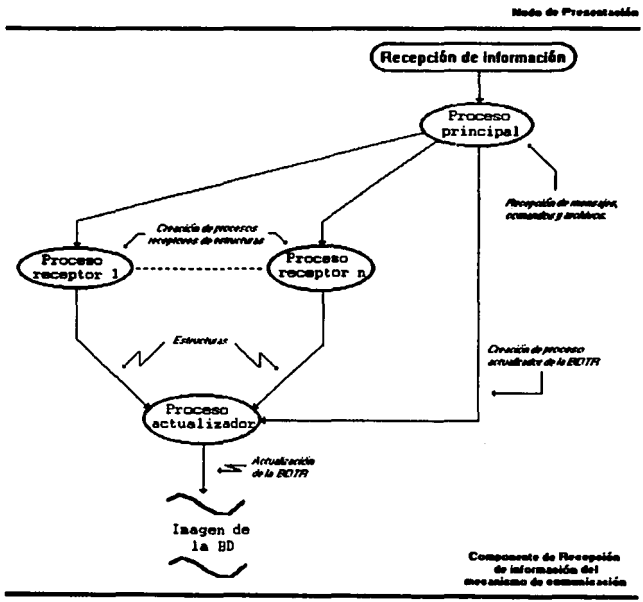

Figura 5.24. Diagrama de flujo de datos para la recepción de estructuras.

Los módulos que constituyen al componente de Recepción de información se muestran en la figura 5.25.

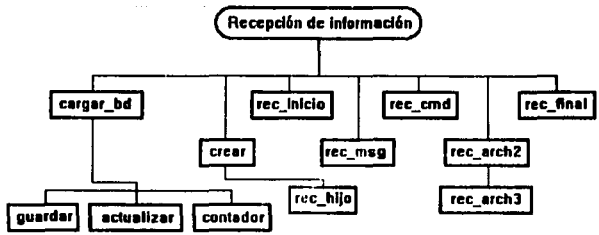

Figura 5.25. Módulos de Recepción de información para los NP's.

1) Módulo **cargar_bd()**.

Este módulo tiene la función de cargar la BDTR a la memoria. Debido a que en la estructura *msg2*, definida en el punto 3.52 del capítulo 3, se encuentran definidas las características principales de las varibles adquiridas por el SCD, y algunas de las características definidas, son apuntadores a los archivos donde se encuentran las descripciones de dichas características; teniendo en cuenta que estas características serán las que se estén actualizandose (valores, estado operativo, etc.) y no sus descripciones, además que, uno de los objetivos primordiales del prototipo del mecanismo de comunicación es la transferencia de estructuras, el módulo **cargar_bd()** sólo se encargará de cargar a memoria el archivo que define la estructura *msg2*, ya que se trata del archivo principal (de configuración) de la BD. Este archivo tiene el nombre de "config.db", y se encontrará en cada Nodo de Presentación.

Una vez que se ha cargado la estructura que define a la BDTR, el módulo estará en espera de todas las estructuras provenientes de los Nodos de Adquisición, para ser recibidas por los procesos de recepción, del componente de Recepción de información.

Cada vez que se reciba una estructura nueva, el módulo **cargar_bd()** llamará al submódulo **actualizar()**, que como su nombre lo indica, tiene la función de actualizar la estructura en memoria con los nuevos valores recibidos (los valores actuales son remplazados por los valores recibidos).

Una vez que finalice la operación del mecanismo de comunicación, o cada determinado tiempo, se debe guardar la estructura que se encuentre en memoria a disco. Para este caso, se utiliza la llamada a la función **guardar()**. Cuando se termine una sesión del mecanismo, éste generará una señal que será "atrapada" por el módulo **cargar_bd()**. Al recibir la señal, el módulo llamará a la función **guardar()**, que tiene por objetivo guardar la información en disco. Esta información será almacenada en el archivo "config.db", de cada Nodo de Presentación.

El vaciado de datos puede realizarse, también, cada determinado tiempo, pero esto dependerá de los requerimientos de disponibilidad de información en los Nodos de Presentación. Para demostrar este tipo de vaciado, el mecanismo vaciará los datos cada 10 minutos. Para realizar lo anterior, se llevará un contador (módulo **contador()**) que generará, cada 10 minutos, una señal requerida por el módulo **cargar_bd()** para llamar a la función **guardar()**. Enseguida se muestra el pseudocódigo de los módulos anteriores. En el módulo **cargar_bd()**, la variable *fpid* contendrá el identificador del proceso **cargar_db()** y la variable *buffer* contendrá, en memoria, el contenido del archivo "config.db" (donde *buffer* será una estructura idéntica a *msg2*). Los módulos **actualizar()** y **guardar()**, fueron resumidos por razones de brevedad en la descripción.

MÓDULO cargar_bd():
Inicio
  Generar manejador de señales **guardar()** para la señal *SIGUSR1*;
  Asignar a *fpid* el identificador del proceso;
  Abrir archivo "config.db" para lectura;
  Leer contenido del archivo;
  Asignar contenido a *buffer*;
  Cerrar archivo "config.db";
  Generar proceso **contador(fpid)**;
  Para ( )
    Recibir mensaje (msg2);
    Llamar a **actualizar()**;
    Replicar mensaje recibido;
  FinPara
Fin

MÓDULO contador(fpid):
Inicio
  Para ( )
    Contar 10 minutos;
    Enviar señal *SIGUSR1* hacia *rpid*;
  FinPara
Fin

MÓDULO actualizar():
Inicio
  Actualizar valores en *buffer* con los valores de msg2;
Fin

MÓDULO guardar():
Inicio
  Abrir archivo "config.db" para escritura;
  Vaciar valores de *buffer* en el archivo "config.db";
  Cerrar archivo "config.db";
Fin

2) Módulo crear(n,i,*ziid, fpid).

El módulo **crear()** tiene la responsabilidad de crear *n* procesos receptores de estructuras, donde *n* indica el número de Nodos de Adquisición en el SCD (en nuestro caso *n* = 3). El parámetro *i* sirve como identificador numérico del proceso y servirá para indicar a que Nodo de Adquisición estará dedicado. El parámetro *ziid* es un puntero a un arreglo de direcciones, donde se almacenarán los identificadores de cada proceso receptor creado.

Cada proceso receptor llamará a la función **rec_hijo()**, que se dedicará al registro de los procesos y a la recepción de estructuras. El parámetro *fpid* es el identificador del proceso **cargar_bd()**. El pseudocódigo de este módulo es el siguiente:

```
MÓDULO crear(n,i,*ziid,fpid):
Inicio
  Si ( n es mayor que 0 ) Entonces
    Generar proceso rec_hijo(i,fpid);
    Asignar identificador de proceso a *ziid;
    Incrementar i;
    Incrementar ziid;
    Decrementar n;
    Llamar a crear(n,i,ziid);
  FinSi
Fin
```

Módulo **rec_hijo(i,fpid).**

Este módulo registra a cada proceso receptor, en el Nodo de Presentación donde se encuentren ejecutándose (*nid*), para que cada Nodo de Adquisición pueda localizarlo y establecer el enlace con ellos. Después de este registro, se esperará por la estructura, proveniente del NA asignado al proceso receptor. Una vez que se recibe la estructura, se envía hacia el proceso **cargar_bd()**, que se encargará de actualizar los datos recibidos. Su pseudocódigo es:

```
MÓDULO rec_hijo(i,fpid):
Inicio
  Asignar a proceso = "/nodonid";
  Opciones (i)
    Caso 0: Concatenar(proceso, "a");
            FinCaso;
    Caso 1: Concatenar(proceso, "b");
            FinCaso;
    Caso 2: Concatenar(proceso, "c");
            FinCaso;
  FinOpciones
```

```
  Registrar a proceso;
  Para ()
    Recibir mensaje (msg2);
    Enviar mensaje (msg2) hacia proceso fpid;
  FinPara
Fin
```

Los demás módulos de recepción (rec_inicio(), rec_msg(), etc.) se comportan de igual forma que los módulos de recepción del subsistema de los Nodos de Adquisición.

5.3.2.4 Tolerancia a fallas.

La tolerancia a fallas en los Nodos de Presentación, está dada por el módulo **conexión()**, que fue explicado en la sección de Tolerancia a fallas para los Nodos de Adquisición. Aquí se omite su descripción, ya que se trata del mismo módulo.

5.4 Interface entre el mecanismo de comunicación y programas de aplicación.

Se ha mencionado que los servicios de transferencia de información, ofrecidos por el mecanismo de comunicación (*mc*), pueden ser solicitados por usuarios del mecanismo (a través del desplegado de un menú de servicios) o por programas de aplicación, que se encuentren ejecutándose en el mismo nodo, en el que se encuentra ejecutándose el *mc*.

Para tener acceso a estos servicios de transferencia, los programas de aplicación deben contar con un mecanismo, que les permita comunicarse con el *mc*. La Interface Aplicación-Aplicación (IAA), es el medio por el cual los programas de aplicación tienen acceso a los servicios de transferencia de información, disponibles en el *mc*. La IAA es un componente independiente al *mc*, y debe ser incluido en cada programa que vaya a requerir la transmisión de información hacia algun nodo remoto. La IAA está conformada por 6 módulos, que se muestran en la figura 5.26.

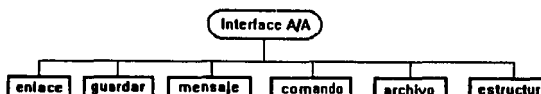

Figura 5.26. Módulos de la interface A/A.

Estos módulos serán vistos, por los programas de aplicación, como si fueran funciones estándar de librería. Lo único que deben conocer los programas de aplicación, son los parámetros de entrada de dichos módulos.

1) Módulo enlace(ban).

El módulo **enlace()**, tiene la función de establecer un enlace entre el programa de aplicación y el *mc*. Este módulo es llamado, al igual que el módulo **guardar()**, por los otros cuatro módulos de solicitud de transferencia (**mensaje()**, **comando()**, etc.).

Primeramente, el módulo obtiene el número de nodo en el cual se está corriendo el programa de aplicación. Después, localiza al *mc* en este nodo. Hay que recordar que el *mc* se registrará con el nombre de "/nodo*nodo*", donde *nodo* indica el nodo de trabajo del *mc*. El resultado del enlace se retorna en la variable *ppid*. El parámetro *ban* indica si la solicitud es de transferencia de estructura (*ban* = 1). En este caso, el enlace se realiza con el proceso receptor de solicitudes de transmisión de estructuras del *mc*, es decir, con el proceso "/nodo*nodo*b". El pseudocódigo de este módulo es el siguiente:

```
MÓDULO enlace(ban):
Inicio
  Obtener número de nodo actual, nodo;
  Si (ban = 1) Entonces
    Localizar al proceso "nodonodob";
  Sino
    Localizar al proceso "nodonodo";
  FinSi
  Si (se localizo al proceso) Entonces
    Asignar a ppid = identificador del proceso localizado;
    Retornar ppid;
  Sino
    Retornar ERROR;
  FinSi
Fin
```

2) Módulo guardar(nodo_rx).

Este módulo realiza la inicialización del mensaje de inicio de transmisión, asignandole los valores correspondientes al nodo transmisor, nodo receptor, tipo de solicitud, etc. El parámetro *nodo_rx*, indica el nodo receptor de la información. Su pseudocódigo es:

```
MÓDULO guardar(nodo_rx):
Inicio
  Asignar a msg0.nm = "/nodonodo";
```

```
  Asignar a msg0.tx = nodo;
  Asignar a msg0.rx = nodo_rx;
  Asignar a msg0.ban = 1;
Fin
```

3) Módulo mensaje(**nodo, msg, resp**).

El módulo **mensaje()**, es el que permite solicitar la transferencia de mensajes. El programa de aplicación debe indicar el nodo destino (el parámetro *nodo_rx*), y el mensaje que requiere transferir (*msg*). El parámetro *resp* es un puntero a un buffer, donde se depositará la respuesta que retornará el nodo destino. El pseudocódigo de este módulo es el siguiente:

```
MÓDULO mensaje(nodo_rx, msg, resp):
Inicio
  Asignar a ppid = enlace(0);
  Si (ppid = ERROR) Entonces
    Retornar ERROR;
  Sino
    Llamar a guardar(nodo_rx);
    Asignar a msg0.inicio = "mg";
    Asignar a msg0.inicio = 1;
    Enviar mensaje(msg0) hacia proceso ppid;
    Si (falla en el envío) Entonces
      Retornar ERROR;
    Sino
      Asignar a msg5.mss = msg;
      Enviar mensaje (msg5) hacia proceso ppid;
      Si (falla en el envío) Entonces
        Retornar ERROR;
      Sino
        Asignar a resp = respuesta retornada (msg7);
        Retornar EXITO;
      FinSi
    FinSi
  FinSi
Fin
```

4) Módulo **comando(nodo_rx, cmd, resp, flag)**.

Aquí se realiza la solicitud de transmisión de comandos. El parámetro *nodo_rx* indica el nodo receptor del comando, *cmd* el comando, *resp* es el buffer de respuesta y *flag* indica el tipo de respuesta que requiere el programa de aplicación (*flag* = 0, sólo reconocimiento de comando ejecutado o *flag* = 1, información generada por el comando).

```
MÓDULO comando(nodo_rx, cmd, resp,flag):
Inicio
  Asignar a ppid = enlace(0);
  Si (ppid = ERROR) Entonces
    Retornar ERROR;
  Sino
    Llamar a guardar(nodo_rx);
    Asignar a msg0.inicio = "cd";
    Asignar a msg0.inicio = 2;
    Enviar mensaje(msg0) hacia proceso ppid;
    Si (falla en el envío) Entonces
      Retornar ERROR;
    Sino
      Asignar a msg6.cmd = cmd;
      Asignar a msg6.flag = flag;
      Enviar mensaje (msg6) hacia proceso ppid;
      Si (falla en el envío) Entonces
        Retornar ERROR;
      Sino
        Asignar a resp = respuesta retornada (msg7);
        Retornar EXITO;
      FinSi
    FinSi
  FinSi
Fin
```

5) Módulo **archivo(nodo_rx, nombre, resp)**.

En este módulo se solicita la transferencia de archivos. El parámetro nombre indica el archivo que se requiere transmitir. El funcionamiento de este módulo es similar a los anteriores, como se muestra a continuación:

```
MÓDULO archivo(nodo_rx, nombre, resp):
Inicio
  Asignar a ppid = enlace(0);
  Si (ppid = ERROR) Entonces
    Retornar ERROR;
  Sino
    Llamar a guardar(nodo_rx);
    Asignar a msg0.inicio = "db";
    Asignar a msg0.inicio = 4;
    Enviar mensaje(msg0) hacia proceso ppid;
    Si (falla en el envío) Entonces
      Retornar ERROR;
```

```
    Sino
      Asignar a msg9.nombre = nombre;
      Enviar mensaje (msg9) hacia proceso ppid;
      Si (falla en el envio) Entonces
        Retornar ERROR;
      Sino
        Retornar EXITO;
      FinSi
    FinSi
  FinSi
Fin
```

6) Módulo estructura(est, resp).

Con este módulo los procesos de adquisición, de los Nodos de Adquisición, pueden solicitar la difusión de la información que están adquiriendo, hacia todos los Nodos de Presentación. Esta información debe estar actualizandose y almacenandose, continuamente, en estructuras en memoria. El módulo **estructura()** debe tener como parámetros de entrada, la estructura que se requiere difundir y el puntero al buffer de respuesta. Su pseudocódigo es el siguiente:

```
MÓDULO estructura(est, resp):
Inicio
  Asignar a ppid = enlace(1);
  Si (ppid = ERROR) Entonces
    Retornar ERROR;
  Sino
    Asignar a msg2 = est;
    Enviar mensaje (msg2) hacia proceso ppid;
    Si (falla en el envio) Entonces
      Retornar ERROR;
    Sino
      Asignar a resp = respuesta retornada (msg7);
      Retornar EXITO;
    FinSi
  FinSi
Fin
```

CAPÍTULO 6: EVALUACIÓN DEL MECANISMO DE COMUNICACIÓN CON TOLERANCIA A FALLAS.

6.1 Introducción.

Una vez terminada la fase de diseño e implantación de una aplicación de software, se procede a la evaluación del funcionamiento de dicho sistema; es decir, se realiza la verificación del cumplimiento de los requerimientos del sistema, inicialmente establecidos.

La prueba de programas, que constituyen a una aplicación de software, consiste en ejercitar a los programas, utilizando datos y condiciones similares a los datos y condiciones reales que habrán de ser ejecutados, observar los resultados y descubrir la existencia de errores o insuficiencias de los programas, a partir de las posibles anomalías surgidas en los resultados obtenidos de la prueba.

De acuerdo con lo anterior, se dice que la prueba es el proceso de detectar la existencia de errores en un programa. La depuración es el proceso de localizar dónde se produjeron esos errores y corregir el código incorrecto [10]. La prueba de programas se diseña para hacer que el comportamiento de un programa sea lo más próximo al ambiente real en el cual se desenvolverá.

El programador responsable del sistema, no es la persona más apropiada para probar un programa, ya que, de manera conciente o inconciente, aplicará pruebas que no fallarán, por lo que no serán adecuadas para demostrar la presencia de errores en el sistema [10]. Sin embargo, el conocimiento detallado de la estructura de un programa o aplicación de software, hacen al programador la persona apropiada para identificar los casos de prueba del sistema.

La clave de una prueba de programa acertada, es establecer un ambiente de trabajo donde el diseñador y programador del sistema y las personas implicadas en aplicar las pruebas al sistema, realicen una función complementaria [10]. Esto debe incluir la premisa de que los errores de los programas son inevitables dada la complejidad de los sistemas implicados y que los errores no deben ser condenables.

6.2 Etapas de evaluación.

Excepto para programas pequeños, no es recomendable intentar la prueba de las aplicaciones de software como si se tratara de una sola unidad. Como se vio en los anteriores capítulos, las aplicaciones de software se componen de subsistemas formados por componentes que, a su vez, pueden estar constituidos por módulos. Si se realiza la prueba del sistema como una sola unidad, es probable que no se identifique más que un pequeño porcentaje de errores, si es que éstos existen.

La evaluación de aplicaciones de software debe avanzar en etapas, siendo cada una de ellas la continuación lógica de la etapa anterior. En la evaluación de un sistema, se pueden identificar cuatro etapas, que son:

1) Prueba de módulos. La prueba de módulos es el nivel básico, en donde se prueban los módulos que constituyen a un componente, para garantizar que operan de manera correcta. Los módulos no deben depender de otros módulos de su mismo nivel, para posibilitar la prueba de cada módulo como una entidad aislada.

2) Prueba de componentes. Un componente se forma por medio de varios módulos, que pueden cooperar entre sí. Después de haber probado cada módulo individualmente, se prueba la integración de estos módulos como una entidad.

3) Prueba de subsistemas. Aquí, los componentes se agrupan para formar subsistemas. Ya que los componentes cooperan y se comunican, la prueba de subsistemas se debe enfocar en la verificación del cumplimiento de los objetivos de cada subsistema.

4) Prueba del sistema. La prueba del sistema, también llamada prueba de integración, se lleva a cabo cuando se integran los subsistemas para conformar el sistema completo. En esta etapa, la evaluación se relaciona con la confirmación del cumplimiento de los requerimientos definidos para todo el sistema. También se relaciona con el hallazgo de errores en el diseño y la codificación.

Generalmente, las pruebas de módulos y componentes son realizadas por el programador, sin una especificación formal de prueba. El programador genera sus propios datos de prueba y comprueba gradualmente su código durante el desarrollo.

En las dos últimas etapas, las pruebas las realiza un equipo (o una persona) independiente. El procedimiento de prueba se debe especificar y establecer de manera formal. A continuación se realiza la especificación formal de pruebas para el mecanismo de comunicación desarrollado en este trabajo de tesis.

6.3 Pruebas al mecanismo de comunicación.

En los siguientes puntos se especifican y realizan, de manera formal, las pruebas que deberán ser aplicadas al prototipo de mecanismo de comunicación.

6.3.1 Objetivos.

El objetivo de probar los servicios proporcionados por el mecanismo de comunicación, consiste en verificar que:

a.- Se realice el establecimiento correcto del modo operativo (*Primario-Primario* o *Primario-Respaldo*) de un nodo, de acuerdo con el tipo de nodo (de Adquisición o de Presentación) y al número de nodos dados de alta.

b.- Se realice la transferencia de mensajes entre nodos. El mensaje transmitido y la respuesta retornada deberán desplegarse en pantalla, en el nodo receptor y nodo transmisor, respectivamente.

c.- Se realice la transferencia de comandos entre nodos. El comando transmitido deberá ejecutarse en el nodo receptor y la información generada deberá desplegarse en pantalla, ya sea en el nodo receptor o en el nodo transmisor.

d.- Se realice la transferencia de archivos texto entre nodos.

e.- Se realice la transferencia, recepción y actualización de estructuras en memoria desde un nodo transmisor (Nodo de Adquisición) hacia más de un nodo receptor (Nodos de Presentación).

f.- Se mantenga la disponibilidad en la recepción de información, cuando se realice una transferencia de mensaje, comando o archivo, en caso de una falla en el nodo receptor.

g.- Se mantenga la disponibilidad en la transferencia de estructuras, aún con la caída del nodo transmisor de la estructura.

6.3.2 Requisitos.

Para la ejecución del mecanismo de comunicación se requieren los siguientes elementos:

1) Por lo menos dos PC's 386 o superiores, enlazadas por medio de la red ARCNET.

2) La plataforma software del Sistema Operativo en Tiempo Real QNX, versión 4.0.

3) La existencia del siguiente directorio: */usr/gonix/proyecto*.

4) La existencia del código ejecutable del subsistema del mecanismo de comunicación para los Nodos de Presentación. Este código tiene el nombre de "*pres*".

5) La existencia del código ejecutable del subsistema del mecanismo de comunicación para los Nodos de Adquisición. Este código tiene el nombre de "*adq*".

6) La existencia del código fuente de la Interface Aplicación-Aplicación. Este código tiene el nombre de "interface.c".

7) La existencia del archivo para la configuración de la BDTR. Este archivo deberá estar replicado con los nombres de "*config.db*", "*config1.db*" y "*config3.db*".

8) El código ejecutable del programa que contiene las funciones de adquisición del SCD y responsable de la solicitud de transferencia de archivos. Este código tiene el nombre de "*adquiere*".

9) Deberá existir el archivo "*config.daq*", utilizado por el programa de adquisición para configurar los archivos de adquisición.

10) El código ejecutable del programa que simula la solicitud de transferencias de estructuras en memoria. Este código tiene el nombre de "*cliente*".

6.3.3 Supuestos.

Para la realización de las pruebas, se asume que la persona encargada de realizar las pruebas conoce o se encuentra familiarizado con el Sistema Operativo QNX, el editor Vedit de QNX, así como la localización de los directorios de trabajo y funcionamiento global del programa con funciones de adquisición "*adquiere*", el programa simulador de generación de estructuras "*cliente*", y el mecanismo de comunicación (programas "*interface.c*", "*adq*" y "*pres*"). Además, la persona encargada de realizar las pruebas deberá tener conocimientos acerca de tolerancia a fallas, funciones de adquisición de datos, mecanismos de comunicación (protocolos) y sistemas distribuidos.

6.3.4 Metodología de las pruebas.

Las pruebas al mecanismo de comunicación, se derivan de la Matriz Relacional de Requisito-Prueba mostrada en la figura 6.1.

R/P	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
R1	•	•								
R2	•									
R3		•								
R4			•	•	•	•	•	•		
R5			•							
R6				•	•					
R7						•	•			
R8								•		
R9									•	•
R10									•	
R11										•

Figura 6.1. Matriz Relacional Requisito-Prueba para el mecanismo de comunicación.

Los requisitos identificados de la Matriz Relacional anterior, son los siguientes:

R1: Establecimiento de Modo Operativo.
R2: Verificación de Modo Operativo en Nodos de Adquisición.
R3: Verificación de Modo Operativo en Nodos de Presentación.
R4: Transferencia de información.
R5: Servicio de transferencia de mensajes.
R6: Servicio de transferencia de comandos.
R7: Servicio de transferencia de archivos.
R8: Servicio de transferencia de estructuras en memoria.
R9: Tolerancia a fallas.
R10: Mantenimiento de la disponibilidad en la recepción de información (mensajes, comandos, archivos), con la caida del nodo receptor.
R11: Mantenimiento de la disponibilidad en la transmisión de información (estructuras en memoria), con la caida del nodo transmisor.

6.3.5 Guión de pruebas.

Prueba: P1.
Título: Establecimiento de Modo Operativo en Nodos de Adquisición.
Objetivo: Probar que se cumple el establecimiento correcto del Modo Operativo para los Nodos de Adquisición, formando un anillo lógico.
Pasos:

En el Nodo de Adquisición 2:

1) Introducirse en el directorio */usr/gonix/proyecto*.

2) Ejecutar el programa "*adq*".

3) Verificar que el programa establece el Modo Operativo *Primario-Primario* para el nodo 2, mediante el desplegado de: "*NA 2 trabajando con NA Respaldo 6: Inactivo y NA Respaldado 4: Inactivo*".

En el Nodo de Adquisición 4:

4) Realizar los pasos 1 y 2.

5) Verificar que el programa establece el Modo Operativo *Primario-Primario* para el nodo 4, mediante el desplegado de: "*NA 4 trabajando con NA Respaldo 2: Activo y NA Respaldado 6: Inactivo*".

En el Nodo de Adquisición 2:

6) Verificar el cambio de Modo Operativo a *Primario-Respaldo* en el nodo 2, mediante el desplegado de: "*NA 2 trabajando con NA Respaldo 6: Inactivo y NA Respaldado 4: Activo*".

En el Nodo de Adquisición 6:

7) Realizar los pasos 1 y 2.

8) Verificar que el programa establece el Modo Operativo *Primario-Respaldo* para el nodo 6, mediante el desplegado de: "*NA 6 trabajando con NA Respaldo 4: Activo y NA Respaldado 2: Activo*".

En el Nodo de Adquisición 4:

9) Verificar el cambio de Modo Operativo a *Primario-Respaldo* en el nodo 4, mediante el desplegado de: "*NA 4 trabajando con NA Respaldo 2: Activo y NA Respaldado 6: Activo*".

Prueba: P2.
Título: Establecimiento de Modo Operativo en Nodos de Presentación.
Objetivo: Probar que se cumple el establecimiento correcto del Modo Operativo para los Nodos de Presentación, respaldándose por parejas.
Pasos:

En el Nodo de Presentación 1:

1) Introducirse en el directorio */usr/gonix/proyecto*.

2) Ejecutar el programa "*pres*".

3) Verificar que el programa despliega el Modo Operativo *Primario-Primario* para el nodo 1.

En el Nodo de Presentación 3:

4) Realizar los pasos 1 y 2.

5) Verificar que el programa despliega el Modo Operativo *Primario-Respaldo* para el nodo 3.

6) Verificar el cambio de Modo Operativo a *Primario-Respaldo* en el nodo 1.

Prueba: P3.
Título: Transferencia de mensajes entre nodos.
Objetivo: Probar que se cumple la transferencia de mensajes entre nodos.
Pasos:

En el Nodo de Adquisición 2:

1) Introducirse en el directorio */usr/gonix/proyecto*.

2) Ejecutar el programa "*adq*".

En el Nodo de Presentación 1:

3) Realizar el paso 1.

4) Ejecutar el programa "*pres*".

En el Nodo de Adquisición 2:

5) En el menú de servicios del programa "*adq*", **seleccionar** "*Enviar Mensaje*".

6) Elegir nodo receptor de mensaje "1".

7) Teclear, como mensaje a transmitir, "*Esto es una prueba*".

En el Nodo de Presentación 1:

8) Verificar la recepción del mensaje "*Esto es una prueba*".

9) Teclear, como mensaje de respuesta, "*Mensaje recibido*".

En el Nodo de Adquisición 2:

10) Verificar la recepción del mensaje de respuesta "*Mensaje recibido*".

Prueba: P4.
Título: Transferencia de comandos entre nodos, con desplegado de la ejecución del comando en el nodo receptor (Comando *ls*).
Objetivo: Probar que se cumple la transferencia de comando entre nodos, la ejecución del comando en el nodo receptor y el desplegado de la información generada en el nodo receptor.
Pasos:

1) Realizar los pasos 1, 2, 3 y 4 de la prueba P3.

En el Nodo de Adquisición 2:

2) En el menú de servicios del programa "*adq*", seleccionar "*Enviar Comando*".

3) Elegir nodo receptor de comando "1".

4) Teclear, como comando a transmitir, "*ls*" (Comando para listar archivos del directorio actual).

5) Elegir la opción de "*Bandera de comando ejecutado*".

En el Nodo de Presentación 1:

6) Verificar la recepción del comando "*ls*".

7) Verificar la ejecución del comando, con el despliegue de los archivos del directorio actual.

En el Nodo de Adquisición 2:

8) Verificar el desplegado del mensaje "*Comando ejecutado!*".

Prueba: P5.
Título: Transferencia de comandos entre nodos, con desplegado de la ejecución del comando en el nodo transmisor (Comando *sin*).
Objetivo: Probar que se cumple la transferencia de comando entre nodos, la ejecución del comando en el nodo receptor y el desplegado de la información generada en el nodo transmisor.
Pasos:

1) Realizar los pasos 1, 2, 3 y 4 de la prueba P3.

En el Nodo de Adquisición 2:

2) En el menú de servicios del programa "*adq*", seleccionar "*Enviar Comando*".

3) Elegir nodo receptor de comando "1".

4) Teclear, como comando a transmitir, "*sin*" (Comando para listar a los procesos que se encuentran ejecutándose en el nodo atual).

5) Elegir la opción de "*Información generada*".

En el Nodo de Presentación 1:

6) Verificar la recepción del comando "*sin*".

7) Verificar el desplegado del mensaje "*Comando ejecutado !*".

En el Nodo de Adquisición 2:

8) Verificar la ejecución del comando, con el desplegado de los procesos que se encuentran ejecutándose en el Nodo de Presentación 1.

Prueba: P6.
Título: Transferencia de archivos entre nodos.
Objetivo: Probar que se cumple la transferencia de archivos de texto entre nodos, siendo el solicitante de este servicio un usuario.
Pasos:

1) Relizar los pasos 1, 2, 3 y 4 de la prueba P3.

En el Nodo de Adquisición 2:

2) En el menu de servicios del programa "*adq*" seleccionar "*Enviar Archivo*".

3) Elegir nodo receptor de archivo "1".

4) Teclear, como archivo a transmitir, "*config.db*".

En el Nodo de Presentación 1:

5) Verificar el desplegado del mensaje "*Archivo recibido*".

6) En el menú de servicios del programa "*pres*", seleccionar "*Salir de la aplicación*".

7) Verificar con el editor de texto Vedit, la existencia del archivo "*config.db*".

Prueba: P7.
Título: Difusión de archivos hacia Nodos de Presentación.
Objetivo: Probar que se cumple la transferencia de archivos de texto entre nodos, siendo el solicitante de este servicio un programa de aplicación y los receptores todos los Nodos de Presentación.
Pasos:

En el Nodo de Adquisición 2:

1) Introducirse en el directorio */usr/gomix/proyecto.*

2) Ejecutar el programa "*adq*".

3) Ejecutar el programa "*adquiere*", donde se encuentran funciones de adquisición de datos, almacenamiento de dichos datos en archivos ("*datosanal.txt*" y "*datosdig.txt*") y solicitud del servicio de transferencia de archivos. En este programa se ha incluido el programa "*interface.c*".

En el Nodo de Presentación 1:

4) Realizar el paso 1.

5) Ejecutar el programa "*pres*".

En el Nodo de Presentación 3:

6) Realizar los pasos 4 y 5.

En el Nodo de Adquisición 2:

7) En el menú de servicios del programa "*adq*", seleccionar "*Enviar mensaje*".

8) Elegir como proceso receptor "5" (identificador del proceso de adquisición "*adquiere*").

9) Teclear, como mensaje a transmitir, "*activar*".

10) Verificar, en el programa "*adquiere*", el inicio de la adquisición de datos.

11) Esperar a que el programa "*adquiere*" solicite el servicio de transferencia de archivos (aproximadamente cada 10 segundos) mediante la IAA.

12) Verificar, en el programa "*adq*", el inicio de transmisión de los archivos "*datosanal.txt*" y "*datosdig.txt*" hacia los Nodos de Presentación.

En los Nodos de Presentación 1 y 3:

13) Verificar el desplegado del mensaje "*Archivo recibido*".

14) En el menú de servicios del programa "*pres*", seleccionar "*Salir de la aplicación*".

15) Verificar con el editor Vedit, la existencia de los archivos "*datosanal.txt*" y "*datosdig.txt*".

En el Nodo de Adquisición 2:

16) Realizar los pasos 7 y 8.

17) Teclear, como mensaje a transmitir, "*desactivar*".

18) Verificar, en el programa "*adquiere*", la terminación de la adquisición de datos.

Prueba: P8.
Título: Transferencia de estructuras en memoria.
Objetivo: a) Probar que se cumple la transferencia de estructuras en memoria de un Nodo de Adquisición hacia todos los Nodos de Presentación.
b) Probar la recepción y actualización de estructuras en memoria en los Nodos de Presentación.

Pasos:

En el Nodo de Adquisición 2:

1) Introducirse en el directorio */usr/gonix/proyecto*.

2) Ejecutar el programa "*adq*".

En los Nodos de Presentación 1 y 3:

3) Realizar el paso 1.

4) Ejecutar el programa "*pres*".

En el Nodo de Adquisición 2:

5) Ejecutar el programa "*cliente*", generador de la estructura que define a la BDTR (por medio del archivo "*config.db*"), simulador de cambios en el contenido de la estructura y solicitante del envío de la estructura. En este programa también se incluyó el programa "*interface.c*"

6) En el programa "*cliente*", introducir el número de estructuras que se desea generar. Por ejemplo, 10 estructuras.

En los Nodos de Presentación 1 y 3:

7) En el menú de servicios del programa "*pres*", seleccionar "*Salir de la aplicación*".

8) Verificar con el editor de texto Vedit, que el contenido de las estructuras que definen a los archivos "*config1.db*" y "*config3.db*", para el nodo 1 y nodo 3 respectivamente, han sido actualizadas.

Prueba: P9.
Título: Tolerancia a fallas en la recepción de información.
Objetivo: Probar que se mantiene la disponibilidad en la recepción de un mensaje, un comando o un archivo, aún con la caída del nodo receptor, mediante el cambio de flujo de información hacia el nodo respaldo del nodo receptor caído.
Pasos:

En los Nodos de Presentación 1 y 3:

1) Introducirse al directorio */usr/gonix/proyecto*.

2) Ejecutar el programa "*pres*".

En el Nodo de Adquisición 2:

3) Realizar el paso 1.

4) Ejecutar el programa "*adq*".

5) En el menú de servicios del programa "*adq*", seleccionar "*Enviar Mensaje*".

6) Elegir nodo receptor de mensaje "**1**".

7) Teclear, como mensaje a transmitir, "*Prueba de falla*".

En el Nodo de Presentación 1:

8) Simular caída del nodo (cortar la ejecución del programa "*pres*", apagar o desconectar el Nodo de Presentación 1).

En el Nodo de Adquisición 2:

9) Verificar la caída del nodo receptor, mediante el despliegue del mensaje "*ERROR 1: Caída del nodo receptor # 1*".

10) Verificar la conexión con el nodo respaldo del nodo receptor caído, mediante el despliegue del mensaje "*Inicio de conexión con nodo receptor respaldo # 3*".

En el Nodo de Presentación 3:

11) Verificar la recepción del mensaje "*Prueba de falla*".

12) Teclear, como mensaje de respuesta, "*Mensaje recibido en nodo respaldo*".

En el Nodo de Adquisición 2:

13) Verificar la recepción del mensaje de respuesta "*Mensaje recibido en nodo respaldo*".

Esta prueba también es válida cuando se transmiten comandos o archivos, y cuando el nodo receptor sea algún Nodo de Adquisición.

Prueba: P10.
Título: Tolerancia a fallas en la transmisión de información.
Objetivo: Probar que se mantiene la disponibilidad en la transferencia de estructuras, aún con la caída del Nodo de Adquisición transmisor, mediante la réplica de las funciones de transmisión en un Nodo de Adquisición respaldo.

Pasos:

En los Nodos de Presentación 1 y 3:

1) Introducirse en el directorio */usr/gonix/proyecto*.

2) Ejecutar el programa "*pres*".

En los Nodos de Adquisición 2 y 6:

3) Realizar el paso 1.

4) Ejecutar el programa "*adq*".

En el Nodo de Adquisición 2:

5) Ejecutar el programa "*cliente*" y generar 1 estructura.

6) Simular caída del nodo (cortar la ejecución del programa "*adq*", etc.).

En los Nodos de Presentación 1 y 3:

7) En el menú de servicios del programa "*pres*", seleccionar "*Salir de la aplicación*".

8) Verificar con el editor de texto Vedit, que el contenido de las estructuras que definen a los archivos "*config1.db*" y "*config3.db*", para el nodo 1 y nodo 3 respectivamente, han sido actualizadas.

6.3.6 Observaciones.

Una vez realizadas las pruebas anteriores, se pudo comprobar que el mecanismo de comunicación desarrollado cumplió correctamente con todos los requerimientos de comunicación y tolerancia a fallas, establecidos previamente.

Todas las pruebas especificadas en este capítulo, fueron documentadas y aprobadas por la Unidad de Resultados de Automatización de Procesos (URAP), del Instituto de Investigaciones Eléctricas (IIE), en el documento: "Pruebas funcionales al Prototipo de Mecanismo de Comunicación para un SCD de una CTE".

CAPÍTULO 7: CONCLUSIONES.

Como resultado del incremento en el nivel de automatización de las Centrales Termoeléctricas (CTE's), relacionado con los requerimientos de eficiencia, confiabilidad y disponibilidad, así como la presencia de nuevas tecnologías de comunicación, como Redes locales (LAN's), aunado con la disminución de los costos del hadware, ha permitido llevar a cabo el desarrollo e implantación de nuevas arquitecturas orientadas al mejoramiento de los sistemas que controlan a estas CTE's.

Las nuevas arquitecturas deben ofrecer características que permitan un mejoramiento en la operación y mantenimiento en las CTE's, haciéndolas más eficientes y confiables. Entre las nuevas altemativas de arquitectura para el control de las CTE's, se encuentran los Sistemas de Control Distribuido (SCD's). En un SCD las funciones de procesamiento y control se encuentran distribuidas entre varios elementos de cómputo, donde cada elemento cuenta con recursos propios o compartidos, pero que en conjunto realizan un objetivo común.

La comunicación entre procesos es una de las características más importantes dentro de un SCD, ya que mediante ésta, se logra transferir información y permite llevar a cabo las funciones de control en todas las actividades que se realizan en el sistema. Por esta razón, los SCD's requieren de mecanismos de comunicación que les permitan mantener un intercambio de información eficiente y confiable. Para el desarrollo de estos mecanismos de comunicación, se requiere contar con un medio ambiente que proporcione los ingredientes esenciales para trabajar con múltiples procesos en tiempo real.

De los sistemas operativos existentes en el mercado, el sistema operativo QNX provee los elementos necesarios para el desarrollo de aplicaciones en tiempo real. El S.O. QNX es multiusuario y multitareas, además de que provee una gran variedad de servicios para implantar la comunicación entre procesos (locales o remotos), dichos servicios permiten que los procesos compartan recursos y dispositivos. También proporciona la creación de procesos, dentro de un programa de aplicación, que pueden ejecutarse concurrentemente, donde cada uno de ellos realiza una tarea distinta. Las características descritas ofrecen la capacidad de que un conjunto de procesos distribuidos físicamente, sean tratados como una sola unidad lógica.

Durante el desarrollo del trabajo presentado en esta tesis, se conoció y aplicó una metodología formal para el desarrollo de un mecanismo de comunicación, y se llevaron a cabo, paso a paso, las fases de definición de requerimientos, especificación, diseño, implantación y pruebas, logrando que el desarrollo del mecanismo se realizara de una manera sencilla y fluida.

El mecanismo de comunicación desarrollado, representa sólo una etapa dentro de un proyecto global, concerniente a los SCD's para CTE's. Dentro de este proyecto, se contempla el desarrollo de diversas etapas, entre las que se encuentran: el desarrollo de las funciones de adquisición, la integración del mecanismo de comunicación a las funciones de adquisición (etapa que se encuentra incluida en este trabajo), el diseño de la Base de Datos del sistema y la implantación de la Interfaz Hombre-Máquina, entre otras. La última etapa consistirá en la integración total de todos estos elementos, para conformar a un SCD.

El mecanismo de comunicación se desarrolló a nivel de prototipo, lo cual proporciona las siguientes ventajas:

1) Se pueden detectar servicios faltantes del usuario, que no se habían tomado en cuenta.

2) Se pueden identificar y redefinir los servicios del usuario, que así lo requieran.

3) El diseñador del sistema puede encontrar requisitos incompletos, inconsistentes o confusos durante el desarrollo del prototipo.

4) Se dispone con cierta rapidez de un sistema, para demostrar la viabilidad y utilidad de la aplicación.

5) Existe una disminución de los costos de desarrollo, ya que se detectan los errores en las fases iniciales del ciclo de vida de un sistema.

Una vez terminado el sistema prototipo, éste representa la base del sistema final que será implantado o que será entregado al usuario.

Entre las características más notables del mecanismo de comunicación, desarrollado en este trabajo de tesis, se pueden mencionar las siguientes:

a) El mecanismo proporciona servicios completos de transferencia de información entre nodos, que incluye: la transferencia de mensajes, de comandos, de archivos de texto y de estructuras en memoria. Cualquier programa de aplicación y los usuarios del sistema, en sí, pueden hacer uso de estos servicios, mismos que son ofrecidos de una manera "amigable", tanto para los usuarios, mediante un menú de servicios, como para los programas de aplicación, mediante una Interface aplicación-aplicación, que permite ver a los servicios de transferencia de información, como si fueran funciones de librería.

b) Debido a que uno de los requerimientos más importantes, en aplicaciones de SCD's, es el contar con una disponibilidad operacional superior al 90%, el mecanismo de comunicación se presenta como una buena alternativa para los sistemas con arquitectura distribuida y que trabajen en ambiente de tiempo real, ya que proporciona esquemas de tolerancia a fallas, que satisfacen los requerimientos de transferencia de información, aún con la presencia de falla en algún nodo del sistema. Esta característica es muy importante para sistemas donde la disponibilidad de la información, entre todos sus nodos integrantes, sea un factor determinante para la realización de las funciones del sistema.

Considerando lo anterior, el prototipo de mecanismo de comunicación desarrollado tiene su principal aplicación en el proyecto de modernización de los SADRES, con la implantación de una arquitectura distribuida. Sin embargo, la metodología de desarrollo y el mecanismo de comunicación en sí, pueden servir como base para el desarrollo de mecanismos de comunicación para sistemas distribuidos con requerimientos específicos, y que requieran de una transferencia de información en tiempo real, así como de una alta disponibilidad (>95%) en la transferencia de información.

APÉNDICE: SUBSISTEMA DEL MECANISMO DE COMUNICACIÓN PARA LOS NODOS DE ADQUISICIÓN.

```
/*
Nombre del programa:    ADQ.C
Versión:                8.0
Diseño y programación:  Gonzalo Martínez Hurtado
Institución:            Instituto de Investigaciones Eléctricas
Ultima revisión:        25 de mayo de 1995
Descripción:            Subsistema que permite la comunicación, transmisión y recepción de información (mensajes,
                        comandos, archivos y estructuras en memoria), entre procesos localizados en distintos nodos
                        de un Sistema de Control Distribuido, en ambiente del S.O. Q.N.X. Contiene mecanismos de
                        detección y tolerancia a fallas de nodos. Programa diseñado para los Nodos de Adquisición
                        de un SCD para una CTE normalizada. */

/* DECLARACIÓN DE LOS ARCHIVOS CABECERA */

#include <conio.h>
#include <ctype.h>
#include <fcntl.h>
#include <i86.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/dev.h>
#include <sys/kernel.h>
#include <sys/name.h>
#include <sys/proxy.h>
#include <sys/psinfo.h>
#include <sys/qnxterm.h>
#include <sys/types.h>
#include <sys/vc.h>
#include <sys/wait.h>
#include <unistd.h>
#include "/usr/gonit/proyecto/mensajes.h"

/* DEFINICIÓN DE VARIABLES PARA LA COMPROBACIÓN DE CONDICIONES */

#define TRUE    1
#define FALSE   0
#define ERROR   -1
#define RENGLON 50
#define COLUMNA 100

/* DECLARACIÓN DE LAS FUNCIONES QUE COMPONEN EL PROGRAMA */
/* FUNCIONES DE INICIALIZACIÓN Y CONFIGURACION */

int liga_nombre(void);   /* Registro del Nodo de Adquisición */
int liga_proxy(void);    /* Generación de un proceso 'proxy' de desbloqueo */
void red(int nid_tx);    /* Configuración de los Nodos de Adquisición */
void alta(void);         /* Aviso de alta del nodo en la red */
int consola(int proxy);  /* Inicialización de la consola de trabajo */
void myhand(int sig);    /* Manejo de la señal 'SIGQUIT' (terminación) */
void espera(int fpid);   /* Espera de 5 segundos para 'Send()' */
```

```
/* FUNCIONES PARA EL ENVÍO DE INFORMACIÓN */

void envio(int tipo);                                   /* Captura del tipo de información por transmitir */
void pids(int nodo, int tipo, int falla_rx);            /* Módulo para la transmisión de información */
int conexion(int nodo);                                 /* Búsqueda del proceso receptor primario o respaldo */
int conexion2(int i, int env_resp);                     /* Búsqueda de procesos receptores secundarios */
int env_inicio(int rpid, int nodo, int tipo);           /* Envío del inicio de transmisión */
int env_msg(int rpid, int falla_rx);                    /* Transmisión de un mensaje simple */
int env_cmd(int rpid, int falla_rx);                    /* Transmisión de un comando */
int env_arch2(int rpid, int falla_rx);                  /* Obtención del nombre del archivo por transmitir */
void obten_nombre(void);                                /* Obtención de los nombres de archivos por difundir */
void env_archs(int i,int j);                            /* Módulo para el envío de archivos */
int env_arch3(int rpid);                                /* Envío de un archivo por bloques */
int env_final(int rpid);                                /* Envío del final de transmisión */
void rec_sol(int prim, pid_t *pids, pid_t *flgs);       /* Recepción de solicitud de transmisión de estructuras */
void crear(int n, int i, pid_t *zid, pid_t *flg, pid_t *flg2);  /* Creación de procesos transmisores de estructuras */
void env_est(int i, int env_resp);                      /* Transmisión de estructura */

/* FUNCIONES PARA TOLERANCIA A FALLAS */

int aviso(void);                                        /* Aviso de transmisión de estructuras hacia nodo respaldo */
void aviso2(int pid_ur);                                /* Aviso de fin de transmisión de estructuras */
void vigia(int ndf, int zid2);                          /* Proceso vigía de nodo respaldado */
void control(int prim, pid_t *pids, pid_t *flgs);       /* Proceso controlador del proceso vigía */
void crear2(int n, int i, pid_t *zid, pid_t *flg);      /* Creación de procesos respaldo transmisores de estructuras */

/* FUNCIONES PARA LA RECEPCIÓN DE INFORMACIÓN */

void recibir(int recep);                                /* Módulo para la recepción de información */
void rec_inicio(int rpid, int proxy, int recep);        /* Recepción del mensaje de inicio de transmisión */
int rec_msg(void);                                      /* Recepción de un mensaje simple */
int rec_cmd(void);                                      /* Recepción de un comando */
int rec_arch2(void);                                    /* Recepción de un archivo en bloques */
int rec_arch3(void);                                    /* Recepción de un archivo en bloques */
int rec_final(void);                                    /* Recepción del mensaje de fin de transmisión */

/* FUNCIONES GENERALES */

void disparo(int sig);  /* Disparo del 'proxy' para desbloqueo */
void modo(void);        /* Determinación del modo de trabajo del nodo */
void desligado(void);   /* Desligamiento del proceso receptor */
void alerta1(void);     /* Alerta de caída del nodo receptor */
void alerta2(void);     /* Alerta de caída del nodo transmisor */
void pausa(void);       /* Generación de una pausa */
void borrar(void);      /* Borrado de mensajes de alerta */
void sonido(void);      /* Generación de un sonido de alerta */

/* DECLARACIÓN DE PUNTEROS DE ARCHIVO */

FILE *fp, *fp1;

/*****************************************************************************
MÓDULO: main().
OBJETIVO: Llamar a los módulos de inicialización de pantalla, registro del proceso, configuración del anillo lógico de los
          Nodos de Adquisición y creación de procesos respaldo transmisores de estructuras.
*****************************************************************************/
```

```
pid_t pid;
char nodo_tx[7];
int id, nr, nrd, ban, nid_tx;
void main()
{
 pid_t *zd, zzid[5];

 if (term_load() == -1) {        /* Inicializacion De Pantalla */
  fprintf(stderr, " term_load error !!, errno = %d\n", errno);
  exit(-1);
  }
 term_clear(TERM_CLS_SCRH);
 if ((id = liga_nombre()) == -1) {       /* Registro Del Proceso */
  pausa();
  term_cur(24,0);
  }
 else {
  red(nid_tx);        /* Configuracion De Los Nodos de Adquisicion */
  zd = &zzid[0];
  crear2(2,0,zd,zd);  /* Creación De Procesos Respaldo Transmisores De Estructuras */
  }/*else-id*/
}/*main*/

/*****************************************************************************************
MÓDULO:          crear2().
OBJETIVO.        Crear procesos respaldo transmisores de estructuras, en caso de falla del Nodo de Adquisición
                 respaldado. Llamar al módulo de envío de estructuras y al módulo receptor de avisos de inicio
                 de transmisión de estructuras, por parte del Nodo de Adquisición respaldado. Llamar al módulo
                 de creación de lo procesos encargados de enviar las estructuras hacia Nodos de Presentación.
PARÁMETROS:      n, número de Nodos de Presentación en el sistema.
                 i, contador del número de procesos creados.
                 *zid, *flg, apuntadores al primer elemento de un arreglo de identificadores de procesos.
*****************************************************************************************/
int zid1;
void crear2(int n, int i, pid_t *zid, pid_t *flg)
{
 pid_t *ziid, ziid[5];

 if (n > 0) {
  *zid = fork();
  if (*zid == 0) env_est(i,TRUE);   /* Envío De Estructuras */
  else {
   i++;
   n--;
   zid++;
   crear2(n,i,zid,flg);
   }
 }
 else {
  zid1 = fork();
  if (zid1 == 0) control(TRUE,flg,flg);  /* Receptor De Aviso De Inicio De Transmisión De Estructuras */
  else {
   ziid = &ziid[0];
   crear2(2,0,ziid,ziid,flg);  /* Creación De Procesos Transmisores De Estructuras */
   }
  }/*else-zid*/
```

```
}/*crear2*/

/*****************************************************************************************
MÓDULO:       crear().
OBJETIVO:     Crear los procesos encargados de la transmisión de estructuras hacia los Nodos de Presentación.
              Crear al proceso receptor de solicitudes de transferencia de estructuras. Llamar a los módulos
              de inicialización de la consola de trabajo y  al de alta de nodo y establecimiento de modo
              operativo.gado del menú de servicios ofrecidos por el mecanismo de comunicación a los
              usuarios del sistema. Recepción de mensajes de alta, de inicio de comunicación y de solicitud
              de servicios de transferencia por parte de programas de aplicación. Llamar a los módulos de
              recepción y transmisión de información.
PARÁMETROS:   n, número de Nodos de Presentación en el sistema.
              i, contador de procesos transmisores creados.
              *zid, *flg, apuntadores al primer elemento de un arreglo de identificadores de procesos.
              *flg2, apuntador al primer elemento del arreglo de identificadores de los procesos respaldo
              transmisores.
*****************************************************************************************/
void crear(int n, int i, pid_t *zid, pid_t *flg, pid_t *flg2)
{
 char opcion;
 pid_t proxy;
 int k, l, ok, loop, zid2;
 if (n > 0) {
  *zid = fork();
  if (*zid == 0) env_est(i,FALSE);  /* Envío De Estructuras */
  else {
   i++;
   n--;
   zid++;
   crear(n,i,zid,flg,flg2);
  }
 }
 else {
  zid2 = fork();
  if (zid2 == 0) rec_sol(TRUE,flg,flg);  /* Receptor De Solicitud De Envío De Estructura */
  else {
   ban = TRUE;
   loop = TRUE;
   strcpy(msg1.resp,"ok");
   while (loop != FALSE) {
    if ((proxy = liga_proxy()) == -1) pausa();
    else {
         alta();                  /* Alta Del Nodo Y Establecimiento De Modo Operativo */
         term_box_fill(0,0,80,25,TERM_CYAN,1,' ');
         term_box_fill(4,19,34,11,TERM_BLUE,2,'');
         for (k = 20; k < 54; k++) term_printf(15,k,TERM_CYAN,"_");
         for (k = 5; k < 16; k++) term_printf(k,53,0,"_");
         term_color(TERM_WHITE);
         term_printf(5,21,2,"MENU DEL NODO DE ADQUISICION %d",nid_tx);
         term_attr_type(7,25,"Enviar $M$ensaje",0,0,2,'$');
         term_attr_type(8,25,"Enviar $C$omando",0,0,2,'$');
         term_attr_type(9,25,"Enviar $A$rchivo",0,0,2,'$');
         term_attr_type(10,25,"$S$alir de la aplicacion",0,0,2,'$');
         term_printf(11,25,0,"Opcion elegida => ");
         term_cur(11,43);
         if ((ok = consola(proxy)) != -1) {       /* Inicialización De La Consola De Trabajo */
```

```
pid = Receive(0,&msg0,sizeof(msg0));
if (pid == proxy) {
 qnx_proxy_detach(proxy);
 opcion = tolower(getch());
 desligado();
 switch (opcion) {
  case 'm': envio(1);       /* Envio De Mensaje */
            break;
  case 'c': envio(2);       /* Envio De Comando */
            break;
  case 'a': envio(4);       /* Envio De Archivo */
            break;
  case 's': loop = FALSE;
            term_cur(24,0);
            kill(zid1,SIGKILL);
            waitpid(zid1,NULL,0);
            kill(zid2,SIGKILL);
            waitpid(zid2,NULL,0);
            for (l = 0; l <= 1; l++) {
             kill(*flg,SIGKILL);
             waitpid(*flg,NULL,0);
             flg++;
             }
            for (l = 0; l <= 1; l++) {
             kill(*flg2,SIGKILL);
             waitpid(*flg2,NULL,0);
             flg2++;
             }
            qnx_name_detach(nid_tx,id);
            break;
  default : term_color(TERM_CYAN);
            borrar();
            sonido();
            term_printf(21,28,TERM_RED_BG,"OPCION INVALIDA !!");
            pausa();
 }/*switch*/
}/*if-opcion*/
else {
 qnx_proxy_detach(proxy);
 if (!strcmp(msg0.inicio,"ok")) recibir(TRUE);      /* Recepción De Información */
 if ((!strcmp(msg0.inicio,"up")) || (!strcmp(msg0.inicio,"ur"))) {
  Reply(pid,&msg1,sizeof(msg1));
  term_color(TERM_CYAN);
  borrar();
  sonido();
  if (!strcmp(msg0.inicio,"up"))
       term_printf(21,18,TERM_RED_BG,"AVISO: Nodo respaldado #%d en funcionamiento",nrd);
  if (!strcmp(msg0.inicio,"ur"))
       term_printf(21,19,TERM_RED_BG,"AVISO: Nodo respaldo #%d en funcionamiento",nr);
  pausa();
  desligado();
 }/*if-strcmp*/
 if (!strcmp(msg0.inicio,"db")) {
  Reply(pid,&msg1,sizeof(msg1));
  obten_nombre();
 }
```

```
            if ((!strcmp(msg0.inicio,"mg")) || (!strcmp(msg0.inicio,"cd"))) {
              Reply(pid,&msg1,sizeof(msg1));
              pids(msg0.nd_rx,msg0.tipo,TRUE);
              desligado();
             }
            }/*else-opcion*/
           }/*if-ok-consola*/
          else {
           term_color(TERM_CYAN);
           borrar();
           sonido();
           term_printf(21,19,TERM_RED_BG,"ERROR: No se pudo inicializar la consola !");
           term_cur(24,0);
           kill(zid1,SIGKILL);
           waitpid(zid1,NULL,0);
           kill(zid2,SIGKILL);
           waitpid(zid2,NULL,0);
           for (i = 0; i <= 1; i++) {
            kill(*flg,SIGKILL);
            waitpid(*flg,NULL,0);
            flg++;
            }
           for (i = 0; i <= 1; i++) {
            kill(*flg2,SIGKILL);
            waitpid(*flg2,NULL,0);
            flg2++;
            }
           qnx_name_detach(nid_tx,id);
           exit(0);
          }/*else-consola*/
         }/*else-proxy*/
    }/*while*/
   }/*else-zid2*/
  }/*else-n>0*/
}/*crear*/

/*****************************************************************************
MÓDULO:         liga_nombre().
OBJETIVO:       Registrar la ejecución del mecanismo de comunicación, bajo un nombre.
SALIDAS:        id, el identificador del proceso o ERROR, si no se pudo registrar el proceso.
*****************************************************************************/
int liga_nombre(void)
{
 int id;
 pid_t info;
 struct _psinfo data;
 char *nids, buffer[10];
 info = qnx_psinfo(0,0,&data,0,0);
 nids = itoa(data.sid_nid,buffer,10);
 nid_tx = data.sid_nid;
 strcpy(nodo_tx,"nodo");
 strcat(nodo_tx,nids);
 if ((id = qnx_name_attach(nid_tx,nodo_tx)) == -1) {
  sonido();
  term_printf(21,21,TERM_RED_BG,"ERROR: Registro de proceso fallido !");
  return ERROR;
```

```
 }
 else return id;
}/*liga_nombre*/

/*****************************************************************************
MÓDULO:         red().
OBJETIVO:       Obtener la configuración (nodos respaldo y respaldado) del anillo lógico de los
                Nodos de Adquisición.
PARÁMETROS:     nid, número de Nodo de Adquisición actual.
*****************************************************************************/
void red(int nid)
{
 switch(nid) {
  case 2: nr = 6;
          nrd = 4;
          break;
  case 4: nr = 2;
          nrd = 6;
          break;
  case 6: nr = 4;
          nrd = 2;
          break;
 }/*switch*/
}/*red*/

/*****************************************************************************
MÓDULO:         alta().
OBJETIVO:       Avisar, al Nodo de Adquisición respaldo y respaldado, el inicio de operaciones del
                mecanismo de comunicación. Establecer el modo operativo del nodo actual (Primario-Primario
                o Primario-Respaldo).
*****************************************************************************/
pid_t sid;
void alta(void)
{
 int zid, fpid, status, rpid_nr, rpid_nrd;
 char *nr_c, nr_s[7], buffer1[10], *nrd_c, nrd_s[7], buffer2[10];

 signal(SIGQUIT,myhand);
 term_printf(21,4,TERM_WHITE,"NA #%d trabajando con NA Respaldo #%d       y NA Respaldado #%d
",nid_tx,nr,nrd);
 fpid = getpid();
 nr_c = itoa(nr,buffer1,10);        /* Aviso Al Nodo Respaldo */
 strcpy(nr_s,"/nodo");
 strcat(nr_s,nr_c);
 if ((rpid_nr = qnx_name_locate(0,nr_s,0,0)) == -1) term_printf(21,40,TERM_RED_BG,"Inactivo");
 else {
  term_printf(21,40,TERM_WHITE_BG,"Activo");
  if (ban == TRUE) {
   strcpy(msg0.inicio,"up");
   if (zid = fork()) sid = Send(rpid_nr,&msg0,&msg1,sizeof(msg0),sizeof(msg1));
   else espera(fpid);
   if (sid != -1) kill(zid,SIGQUIT);
   waitpid(zid,&status,0);
  }/*if-ban*/
 }/*else-rpid*/
desligado();
```

```
nrd_c = itoa(nrd,buffer2,10);        /* Aviso Al Nodo Respaldado */
strcpy(nrd_s,"/nodo");
strcat(nrd_s,nrd_c);
if ((rpid_nrd = qnx_name_locate(0,nrd_s,0,0)) == -1) term_printf(21,68,TERM_RED_BG,"Inactivo");
else {
  term_printf(21,68,TERM_WHITE_BG,"Activo");
  if (ban == TRUE) {
   strcpy(msg0.inicio,"ur");
   if (zid = fork()) sid = Send(rpid_nrd,&msg0,&msg1,sizeof(msg0),sizeof(msg1));
   else espera(fpid);
   if (sid != -1) kill(zid,SIGQUIT);
   waitpid(zid,&status,0);
   }/*if-ban*/
 }/*if-rpid*/
desligado();
ban = FALSE;
}/*alta*/

/*****************************************************************************
MÓDULO:          liga_proxy().
OBJETIVO:        Crear un 'proxy' para desbloqueo.
SALIDAS:         proxy, el identificador del 'proxy' o ERROR, si no se pudo crear el 'proxy'.
*****************************************************************************/
int liga_proxy(void)
{
 int proxy;

 if ((proxy = qnx_proxy_attach(0,0,0,-1)) == -1) {
  sonido();
  term_printf(21,10,TERM_RED_BG,"ERROR !!: No se pudo enlazar proxy");
  return ERROR;
  }
 else return proxy;
}/*liga_proxy*/

/*****************************************************************************
MÓDULO:          consola().
OBJETIVO:        Inicializar la consola de trabajo, para que pueda recibir mensajes y solicitudes desde el teclado.
PARÁMETROS:      proxy, el identificador de un 'proxy' para desbloqueo.
SALIDAS:         0, si se logro inicializar la consola o ERROR si la inicialización fue fallida.
*****************************************************************************/
int consola(int proxy)
{
 int kb_fd;
 struct _dev_info_entry info;
 char *con, buffer[10], consol[10];

 if (dev_info(0,&info) == 0) {
  con = itoa(info.unit,buffer,10);    /*Obtencion Del Numero De La Consola Actual*/
  strcpy(consol,"/dev/con");
  strcat(consol,con);
  kb_fd = open(consol,O_RDWR);        /*Descriptor De Archivo Se Asocia A La Consola*/
  if (dev_mode(kb_fd,0,_DEV_MODES) == -1)   /*Limpiado De Los Canales De E/S De La Consola*/
   return ERROR;
  else
   if (dev_state(kb_fd,0,_DEV_EVENT_INPUT) == -1)   /*Limpiado De Estados De Eventos Pendientes*/
```

```
    return ERROR;
  else                          /*armado del proxy para futuros eventos*/
   if (dev_arm(kb_fd,proxy,_DEV_EVENT_INPUT) == -1)
        return ERROR;
   else return 0;
 }
else return ERROR;
}/*consola*/

/*****************************************************************************
MÓDULO:         myhand().
OBJETIVO:       Manejador de la señal SIGQUIT.
PARÁMETROS:     sig, número de la señal SIGQUIT.
*****************************************************************************/
void myhand(int sig)
{
 sig = 0;
 sid = -1;
}/*myhand*/

/*****************************************************************************
MÓDULO:         espera().
OBJETIVO:       Espera de 5 segundos para el reconocimiento al mensaje de inicio de transmisión.
PARÁMETROS:     fpid, identificador del proceso transmisor.
*****************************************************************************/
void espera(int fpid)
{
 signal(SIGQUIT,SIG_DFL);
 sleep(5);
 if ((sid == 0) || (sid == -1)) kill(fpid,SIGQUIT);
 exit(0);
}/*espera*/

/*****************************************************************************
MÓDULO:         envio().
OBJETIVO:       Obtener el número del nodo receptor de información y llamar al módulo de transmisión de
                información.
PARÁMETROS:     tipo, tipo de información por transmitir.
*****************************************************************************/
void envio(int tipo)
{
 int nodo;

 term_clear(TERM_CLS_SCRH);
 term_box_fill(0,0,80,25,TERM_MAGENTA,1,'_');
 term_box_fill(2,2,76,19,TERM_CYAN,2,' ');
 term_color(TERM_WHITE);
 term_printf(3,23,2,"TRANSMISION DE INFORMACION");
 term_printf(5,10,0,"Numero de Nodo Receptor del ");
 switch(tipo) {
  case 1: term_printf(5,38,0,"mensaje => ");
          break;
  case 2: term_printf(5,38,0,"comando => ");
          break;
  case 4: term_printf(5,38,0,"archivo => ");
          break;
```

```
 }/*switch*/
 term_cur(5,49);
 scanf("%d",&nodo);
 pids(nodo,tipo,FALSE);
}/*envio*/

/*****************************************************************************************
MÓDULO:          pids().
OBJETIVO:        Llamar a los módulos transmisores del mensaje de inicio de transmisión, de mensajes, de
                 comandos, de archivos y del mensaje de fin de transmisión.
PARÁMETROS:      nodo, nodo receptor de la información.
                 tipo, tipo de información por transmitir.
                 falla_rx, bandera que indica si existió una falla en el nodo receptor.
*****************************************************************************************/
int ppid;
void pids(int nodo, int tipo, int falla_rx)
{
 int ok, rpid, spid;

 term_box_fill(0,0,80,25,TERM_MAGENTA,1,'_');
 term_box_fill(2,2,76,19,TERM_CYAN,2,'');
 term_color(TERM_WHITE);
 switch(tipo) {
  case 1: term_printf(3,25,2,"TRANSMISION DE MENSAJE");
          break;
  case 2: term_printf(3,25,2,"TRANSMISION DE COMANDO");
          break;
  case 4: term_printf(3,25,2,"TRANSMISION DE ARCHIVO");
          break;
 }
 spid = getpid();
 rpid = conexion(nodo);
 if (rpid != -1) {
  term_color(TERM_WHITE);
  term_printf(9,10,0,"Soy el proceso PID:      %d", spid);
  term_printf(10,10,0,"PID del proceso receptor:  %d", rpid);
  term_printf(11,10,2,"Iniciando comunicacion con Nodo Receptor...");
  if ((ok = env_inicio(rpid,nodo,tipo)) == -1) {        /* Envio Del Mensaje De Inicio De Transmision */
   strcpy(msg7.nck,"Nodo receptor ocupado !\n");
   if (msg0.ban == TRUE) {
    if (msg0.tipo == 1) ppid = Receive(pid,&msg5,sizeof(msg5));
    if (msg0.tipo == 2) ppid = Receive(pid,&msg6,sizeof(msg6));
    }
   Reply(ppid,&msg7,sizeof(msg7));
   pausa();
  }/*if-ok*/
  else {
   switch(tipo) {
    case 1: if ((ok = env_msg(rpid,falla_rx)) == -1) {        /* Envio De Un Mensaje */
                    term_box_fill(0,0,80,25,TERM_MAGENTA,1,'_');
                    term_box_fill(2,2,76,19,TERM_CYAN,2,'');
                    term_color(TERM_WHITE);
                    term_printf(3,25,2,"TRANSMISION DE MENSAJE");
                    pids(nodo,tipo,TRUE);
                    }
                 else {
```

```
                    ok = env_final(rpid);
                    pausa();
                    }
                break;
    case 2: if ((ok = env_cmd(rpid,falla_rx)) == -1) {      /* Envio De Un Comando */
                    term_box_fill(0,0,80,25,TERM_MAGENTA,1,'_');
                    term_box_fill(2,2,76,19,TERM_CYAN,2,' ');
                    term_color(TERM_WHITE);
                    term_printf(3,25,2,"TRANSMISION DE COMANDO");
                    pids(nodo,tipo,TRUE);
                    }
                else {
                    ok = env_final(rpid);
                    pausa();
                    }
                break;
    case 4: if ((ok = env_arch2(rpid,falla_rx)) == -1) {      /* Envio De Un Archivo */
                    term_box_fill(0,0,80,25,TERM_MAGENTA,1,'_');
                    term_box_fill(2,2,76,19,TERM_CYAN,2,' ');
                    term_color(TERM_WHITE);
                    term_printf(3,25,2,"TRANSMISION DE ARCHIVO");
                    pids(nodo,tipo,TRUE);
                    }
                else {
                    ok = env_final(rpid);
                    pausa();
                    }
                break;
   }/*switch-tipo*/
  }/*else-ok-env_inicio*/
 }/*if-rpid-conexion*/
 else {
  strcpy(msg7.ack,"Enlace fallido !\n");
  if (msg0.ban == TRUE) {
   if (msg0.tipo == 1) ppid = Receive(pid,&msg5,sizeof(msg5));
   if (msg0.tipo == 2) ppid = Receive(pid,&msg6,sizeof(msg6));
  }
  Reply(ppid,&msg7,sizeof(msg7));
 }
}/*pids*/

/*****************************************************************************
MÓDULO:         conexión()
OBJETIVO:       Establecer circuitos virtuales con procesos receptores en nodos remotos.
PARÁMETROS:     nodo, nodo receptor con el que se requiere establecer el circuito virtual.
SALIDAS:        rpid, identificador virtual del proceso en el nodo remoto o ERROR si no se pudo establecer el
                circuito virtual.
*****************************************************************************/
int conexion(int nodo)
{
 char prim[7], resp[7];
 int rp, rpid, igual, mismo;

 igual = FALSE;
 mismo = FALSE;
 term_printf(6,10,TERM_WHITE,"Iniciando rutina de conexion con Nodo Receptor #%d",nodo);
```

```
switch(nodo) {
 case 1: if (nodo != nid_tx) {
            strcpy(prim,"/nodo1");
            strcpy(resp,"/nodo3");
            rp = 3;
            }
           else igual = TRUE;
           break;
 case 2: if (nodo != nid_tx) {
            strcpy(prim,"/nodo2");
            strcpy(resp,"/nodo6");
            rp = 6;
            }
           else igual = TRUE;
           break;
 case 3: if (nodo != nid_tx) {
            strcpy(prim,"/nodo3");
            strcpy(resp,"/nodo1");
            rp = 1;
            }
           else igual = TRUE;
           break;
 case 4: if (nodo != nid_tx) {
            strcpy(prim,"/nodo4");
            strcpy(resp,"/nodo2");
            rp = 2;
            }
           else igual = TRUE;
           break;
 case 5: if (nodo != nid_tx) {
            strcpy(prim,"/nodo5");
            strcpy(resp,"/nodo5");
            rp = 5;
            }
           else igual = TRUE;
           break;
 case 6: if (nodo != nid_tx) {
            strcpy(prim,"/nodo6");
            strcpy(resp,"/nodo4");
            rp = 4;
            }
           else igual = TRUE;
           break;
default: sonido();
           term_printf(21,20,TERM_RED_BG,"ERROR !: El nodo %d no existe en la red",nodo);
           pausa();
           return ERROR;
 }/*switch*/
if (rp == nid_tx) mismo = TRUE;
if (igual == TRUE) {
 sonido();
 term_printf(21,16,TERM_RED_BG,"ERROR !: Se encuentra trabajando en el nodo #%d",nid_tx);
 pausa();
 return ERROR;
 }
else {
```

```
 if ((rpid = qnx_name_locate(0,prim,0,0)) == -1) {
   sonido();
   term_printf(21,4,TERM_RED_BG,"!! WARNING !!: Nodo Receptor Primario #%d fuera de servicio o indispuesto",nodo);
   term_printf(7,10,TERM_WHITE,"Inicio de Conexion con el Nodo Receptor Respaldo #%d",rp);
   pausa();
   term_color(TERM_MAGENTA);
   borrar();
   if (mismo == TRUE) {
    sonido();
    term_printf(21,10,TERM_RED_BG,"!! WARNING !!: Se encuentra trabajando en el Nodo respaldo");
    term_printf(8,10,TERM_WHITE,"No hay nodo receptor disponible !");
    term_printf(9,10,2,"FIN  DE  COMUNICACION");
    pausa();
    return ERROR;
   }/*if-mismo*/
  else {
   if ((rpid = qnx_name_locate(0,resp,0,0)) == -1) {
       sonido();
       term_printf(21,5,TERM_RED_BG,"!! ALERTA !!: Nodo Receptor Respaldo #%d fuera de servicio o indispuesto ",rp);
       term_printf(8,10,TERM_WHITE,"No hay nodo receptor disponible !");
       term_printf(9,10,2,"FIN  DE  COMUNICACION");
       pausa();
       return ERROR;
       }/*if-rpid-resp*/
    else {
          term_color(TERM_WHITE);
          term_printf(8,10,2,"Conexion Establecida con Nodo Receptor Respaldo #%d !",rp);
          return rpid;
          }/*else-rpid-resp*/
   }/*else-mismo*/
  }/*if-rpid-prim*/
 else {
   term_color(TERM_WHITE);
   term_printf(7,10,2,"Conexion Establecida con Nodo Receptor Primario #%d !",nodo);
   return rpid;
   }/*if-rpd-prim*/
 }/*else-igual*/
}/*conexion*/

/*****************************************************************************************
MÓDULO:          conexión2().
OBJETIVO:        Establecer circuitos virtuales con los procesos receptores de estructuras localizados en los
                 Nodos de Presentación.
PARÁMETROS:      i, número del proceso transmisor de estructura.
                 env_resp, bandera que indica si el proceso transmisor es principal o de respaldo.
SALIDAS:         rrpid, identificador virtual del proceso en el nodo remoto, -1 si no se pudo establecer el enlace.
*****************************************************************************************/
int conexion2(int i, int env_resp)
{
 int nodo, rrpid;
 char proc[8];

 strcpy(proc,"/nodo");
 switch(i) {
  case 0: strcat(proc,"1");
          break;
```

```
  case 1: strcat(proc,"3");
          break;
 }
 if (env_resp == FALSE) nodo = nid_tx;
 else nodo = nrd;
 switch(nodo) {
  case 2: strcat(proc,"a");
          break;
  case 4: strcat(proc,"b");
          break;
  case 6: strcat(proc,"c");
          break;
 }
 rpid = qnx_vc_name_attach(1,0,proc);
 return rpid;
 }/*conexion2*/

/*****************************************************************************************
MÓDULO:         env_inicio().
OBJETIVO:       Inicializar y enviar el mensaje de inicio de transmisión.
PARÁMETROS:     rpid, el identificador del proceso receptor.
                nodo, nodo receptor del mensaje.
                tipo, tipo de información por transmitir.
SALIDAS:        rpid, si la transmisión se realiza correctamente o ERROR si existe una falla en el proceso
                receptor.
*****************************************************************************************/
pid_t sid;
int env_inicio(int rpid,int nodo,int tipo)
{
 int zid, fpid, status;

 signal(SIGQUIT,myhand);
 fpid = getpid();
 term_printf(12,10,TERM_WHITE,"Enviando mensaje de inicio de transmision...");
 strcpy(msg0.inicio,"ok");
 strcpy(msg0.nm,nodo_tx);
 msg0.nd_tx = nid_tx;
 msg0.nd_rx = nodo;
 msg0.tipo = tipo;
 if (zid = fork()) sid = Send(rpid,&msg0,&msg1,sizeof(msg0),sizeof(msg1));
 else espera(fpid);
 if (sid == -1) {
  waitpid(zid,&status,0);
  sonido();
  term_printf(21,21,TERM_RED_BG,"WARNING !: Nodo receptor #%d ocupado",nodo);
  return ERROR;
 }
 else {
  kill(zid,SIGQUIT);
  waitpid(zid,&status,0);
  term_printf(13,10,2,"Mensaje de inicio de transmision recibido por Nodo Receptor !");
  return rpid;
 }
}/*env_inicio*/
```

```
/*****************************************************************************
MÓDULO:         env_msg().
OBJETIVO:       Transmitir un mensaje.
PARÁMETROS:     rpid, el identificador del proceso receptor.
                falla_rx, bandera que indica si existió una falla en el proceso receptor.
SALIDAS:        rpid, si la transmisión se realiza correctamente o ERROR, si existe falla en el proceso receptor.
*****************************************************************************/
int env_msg(int rpid, int falla_rx)
{
 int falla;

 if (falla_rx == FALSE) {
  term_printf(14,10,TERM_WHITE,"Mensaje a transmitir => ");
  term_cur(14,34);
  gets(msg5.mss);
  }
 else {
  if (msg0.ban == TRUE) ppid = Receive(pid,&msg5,sizeof(msg5));
  term_printf(14,10,TERM_WHITE,"Mensaje a transmitir => %s",msg5.mss);
  }
 term_printf(15,10,0,"Esperando respuesta...");
 if ((falla = Send(rpid,&msg5,&msg7,sizeof(msg5),sizeof(msg7))) == -1) {
  alerta1();
  msg0.ban = FALSE;
  return ERROR;
  }
 else {
  term_color(TERM_WHITE);
  term_printf(16,10,2,"Mensaje recibido !");
  term_printf(17,10,0,"Respuesta del N. Receptor: %s",msg7.ack);
  term_printf(18,10,2,"FIN  DE  COMUNICACION");
  desligado();
  Reply(ppid,&msg7,sizeof(msg7));
  return falla;
  }
}/*env_msg*/

/*****************************************************************************
MÓDULO:         env_cmd().
OBJETIVO:       Transmitir un comando
PARÁMETROS:     rpid, el identificador del proceso receptor.
                falla_rx, bandera que indica si existió una falla en el proceso receptor.
SALIDAS:        rpid, si la transmisión se realiza correctamente o ERROR, si existe falla en el proceso receptor.
*****************************************************************************/
int env_cmd(int rpid, int falla_rx)
{
 char band;
 int falla, loop;

 if (falla_rx == FALSE) {
  term_printf(14,10,TERM_WHITE,"Comando a ejecutar => ");
  term_cur(14,32);
  gets(msg6.cmd);
  }
 else term_printf(14,10,TERM_WHITE,"Comando a ejecutar => %s",msg6.cmd);
 if (falla_rx == FALSE) {
```

```
 loop = TRUE;
 while (loop != FALSE) {
  term_color(TERM_WHITE);
  term_attr_type(15,10,"$B$andera de comando ejecutado || $I$nformacion generada ?",0,0,2,'$');
  term_cur(15,65);
  band = tolower(getch());
  switch(band) {
   case 'b': loop = FALSE;
             msg6.flag = 0;
             break;
   case 'i': loop = FALSE;
             msg6.flag = 1;
             break;
   default:  sonido();
             term_printf(21,28,TERM_RED_BG,"OPCION INVALIDA !!");
             pausa();
             term_color(TERM_MAGENTA);
             borrar();
   }/*switch-band*/
  }/*while*/
 }/*if-falla_rx*/
else {
 if (msg0.ban == TRUE) ppid = Receive(pid,&msg6,sizeof(msg6)); }
term_printf(16,10,TERM_WHITE,"Esperando respuesta...");
if ((falla = Send(rpid,&msg6,&msg7,sizeof(msg6),sizeof(msg7))) == -1) {
 alerta1();
 msg0.ban = FALSE;
 return ERROR;
}
else {
 if (msg6.flag == 0) {
  term_color(TERM_WHITE);
  term_printf(17,10,0,"Respuesta del N. Receptor: %s",msg7.ack);
  term_printf(18,10,2,"FIN  DE  COMUNICACION");
  desligado();
  Reply(ppid,&msg7,sizeof(msg7));
  return falla;
 }
 else {
  if (!strcmp(msg7.ack,"Comando erroneo")) {
   term_color(TERM_WHITE);
   term_printf(17,10,0,"Respuesta del N. Receptor: %s",msg7.ack);
   term_printf(18,10,2,"FIN  DE  COMUNICACION");
   desligado();
   Reply(ppid,&msg7,sizeof(msg7));
   return falla;
   }
  else {
   term_clear(TERM_CLS_SCRH);
   system("more /txt.cmd");
   desligado();
   Reply(ppid,&msg7,sizeof(msg7));
   return falla;
   }/*else-strcmp*/
  }/*else-msg6.flag*/
}/*else-falla*/
```

```
}/*env_cmd*/

/*****************************************************************************************
MÓDULO:         env_arch2().
OBJETIVO:       Obtener el nombre del archivo que se requiere transmitir y llamar al módulo de transmisión de
                archivos.
PARÁMETROS:     rpid, identificador del proceso receptor.
                falla_rx, bandera que indica si existió una falla en el proceso receptor.
SALIDAS:        ok, valor retornado por el módulo env_arch3()
*****************************************************************************************/
int env_arch2(int rpid, int falla_rx)
{
 int ok;

 if (falla_rx == FALSE) {
  term_printf(14,10,TERM_WHITE,"Archivo por transmitir => ");
  term_curr(14,36);
  gets(msg9.nombre);
  }
 else term_printf(14,10,TERM_WHITE,"Archivo por transmitir => %s",msg9.nombre);
 term_printf(15,10,0,"Abriendo archivo...");
 ok = env_arch3(rpid);
 return ok;
}/*env_arch2*/

/*****************************************************************************************
MÓDULO:         obten_nombre().
OBJETIVO:       Obtener el o los nombres de los archivos que se requieren transmitir, por medio de la IAA y
                llamar al módulo de difusión de archivos hacia Nodos de Presentación.
*****************************************************************************************/
void obten_nombre(void)
{
 int i, j, caida;

 if (msg0.han == 2) {
  caida = Receive(pid,&msg8,sizeof(msg8));
  strcpy(msg1.resp,"ok");
  caida = Reply(pid,&msg1,sizeof(msg1));
  for (i = 1; i <= 3; i++) {
   if ((i != nid_tx) && (i != nr) && (i != nrd)) {
    for (j = 1; j <=2; j++) env_archs(i,j);  /* Difundir Archivo Hacia Nodos De Presentación */
    }
   }/*for*/
  }/*if-han*/
 else {
  ppid = Receive(pid,&msg9,sizeof(msg9));
  env_archs(msg0.nd_rx,0);     /* Transmitir archivo */
  }
}/*obten_nombre*/

/*****************************************************************************************
MÓDULO:         env_archs().
OBJETIVO:       Administrar la difusión de archivos hacia Nodos de Presentación. Llamar al módulo de
                establecimiento de enlaces y al módulo de transmisión de archivos.
PARAMETROS:     i, número de Nodo de Presentación o nodo receptor y  j, número de archivo.
*****************************************************************************************/
```

```
void env_archs(int i, int j)
 {
 int ok, rpid;

 term_box_fill(0,0,80,25,TERM_MAGENTA,1,'_');
 term_box_fill(2,2,76,19,TERM_CYAN,2,' ');
 term_color(TERM_WHITE);
 term_printf(3,25,2,"TRANSMISION DE ARCHIVO");
 rpid = conexion(i);          /* Establecimiento De Enlace Con Nodo Receptor */
 if (rpid != -1) {
  if ((ok = env_inicio(rpid,i,4) == -1)) {        /* Envio Del Mensaje De Inicio De Transmisión */
    strcpy(msg7.ack,"Nodo receptor ocupado !\n");
    Reply(ppid,&msg7,sizeof(msg7));
    pausa();
   }
  else {
    switch(j) {
     case 1: strcpy(msg9.nombre,msg8.nom1);
             break;
     case 2: strcpy(msg9.nombre,msg8.nom2);
             break;
    }
   if ((ok = env_arch3(rpid)) == -1) {        /* Transmisión De Archivo */
    alerta1();
    env_archs(i,j);
    }
   else ok = env_final(rpid);      /* Envio Del Mensaje De Fin De Transmisión */
  }/*else-ok-env_inicio*/
 }/*if-rpid*/
 else {
  if (msg0.ban == TRUE) {
   strcpy(msg7.ack,"Enlace fallido !");
   Reply(ppid,&msg7,sizeof(msg7));
   }
 }
}/*env_archs*/

/*****************************************************************************************
MÓDULO:          env_arch3().
OBJETIVO:        Transmitir un archivo.
PARÁMETROS:      rpid, identificador del proceso receptor.
SALIDAS:         rpid, si la transmisión se realiza correctamente, ERROR si existe falla en el proceso receptor o
                 0, si no existe el archivo.
*****************************************************************************************/
int env_arch3(int rpid)
{
 int i, falla;

 if ((fp1 = fopen(msg9.nombre,"r")) == NULL) {
  msg9.falla = TRUE;
  falla = Send(rpid,&msg9,&msg1,sizeof(msg9),sizeof(msg1));
  sonido();
  term_color(TERM_WHITE);
  term_printf(18,10,2,"FIN  DE  COMUNICACION");
  term_printf(21,24,TERM_RED_BG,",ERROR !: No existe el archivo");
  desligado();
```

```
 strcpy(msg7.ack,"No existe el archivo !");
 Reply(ppid,&msg7,sizeof(msg7));
 return 0;
}
else {
 msg9.falla = FALSE;
 i = 0;
 while (!feof(fp1)) {
  fgets(msg9.cadena[i++],COLUMNA,fp1);
  if (!feof(fp1)) {
   if (i == 10) {
         msg9.indice = i;
         msg9.bandera = FALSE;
         i = 0;
         if ((falla = Send(rpid,&msg9,&msg1,sizeof(msg9),sizeof(msg1))) == -1) {
          alerta1();
          fclose(fp1);
          return ERROR;
         }
        }/*if-i=10*/
  }/*if-feof*/
  else {
   msg9.indice = i-1;
   msg9.bandera = TRUE;
   if ((falla = Send(rpid,&msg9,&msg1,sizeof(msg9),sizeof(msg1))) == -1) {
         alerta1();
         fclose(fp1);
         return ERROR;
        }
   else {
         term_color(TERM_WHITE);
         term_printf(16,10,2,"Archivo recibido !");
         term_printf(17,10,0,"Respuesta del N. Receptor: %s",msg1.resp);
         term_printf(18,10,2,"FIN DE COMUNICACION");
         desligado();
         fclose(fp1);
         strcpy(msg7.ack,"Archivo recibido !");
         Reply(ppid,&msg7,sizeof(msg7));
         return falla;
        }
  }/*else-feof*/
 }/*while*/
}/*else-fopen*/
}/*env_arch3*/

/*****************************************************************************************
MÓDULO:          env_final().
OBJETIVO:        Enviar el mensaje de fin de transmisión.
PARÁMETROS:      rpid, identificador del proceso receptor.
SALIDAS:         rpid, si la transmisión se realiza correctamente o ERROR, si existe falla en el proceso receptor.
*****************************************************************************************/
int env_final(int rpid)
{
 int falla;

 strcpy(msg4.final,"ok");
```

```
 strcpy(msg7.ack,"No existe el archivo !");
 Reply(ppid,&msg7,sizeof(msg7));
 return 0;
 }
else {
 msg9.falla = FALSE;
 i = 0;
 while (!feof(fp1)) {
  fgets(msg9.cadena[i++],COLUMNA,fp1);
  if (!feof(fp1)) {
   if (i == 10) {
         msg9.indice = i;
         msg9.bandera = FALSE;
         i = 0;
         if (((falla = Send(rpid,&msg9,&msg1,sizeof(msg9),sizeof(msg1))) == -1) {
          alerta1();
          fclose(fp1);
          return ERROR;
         }
        }/*if-i=10*/
  }/*if-feof*/
  else {
   msg9.indice = i-1;
   msg9.bandera = TRUE;
   if ((falla = Send(rpid,&msg9,&msg1,sizeof(msg9),sizeof(msg1))) == -1) {
         alerta1();
         fclose(fp1);
         return ERROR;
         }
   else {
         term_color(TERM_WHITE);
         term_printf(16,10,2,"Archivo recibido !");
         term_printf(17,10,0,"Respuesta del N. Receptor: %s",msg1.resp);
         term_printf(18,10,2,"FIN DE COMUNICACION");
         desligado();
         fclose(fp1);
         strcpy(msg7.ack,"Archivo recibido !");
         Reply(ppid,&msg7,sizeof(msg7));
         return falla;
         }
  }/*else-feof*/
 }/*while*/
}/*else-fopen*/
}/*env_arch3*/

/*****************************************************************************
MÓDULO:          env_final().
OBJETIVO:        Enviar el mensaje de fin de transmisión.
PARÁMETROS:      rpid, identificador del proceso receptor.
SALIDAS:         rpid, si la transmisión se realiza correctamente o ERROR, si existe falla en el proceso receptor.
*****************************************************************************/
int env_final(int rpid)
{
int falla;

strcpy(msg4.final,"ok");
```

```
if(((falla = Send(rpid,&msg4,&msg1,sizeof(msg4),sizeof(msg1))) == -1) {
  alerta1();
  return ERROR;
 }
 else {
  desligado();
  return falla;
 }
}/*env_final*/

/*****************************************************************************************
MÓDULO:          rec_sol().
OBJETIVO:        Recepción de la solicitud de envío de estructuras, por parte de un programa de aplicación, por
                 medio de la IAA. Avisar el inicio de transferencia de estructuras al NA respaldo.
PARÁMETROS:      prim, bandera que indica si es la primera vez que se ejecuta este módulo.
                 *pids y *flgs, apuntadores al arreglo de identificadores de los procesos transmisores de
                 estructuras.
*****************************************************************************************/
void rec_sol(int prim, pid_t *pids, pid_t *flgs)
{
 char hj2[8];
 int i, clnt, pid_nr, hj2_id, caida;

 signal(SIGKILL,SIG_DFL);
 if (prim == TRUE) {
  strcpy(hj2,nodo_tx);
  strcat(hj2,"b");
  hj2_id = qnx_name_attach(0,hj2);
  strcpy(msg1.resp,"ok");
 }
 clnt = Receive(0,&msg2,sizeof(msg2));
 pid_nr = aviso();          /* Aviso De Inicio De Transferencia De Estructuras */
 for (i = 0; i <= 1; i++) {
  caida = Send(*flgs,&msg2,&msg1,sizeof(msg2),sizeof(msg1));
  flgs++;
 }
 if (pid_nr != -1) aviso2(pid_nr);
 Reply(clnt,&msg1,sizeof(msg1));
 rec_sol(FALSE,pids,pids);
}/*rec_sol*/

/*****************************************************************************************
MÓDULO:          env_est().
OBJETIVO:        Llamar al módulo de establecimiento de enlaces con los procesos receptores de estructuras y
                 envío de estructuras hacia Nodos de Presentación.
PARÁMETROS:      i, número de proceso trasmisor de estructura.
                 env_resp, bandera que indica si el proceso transmisor es principal o de respaldo.
*****************************************************************************************/
void env_est(int i, int env_resp)
{
 int rpid, caida;

 signal(SIGKILL,SIG_DFL);
 strcpy(msg1.resp,"ok");
 for (;;) {
  caida = Receive(0,&msg2,sizeof(msg2));
```

```
  rpid = conexion2(i,env_resp);
  if (rpid != -1) {
   Send(rpid,&msg2,&msg1,sizeof(msg2),sizeof(msg1));
   qnx_vc_detach(rpid);
   }
  Reply(caida,&msg1,sizeof(msg1));
  }
}/*env_est*/

/*****************************************************************************************
MÓDULO:         aviso().
OBJETIVO:       Avisar, al Nodo de Adquisición respaldo, que se inicia una transferencia de estructuras.
SALIDAS:        pid_nr, identificador del proceso respaldo o ERROR, si no se encuentra en funcionamiento el
                Nodo de Adquisición respaldo.
*****************************************************************************************/
int aviso(void)
{
 int pid_nr;
 char *na, nr_c[7], buffer[10];

 na = itoa(nr,buffer,10);
 strcpy(nr_c,"/nodo");
 strcat(nr_c,na);
 strcat(nr_c,"a");
 if ((pid_nr = qnx_name_locate(0,nr_c,0,0)) == -1 ) return ERROR;
 else {
  msg10.nid_tx = nid_tx;
  Send(pid_nr,&msg10,&msg1,sizeof(msg10),sizeof(msg1));
  Send(pid_nr,&msg2,&msg1,sizeof(msg2),sizeof(msg1));
  return pid_nr;
  }
}/*aviso*/

/*****************************************************************************************
MÓDULO:         aviso2().
OBJETIVO:       Avisar, al Nodo de Adquisición respaldo, de la terminación de transferencia de estructuras.
PARÁMETROS:     pid_nr, identificador del proceso respaldo.
*****************************************************************************************/
void aviso2(int pid_nr)
{
 strcpy(msg4.final,"ok");
 Send(pid_nr,&msg4,&msg1,sizeof(msg4),sizeof(msg1));
}/*aviso2*/

/*****************************************************************************************
MÓDULO:         control().
OBJETIVO:       Recibir los avisos de inicio y final de transmisión de estructuras, crear el proceso vigía del NA
                respaldado y llamar al módulo de envio de estructuras en caso de falla en el NA respaldado.
PARÁMETROS:     prim, bandera que indica si es la primera vez que se ejecuta este módulo.
                *pids y *flgs, apuntadores al arreglo de identificadores de los procesos respaldo transmisores de
                estructuras.
*****************************************************************************************/
void control(int prim, pid_t *pids, pid_t *flgs)
{
 char hj[8];
 int t, zid, pid1, mpid, spid, hj_id, caida, cliente;
```

```
signal(SIGKILL,SIG_DFL);
if (prim == TRUE) {
 mpid = getpid();
 strcpy(hj,nodo_tx);
 strcat(hj,"a");
 strcpy(msg1.resp,"ok");
 hj_id = qnx_name_attach(0,hj);
 }
spid = Receive(0,&msg10,sizeof(msg10));
Reply(spid,&msg1,sizeof(msg1));
caida = Receive(spid,&msg2,sizeof(msg2));
Reply(spid,&msg1,sizeof(msg1));
zid = fork();
if (zid == 0) vigia(msg10.nid_tx,mpid);
else {
 pid1 = Receive(0,&msg4,sizeof(msg4));
 kill(zid,SIGKILL);
 waitpid(zid,NULL,0);
 Reply(pid1,&msg1,sizeof(msg1));
 if (!strcmp(msg4.final,"fl")) {
  for (i = 0; i <= 1; i++) {
   caida = Send(*flgs,&msg2,&msg1,sizeof(msg2),sizeof(msg1));
   flgs++;
   }/*for*/
  }/*if-msg4.final*/
 cliente = qnx_name_locate(0,"/cliente",0,0);
 Send(cliente,&msg2,&msg1,sizeof(msg2),sizeof(msg1));
 control(FALSE,pids,pids);
 }/*else-zid*/
}/*control*/

/*****************************************************************************************
MÓDULO:          vigia().
OBJETIVO:        Checar el estado operativo del Nodo de Adquisición respaldado, mientras éste realiza una
                 transferencia de estructura y avisar la falla en este nodo.
PARÁMETROS:      ndf, número del NA respaldado.
                 zidd2, identificador del módulo control().
*****************************************************************************************/
void vigia(int ndf, int zidd2)
{
 int fail;
 char *na, nm[8], buffer[10];
 signal(SIGKILL,SIG_DFL);
 na = itoa(ndf,buffer,10);
 strcpy(nm,"/nodo");
 strcat(nm,na);
 strcat(nm,"b");
 for(;;) {
  if ((fail = qnx_vc_name_attach(ndrl,0,nm)) == -1) {
   strcpy(msg4.final,"fl");
   Send(zidd2,&msg4,&msg1,sizeof(msg4),sizeof(msg1));
   }
  else qnx_vc_detach(fail);
 }
exit(0);
}/*vigia*/
```

```
/*****************************************************************************************
MÓDULO:         recibir().
OBJETIVO:       Administrar la recepción de información.
PARÁMETROS:     recep, bandera que indica si exitió falla en el nodo transmisor.
*****************************************************************************************/
void recibir(int recep)
{
 int ok, rpid, proxy;

 term_box_fill(0,0,80,25,TERM_BLUE,1,' ');
 term_box_fill(2,2,76,19,TERM_RED,2,' ');
 term_color(TERM_WHITE);
 term_printf(3,24,2,"RECEPCION  DE  INFORMACION");
 rpid = getpid();
 if((proxy = liga_proxy()) == -1) pausa();       /* Generacion De Un 'Proxy' Para Desbloqueo*/
 else {
   modo();        /* Identificacion De Modo De Trabajo */
   rec_inicio(rpid,proxy,recep);        /* Recepcion Del Inicio De Transmision */
   qnx_proxy_detach(proxy);
   term_color(TERM_BLUE);
   borrar();
   if (pid != -1) {
    switch(mag0.tipo) {
          case 1: term_color(TERM_WHITE);
                  term_printf(11,10,2,"Recepcion de mensaje !");
                  if ((ok = rec_msg()) == -1) recibir(FALSE);
                  else {
                    ok = rec_final();
                    desligado();
                    sleep(5);
                  }
                  break;
          case 2: term_color(TERM_WHITE);
                  term_printf(11,10,2,"Recepcion de comando !");
                  if ((ok = rec_cmd()) == -1) recibir(FALSE);
                  else {
                    ok = rec_final();
                    desligado();
                    sleep(5);
                  }
                  break;
          case 4: term_color(TERM_WHITE);
                  term_printf(11,10,2,"Recepcion de archivo !");
                  if ((ok = rec_arch2()) == -1) recibir(FALSE);
                  else {
                    ok = rec_final();
                    desligado();
                    sleep(5);
                  }
                  break;
        }/*switch*/
   }/*if-pid*/
 }/*else-proxy*/
}/*recibir*/
```

```
/*****************************************************************************
MÓDULO:          modo().
OBJETIVO:        Identificar modo operativo (PP o PR), mientras se recibe información.
*****************************************************************************/
void modo(void)
{
 int resp;
 char *nd_c, nd_s[7], buffer[10];

 nd_c = itoa(nrd,buffer,10);
 strcpy(nd_s,"/nodo");
 strcat(nd_s,nd_c);
 if ((resp = qnx_name_locate(0,nd_s,0,0)) == -1)
  term_printf(6,10,TERM_WHITE,"Nodo #%d en modo PRIMARIO/PRIMARIO del nodo #%d",nid_tx,nrd);
 else
  term_printf(6,10,TERM_WHITE,"Nodo #%d en modo PRIMARIO/RESPALDO del nodo #%d",nid_tx,nrd);
}/*modo*/

/*****************************************************************************
MÓDULO:          rec_inicio().
OBJETIVO:        Recibir el mensaje de inicio de transmisión.
PARÁMETROS:      rpid, identificador del proceso receptor.
                 proxy, identificador de 'proxy' para desbloqueo.
                 recep, bandera que indica si existió falla en el proceso transmisor.
*****************************************************************************/
int prox;
void rec_inicio(int rpid, int proxy, int recep)
{
 int caida;

 signal(SIGINT,disparo);
 prox = proxy;
 term_printf(5,10,TERM_WHITE,"Proceso Receptor esperando inicio de transmision...");
 term_printf(22,15,TERM_WHITE_BG,"Oprima <ctrl><e> para regresar al menu principal.");
 if (recep == FALSE) pid = Receive(0,&msg0,sizeof(msg0));
 if (!strcmp(msg0.inicio,"ok")) {
  term_color(TERM_BLUE);
  borrar();
  term_color(TERM_WHITE);
  term_printf(7,10,2,"Mensaje de inicio de transmision recibido !");
  term_printf(8,10,0,"Informacion para el nodo #%d",msg0.nd_rx);
  term_printf(9,10,0,"Proceso Receptor:   %d",rpid);
  term_printf(10,10,0,"Proceso Transmisor: %d",pid);
  strcpy(msg1.resp,"ok");
  if ((caida = Reply(pid,&msg1,sizeof(msg1))) == -1) {
   alerta2();
   recibir(FALSE);
  }
 }/*if-"ok"*/
 else {
  if ((!strcmp(msg0.inicio,"up")) || (!strcmp(msg0.inicio,"ur"))) {
   strcpy(msg1.resp,"ok");
   Reply(pid,&msg1,sizeof(msg1));
   term_color(TERM_BLUE);
   borrar();
   sonido();
```

```
    if (!strcmp(msg0.inicio,"up"))
      term_printf(21,18,TERM_RED_BG,"AVISO: Nodo respaldado #%d en funcionamiento",nrd);
    if (!strcmp(msg0.inicio,"ur"))
      term_printf(21,19,TERM_RED_BG,"AVISO: Nodo respaldo #%d en funcionamiento",nr);
    pausa();
    desligado();
   }/*if-strcmp*/
  else desligado();
 }/*else-"ok"*/
}/*rec_inicio*/

/*****************************************************************************************
MÓDULO:         rec_msg().
OBJETIVO:       Recibir un mensaje.
SALIDAS:        caida, el identificador del proceso transmisor, si se recibe correctamente la información o
                ERROR, si existe falla en el proceso transmisor.
*****************************************************************************************/
int rec_msg(void)
{
 char resp[30];
 int caida;

 if ((caida = Receive(pid,&msg5,sizeof(msg5))) == -1) {
  alerta2();
  return ERROR;
 }
 else {
  term_printf(12,10,TERM_WHITE,"Mensaje recibido: %s", msg5.mss);
  term_printf(13,10,0,"Respuesta => ");
  term_cur(13,23);
  gets(resp);
  strcpy(msg7.ack,resp);
  caida = Reply(pid,&msg7,sizeof(msg7));
  if (caida == -1) {
   alerta2();
   return ERROR;
  }
  else {
   term_printf(14,10,2,"FIN DE COMUNICACION");
   return caida;
  }
 }/*else-caida*/
}/*rec_msg*/

/*****************************************************************************************
MÓDULO:         rec_cmd().
OBJETIVO:       Recibir un comando.
SALIDAS:        caida, el identificador del proceso transmisor, si se recibe correctamente la información o
                ERROR, si existe falla en el proceso transmisor.
*****************************************************************************************/
int rec_cmd(void)
{
 int caida, error;
 char *tx, buffer[5], comando[30];

 if ((caida = Receive(pid,&msg6,sizeof(msg6))) == -1) {
```

```
  alerta2();
  return ERROR;
 }
 else {
  term_printf(12,10,TERM_WHITE,"Comando a ejecutar: %s", msg6.cmd);
  if (msg6.flag == 0) {
   term_clear(TERM_CLS_SCRH);
   error = system(msg6.cmd);
   if (error == 0) strcpy(msg7.ack,"Comando ejecutado");
   else {
     term_printf(0,0,TERM_WHITE,"                ");
     term_printf(10,25,2,"Comando erroneo !");
     strcpy(msg7.ack,"Comando erroneo");
    }
  }
  else {
    tx = itoa(msg0.ndtx,buffer,10);
    strcpy(comando,msg6.cmd);
    strcat(comando," >//");
    strcat(comando,tx);
    strcat(comando,"/txt.cmd");
    error = system(comando);
    if (error == 0) {
     strcpy(msg7.ack,"Comando ejecutado");
     term_printf(13,10,TERM_WHITE,"Comando ejecutado !");
     }
    else {
     strcpy(msg7.ack,"Comando erroneo");
     term_printf(13,10,TERM_WHITE,"Comando erroneo !");
     }
   }
  caida = Reply(pid,&msg7,sizeof(msg7));
  strcpy(msg7.ack,"");
  if (caida == -1) {
   alerta2();
   return ERROR;
   }
  else return caida;
 }
}/*rec_cmd*/

/*****************************************************************************************
MÓDULO:          rec_arch2().
OBJETIVO:        Recibir el nombre del archivo por transferir, abrir un archivo para escritura y llamar al módulo
                 de recepción de archivo..
SALIDAS:         ERROR, si existe falla en el proceso transmisor.
                 0 si existe falla en la apertura del archivo en el nodo transmisor o si se recibe el archivo en un
                 sólo paquete.
                 ok, valor retornado por el módulo de recepción de archivo.
*****************************************************************************************/
int rec_arch2(void)
{
 int j, ok, caida;

 if ((caida = Receive(pid,&msg9,sizeof(msg9))) == -1) {
  alerta2();
```

```
  return ERROR;
 }
 else {
  strcpy(msg1.resp,"ok");
  if ((caida = Reply(pid,&msg1,sizeof(msg1))) == -1) {
   alerta2();
   return ERROR;
  }
  else {
   if (msg9.falla == TRUE) {
    term_printf(12,10,TERM_WHITE,"Apertura fallida de archivo en Nodo Transmisor");
    term_printf(13,10,2,"FIN DE COMUNICACION");
    return 0;
   }
   else {
    if ((fp1 = fopen(msg9.nombre,"w")) == NULL) {
         sonido();
         term_printf(21,10,TERM_RED_BG,"No se pudo abrir archivo para escritura");
         return ERROR;
         }
    else {
         for (j = 0; j < msg9.indice; j++) fputs(msg9.cadena[j],fp1);
         if (msg9.bandera == TRUE) {
          term_printf(12,10,TERM_WHITE,"Archivo recibido");
          term_printf(13,10,2,"FIN DE COMUNICACION");
          fclose(fp1);
          return 0;
         }
         else {
          ok = rec_arch3();
          return ok;
         }/*else-msg9.bandera*/
        }/*else-fopen*/
   }/*else-msg9.falla*/
  }/*else-caida-reply*/
 }/*else-caida-receive*/
}/*rec_arch2*/

/*****************************************************************************
MÓDULO:          rec_arch3().
OBJETIVO:        Recibir un archivo.
SALIDAS:         0, si se recibe correctamente la información o ERROR, si existe falla en el proceso transmisor.
*****************************************************************************/
int rec_arch3(void)
{
 int j, ok, caida;

 if ((caida = Receive(pid,&msg9,sizeof(msg9))) == -1) {
  alerta2();
  return ERROR;
 }
 else {
  strcpy(msg1.resp,"ok");
  if ((caida = Reply(pid,&msg1,sizeof(msg1))) == -1) {
   alerta2();
   return ERROR;
```

```
    }
    else {
     for (j = 0; j < msg0.indice; j++) fputs(msg9.cadena[j],fp1);
     if (msg9.bandera == TRUE) {
      term_printf(12,10,TERM_WHITE,"Archivo recibido");
      term_printf(13,10,2,"FIN DE COMUNICACION");
      fclose(fp1);
      return 0;
     }
     else {
      ok = rec_arch3();
      return ok;
     }
    }/*else-caida-reply*/
   }/*else-caida-receive*/
}/*rec_arch3*/

/*****************************************************************************************
MÓDULO:          rec_final()
OBJETIVO         Recibir el mensaje de fin de transmisión.
SALIDAS          caida, el identificador del proceso transmisor, si se recibe correctamente la información o
                 ERROR, si existe falla en el proceso transmisor.
*****************************************************************************************/
int rec_final(void)
{
 int caida;
 strcpy(msg1.resp,"ok");
 if ((caida = Receive(pid,&msg4,sizeof(msg4))) == -1)
  return ERROR;
 else {
  caida = Reply(pid,&msg1,sizeof(msg1));
  return caida;
 }/*else-caida*/
}/*rec_final*/

/*****************************************************************************************
MÓDULO:          disparo().
OBJETIVO:        Enviar un 'proxy' de desbloqueo.
PARÁMETROS       sig, número de la señal que generó el envío del 'proxy'.
*****************************************************************************************/
void disparo(int sig) {
 sig = 0;
 Trigger(prox);
}/*disparo*/

/*****************************************************************************************
MÓDULO:          desligado()
OBJETIVO:        'Limpiado' del mensaje de inicio de transmisión.
*****************************************************************************************/
void desligado(void) {
 pid = -1;
 strcpy(msg0.inicio,"");
 strcpy(msg0.nm,"");
 msg0.nd_rx = 0;
 msg0.nd_tx = 0;
}/*desligado*/
```

```
/*****************************************************************************
MÓDULO:         alerta1().
OBJETIVO:       Desplegar un mensaje, indicando la caida del nodo receptor.
*****************************************************************************/
void alerta1(void) {
 sonido();
 term_printf(21,10,TERM_RED_BG,"SE CAYO NODO RECEPTOR O FALLA EN EL CANAL DE COMUNICACION !");
 pausa();
 desligado();
}/*alerta1*/

/*****************************************************************************
MÓDULO:         alerta2().
OBJETIVO:       Desplegar un mensaje, indicando la caida del nodo transmisor.
*****************************************************************************/
void alerta2(void) {
 sonido();
 term_printf(21,9,TERM_RED_BG,"SE CAYO NODO TRANSMISOR O FALLA EN EL CANAL DE COMUNICACION !");
 pausa();
 desligado();
}/*alerta2*/

/*****************************************************************************
MÓDULO:         pausa()
OBJETIVO:       Generar una pausa en la ejecución del mecanismo de comunicación.
*****************************************************************************/
void pausa(void) {
 char ch;
 term_printf(22,19,TERM_WHITE_BG,"Oprima cualquier tecla para continuar...");
 term_cur(22,58);
 ch = getch();
}/*pausa*/

/*****************************************************************************
MÓDULO:         sonido().
OBJETIVO:       Generar una sonida de alerta.
*****************************************************************************/
void sonido(void) {
 sound(1000);
 delay(200);
 nosound();
}/*sonido*/

/*****************************************************************************
MÓDULO:         borrar().
OBJETIVO:       Borrar mensajes.
*****************************************************************************/
void borrar(void) {
 int i;
  for (i = 2; i < 78; i++) {
   term_printf(21,i,0,"_");
   term_printf(22,i,0,"_");
  }
}/*borrar*/
```

REFERENCIAS

[1].- Ruiz Serna, Enrique
Arquitectura distribuida para un sistema de adquisición de datos en tiempo real con un protocolo de comunicaciones por difusión espontánea.
Tesis de Maestría en Ciencias Computacionales, ITESM Morelos, Febrero 1992.

[2].- Ramírez Valenzuela, Carlos E.
Perspectivas de nuevas tecnologías para los SADRES en el IIE.
Boletín IIE, vol. 15, num. 4, 1991.

[3].- Villavicencio Ramírez, Alejandro
Sistema de Adquisición de Datos y Registro de Eventos.
Boletín IIE, vol. 8, num. 2, 1984.

[4].- Documento de especificación CFE ICA-J100 para Centrales Termoeléctricas Normalizadas de 350 MW.

[5].- Manual de Usuario de las funciones de la Interfaz Hombre-Máquina para la Central de Ciclo Combinado de Gómez Palacios, Durango.
Unidad de Resultados de Automatización de Procesos, I.I.E., Agosto 1994.

[6].- Prince, S.M. y Sloman, M.
Communication Requirements of a Distributed Computer Control System.
IEE PROC., Vol. 128, Pt. E, No. 1, January 1981.

[7].- Simón, Gerald A. y Kaufman, David J.
Protocol Specification, Testing and Verification.
North-Holland Publishing Company, 1982.

[8].- Bochmann, Gregor V. y Sunshine, Carl A.
Formal Methods in Communication Protocol Design.
IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980.

[9].- Holzmann, Gerald J.
Design and Validation of Computer Protocols.
Prentice Hall, Software Series, 1991.

[10].- Sommerville, Ian
Ingeniería de Software.
Addison-Wesley Iberoamericana, 1985.

BIBLIOGRAFÍA

1).- Stallings, William
Local Networks, An Introduction.
Macmillan Publishing Company, 1984.

2).- Williams, Theodore J.
The use of digital computers in process control.
Instrument Society of America, 1982.

3).- Pressman, Roger S.
Ingeniería de Software: Un enfoque práctico.
Mc. Graw Hill, 1989.

4).- Fairley, Richard
Ingeniería de Software.
Mc. Graw Hill, 1989.

5).- Lamport, Leslie
Time, Clocks and the Ordering of Events in a Distributed System.
Communications of the ACM, Vol. 21, No. 7, July 1978.

6).- Melliar-Smith, P. y Moser, Louise
Broadcast Protocols for Distributed Systems.
IEEE Transactions on Parallel and Distibuted Systems, Vol. 1, No. 1, January 1990.

7).- Narayan, Ajit P.
Reliable Multidestination Transfer of Data in a Local Area Network.
Phoenix Conference on Computers and Communications, IEEE, March 27-30, 1991.

8).- Gopal, Inder y Jaffe, Jeffrey
Point-to-Multipoint Communication Over Broadcast Links.
IEEE Transactions on Communications, Vol. COM-32, No. 9, September 1984.

9).- Chang, Jo-Mei y Maxemchuk, N. F.
Reliable Broadcast Protocols.
ACM Transactions on Computer Systems, Vol. 2, No. 3, August 1984.

10).- Ananda, A. L. y Tay, B. H.
SRDTP: A reliable datagram transport protocol for distributed computing.
Computer Communications, Vol. 16, No. 4, April 1993.

11).- Kopetz, Hermann y Grünsteidl, Günter
TTP - A Protocol for Fault-Tolerant Real-Time Systems.
Computer, Vol. 27, No. 1, January 1994.

12).- Calo, S. B. y Easton M. C.
A Broadcast Protocol for File Transfers to Multiple Sites.
IEEE Transactions on Communication, Vol. COM-29, No. 11, November 1981.

13).- Ramakrishna, S.
Design of broadcast programming primitives for distributed systems.
Computer Communications, Vol. 16, No. 9, September 1993.

14).- Verissimo, Paulo y Marques, José Alves
Reliable Broadcast for Fault-Tolerance on Local Computer Networks.
Ninth Symposium on Reliable Distributed Systems, IEEE, Los Alamitos Calif., October 1990.

15).- Danthine, André A.
Protocol Representation with Finite-State Models.
IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980.

16).- QNX 4.0 Operating System User's Guide.
Quantum Software Systems Ltd, 1992.

17).- QNX 4.0 System Arquitecture.
Quantum Software Systems Ltd, 1992.

18).- McDowell, S.G.
Watcom C: Languaje Reference & Programmer's Guide.
Watcom Publications Limited, 1991.

19).- Crigger, F. W., Welch, J. W. y Schueler, J. B.
Watcom C: Library Reference for QNX, Vol. I, II.
Watcom Publications Limited, 1991.

20).- Coschi, G. y Schueler, J. B.
Watcom C: Optimizing Compiler and Tools User's Guide for QNX.
Watcom Publications Limited, 1991.

21).- Coschi, G.
Watcom C: Linker User's Guide for QNX.
Watcom Publications Limited, 1991.

22).- Schildt, Herbert
Turbo C/C++.
Mc. Graw-Hill, 1992.

23).- Nance, Barry
Network Programming in C.
Que Corporation, 1990.